MASTER EN SCIENCES INFORMATIQUES

Mise en place d'une architecture distribuée parallèle au service de l'imagerie fonctionnelle par résonance magnétique

Huys, Ludovic

Award date:
2006

Mise en place d'une architecture distribuée parallèle au service de l'imagerie fonctionnelle par résonance magnétique

Ludovic Huys

Mémoire présenté en vue de l'obtention du grade de
Maître en Informatique

Année académique 2005 – 2006

Résumé

Résumé : Les études actuelles en matière d'IRM fonctionnelle sont basées sur des analyses statistiques d'images cérébrales représentant l'activité du cerveau propre à certains protocoles. L'obtention de ces images est, actuellement, un processus relativement long. La raison de ceci n'est pas le temps d'acquisition, qui est le temps où se produit le stimulus propre au protocole de l'expérience. Ce temps est dû au fait que les images demandent un temps de calcul énorme. Avec la technologie actuelle, le résultat de l'analyse n'est pas disponible en temps réel. En effet, les images doivent être transférées manuellement du poste d'acquisition vers un autre poste, afin que le médecin puisse lancer le traitement. S'il était possible de réduire le temps de traitement des images, il serait dès lors possible de mettre une plateforme d'acquisition / traitement des images en temps réel, et ceci afin de réduire les effets de bord en raison de l'anxiété du patient. Le chemin vers l'IRM fonctionnelle en temps réel, comme proposé par Christian Scheiber, est assez pragmatique : implémenter l'ensemble du parcours suivi par l'image, et améliorer la performance des différentes étapes (qui sont, à l'heure actuelle, lancées manuellement par le chercheur). La stratégie de parallélisation est cruciale afin d'améliorer le temps de post-traitement des images. La stratégie actuelle est le développement d'une application destinée à cacher de l'utilisateur les aspects liés à la parallélisation, à l'utilisation de moyens de calcul mutualisés et ceux liés à la technique de stockage des données. Cette technique repose sur une technologie récente, Storage Resource Broker, qui permet d'avoir une interface unique d'accès à des ressources hétérogènes.

Mots clés : IRM fonctionnelle, temps réel, parallélisation, moyens de calcul mutualisés, SRB

Abstract: Current functional MRI scientific studies are based on the statistical analysis of cerebral image maps representing some protocol-specified brain activity. The achievement of those maps, is for the time being, a delayed process not because of the imaging techniques acquisition time (which must be at the time scale of the brain physiological event) but because the images require heavy computational post-treatment in order to obtain the desired maps. With the current technology the maps are not available in real time. Indeed, the maps must be transferred manually from the acquisition post to another post, in order to allow the medical doctor to do the treatment. If it was possible to shorten the post-treatment delay (down to reaching real time fMRI i.e. a couple seconds) it would then be possible to adapt an on going experiment/acquisition in order to avoid undesired patient's side effects due to anxiety. The path to real time fMRI, as proposed by Christian Scheiber is quite pragmatic: implement the full image pipeline and improve the performance of each stage until the quest is fulfilled. In the meantime, the tool (however slow) is still at MD researcher's disposal for their studies. The parallelization strategy seems crucial in order to tackle the delay problem of post-treatment pipeline. The current strategy is to develop an application intended to hide from the user the aspects related to the parallelization, the using of mutual processing middles and the data storage's technique. This technique uses a recent technology, Storage Resource Broker, which allows having a single interface to access data stored in heterogeneous resources.

Keywords: Functional MRI, real time, parallelization, mutual processing middles, SRB

Avant-propos

Nous voudrions remercier les personnes qui ont, de près ou de loin, permis à ce travail d'aboutir. Tout d'abord Mr Vincent Englebert, le promoteur de ce mémoire, pour sa guidance attentive et efficace tout au long de l'élaboration de ce travail ; ensuite les personnes qui ont encadré ce travail, en particulier Messieurs Eric Boix, Pascal Calvat , Eddy Caron, Jean-Yves Nief et Christian Scheiber ; mais aussi les professeurs et assistants des FUNDP qui ont contribué à notre formation de qualité ; sans oublier la personne qui a relu attentivement ce document, Mr Benjamin Noiset ; et enfin nos parents et amis qui nous auront toujours soutenus au long de ces cinq années de vie universitaire.

Glossaire

AFS	Andrew File System, système de fichiers utilisé à l'IN2P3.
BIRNy	Nom du projet destiné à envoyer les images fonctionnelles d'une console d'acquisition vers un serveur SRB.
BQS	Batch Queueing System, Système batch de l'IN2P3.
CC-IN2P3	Centre de Calcul de l'Institut National de Physique Nucléaire et de Physique des Particules.
CERMEP	Centre d'Exploration et de Recherche Médicales par Émission de Positons.
IRMf ou fMRI	Imagerie fonctionnelle par Résonance Magnétique.
LyNDA	Lyon Neuroimaging Database and Application, nom du projet développé dans ce document.
MCAT	Méta-CATalogue, base de données gérant notamment les ressources, utilisateurs et données de SRB.
NFS	Network File System, protocole de partage de fichiers utilisé à l'IN2P3.
SPM	Statistical Parametric Mapping, nom du logiciel de référence pour le traitement des images fonctionnelles.
PSPM	Parallel SPM, nom du logiciel parallèle pour SPM.
PSPMJOB	Parallel SPMJOB, nom de l'application développée dans le cadre du projet LyNDA.

Table des matières

Résumé *3*
Avant-propos *5*
Glossaire *7*
Table des matières *9*
Introduction *13*
Chapitre 1 ..... *15*
IRM fonctionnelle *15*
1.1 Qu'est-ce que l'IRM fonctionnelle ? 15
1.2 Statistical Parametric Mapping (SPM) 16
1.3 Les prétraitements 17
1.3.1 Le réalignement spatial (realign) 17
1.3.2 Le réalignement temporel (slice timing) 17
1.3.3 La normalisation (normalize) 18
1.3.4 Le lissage (smooth) 18
1.4 L'analyse statistique 18
1.5 Récapitulatif 19
1.6 SPM parallèle et parallèle SPM 20
1.6.1 Pourquoi paralléliser SPM ? 20
1.6.2 Solutions existantes 20
Chapitre 2 ..... *21*
Une plateforme au service de la neuroscience *21*

2.1 Design de l'architecture 21
2.2 Exigences et fonctionnalités fournies par l'architecture 23
2.3 Analyse des différents blocs et liens 23
2.4 Perspectives futures 24
Chapitre 3 ..... *25*
Storage Resource Broker (SRB) *25*
3.1 Présentation de SRB 25
3.2 Le méta-catalogue (MCAT) 26
3.2.1 Concepts de MCAT 26
3.2.2 MCAT et SRB 28
3.3 Les caractéristiques principales de SRB 29
3.3.1 Interface unique de stockage 29
3.3.2 Méta-catalogue (MCAT) 29
3.3.3 Hiérarchie de collection 29
3.3.4 Contrôle d'accès hiérarchique 30
3.3.5 Le système de tickets 30
3.3.6 Ressources de stockage 30
3.3.7 Opérations proxy 31
3.3.8 Opérations fédérées 31
3.3.9 Authentification et cryptage 31
3.3.10 Fichiers log 32
3.3.11 Applications 32
3.4 Architecture du système 32
3.4.1 Le service du méta-catalogue 34
3.4.2 Fédération de serveurs SRB 35
3.5 Les modèles de fédération 37
3.5.1 Nomadic Zones (SRB in a Box) 39
3.5.2 Archival Zone (BackUp zone) 40
3.5.3 Replicated Data Zones 41
3.5.4 Master-Slave Zones 42

3.5.5 Snow-Flake Zones ... 44
3.5.6 Replicated Catalog ... 45
3.5.7 Free Floating Zones (my zone) ... 46
3.5.8 User and Data Replica Zones ... 47
3.5.9 Ressource Interaction ... 48
3.5.10 Occasional Interchange ... 50
3.5.11 Schémas récapitulatifs ... 51
3.6 Réplication ... 52
3.7 Performances ... 52
3.8 Design d'un agent SRB ... 54
3.9 API à disposition du client ... 55
3.9.1 (Dé)Connexion au serveur ... 55
3.9.2 Requête / mise à jour de méta-données ... 56
3.9.3 Création d'objets ... 56
3.9.4 Ouvrir / lire / écrire / effacer des objets ... 56
3.10 Les S-Commandes ... 56
3.11 Les clients SRB ... 56
3.12 Extensibilité de SRB ... 57
Chapitre 4 ... *59*
Lyon Neuroimaging Database and Application (LyNDA) ... *59*
4.1 Contexte du projet ... 59
4.2 Architecture de la plateforme LyNDA ... 60
4.3 Analyse de l'existant ... 61
4.3.1 Etat initial de la plateforme ... 61
4.3.1.1 Composant « CERMEP » ... 61
4.3.1.2 Composant « Users – Internet » ... 62
4.3.1.3 Composant « CC-IN2P3 » ... 63
4.3.1.4 Composant « Storage (CC-IN2P3) » ... 64
4.3.1.5 Composant « Processing (CC-IN2P3) » ... 68
4.3.2 Moyens applicatifs ... 69
4.4 Objectifs à atteindre ... 70
4.5 Comment atteindre les objectifs ? ... 71

4.5.1 Objectif « partage de données » 71
4.5.2 Objectif « utilisation d'un programme parallèle » 71
4.5.3 Objectif « Utilisation des ressources mutualisées » 73
4.6 Implémentation 73
4.6.1 Objectif « partage de données » 74
4.6.2 Objectif « utilisation d'un programme parallèle » 78
4.6.3 Objectif « Utilisation des ressources mutualisées » 82
4.7 Application PSPMJOB 85
4.7.1 Interface de base 85
4.7.2 Objectif « partage de données » 86
4.7.3 Objectif « utilisation d'un programme parallèle » 87
4.7.4 Objectif « Utilisation des ressources mutualisées » 87
4.8 Des algorithmes d'envoi de données dans SRB 89
4.8.1 Nature du problème 89
4.8.2 Solutions proposées 89
4.8.2.1 Utilisation de fichiers Tarball 90
4.8.2.2 Découpage du dossier à envoyer 91
4.8.2.3 Comparaison des performances 93
4.9 Perspectives futures de LyNDA 95
Conclusion *97*
Annexe A *99*
Real-Time Functional MRI Using a PC Cluster *99*
Annexe B *103*
A Platform for Distributed Analysis of Neuroimaging Data on Global Grids *103*
Annexe C *108*
Listes des S-Commandes *108*
Annexe D *109*
Exemple de fichiers relatifs à l'exécution d'un job *109*
Bibliographie *115*

Introduction

L'imagerie par résonance magnétique, technique d'imagerie médicale se basant sur la résonance magnétique nucléaire, a pour principal objectif d'obtenir des vues bidimensionnelles ou tridimensionnelles d'une partie spécifique du corps humain. Une de ses applications, l'imagerie fonctionnelle par résonance magnétique (IRMf), a quant à elle pour objectif d'étudier le fonctionnement du cerveau humain.

Pour comprendre l'application de cette dernière discipline, nous pouvons citer l'exemple suivant : « Prenez un être humain, joueur invétéré de casino, montrez lui une machine à sous de casino, et regardez quelle(s) zone(s) de son cerveau est (sont) activée(s) à la vue de cette machine ; ensuite, prenez un autre être humain, non joueur (habituel) de casino, réalisez la même opération, la (les) même(s) zone(s) de est (sont) elle(s) activée(s) ? Dans l'affirmative, ce deuxième être humain est un joueur potentiel de casino ». Cet exemple est volontairement « caricatural » par rapport aux études réalisées dans le domaine de la neuroscience (étude du cerveau), mais pourrait devenir plus crédible avec les avancées futures dans ce dernier domaine. En outre, il a le mérite de montrer l'enjeu des études en matière d'IRM fonctionnelle.

Au-delà des aspects liés au domaine médical en lui-même, un concept est de plus en plus à la mode, celui d'imagerie fonctionnelle par résonance magnétique en temps réel : traitement des images acquises en même temps que leur acquisition, permettant de d'accélérer d'une part les études théoriques en neuroscience, et d'autre part d'avoir une appréciation temps réel de la qualité des images acquises lorsqu'un patient subit une résonance magnétique.

Grâce à une analyse temps réel, le monitoring (qualité, détection du mouvement du patient, …) de l'acquisition des images fonctionnelles par résonance magnétique sera rendu plus performant, permettant aussi une adaptation du protocole d'acquisition au cours de celle-ci (l'IRMf deviendrait ainsi plus flexible et dynamique puisque le chercheur peut décider de changer le protocole d'acquisition en fonction des premiers résultats obtenus).

D'un point de vue clinique, certaines applications de l'imagerie en temps réel pourraient voir le jour. Par exemple, une chirurgien pratiquant une intervention chirurgicale pourrait s'aider d'images obtenues en temps réel. Aussi, il serait possible, via un feedback immédiat de l'activité cérébrale du patient, d'évaluer la nécessité d'un traitement après une perte temporaire de certaines facultés motrices.

Le but de ce mémoire est de présenter les premiers pas vers cette imagerie fonctionnelle par résonance magnétique en temps réel, et ceci dans le cadre d'un projet ambitieux : Lyon Neuroimaging Database and Application (LyNDA). Par premiers pas, nous entendons la mise en place des premières « briques » vers l'IRMf en temps réel, objectif final du projet qui devra être atteint à plus long terme.

Dans un premier temps (chapitre 1), nous présenterons le domaine d'application du projet LyNDA. Lorsque les images ont été acquises dans un centre d'acquisition, celles-ci ne sont pas utilisables en tant que telles, mais doivent subir certaines transformations afin d'être exploitées par la suite. Cette exploitation se fait grâce à un logiciel spécifique, utilisé par la plupart des chercheurs en neuroscience. Nous verrons aussi que des versions parallèles de ce logiciel existent.

Après avoir présenté le « Business » de la neuroscience, nous proposerons une première version de plateforme au service de la neuroscience. Celle-ci sera volontairement simplifiée et contiendra le moins possible de choix technologique, le but étant de montrer les enjeux et les exigences qu'une telle plateforme doit comporter.

Le chapitre trois présentera en détails une technologie de stockage de données : Storage Resource Broker (SRB). Le but de cette technologie est de fournir une interface unique d'accès à des données stockées dans des ressources hétérogènes (systèmes de fichiers divers). Nous passerons en revue les détails relatifs à son élaboration, fonctionnement, et utilisation. Si nous présentons en long et en large cette technologie, c'est tout simplement parce que celle-ci constitue un choix technologie cruciale de l'architecture du projet LyNDA.

Enfin, le dernier chapitre aura pour but de présenter en détails l'architecture de la plateforme mise en œuvre dans le cadre du projet LyNDA. A cette fin, nous présenterons ce qui préexistait à la contribution faisant l'objet de ce mémoire. Nous développerons ensuite les objectifs à remplir, ainsi que le choix réalisés afin de remplir ceux-ci (accompagnés de détails d'implémentation).

Nous exposons enfin notre contribution qui s'est matérialisée par une application à destination des chercheurs en neuroscience, application appelée à se développer dans le futur, comme nous le verrons dans les perspectives futures de LyNDA.

Après avoir présenté les enjeux du présent document, ainsi que le plan de ce dernier, nous pouvons maintenant rentrer dans le vif du sujet.

Chapitre 1

IRM fonctionnelle

Dans ce premier chapitre, nous donnons une définition de l'IRM fonctionnelle, domaine d'application de ce mémoire. Nous nous intéressons ensuite à une brève introduction aux prétraitements d'images fonctionnelles, et à leur analyse. Nous terminons ce chapitre par la présentation de SPM, un logiciel permettant l'analyse d'images d'IRM fonctionnelle, ainsi qu'à PSPM et SPM parallèle, deux versions parallèles de ce dernier.

1.1 Qu'est-ce que l'IRM fonctionnelle ?

Pour comprendre l'IRM fonctionnelle (IRMf), nous pouvons l'envisager à 2 niveaux : le niveau physique et le niveau biologique [DMV99].

Au niveau physique, l'IRMf est fondée sur le principe physique de la résonance magnétique nucléaire, c'est-à-dire l'absorption d'énergie électromagnétique radiofréquence (50-100 MHz) par un matériau dans un champ magnétique intense. Les images produites du cerveau sont de très bonne qualité, précises (de l'ordre de 3mm pour des images fonctionnelles), et acquises en un temps très bref (3 secondes par volume).

Au niveau biologique, l'IRMf se base sur les variations du débit et de l'oxygénation du sang dans le tissu cérébral impliqué dans une tâche cognitive particulière. Vu que la consommation d'oxygène augmente dans une moindre proportion dans ces mêmes régions, il en résulte un excès d'oxyhémoglobine (combinaison de l'hémoglobine avec l'oxygène), qui perturbe les propriétés magnétiques locales et donne lieu à une hyperintensité du signal de résonance magnétique nucléaire. Une soustraction entre une image prise dans un état « activé » et un état de « repos » mettra donc en évidence les zones activées.

L'acquisition des images se déroule suivant un certain protocole (par un imageur par RMN, figure 1.1), définissant les stimulations qui caractérisent l'expérience du point de vue du sujet. L'analyse par la suite se fait bien sûr au regard de ce protocole.

Figure 1.1 - Imageur par RMN

1.2 Statistical Parametric Mapping (SPM)

A la suite de l'acquisition des images fonctionnelles, nous pouvons analyser celles-ci via un logiciel spécifique : SPM (figure 1.2, [SPM]). Ce logiciel a pour but d'identifier les réponses fonctionnelles du cerveau, et est d'ailleurs un moyen répandu d'étude du changement de l'anatomie fonctionnelle[1] dû à certaines maladies. Il analyse les voxels[2] par le biais de méthodes d'inférences afin de mettre en évidence des réponses locales du cerveau aux stimuli propres à l'expérience, c'est ce qui est communément appelé l'analyse statistique.

Mais avant d'en arriver là, il faut pré-traiter les données afin que celles-ci soient conformes à un espace anatomique connu. Sans trop entrer dans les détails liés au domaine médical, en particulier la neuroscience, nous allons définir ce que sont les différents prétraitements. Nous nous pencherons ensuite sur l'étape ultime qu'est l'analyse statistique.

[1] Définition du Wikipédia : L'anatomie est une science descriptive étudiant la structure, la topographie et le rapport des organes entre eux. Elle désigne à la fois la structure d'un organisme vivant et la branche de la biologie (ou de la médecine, pour l'anatomie humaine) qui étudie cette structure. L'anatomie fonctionnelle est liée à une branche de l'anatomie : La neuroanatomie (étude anatomique du système nerveux central et périphérique).

[2] Définition du Wikipédia : Le voxel (contraction de "volumetric pixel") est un pixel en 3 dimensions.

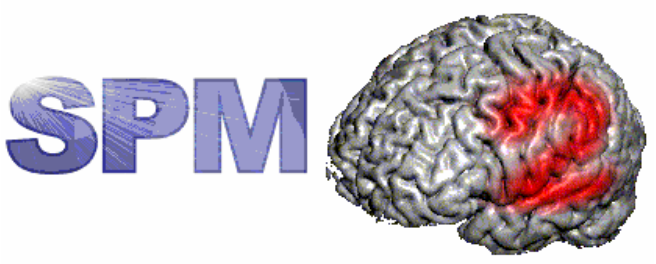


Figure 1.2 – Logo SPM

1.3 Les prétraitements

Nous allons donc nous intéresser, dans un premier temps, aux prétraitements des images, aussi appelées volumes, avant l'analyse statistique. Ces prétraitements ont pour but de réduire des effets indésirables induits par le mouvement du sujet, ou par le nombre de séquences d'acquisition d'images.

1.3.1 Le réalignement spatial (realign)

L'objectif est de corriger les artefacts dus aux mouvements de la tête du sujet, inévitables pendant une acquisition qui peut durer longtemps. Le principe est de prendre, dans chaque série, une image de référence et de recaler les autres images par rapport à celle-ci. Ce recalage peut se faire de deux façons avec SPM2 (Version 2002 de SPM). C'est-à-dire à l'intérieur de chaque série acquise (série d'images par série d'images) ou alors entre les différentes séries acquises (toutes les séries en même temps) pour un même sujet.

Le recalage s'effectue en deux étapes. Une étape "coregister" où une matrice qui contient la transformation à effectuer est calculée pour chaque image. L'autre étape est le "reslice" où la transformation calculée précédemment est appliquée aux images.

Même en cas d'un réalignement d'excellente qualité, il arrive que des effets dus aux mouvements du sujet subsistent. Il faut ajouter à ça le problème dû au temps d'acquisition entre chaque volume. Ceci soulève la nécessité de l'étape suivante : le slice timing.

1.3.2 Le réalignement temporel (slice timing)

Après avoir effectué un réalignement, nous pouvons envisager le *slice timing*. Le problème ici est que, lors d'une acquisition fonctionnelle, les différentes coupes d'un même volume (images correspondantes au même stimulus) ne sont pas acquises simultanément, mais successivement pendant une durée égale au temps de répétition TR (Repetition Time).

SPM se propose de corriger ce décalage lors du prétraitement des images. Lors des analyses statistiques, on pourra considérer alors que toutes les coupes d'un même volume ont été acquises en même temps. Il faut tout de même noter que cette étape n'est pas possible si le TR est trop grand, l'interpolation étant impossible.

Nous savons maintenant comment procéder pour tenir compte du mouvement du sujet, ainsi que de l'ordre d'acquisition des coupes d'un même volume. Il faut maintenant calibrer les images par rapport à un espace connu, d'où la normalisation.

1.3.3 La normalisation (normalize)

Ayant en main des images réalignées, nous devons maintenant ramener chacune d'elles dans un modèle connu, c'est-à-dire un template. Suivant l'étude que mène le chercheur en neuroscience, un template approprié est utilisé.

Une fois que toutes les images sont plongées dans un repère commun, l'étude et la comparaison de celles-ci sont rendues possibles.

Nous pouvons maintenant envisager le dernier prétraitement nécessaire à l'analyse statistique : le lissage.

1.3.4 Le lissage (smooth)

La nécessité de cette étape provient du fait que les données de l'IRM fonctionnelle présentent des corrélations spatiales. Le signal acquis dans un voxel n'est pas indépendant du signal des voxels voisins. On lisse donc les données avec un filtre de façon à ce que, dans les images lissées, les caractéristiques des corrélations spatiales soient connues et imposées par les propriétés du filtre.

Une fois les images réalignées, normalisées et lissées, nous pouvons envisager la partie la plus importante et intéressante du travail, en d'autres termes, l'analyse statistique.

1.4 L'analyse statistique

Nous en arrivons donc à la partie la plus intéressante et cruciale dans la phase d'analyse d'images d'IRMf. C'est ici que nous allons obtenir réponse aux questions d'ordre cognitive, raison d'être de l'expérience.

Ces questions sont, en général, du style de « Quelle(s) zone(s) du cerveau est (sont) activée(s) par tel ou tel stimulus ? Quelle(s) zone(s) est (sont) concernée(s) par un stimulus et pas par un autre ?

L'analyse statistique se déroule en trois temps : développement d'un modèle (modèle général linéaire), indication des images concernées, estimation. Ensuite le chercheur doit interpréter les résultats qu'il a ainsi obtenus.

1.5 Récapitulatif

Ce récapitulatif (figure 1.3) illustre l'enchaînement des étapes d'analyse d'images fonctionnelles (prétraitements et analyse statistique). Les images sont tout d'abord réalignées, normalisées (grâce au modèle de référence) et lissées (grâce à un filtre, *Kernel*). L'analyse statistique (sur base d'un modèle général linéaire) permet ensuite d'obtenir les résultats à interpréter par le chercheur en neuroscience.

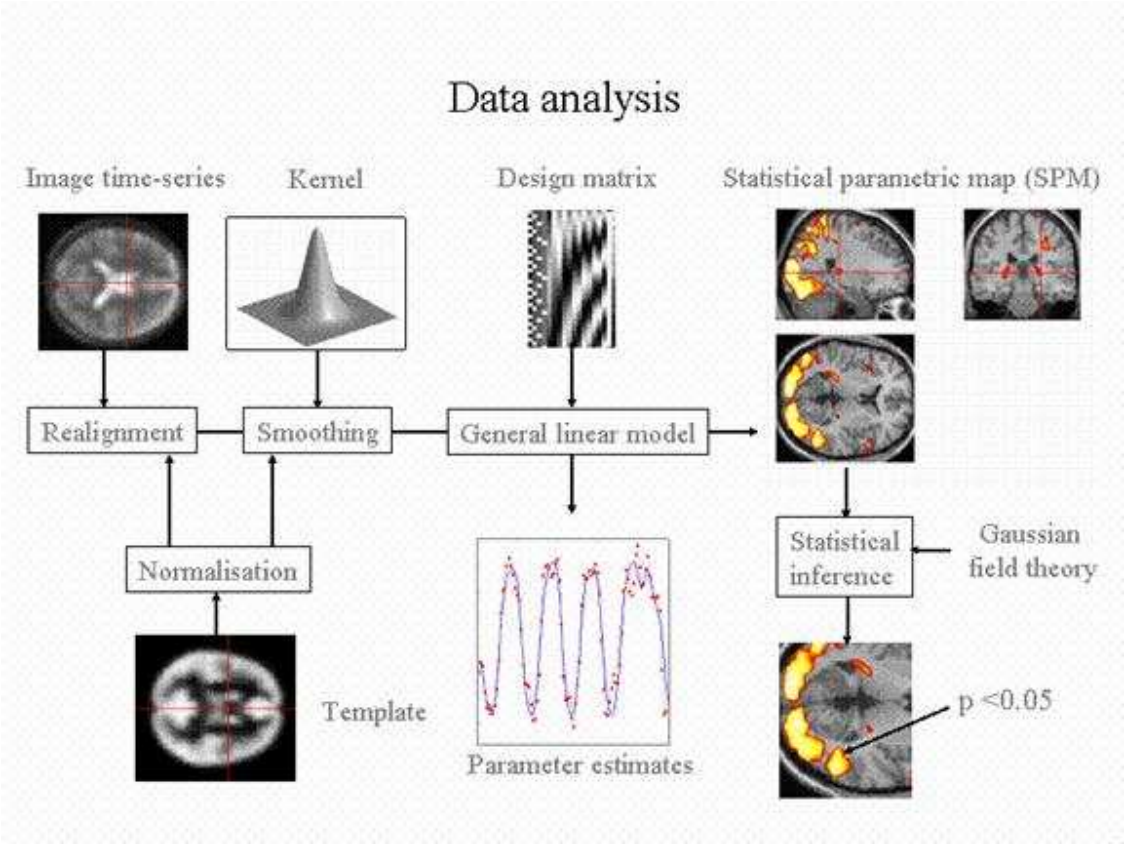


Figure 1.3 - Récapitulatif des prétraitements et statistique

1.6 SPM parallèle et parallèle SPM

1.6.1 Pourquoi paralléliser SPM ?

La justification de la parallélisation des services rendus par SPM est aussi évidente que multiple. Premièrement, la parallélisation efficace d'un logiciel a pour effet d'accélérer l'exécution de celui-ci. Dans le cas de SPM, ce besoin d'accélération provient de 2 besoins. D'une part, un besoin opérationnel puisqu'il est évident qu'un gain de temps n'est nullement refusable. D'autre part, un besoin de recherche puisque le fait d'accélérer le temps de traitement des jobs du chercheur aura pour conséquence d'améliorer sa productivité, celui-ci ayant ses résultats en un temps réduit grâce au gain de la parallélisation.

Deuxièmement, le summum à atteindre dans le domaine de l'IRMf, est de le faire en temps réel. C'est-à-dire, le traitement des résultats en même temps que l'acquisition. Ce qui est plus confortable pour le patient, et plus efficace pour le médecin.

Enfin, la dernière raison est moins évidente, mais pourtant réelle : la rationalisation des ressources informatiques. Les moyens informatiques à mettre en œuvre, tant logiciels que matériels sont colossaux. La solution la plus élégante et efficace est de former une communauté d'utilisateurs de ressources de qualité, maintenues, et faciles à utiliser.

1.6.2 Solutions existantes

Deux solutions de parallélisation de SPM existent actuellement : SPM parallèle et parallèle SPM.

La première est, à la base, une version parallèle de SPM99 (version 1999 de SPM) développée à Strasbourg par Gérald Vétois ([SPMP]). Cette version est encore utilisée à l'heure actuelle sur un cluster de 12 machines (situé à Strasbourg).

La deuxième version est plus récente Elle est compatible avec SPM2 (dernière version de SPM). Mais celle-ci est peu utilisée.

Nous verrons, dans le chapitre 4, les avantages et désavantages de ces 2 solutions, et surtout, nous opterons pour l'une d'elles dans le cadre du projet LyNDA. Mais avant cela, nous allons proposer une première architecture permettant d'exploiter au mieux les outils et concepts que nous avons présentés.

Chapitre 2

Une plateforme au service de la neuroscience

Après avoir étudié les différents outils à disposition des chercheurs en neuroscience, nous proposons une plateforme au service de ces derniers. Celle-ci est définie dans un premier de façon sommaire, puis sera détaillée au chapitre 4.

2.1 Design de l'architecture

Nous allons donc élaborer le design relativement sommaire (un design détaillé sera proposé au chapitre 4) d'une plateforme destinée à satisfaire les besoins des chercheurs en neuroscience.

Plusieurs exigences seront satisfaites par le biais de cette première ébauche. Voici donc une vision schématique de celle-ci (figure 2.1), nous détaillerons par la suite les différents composants et liens qui la composent, de même que les exigences et certaines particularités techniques.

Nous voyons que les images fonctionnelles sont acquises par l'imageur (FMRI system) afin d'être traitées en temps réel ou par la suite (à la demande d'un chercheur). Il est possible de se connecter aux ressources (de calcul, stockage, traitement) via une machine jouant le rôle de passerelle (system interface). Les moyens applicatifs mis en œuvre permettront de sélectionner les données et de les traiter (de manière parallèle ou non) sur une ferme de calcul [1](processing farm).

[1]Définition du Wikipédia : On parle de grappe de serveurs ou de ferme de calcul (cluster en anglais) pour désigner des architectures consistant à regrouper plusieurs ordinateurs indépendants (appelés nœuds, node en anglais) pour permettre une gestion globale et dépasser les limitations d'un ordinateur.

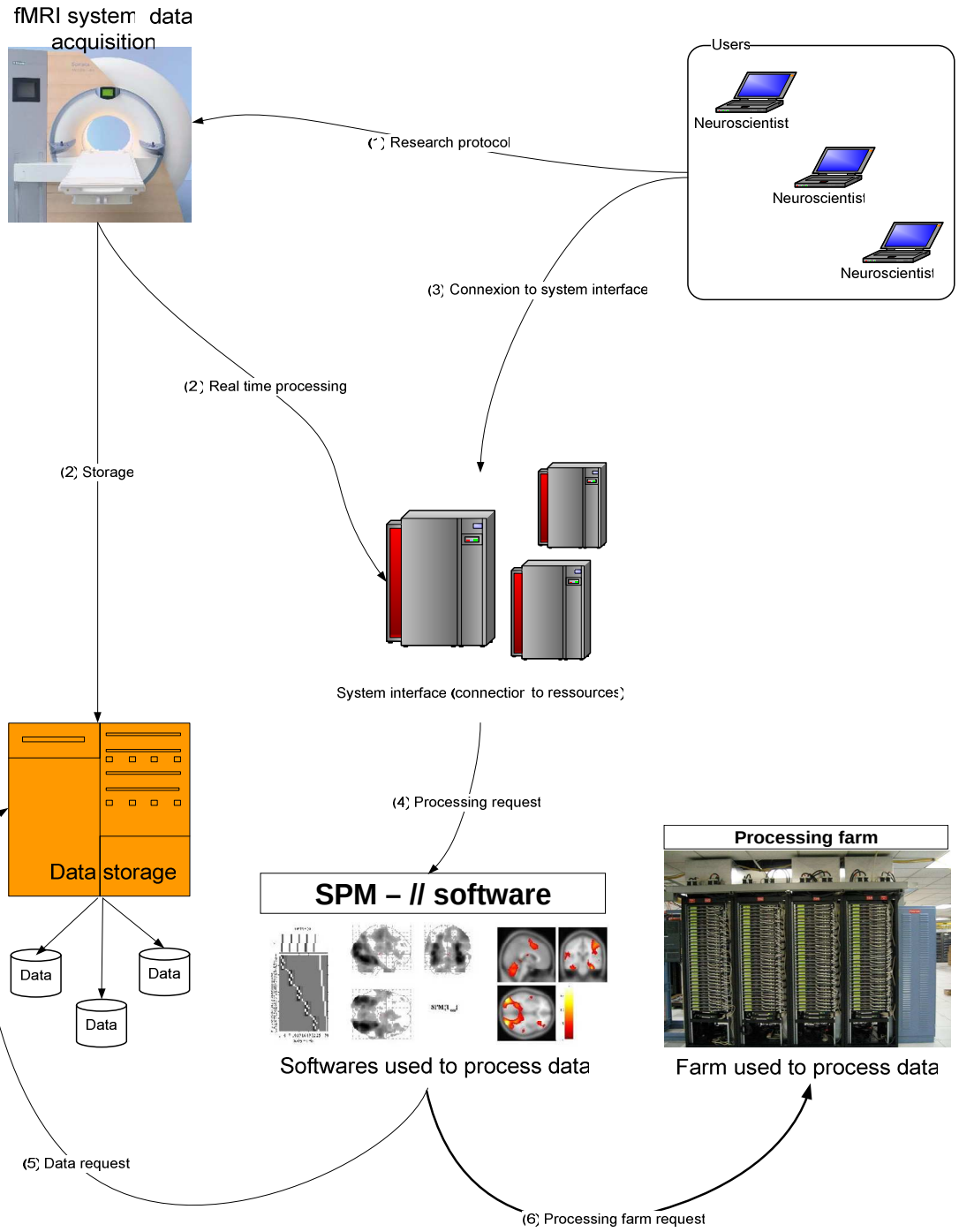


Figure 2.1 – Ebauche d'architecture

2.2 Exigences et fonctionnalités fournies par l'architecture

L'architecture proposée au point 2.1 est porteuse de plusieurs exigences et fonctionnalités, comme :

- Gain de temps (ferme de calcul et application parallèle).
- Gain de sécurité (toutes les ressources sont communautarisées, sous la surveillance d'une autorité compétente).
- Stockage de larges quantités de données, de manière sécurisée. Accès aux ressources 24h/24h.
- Partage des données et de ressources (faible coût pour l'utilisateur).
- Management des données, et ce, de n'importe quel endroit disposant d'une connexion à Internet.
- Temps réel (discuté dans la suite).

Nous voyons donc que les avantages d'une telle plateforme sont aussi nombreux qu'intéressants, tant pour les chercheurs que pour les institutions de recherche. La complexité du système se situe en effet chez l'autorité gérant celui-ci.

2.3 Analyse des différents blocs et liens

Ici, nous allons détailler les différents composants de notre plateforme, ainsi que les liens entre ceux-ci.

2.3.1 Analyse des blocs

- fMRI system : sans entrer dans les détails techniques du domaine, il s'agit d'un imageur par résonance magnétique.
- Users : Il s'agit bien entendu des utilisateurs de la plateforme : chercheurs, techniciens, développeurs, …
- System interface : Il s'agit de l'interface de connexion, à disposition des utilisateurs. Via cette interface, l'utilisateur a accès aux données et ressources de calcul.
- Data storage : Il s'agit du système de stockage (et de partage) des données.
- SPM & // software : SPM est l'application de traitement d'images présentée au chapitre 1. En ce qui concerne l'application parallèle, nous détaillerons cela au chapitre 4.
- Processing farm : ferme de calcul destinées à fournir les capacités de calcul nécessaires à SPM et à l'application parallèle.

2.3.2 Analyse des liens

- (1) Research protocol : l'acquisition des images fonctionnelles via l'imageur se fait toujours au regard d'un protocole d'acquisition (stimulus de l'expérience, …).
- (2) Storage : une fois l'acquisition des images terminée, celles-ci peuvent être stockées dans un système de stockage. Ce système de stockage de données fera l'objet du chapitre 3 : Storage Resource Broker.
- (2') Real time processing : l'analyse des images fonctionnelles en temps réel est une des perspectives futures de notre architecture. Par temps réel, nous entendons traitement des images en même temps que leur acquisition.
- (3) Connexion to system interface : les utilisateurs de notre plateforme se connecteront concrètement à celle-ci via une connexion SSH sur une machine « passerelle ».
- (4) Processing request : les utilisateurs, ayant accès aux données et capacités de calcul, pourront utiliser des logiciels comme SPM afin d'effectuer certaines opérations sur des données choisies préalablement.
- (5) Data request : comme nous l'avons annoncé au point précédent, des données peuvent être sélectionnées dans les ressources de stockage.
- (6) Processing farm request : les logiciels utilisés par les chercheurs nécessitent très souvent une puissance de calcul considérable, d'où l'utilisation d'une ferme de calcul.

2.4 Perspectives futures

La finalité ultime d'une telle architecture a déjà été annoncée : une analyse des images en temps réel. En guise d'état de l'art en la matière, nous présentons les résultats de travaux ayant des objectifs similaires (annexe A [BMN03] et annexe B [KMLSBE05]).

Le chapitre 3 est destiné à présenter et détailler SRB (Storage Resource Broker) : une architecture permettant le stockage et le partage de données, qui se révélera intéressante pour notre brique « Data Storage ». Le chapitre 4 quant à lui présente tous les détails concernant les différentes « briques » à mettre en place.

Chapitre 3

Storage Resource Broker (SRB)

Ce chapitre est destiné à présenter, en détails, Storage Resource Broker [SRB]. Nous étudions ses caractéristiques, son architecture et les modèles de fédération de serveurs SRB.

3.1 Présentation de SRB

Les équipes de San Diego Supercomputer Center (SDSC) sont impliquées dans le développement d'une plateforme distribuée offrant des capacités de calcul conséquentes, et cela, dans le contexte de leur partenariat avec la NSF (National Science Fondation), et plus particulièrement dans le cadre du projet NPACI (National Partnership for Advanced Computational Infrastructure). Une des exigences de ce projet est d'avoir une capacité de stockage très large, permettant de stocker, retirer, trier de larges quantités de données. C'est donc dans cette optique que SDSC a développé Storage Resource Broker [SRB].

SRB propose une API (Application Program Interface) unique d'accès à des données stockées dans des ressources de stockage hétérogènes. De plus, il est possible de découvrir, chercher, sélectionner des données se trouvant dans des ressources ou des sites différents, et ce, grâce à un concept-clé de cette technologie : le méta-catalogue (noté MCAT). Avant d'aller plus loin dans la découverte de SRB, nous allons nous intéresser à ce MCAT.

3.2 Le méta-catalogue (MCAT)

3.2.1 Concepts de MCAT

Le concept de MCAT a été élaboré dans le cadre de la création d'un MDAS (Massive Data Analysis System, [CFLMMRW96]), une plateforme sur laquelle on trouve notamment des API destinées aux programmeurs, des aspects liés au fonctionnement du MDAS (matérialisées sous forme de processus démons), une collection de drivers liés aux différents types de ressources de stockage et, bien entendu, le méta-catalogue.

Une exigence importante d'un tel système est qu'il soit « scalable », c'est-à-dire qu'il supporte le management, l'ordonnancement (gestion des réplicas, voir plus loin), la découverte, l'insertion, la suppression et le déplacement de larges quantités de données, et ce, au travers de ressources hétérogènes. D'autre part, il doit être capable d'invoquer lui-même des méthodes propres au système, allouer un ensemble de ressources (par exemple, pour garantir une demande d'enregistrement de fichier d'un utilisateur), accéder à des données de manière efficace, permettre le transit de données entre zones d'archivage et temporaires (ou tout simplement entre deux ressources quelconques), fournir des mécanismes de réplication.

Le MDAS définit 4 types d'entités système : Data Sets, Servers, Methods, Users.

- Data Sets

  Ce sont des entités accédées par des méthodes. Ces entités sont des données stockées dans divers systèmes de fichiers, base de données,… Des informations sont maintenues sur ces entités : nom, localisation, taille, format, droits d'accès, informations sur le partitionnement, structure, historique d'accès (d'un ou plusieurs utilisateur(s)). L'historique d'accès aura pour but de décider quand mettre un data set en cache, ou quand il doit être répliqué.

- Servers

  Ceux-ci incluent des capacités de calcul et de stockage. Celles-ci ont une capacité particulière qu'il faut contrôler afin de faire un ordonnancement efficace.

- Methods

  Les méthodes sont les macros, programmes et utilitaires utilisés pour manipuler les data sets. Elles peuvent être propres à chaque utilisateur ou communes.

- Users

  Les utilisateurs peuvent accéder au système de différentes manières, comme un portail web, un accès direct via une application ou un DBMS (DataBase Management System), et ce, de manière anonyme (mais pouvant s'enregistrer et recevoir un niveau d'accès et de service standard) ou authentifié (quelqu'un déjà enregistré et recevant ses services et son niveau d'accès). Le niveau de service détermine les priorités de l'utilisateur et ses contraintes d'utilisation de ressources.

Son niveau d'accès détermine ses droits d'accès aux ressources. Un historique d'accès aux ressources, méthodes, data sets est enregistré pour chaque user.

Des méta-données particulières sont définies pour chacun de ces types d'entité. Le système doit fournir la capacité d'insérer, mettre à jour, stocker et interroger l'ensemble des méta-données enregistrées pour toutes les entités du système. Les méta-données peuvent être enregistrées à l'installation du système, ou par après, par des utilisateurs ayant les droits liés à cette tâche.

L'ensemble des services que nous avons décrit doit être accessible via une API fournissant plusieurs types de transparence :

- La transparence de localisation

  Les données doivent être identifiables par leur contenu intrinsèque, ou par leurs méta-données spécifiques, et non par leur localisation exacte (comme une URL, un chemin absolu vers un fichier). En outre, des optimisations doivent être fournies quant à l'accès à des fichiers qui peuvent être répliqués en zone cache ou sur différents sites.

- La transparence du format

  Il est possible que plusieurs ensembles de données soient sémantiquement les mêmes, mais différents en terme de leur représentation interne (exemple : un fichier stocké en format doc, html, pdf, ps …). Le système doit donc maintenir des méta-données liées au format des données. Ce qui permet de répondre aux requêtes en fournissant des données aux bons formats, et, au besoin, de faire appel à des mécanismes de conversion entre formats.

- La transparence de la méthode

  Pour une requête dite de haut niveau (par exemple, une fonction de haut niveau appelée par l'utilisateur), différentes implémentations peuvent exister sur le système. Le système doit être capable d'exécuter la méthode (quasi) optimale, selon des critères propres à la méthode ou définis par l'utilisateur. Pour atteindre cette exigence, des méta-données peuvent renseigner le speed-up (gain de temps) et le scale-up (réaction due à la montée en charge) dues à l'appel d'une méthode (par exemple, il sera plus rapide de lire un fichier sur une zone cache que sur une zone d'archivage).

- La transparence de la ressource

  L'utilisateur doit pouvoir spécifier quelle ressource il désire utiliser. Mais d'un autre coté, il doit pouvoir laisser le système choisir la ressource (quasi) optimale à utiliser. Une fois de plus, le système fournit un tel mécanisme grâce à certaines méta-données (Comme un historique de charge d'une ressource, une estimation du temps de queue pour lire dans une ressource, …).

- La transparence de l'utilisateur

  Pour certaines requêtes d'utilisateurs, il peut arriver que celles-ci fassent appel à des données stockées dans différents domaines (se situant sur divers lieux géographiques). Le problème est que ces domaines peuvent avoir des niveaux de sécurité différents.

  Le système de « ticket » est donc destiné à palier à cette difficulté. L'utilisateur faisant la requête doit accéder aux données des multiples sites de manière transparente en termes d'authentification et de sécurité. Si des données doivent être chargées à partir d'un autre domaine, c'est une méthode qui se charge de s'authentifier à la place de l'utilisateur. Ceci permet d'éviter de copier les fichiers.

  La notion de « user space » remplace la notion de home directory Unix. Il n'y a pas de notions de « file » et de « directory », nous en reparlerons plus loin (cf. section 3.3.3). L'utilisateur se réfère toujours à son « user space », qui ne désigne pas nécessairement une localisation physique particulière.

3.2.2 MCAT et SRB

Pour rappel, SRB est un middleware via lequel une quelconque application peut accéder à des ressources hétérogènes (figure 3.1). Le méta-catalogue maintient des méta-données descriptives et systèmes sur le contenu de collection de données ou de données individuelles. Les méta-données descriptives sont celles qui décrivent le contenu, alors que les méta-données systèmes renseignent sur la localisation et les droits d'accès. Pour rappel, il existe aussi des méta-données pour les autres entités telles que les utilisateurs ou ressources de stockage. Tout ceci permet donc un accès aux données basé sur les attributs de celles-ci. Une application désirant accéder à des données rangées dans SRB ne se voit donc pas obligée de fournir des informations de bas niveau (chemin physique, …) sur celles-ci, mais doit simplement fournir des méta-données pertinentes sur ce qu'elle désire, cette application pourra en outre découvrir de nouvelles données dynamiquement.

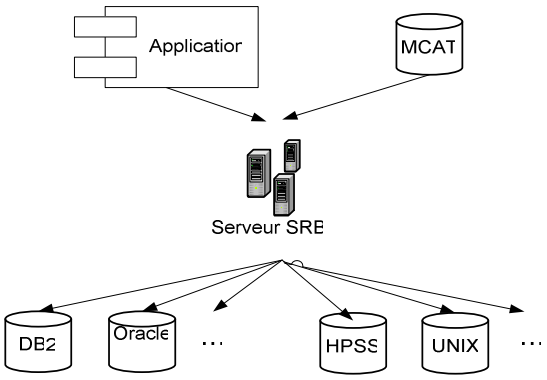


Figure 3.1 : MCAT et SRB

3.3 Les caractéristiques principales de SRB

3.3.1 Interface unique de stockage

Comme nous l'avons vu, SRB propose une interface unique d'accès à des ressources de stockage hétérogènes. L'hétérogénéité des ressources implique généralement des fonctions différentes pour copier, déplacer, enlever, … des données sur celles-ci, mais SRB se propose de masquer cette complexité via une interface unique. L'utilisateur manipulera donc l'ensemble de ses données de la même manière, indépendamment des types de ressources concernées par ses requêtes.

Dans sa configuration actuelle, SRB propose une interface input/output proche de celle utilisée par UNIX, proposant par exemple les fonctions « get » et « put » (envoi et retrait d'objets dans SRB). Le middleware SRB réalise donc la correspondance entre la fonction de haut niveau (le « get » par exemple) et la fonction de bas niveau proposée par le(s) constructeur(s) de(s) ressource(s) concernée(s) par la requête. Ceci est rendu possible par un ensemble de pilotes implémentant les interfaces d'accès aux ressources de stockage.

3.3.2 Méta-catalogue (MCAT)

Nous avons présenté le méta-catalogue précédemment (cf. section 3.2).

Pour rappel, le MCAT permet de supporter l'accès aux données sur base des attributs de celles-ci. Une API fournit les fonctions nécessaires à la manipulation des méta-données. Chaque objet dans SRB se voit attribuer des méta-données descriptives et/ou systèmes. En ce qui concerne les méta-données descriptives, un schéma standard existe, proposant des attributs proches des spécifications prônées par le Dublin Core ([DCM]).En guise de dernière remarque, signalons que le nombre de méta-données peut varier en fonction du type de ressources.

3.3.3 Hiérarchie de collection

Que l'on stocke des données sur son propre poste de travail ou dans un système plus évolué comme SRB, celles-ci doivent être rangées de manière à être lues facilement. SRB a donc choisi une manière identique à celle que l'on trouve sur un système de fichiers comme Unix.

En effet, les données sont stockées sous forme de hiérarchies de collections et sous collections, respectant les spécifications suivantes :

- Une collection (collection) contient un nombre (plus grand ou égal à zéro) de sous-collections et/ou de données (data).

- Une sous-collection (sub-collection) contient un nombre (plus grand ou égal à zéro) de sous-collections et/ou de fichiers (data).
- Une donnée est un fichier ou BLOB (Binary Large OBject, norme de stockage d'informations mixtes (visuelles, sonores, textes) définie par la firme Borland).

3.3.4 Contrôle d'accès hiérarchique

De manière similaire à Unix, le contrôle d'accès est hiérarchique. C'est-à-dire que les droits d'accès peuvent varier de collections en sous-collections. De plus, l'utilisateur peut changer les privilèges à son gré. L'ensemble des privilèges est extensible ([BR98]).

3.3.5 Le système de tickets

Le système de tickets a été créé par l'équipe de San Diego Supercomputer Center ([SDSC]) afin de raffiner les droits de lecture sur les données dans SRB.

Un ticket permet à un utilisateur d'octroyer des droits de lecture à d'autres utilisateurs, mais sous contrainte : le ticket est valable seulement durant un certain intervalle de temps ou pour un nombre d'utilisation limité.

Pour émettre un ticket à destination d'utilisateurs ou groupes d'utilisateurs, un utilisateur doit jouir du droit de contrôle ([BR98]) sur la donnée ou collection.

Le ticket est implémenté sous forme d'une chaîne de caractères (contenant des chiffres, lettres, et caractères de ponctuation) et transmis au destinataire qui se voit octroyé le droit d'ouvrir et lire des objets dans SRB.

Signalons que le méta-catalogue distingue les utilisateurs enregistrés (pour lesquels il possède des méta-données) et non enregistrés (inconnus du MCAT). Ces types d'utilisateurs peuvent recevoir des tickets, mais ceux de la deuxième catégorie devront suivre une procédure de connexion distincte et se verront octroyer des droits restrictifs (pas de home directory, …).

3.3.6 Ressources de stockage

Les concepteurs de SRB ont défini 2 types de ressources de stockage :

- Les ressources physiques parmi lesquelles on retrouve 2 catégories :
    - Celles ayant une interface de type système de fichiers, définies par le doublon (nom d'hôte, chemin) spécifiant un répertoire sur la machine possédant la ressource.

    - Celles ayant une interface de type base de données, définies par le triplet (nom d'hôte, identifiant de base de données, identifiant de table) spécifiant une table d'une base de données sur la machine possédant la ressource.

- Les ressources logiques sont définies comme une combinaison de plusieurs ressources physiques. Par conséquent, une ressource logique peut consister en toute combinaison des ressources physiques citées ci-dessus. Typiquement, un client se réfère toujours à une ressource logique dans laquelle il peut répliquer ses données et utiliser par la suite le réplica qu'il désire.

3.3.7 Opérations proxy

Les opérations « proxy » sont celles qui sont réalisées sans implication du client, c'est-à-dire sans flux de données entre le client et son serveur (ou inversement). On voit donc que le serveur réalise une opération à la place du client.

Des exemples sont notamment les opérations « move » et « copy » entre ressources physiques, où les fichiers concernés ne transiteront pas par le client.

3.3.8 Opérations fédérées

L'une des grandes caractéristiques de SRB est de fournir une plateforme distribuée. Il est donc possible, à partir d'un serveur, d'aller interroger un autre serveur SRB. Ce mécanisme est appelé fédération de serveurs (cf. section 3.4.2).

Nous savons qu'un serveur SRB contrôle un ensemble de ressources physiques, et que les clients de ce serveur ont accès à celles-ci. Il est possible pour un client si, toutefois, il a les droits d'accéder aux ressources d'un autre serveur. Son serveur d'origine jouera le rôle de client envers les autres serveurs de l'environnement distribué.

3.3.9 Authentification et cryptage

SRB supporte trois types d'authentification.

Premièrement, une authentification basée sur un simple mot de passe, mais ce système est progressivement retiré.

Deuxièmement, le système d'authentification appelé Encrypt1. Ce système est aussi basé sur un mot de passe, qui est utilisé dans un protocole de challenge-réponse de manière à ce qu'aucun mot de passe en clair ne passe sur le réseau. Un algorithme de hachage est appliqué sur le mot de passe avant l'envoi, et aucun message n'est crypté vu qu'aucun mot de passe en clair n'est visible sur le réseau (donc l'attaque par écoute n'est pas applicable). Le mot de passe

est enregistré dans le MCAT et, par convention, dans le répertoire « $HOME/.srb/.MdasAuth[1] » de l'utilisateur.

La troisième méthode utilise GSI (Grid Security Infrastructure, [GSI]) de Globus. Ceci requiert que l'utilisateur se procure un certificat.

3.3.10 Fichiers log

Dans le cadre de n'importe quel système, et d'autant plus si celui-ci est distribué, il est primordial d'enregistrer l'ensemble des activités dans des fichiers log. Ceux-ci jouent un rôle primordial, surtout pour la récupération d'erreurs.

Dans SRB, il est possible de garder trace de tout changement des méta-données, et ce, dans le MCAT même. C'est ainsi que, lors de la création de collections ou de données, l'utilisateur peut spécifier s'il désire enregistrer de l'information pour toute manipulation (même une simple lecture) de son objet. Toutefois, les autres utilisateurs peuvent désactiver ce mécanisme, le propriétaire n'en sera pas averti.

3.3.11 Applications

Le but principal du middleware développé par l'équipe du SDSC est clair : rendre possible, de manière transparente et facile, l'accès à d'immenses quantités de données dispersées sur un grand nombre de serveurs déployés sur le globe.

Dans le cas précis de la neuroscience, un chercheur pourrait accéder à un grand nombre d'images sur base d'une seule requête. Mais, à l'heure actuelle, chaque application doit garder de l'information précise sur les fichiers manipulés (chemin, type, …), mais aussi sur la manière d'y accéder (driver spécifique au type de stockage). De plus, il n'est souvent pas possible d'accéder à des données sur des serveurs externes.

SRB permet quant à lui de construire des requêtes ad hoc, basées sur les méta-données des fichiers nécessaires. Ces fichiers sont stockés dans un environnement hétérogène et distribué.

3.4 Architecture du système

Etudions à présent plus en détail l'architecture du middleware SRB. Ce dernier est constitué de deux types de processus :

[1] Par convention, les fichiers sensibles de l'utilisateur (contenant les informations de connexion, dont le mot de passe) sont situés dans un répertoire nommé « .srb », dans le home directory cet l'utilisateur (par exemple « /home/lhuys/.srb/ »).

- Des SRB masters : démons identifiés par le doublon (Nom d'hôte, numéro de port) et contrôlant des ensembles distincts de ressources physiques (Inversement, des ressources physiques distinctes sont contrôlées par des SRB masters distincts). Son rôle principal est d'accueillir les connexions des clients (via son port unique d'écoute).
- Des SRB agents : chaque SRB master a autorité sur un ensemble de SRB agents (et un SRB agent est sous l'autorité d'un seul SRB master), qui sont les démons[1] avec lesquelles les clients discutent après leur connexion.

Voici le déroulement classique de la connexion d'un client à son serveur SRB (figure 3.2) :

- Etape 1 : le client se connecte à son SRB master, qui authentifie le client (vérifie si le client est enregistré dans le MCAT). Remarque : toutes les communications client – SRB master/agent se font via des sockets TCP/IP.
- Etapes 2 et 3 : le master charge un de ses agents de gérer les requêtes du client, et renvoie les coordonnées de cet agent au client (chaque connexion d'un même client est desservie par un agent différent).
- Etapes 4 : le client communique avec son agent pour l'ensemble de ses requêtes.

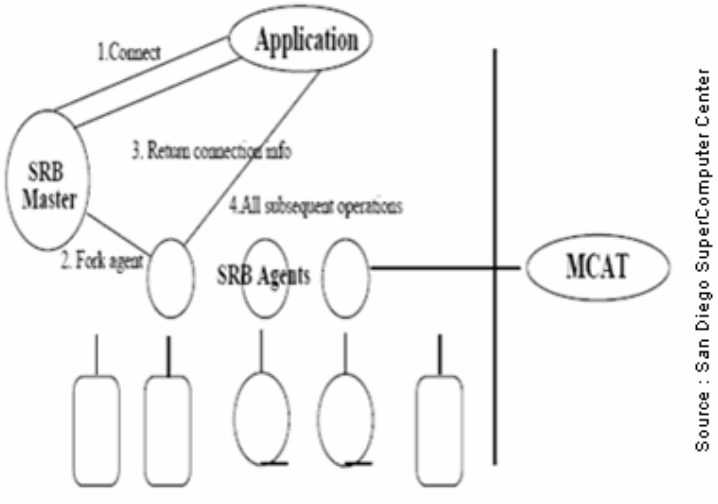


Figure 3.2 : connexion au serveur SRB

Un client ne peut établir qu'une seule connexion avec un master, mais peut très bien établir des connexions différentes (et concurrentes) avec des masters distincts.

[1] Processus système (Linux) qui est exécuté en arrière plan (il n'est pas sous le contrôle direct d'un utilisateur particulier du système).

Les agents SRB utilisent le MCAT afin d'acquérir les méta-données nécessaires au bon déroulement des requêtes des clients. Par exemple, si un client demande de charger un fichier, l'agent SRB devra interroger le MCAT afin de connaître dans quelles ressources physiques (éventuellement logiques) se situe ce fichier afin de connaître la méthode d'interrogation de la ressource physique (pilote à utiliser, localisation …). Par contre, si l'objet demandé par le client se situe sur une ressource non contrôlée par son SRB master, c'est le mécanisme de fédération de serveurs SRB qui est utilisé (voir plus loin).

Nous allons donc maintenant analyser plus en détail comment le méta-catalogue est utilisé, comment des serveurs peuvent se fédérer, la mise en œuvre plus précise d'un agent SRB, ainsi que les API à disposition du client.

3.4.1 Le service du méta-catalogue

Notons tout d'abord qu'il existe une différence majeure entre un fichier stocké dans SRB et un fichier stocké sur son propre poste de travail. Comme sur une machine locale, un fichier est identifié par son nom de collection (dossier), son nom de sous-collection (sous-dossier) ainsi que son nom de fichier. Mais la différence est que, à l'inverse d'une machine locale, le nom de collection (et de sous-collection) et le nom de fichier n'impliquent pas la localisation physique. En effet, si nous prenons comme exemple un fichier stocké dans un home directory Unix comme « /home/lhuys/toto.txt », nous savons que le fichier toto.txt est stocké sur le disque hébergeant le /home. Par contre, si nous prenons le fichier « /neuro/home/lhuys.isc/toto.txt [1]», le fichier toto.txt peut être répliqué sur différents types de systèmes de fichiers (et donc différentes ressources physiques).

Le méta-catalogue aura donc pour but d'enregistrer l'information relative à la localisation des ressources physiques, ainsi que des fichiers stockés dans celles-ci. Il contient aussi les méta-données relatives à l'implémentation du contrôle d'accès hiérarchique aux données, de la hiérarchie de collections / sous-collections et du système de tickets. Enfin, il permet d'enregistrer des méta-données décrivant le contenu des (sous-) collections et données, ce qui permet d'accéder à des données sur base de leurs attributs (méta-données) plutôt que leur nom.

Dans les anciennes versions de SRB, le MCAT était considéré comme une ressource centrale, ce qui constituait bien sûr un goulot d'étranglement, un point d'échec unique et cela engendrait un temps de réponse trop long. Les nouvelles versions supportent la réplication du MCAT, ce qui implique que les applications utilisent un MCAT dépendant du SRB master auquel elles se connectent. L'équipe de San Diego est même allée plus loin, puisqu'elle a créé le mécanisme de fédération de serveurs SRB.

[1] « /neuro » désigne la zone SRB « neuro ». La notion de zone SRB est définie dans la section 3.4.2.

3.4.2 Fédération de serveurs SRB

L'idée de base de la fédération est la coopération entre plusieurs serveurs SRB, ce qui permet d'accroître le nombre d'objets qu'un utilisateur peut percevoir via sa vue logique.

Nous savons que chaque SRB master est responsable pour un ensemble de ressources physiques. La répartition de ces dernières se fait principalement pour des raisons administratives et techniques, comme :

- Si des ressources physiques se trouvent dans des domaines administratifs différents, il est possible que chacun d'entre eux veuille administrer son propre domaine SRB (et donc disposer de son propre SRB master).
- Dans l'optique d'une amélioration des performances de SRB, il est préférable de lancer plusieurs masters (s'occupant de ressources différentes). Un client accédera donc à plusieurs ressources via des masters distincts.
- Il est possible aussi de lancer des masters sur des hôtes différents (améliorant la disponibilité du système), en répliquant les données sur les ressources physiques contrôlées par les différents masters. La réplication pose bien entendu le problème de la synchronisation, ce mécanisme est géré par SRB de la façon suivante : quand une application crée une donnée destinée à être répliquée sur différentes ressources physiques, SRB s'assure de l'écriture dans l'ensemble de celles-ci. De plus, si un problème survient avec un réplica (échec d'écriture, de création, …), SRB active un flag « inconsistent » sur ce réplica, empêchant toute lecture future de celui-ci. Il est néanmoins possible de synchroniser les réplicas « inconsistent » avec les réplicas « consistent », plus particulièrement avec celui qui a été modifié le dernier.
- Certains systèmes de stockage sont peu efficaces en mode client-serveur, il est préférable qu'un master tourne sur le même hôte que ceux-ci.
- Avec des systèmes comme HPPS (Hight Performance Storage System, [HPSS]), le SRB master doit être situé dans le même DCE (Distributed Computing Environment) que le système HPSS (car les utilisateurs de la ressource HPSS doivent être enregistrés dans ce même DCE). Donc, si plusieurs systèmes HPSS sont utilisés, il faut plusieurs masters.

Voici un exemple (figure 3.3) d'opération fédérée (ou le mécanisme de fédération est mis en œuvre), en faisant abstraction de la connexion à SRB (celle-ci ayant été fructueuse préalablement) :

- Étape 1 : Le client demande l'ouverture d'un fichier (en lecture), nommé fic.
- Étape 2 : L'agent SRB (sur l'hôte A) utilise le méta-catalogue afin de déterminer que fic se situe sur une ressource physique contrôlée par un master de l'hôte B.

- Étape 3 : L'agent de l'hôte A se connecte sur le master de l'hôte B (qui l'authentifie), et lui passe la requête originale. Un agent de l'hôte B lui retournera les informations nécessaires à la lecture du fichier.
- Étape 4 : L'agent de l'hôte A envoie ces informations au client.
- Étape 5 : Le client peut lire le fichier, grâce aux informations reçues.
- Remarque : au début, le client peut passer une option permettant de recevoir les coordonnées du master de l'hôte B. Ceci lui permettra de se connecter (s'il a les droits) directement sur l'hôte B, outrepassant ainsi le mécanisme de fédération.

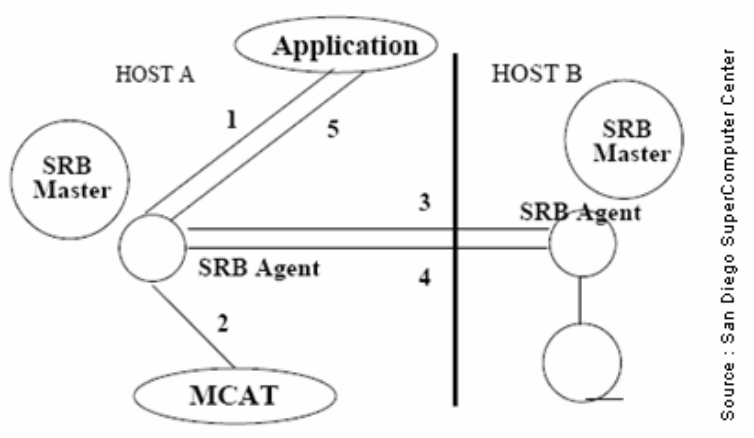


Figure 3.3 : Exemple d'opération fédérée

La manière dont les serveurs SRB peuvent se fédérer a évolué avec le temps. Dans les versions précédentes, tous les serveurs communiquaient avec le même MCAT dans le but d'échanger leurs informations (figure 3.4).

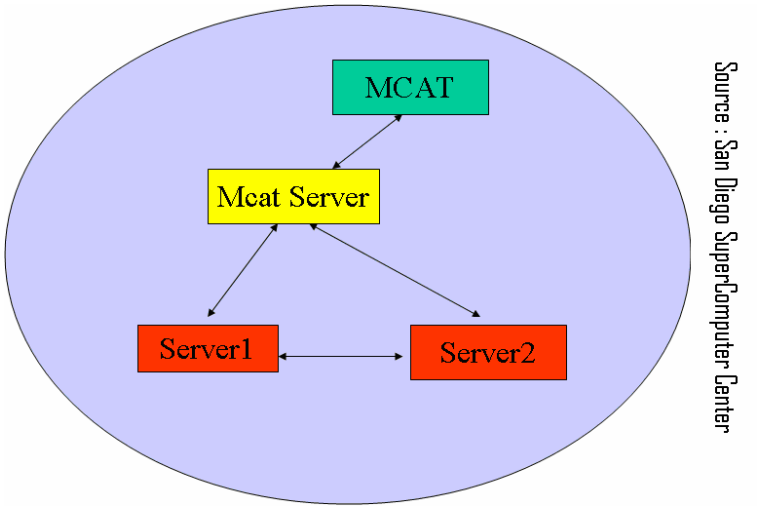


Figure 3.4 : ancien modèle de fédération

Comme nous l'avons annoncé précédemment, cette façon de procéder présente des inconvénients évidents, comme le goulot d'étranglement situé sur le « serveur de MCAT », mais aussi le fait que ce dernier soit un point d'échec unique susceptible de paralyser l'ensemble du système en cas de disfonctionnement.

Les nouvelles versions de SRB ont permis de résoudre ces problèmes. En effet, un nouveau concept est apparu dans les versions plus récentes, celui de zone SRB.

Après avoir évoqué ce mécanisme de fédération, nous allons étudier les différentes manières dont des serveurs SRB peuvent se fédérer.

3.5 Les modèles de fédération

Nous allons donc étudier les différents modèles de fédération de SRB [RWMS04]. Notons que ceux-ci sont d'un niveau théorique, les utilisateurs doivent eux-mêmes configurer leur serveur afin de rentrer dans une catégorie que nous allons citer.

Le mécanisme de fédération permet d'accéder aux ressources physiques situées dans des domaines distincts. Chaque serveur SRB se réfère à un méta-catalogue (MCAT), implémentant ainsi un espace de noms logiques pour les utilisateurs, ressources, fichiers et collections. Ce serveur est accessible à de multiples clients. Nous voyons donc que certaines questions émergent, comme le partage des espaces de noms logiques, la « scalabilité » de ceux-ci, l'optimisation de performance, et la tolérance aux fautes.

Avant d'aller plus loin, définissons le concept de zone SRB : une zone SRB est un ensemble de serveurs SRB sous l'autorité d'un même MCAT (et se référençant donc au même espace de noms logiques).

Plusieurs motivations sont à la base du mécanisme de fédération :

- Améliorer la performance de SRB sur un réseau très large (WAN : Wide Area Network). En effet, dans ce type de réseau, la latence réseau s'accentue fortement. Le fait que le MCAT soit proche est bénéfique vu le nombre d'interactions avec celui-ci.
- Contrôle local : chaque zone est à même de partager ses ressources et collections qui sont administrées par l'administrateur de la zone, et non par un administrateur global.
- « Scalabilité » du MCAT : chaque zone dispose de son MCAT, et donc de sa base de données. Ceci permet de distribuer la charge et d'éviter le phénomène de goulot d'étranglement.
- Pas de point d'échec unique : le fait qu'un site ne dépende pas du méta-catalogue situé sur un autre site implique que, même en cas de problème sur cet autre site, le premier site peut continuer à fonctionner.

Le MCAT a subi certaines modifications dues à l'implémentation de la fédération, puisque chaque zone dispose de son propre MCAT qui doit garder de l'information sur l'existence d'autres zones. Une nouvelle table est donc ajoutée dans chaque MCAT, contenant le nom de zone, l'adresse réseau, un flag local/foreign et des informations d'authentification (dont la méthode d'authentification). De plus, il y a un unique espace de noms d'utilisateur, ce qui fait que chaque nomUtilisateur@NomDomaine doit être unique dans chaque fédération de zones. Chaque MCAT dispose d'informations sur tous les utilisateurs de la fédération, dont un flag renseignant si un utilisateur est local ou pas (un utilisateur n'est local qu'a une zone !), et les informations sensibles (mots de passe) ne sont enregistrées que dans la zone locale des utilisateurs (ce qui implique que si la sécurité d'une zone est compromise, l'instabilité du système ne se répand pas sur d'autres zones).

Des opérations entre serveurs SRB sont possibles, voire même nécessaires. Lorsqu'un utilisateur veut interroger une autre zone, il doit nécessairement passer par sa propre zone, qui s'authentifiera sur la zone étrangère comme utilisateur privilégié de cette dernière (nous perdons donc un peu de transparence, mais qui a un impact mineur sur les opérations). Nous voyons donc que la zone locale de l'utilisateur réalise l'opération à la place de celui-ci, puisqu'une zone peut se connecter sur une autre zone comme utilisateur privilégié (mais ne peut réaliser des opérations qu'à la place de ses utilisateurs locaux).

En ce qui concerne la synchronisation des différentes zones, un script écrit en PERL ([PERL]) doit être lancé périodiquement par l'administrateur de chaque zone afin de synchroniser les méta-données.

Différents scénarios de fédération ont été identifiés (l'architecture formée par les zones fédérées peut prendre différents aspects), mais avant de les détailler, nous devons définir des critères de comparaison (qui permettent de qualifier l'architecture de la fédération). Nous avons :

- Organisation de la zone : peer to peer si aucune hiérarchie n'est imposée entre les zones, hiérarchiques sinon. Si le type est hiérarchique, il est possible que chaque

zone s'occupe de ses contrôles d'accès, ou qu'un super utilisateur s'occupe de l'ensemble de ceux-ci.

- Contrôle d'interactions entre zones : définit qui contrôle l'interaction avec les autres zones.
- Management de la consistance : définit qui contrôle la réplication des données.
- Point d'accès de l'utilisateur : définit où l'utilisateur peut se connecter.
- Contrôle d'accès sur les données : définit qui contrôle les accès aux données.
- Synchronisation des méta-données : définit qui synchronise les méta-données.
- Partage de ressources : définit la manière dont les ressources sont partagées.
- Partage des utilisateurs entre zones : définit comment les zones se partagent les noms d'utilisateur.

Nous pouvons maintenant détailler les différents scénarios de fédération, au travers des critères que nous venons de définir.

3.5.1 Nomadic Zones (SRB in a Box)

Ce modèle (figure 3.5) représente le cas où les utilisateurs installent une zone sur leur machine personnelle, celle-ci n'est pas connectée aux autres zones (nomades) de façon permanente. Durant la période de non-connexion, des nouvelles données et méta-données peuvent être créées. Lors de la connexion à une zone parente, la zone fille échangera ses données et méta-données avec celle-ci. En outre, il est pratique pour les scientifiques nomades ayant une zone sur leur machine et se connectant périodiquement à leur zone parente. Pour un exemple, voir [SIO].

- Organisation de la zone : hiérarchique.
- Contrôle d'interactions entre zones : administrateur local.
- Management de la consistance : fait par l'utilisateur. La zone fille réplique sur la zone parente.
- Point d'accès de l'utilisateur : sur la zone où il est enregistré (donc sa propre zone).
- Contrôle d'accès sur les données : fait par l'utilisateur.
- Synchronisation des méta-données : faite par l'utilisateur.
- Partage de ressources : partiel, car les ressources de la zone fille sont périodiquement synchronisées avec celles de la zone parente, et de façon partielle.
- Partage des utilisateurs entre zones : Un niveau (zone fille – zone parente).

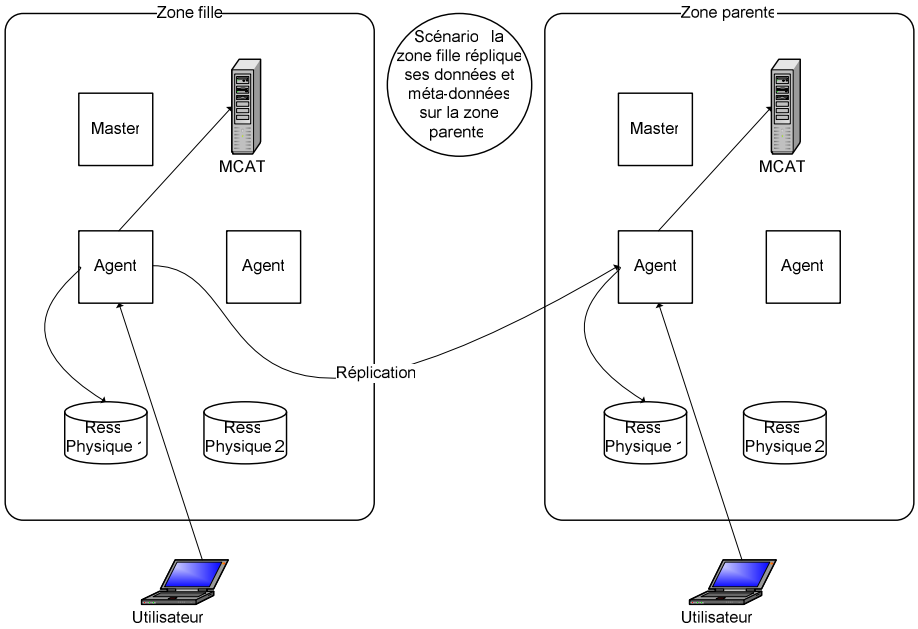


Figure 3.5 : Nomadic Zones

Note : On voit que la zone fille réplique des données sur la zone parente. La zone fille se connecte donc à celle-ci pour y envoyer les données. L'utilisateur doit lui-même synchroniser les méta-données.

3.5.2 Archival Zone (BackUp zone)

Dans ce modèle (figure 3.6), nous avons plusieurs zones accompagnées d'une zone supplémentaire : la zone d'archives (dont le rôle est de créer une archive des autres zones). N'importe quelle zone peut décider d'archiver des données sur la zone de backup, ce qui fournit ensuite un backup pour les autres zones. Pour un exemple, voir [NAS].

Ce modèle est pratique lorsqu'un ensemble de zones est sous l'autorité d'une zone commune, servant de backup.

- Organisation de la zone : hiérarchique.
- Contrôle d'interactions entre zones : « super » administrateur.
- Management de la consistance : système de gestion des versions dans la zone de backup.
- Point d'accès de l'utilisateur : sur la zone où il est enregistré (donc sa propre zone).

- Contrôle d'accès sur les données : fait par le système (de la zone de backup).
- Synchronisation des méta-données : faite par le système (de la zone de backup).
- Partage de ressources : aucun, car il y a archivage, mais pas de synchronisation.
- Partage des utilisateurs entre zones : complet.

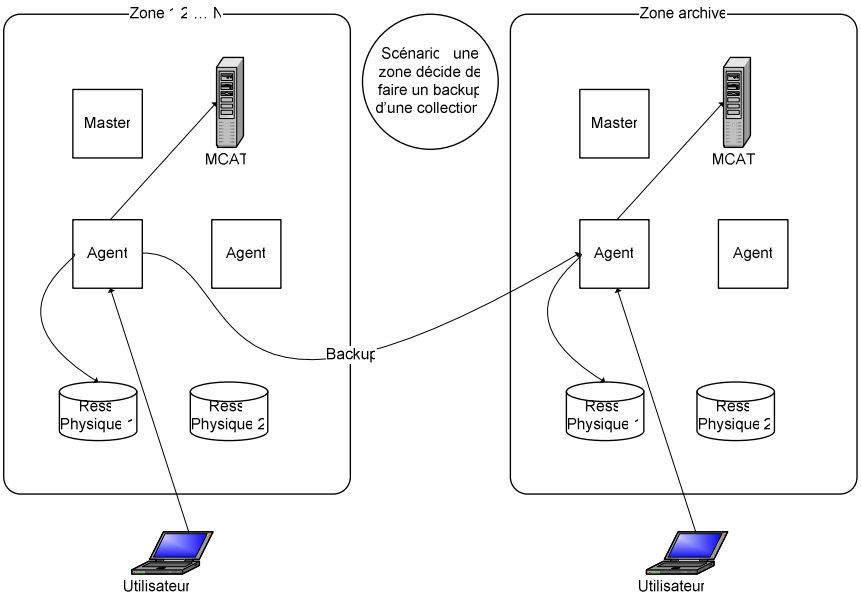


Figure 3.6 : Archival Zone

Note : On voit qu'une zone réalise un backup de données sur la zone de backup. La synchronisation des méta-données est faite par cette dernière.

3.5.3 Replicated Data Zones

Dans ce modèle(figure 3.7), deux zones (ou plus) sont indépendantes, mais maintiennent les mêmes données (réplications de données et méta-données entre zones). Les utilisateurs d'une zone n'ont pas accès aux autres zones (pas de partage des utilisateurs et ressources entre zones). C'est un utilisateur qui dispose d'un compte dans les autres zones qui réplique les données et méta-données. Pour un exemple, voir [BABAR].

Ce modèle est pratique pour des zones éloignées géographiquement, mais désirant partager des données.

- Organisation de la zone : peer to peer.
- Contrôle d'interactions entre zones : administrateur local.

- Management de la consistance : fait par l'utilisateur qui réplique.
- Point d'accès de l'utilisateur : sur la zone où il est enregistré (donc sa propre zone).
- Contrôle d'accès sur les données : fait par l'utilisateur sur la zone où il réplique des données.
- Synchronisation des méta-données : faite par l'utilisateur (lors de la synchronisation)
- Partage de ressources : Partiel. Seules les données répliquées sont partagées.
- Partage des utilisateurs entre zones : Partiel. Certains utilisateurs ont un compte sur plusieurs zones.

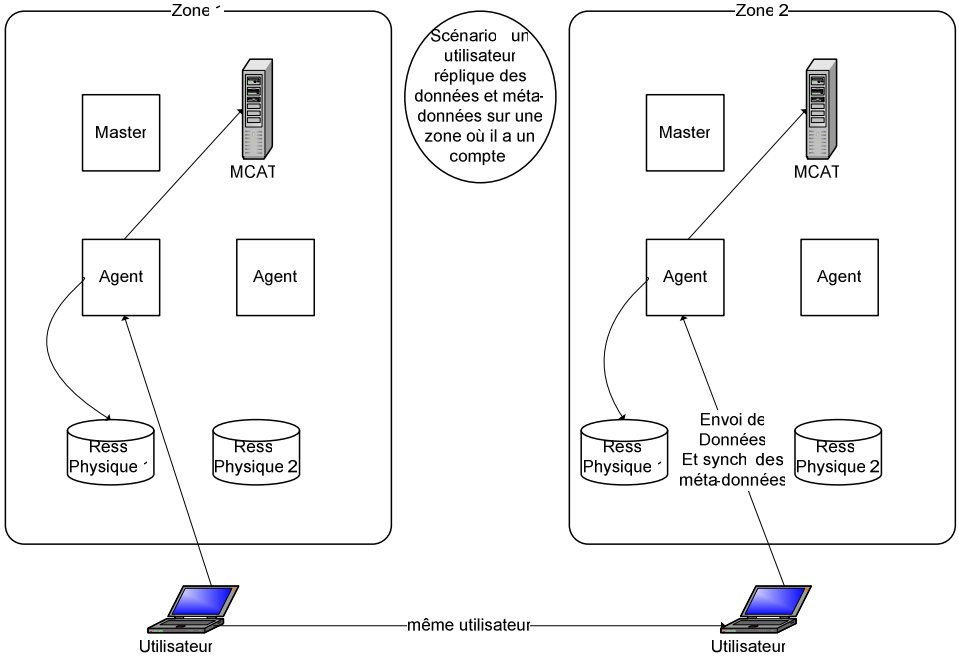


Figure 3.7 : Replicated Data Zones

Note : On voit qu'un utilisateur décide d'envoyer des données et méta-données (déjà présentes dans la zone1) sur la zone2 où il dispose d'un compte.

3.5.4 Master-Slave Zones

Ce modèle (figure 3.8) est une variation du modèle « Replicated Data Zones ». Ici, des données sont créées dans la zone maîtresse, et la zone esclave se synchronise avec celle-ci.

Les utilisateurs ne sont pas partagés entre zones. La zone esclave peut créer de nouveaux objets, dérivés de ceux de la zone maîtresse, mais ceux-ci ne sont pas partagés avec la zone maîtresse. Pour un exemple, voir [PDB].

- Organisation de la zone : hiérarchique.
- Contrôle d'interactions entre zones : super administrateur.
- Management de la consistance : fait par le système (chez les esclaves)
- Point d'accès de l'utilisateur : sur la zone où il est enregistré (donc sa propre zone).
- Contrôle d'accès sur les données : fait par le système.
- Synchronisation des méta-données : faite par le système, mais de manière partielle (pour les données synchronisées avec la zone maîtresse).
- Partage de ressources : aucun (ce qui est créé dans une zone esclave n'est pas propagé sur la zone maîtresse).
- Partage des utilisateurs entre zones : un niveau (zone maîtresse – zone esclave).

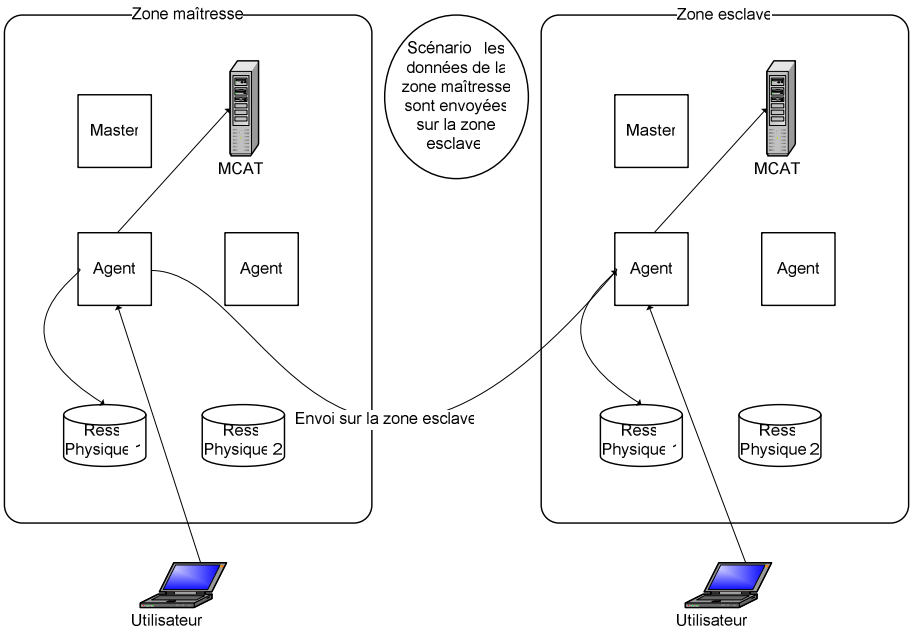


Figure 3.8 : Master Slave Zones

Note : On voit que les données et méta-données de la zone maîtresse sont envoyées sur la zone esclave (et pas inversement).

3.5.5 Snow-Flake Zones

Ce modèle (figure 3.9) est une variation du modèle « Master-Slave Zones ». Il est toujours hiérarchique, mais à plus de deux niveaux. Des données et méta-données d'un niveau sont copiées chez leurs zones esclaves, qui peuvent créer de nouvelles données et méta-données, et propager un sous-ensemble de celles-ci à leurs propres esclaves. Pour un exemple, voir [CMS].

- Organisation de la zone : hiérarchique.
- Contrôle d'interactions entre zones : administrateur local.
- Management de la consistance : fait par le système (chez les esclaves)
- Point d'accès de l'utilisateur : sur la zone où il est enregistré (donc sa propre zone).
- Contrôle d'accès sur les données : fait par le système.
- Synchronisation des méta-données : faite par le système, mais de manière partielle (pour les données synchronisées avec la zone maîtresse).
- Partage de ressources : aucun (ce qui est créé dans une zone esclave n'est pas propagé sur la zone maîtresse).
- Partage des utilisateurs entre zones : un niveau (zone maîtresse – zone esclave).

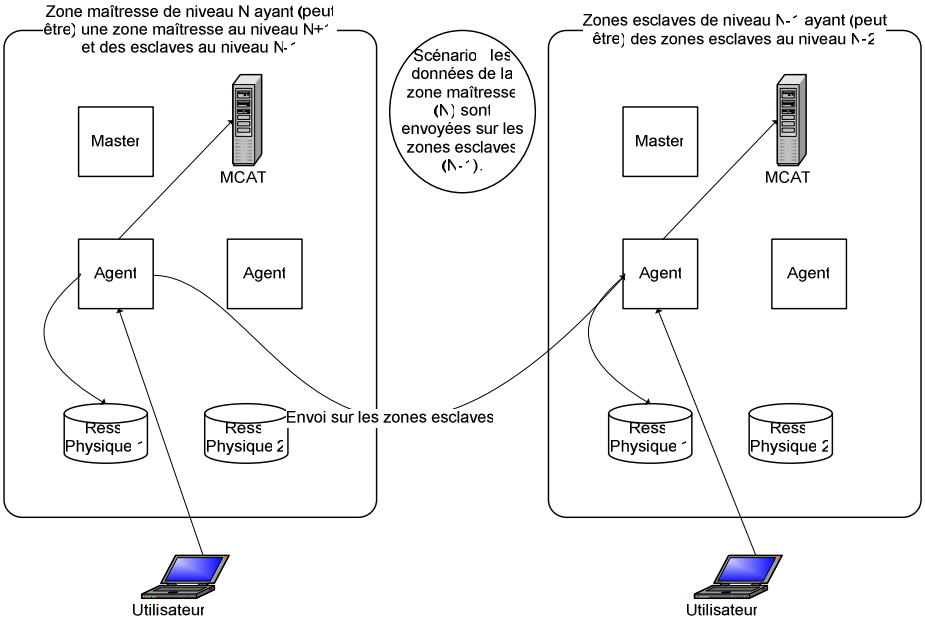


Figure 3.9 : Snow-Flake Zones

Note : On que voit les données et méta-données de la zone maîtresse (niveau N) sont envoyées sur les zones esclaves du niveau N-1 (et pas inversement). Celles-ci peuvent faire de même avec leurs propres esclaves (niveau N-2).

3.5.6 Replicated Catalog

Dans ce modèle (figure 3.10), les méta-catalogues de plusieurs zones sont synchronisés entre eux (toutes les méta-données), le système se comporte donc comme s'il y avait qu'une seule zone. La réplication des MCAT implique que la tolérance aux fautes est très élevée.

Quant à la période de synchronisation des MCAT, elle est convenue entre administrateurs des zones concernées (cette période peut se mesurer en jours si l'activité est faible). Mais ceci a le désavantage d'occasionner des collisions de noms logiques (plusieurs zones peuvent créer un objet de même nom logique, entre 2 synchronisations). Pour les résoudre, une politique est décidée entre les administrateurs de zone afin de déterminer où les utilisateurs peuvent créer leurs fichiers au sein d'une collection. Pour un exemple, voir [NAR].

- Organisation de la zone : peer to peer.
- Contrôle d'interactions entre zones : administrateur local.
- Management de la consistance : fait par le système (synchronisation périodique, politique de gestion des conflits de noms).
- Point d'accès de l'utilisateur : à partir de n'importe quelle zone !
- Contrôle d'accès sur les données : fait par le système.
- Synchronisation des méta-données : faite par le système.
- Partage de ressources : total.
- Partage des utilisateurs entre zones : total.

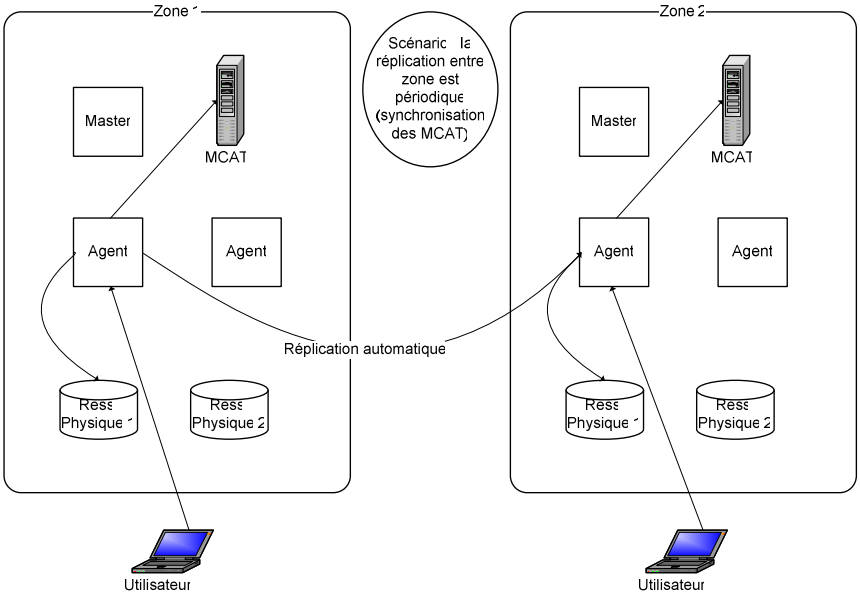


Figure 3.10 : Replicated Catalog

Note : On voit que les MCAT sont synchronisés entre eux afin d'offrir une vue globale (existence de plusieurs zones, mais vues comme une seule).

3.5.7 Free Floating Zones (my zone)

Ce modèle (figure 3.11) est une variation du modèle « Nomadic Zone », où il peut exister plusieurs zones, mais pas de zone parente. Ces zones ont leurs propres utilisateurs et ressources, et vivent indépendamment les unes des autres. A l'occasion, elles échangent des données et collections, ce qui est similaire à l'échange de données via stick USB entre ordinateurs personnels. Les zones sont très autonomes et échangent des données de manière contrôlée et occasionnelle (Modèle de fédération de Napster).

- Organisation de la zone : peer to peer.
- Contrôle d'interactions entre zones : administrateur local.
- Management de la consistance : fait par l'utilisateur, publication de données sur les autres zones (copie dans un endroit accessible).
- Point d'accès de l'utilisateur : sur la zone où il est enregistré (donc sa propre zone).
- Contrôle d'accès sur les données : fait par l'utilisateur qui donne les droits d'accès à ceux qu'il désire.

- Synchronisation des méta-données : faite par l'utilisateur.
- Partage de ressources : non (partage occasionnel).
- Partage des utilisateurs entre zones : aucun.

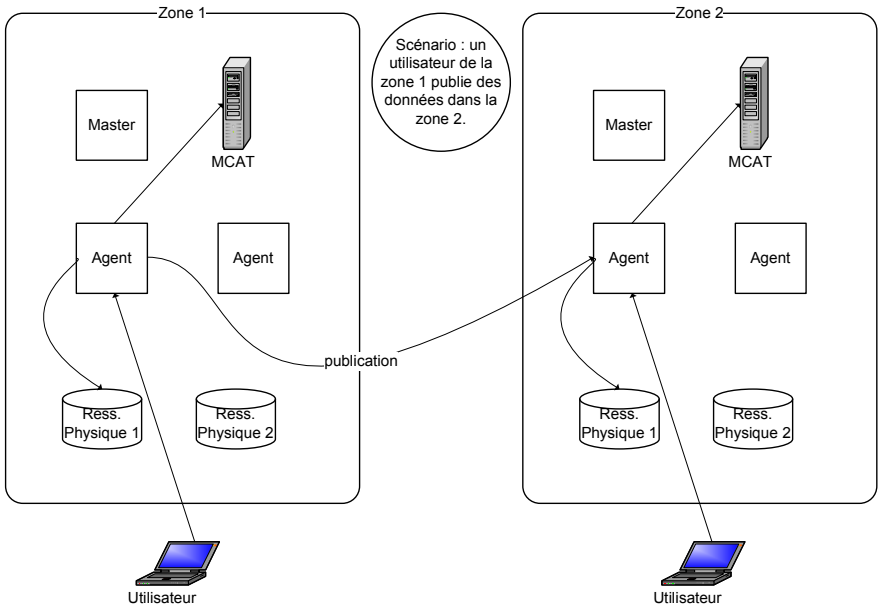


Figure 3.11 : Free Floating Zones

Note : un utilisateur de la zone 1 a des droits d'écriture dans la zone 2 et publie des données dans celle-ci.

3.5.8 User and Data Replica Zones

Ce modèle (figure 3.12) est une variation du modèle « Replicated Data Zones », où les noms d'utilisateur sont aussi échangés entre zones (et pas seulement les données). Les utilisateurs accèdent à leurs données de n'importe où, ce qui est pratique quand un utilisateur voyage entre zones et veut accéder à ses données de manière performante.

- Organisation de la zone : peer to peer.
- Contrôle d'interactions entre zones : administrateur local.
- Management de la consistance : fait par l'utilisateur.
- Point d'accès de l'utilisateur : sur la zone où il est enregistré (donc sa propre zone).

- Contrôle d'accès sur les données : fait par le système.
- Synchronisation des méta-données : faite par le système.
- Partage de ressources : partiel.
- Partage des utilisateurs entre zones : total.

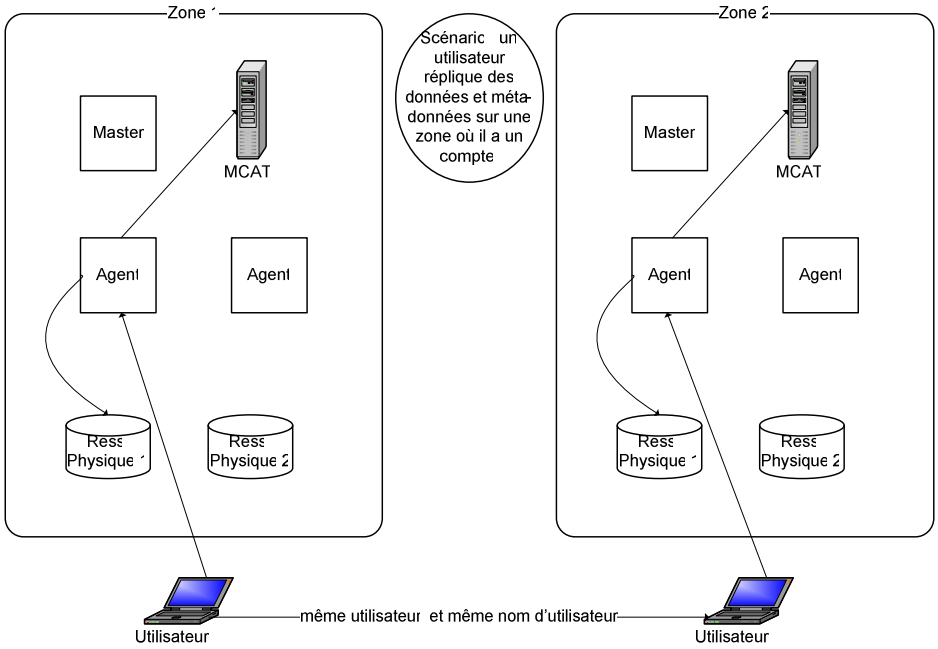


Figure 3.12 : User and Data Replica Zones

Note : comme pour le modèle « Replicated Data Zones », on voit qu'un utilisateur décide de synchroniser des données et méta-données sur une autre zone où il dispose d'un compte, mais cette fois le compte sur l'autre zone existe du fait de l'échange des noms d'utilisateur entre zones (et non parce que l'utilisateur s'est vu créer un compte sur cette autre zone).

3.5.9 Ressource Interaction

Dans ce modèle (figure 3.13), les ressources sont partagées entre zones afin d'être utilisées comme support à la réplication. Ce modèle s'avère utile lorsque des zones éloignées veulent se partager des données ayant un intérêt mutuel. Les utilisateurs répliquent les données à partager dans les ressources partagées, les méta-données sont ensuite synchronisées. Les noms d'utilisateur ne sont pas forcément tous échangés. Pour un exemple, voir [BIRN].

- Organisation de la zone : peer to peer.

- Contrôle d'interactions entre zones : administrateur local.
- Management de la consistance : fait par l'utilisateur, réplication dans les ressources partagées.
- Point d'accès de l'utilisateur : sur la zone où il est enregistré (donc sa propre zone).
- Contrôle d'accès sur les données : fait par l'utilisateur qui donne les droits d'accès à ceux qu'il désire (mais seulement sur sa propre zone).
- Synchronisation des méta-données : aucune car écriture dans les ressources partagées.
- Partage de ressources : partiel (lors de la réplication dans les ressources partagées).
- Partage des utilisateurs entre zones : partiel.

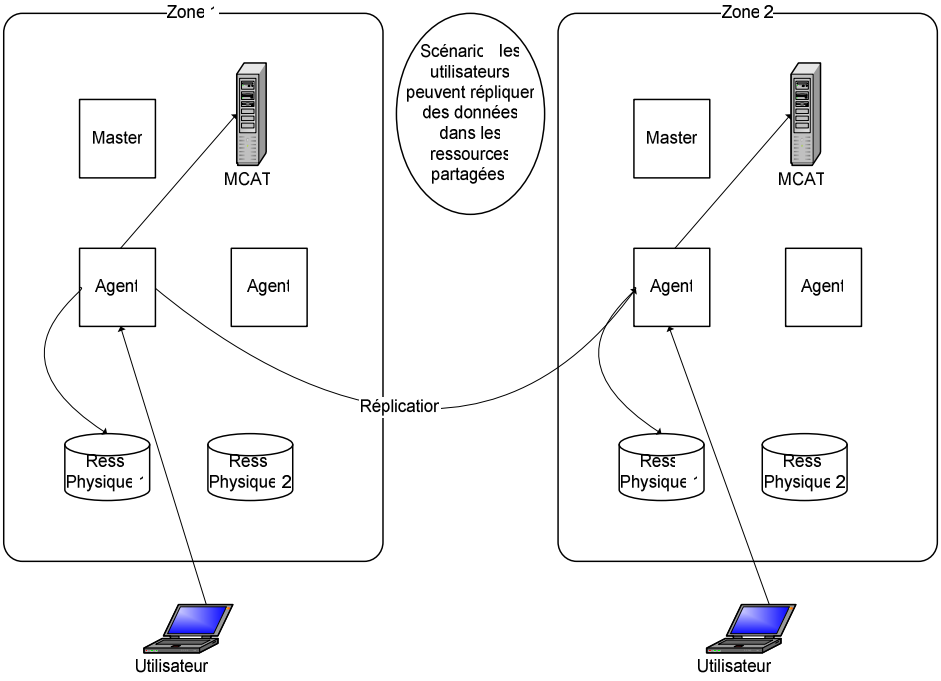


Figure 3.13 : Ressource Interaction

Note : les utilisateurs se servent des ressources partagées (ici, les ressources physiques 1 dans deux zones) entre zones pour mettre à disposition les données pouvant intéresser les autres.

3.5.10 Occasional Interchange

Ici (figure 3.14), plusieurs zones fonctionnent de manière autonome, avec peu d'échanges. Les noms d'utilisateur sont échangés seulement pour ceux qui doivent accéder à d'autres zones, ceux-ci peuvent créer des données dans d'autres zones, qui ne pourront être lues que par les utilisateurs de la zone où sont créées ces données. Ce modèle offre donc un grand degré d'autonomie et de contrôle, les administrateurs locaux détermine quels utilisateurs externes peuvent créer des données dans leur zone, et dans quelles ressources. Pour un exemple, voir [NPACI].

- Organisation de la zone : peer to peer.
- Contrôle d'interactions entre zones : administrateur local.
- Management de la consistance : fait par l'utilisateur.
- Point d'accès de l'utilisateur : sur la zone où il est enregistré (donc sa propre zone).
- Contrôle d'accès sur les données : fait par l'utilisateur.
- Synchronisation des méta-données : faite par l'utilisateur.
- Partage de ressource : aucun.
- Partage des utilisateurs entre zones : partiel.

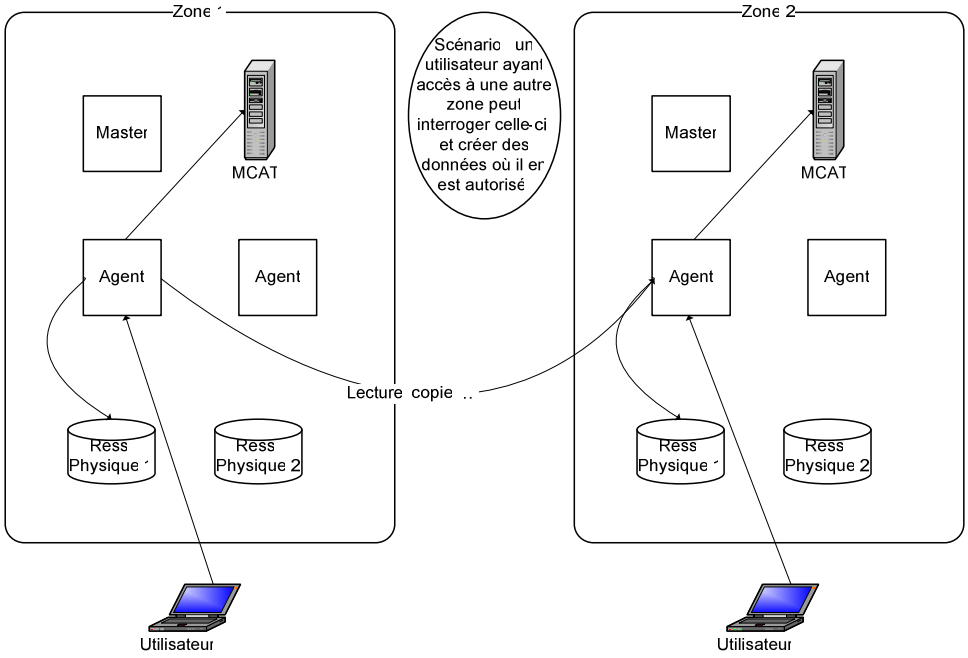


Figure 3.14 : Occasional Interchange

Note : les données créées par un utilisateur externe ne sont pas accessibles aux utilisateurs de la même zone que cet utilisateur, il crée donc des objets pour les utilisateurs locaux à la zone concernée.

3.5.11 Schémas récapitulatifs

Le premier tableau (figure 3.15) reprend les différents scénarios (au nombre de dix) que nous avons détaillé, ainsi que les critères de comparaisons qui les différencient.

Zone SRB	Zone Organization	Zone interaction control	Consistency Management	User Connection Point to access files	Data Access Control Setting	Metadata synchroni-zation	Resource sharing	User-ID sharing between zones
	Zones	Zones	Collections	Files	Files	Metadata	Resources	User names
Free Floating Zones	Peer-to-Peer	Local Admin	User-specified data publication	From home zone	User set access controls	User controlled synchronization	None	None
Occasional Interchange	Peer-to-Peer	Local Admin	User specified	From home zone	User set access controls	User controlled synchronization	None	Partial
Replicated Data Zones	Peer-to-Peer	Local Admin	User-specified replication	From home zone	User set local access controls	User controlled synchronization	Partial	Partial, user establishes own accounts
Resource Interaction	Peer-to-Peer	Local Admin	User-specified replication	From home zone	User set access controls	None	Partial shared resource for replication	Partial
User and Data Replica Zones	Peer-to-Peer	Local Admin	User-specified replication	From home zone	System set access controls	System controlled complete synchronization	Partial	Complete
Replicated Catalog	Peer-to-Peer	Local Admin	System managed name conflict resolution	From any zone	System replicated access controls	System controlled complete synchronization	All zones share resources	Complete
Snow Flake Zones	Hierarchical	Local Admin	System managed replication in hierarchy of zones	From home zone	System set access controls	System controlled partial synchronization	None	One
Master-Slave Zones	Hierarchical	Super Admin	System-managed replication to slave	From home zone	System set access controls	System controlled partial synchronization	None	One
Archival zones	Hierarchical	Super Admin	System-managed versioning to parent zone	From home zone	System set access controls	System controlled complete synchronization	None	Complete
Nomadic Zones	Hierarchical	Local Admin	User-managed replication to parent zone	From home zone	User set access controls	User controlled synchronization	Partial	One

Source : San Diego SuperComputer Center

Figure 3.15 : tableau récapitulatif des modèles de fédération

La figure 3.16 schématise les différences notables entre modèles de fédération. La partie « peer-to-peer Zones » ne comprend que des modèles peer to peer, la partie « Replication Zones » également (mais où des données sont partagées, via des ressources partagées), et la partie « Hierarchical Zones » ne comprend que des zones hiérarchiques.

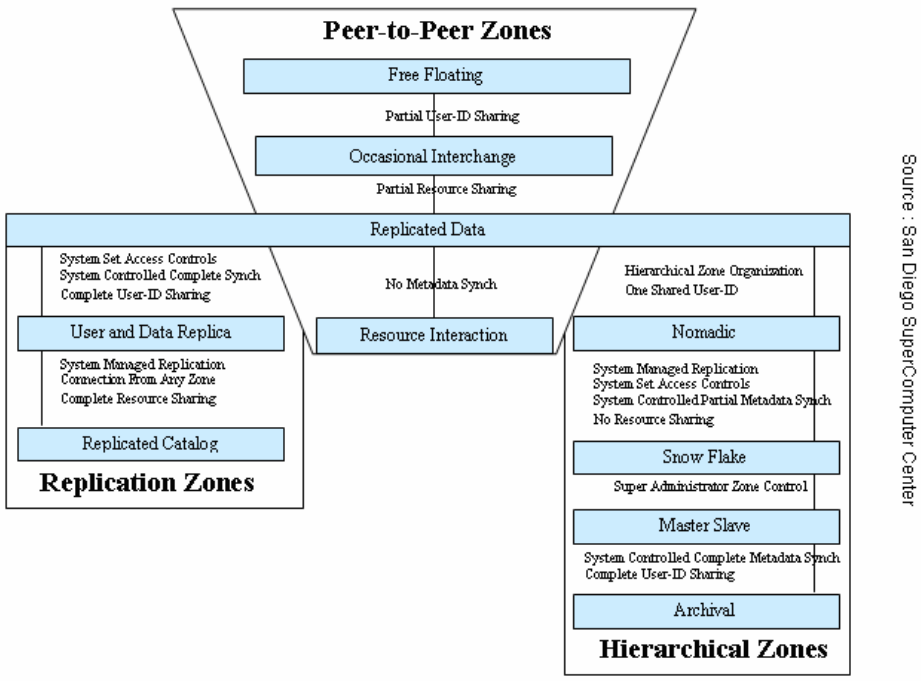


Figure 3.16 : Schéma récapitulatif des modèles de fédération [RWMS04]

Nous avons donc un aperçu des différentes possibilités de fédération de zones SRB. Rappelons que ces modèles sont purement théoriques, et que les administrateurs de zones SRB doivent configurer eux-mêmes leur zone afin de se rapprocher d'un modèle de fédération particulier.

3.6 Réplication

Comme nous l'avons annoncé précédemment, SRB offre des mécanismes de réplication de données. Ce qui veut donc dire qu'un fichier SRB peut avoir plusieurs réplicas, ceux-ci pouvant être stockés sur des ressources physiques différentes.

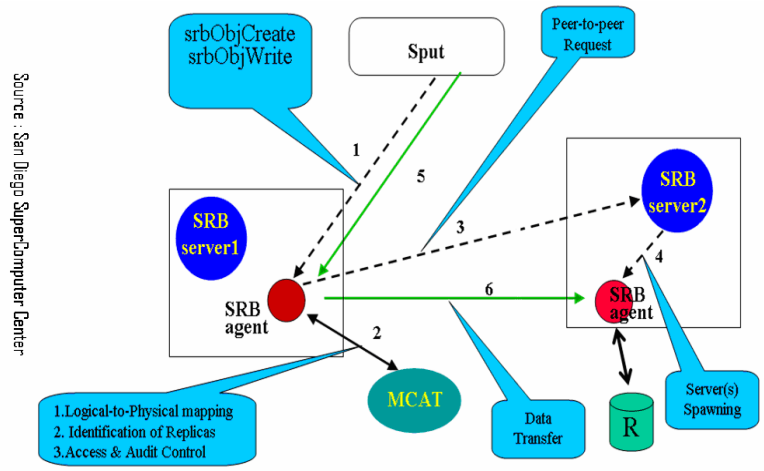
Du point de vue utilisateur, des commandes spécifiques existent afin de répliquer des données, de réaliser un backup sur une ressource spécifique, ainsi que pour synchroniser des réplicas.

3.7 Performances

Au cours de cette section, nous allons découvrir les mécanismes mis en œuvre dans SRB afin d'atteindre un certain niveau de performance.

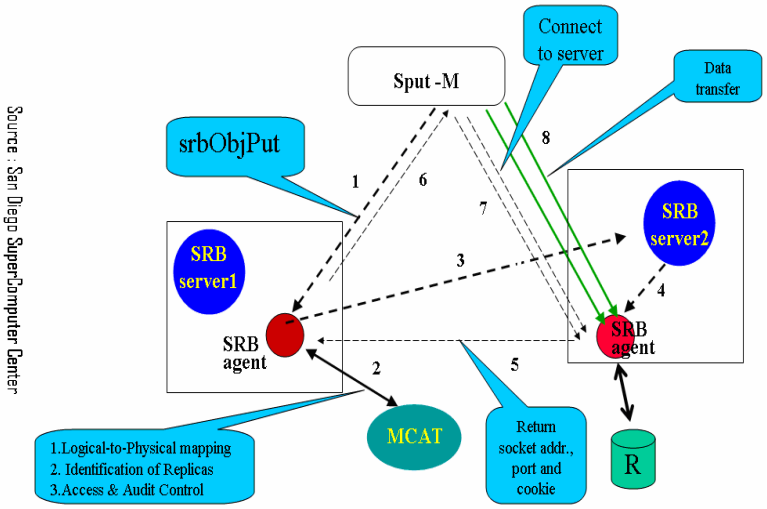
Tout d'abord, certains optimisations ont été développées afin d'accélérer le transfert de fichiers (envoi ou réception). Celles-ci se nomment Parallel I/O et Bulk Operation.

- Parallel I/O : cette option est très pratique pour transférer de gros fichiers. En effet, plusieurs threads sont lancés (plusieurs connexions réseaux) en même temps, et les I/O parallèles disques sont exploités (par exemple, les I/O parallèles offerts par HPSS [HPSS]). Un exemple d'application de ceci est la commande Sput (permettant d'envoyer des fichiers dans SRB) où nous pouvons constater que, à l'inverse du mode classique, le mode parallèle permet au client d'initier plusieurs connexions vers le serveur SRB.
  Les figures 3.17 et 3.18 nous montrent des scénarios de création d'un fichier dans SRB (avec et sans l'option parallel I/O).



1. Demande de création d'un fichier.
2. Accès au MCAT.
3. Accès au serveur responsable de la ressource où on veut créer le fichier.
4. Le master délègue la requête à un agent.
5. Transfert de données.
6. Idem.

Figure 3.17 : Sput (non parallèle)



1. Demande de création d'un fichier.
2. Accès au MCAT.
3. Accès au serveur responsable de la ressource où on veut créer le fichier.
4. Le master délègue la requête à un agent.
5. Transfert des coordonnées de l'agent SRB responsable de la ressource où on veut créer le fichier.
6. Idem.
7. Connexion au serveur SRB (où le fichier sera créé).
8. Transfert de données (avec l'option parallel I/O).

Figure 3.18 : Sput (mode parallèle initié par le client)

- Bulk Operation : l'option Bulk est quant à elle très pratique pour le transfert d'un grand nombre de petits fichiers. Elle utilise également plusieurs threads. Cette option a le grand avantage d'enregistrer (lors d'un envoi) jusqu'à 500 fichiers en une fois dans le MCAT, au lieu de le faire un à un en mode classique. De même, les fichiers sont envoyés sur le réseau par agrégats de plusieurs d'entre eux. Tout ceci donne un facteur d'accélération de 3 à 10 !

Ensuite, une notion très importante et pratique dans SRB est la notion de container. Un container permet de regrouper un ensemble de petits fichiers en un seul fichier physique. Ceci fonctionne très bien avec des ressources physiques sur bandes (comme HPSS). L'ensemble du container est mis en cache sur une ressource temporaire (de type cache) permettant des I/O rapides et, ensuite, le container est à nouveau archivé sur bande(s). De manière plus précise, un container fonctionne avec une ressource logique contenant deux ressources physiques, une ressource d'archive (HPSS, …) et une ressource cache (système de fichiers Unix, …). Un container ressemble donc à un ficher tarball (.tar) puisqu'il rassemble plusieurs fichiers en un seul, et ce container grandit au fur et à mesure de l'ajout de fichiers. Mais à l'inverse d'un fichier tarball, un container peut s'agrandir à la demande, et l'utilisateur peut lire les fichiers un à un sans devoir charger l'ensemble (grâce à l'information sur le container enregistrée dans le MCAT).

3.8 Design d'un agent SRB

Nous allons regarder un peu plus en détails la manière dont fonctionne un agent SRB (figure 3.19), « interlocuteur » principal d'un client SRB. Le design de ce dernier est réalisé en 3 couches :

- La couche supérieure (Dispatcher) : cette couche interagit avec le client (au travers de sockets TCP/IP), elle authentifie aussi ce dernier. Lorsqu'elle reçoit une requête d'un client, elle analyse celle-ci afin d'invoquer la couche intermédiaire ou inférieure, sans oublier d'envoyer le résultat du client.
- La couche intermédiaire (High Level Request Handler) : la plupart des requêtes venant des clients passent par cette couche. Elle reçoit des paramètres sous forme logique (chemin logique, nom de ressource logique) et transforme ceux-ci dans leur représentation physique. De manière générale, les requêtes sont de 2 types : soient des requêtes d'accès aux données, dans ce cas le MCAT est utilisé pour convertir une représentation logique en une représentation physique et une fonction de la couche inférieure est appelée ; soit une requête d'accès aux méta-données, le MCAT est donc utilisé pour y accéder.
- La couche inférieure (Low Request Handler) : c'est à partir de cette couche que tous les accès aux données sont réalisés. Elle est responsable et dispose de pilotes pour 2 types de ressources : Les ressources de type système de fichiers (UNIX, HPSS, …) ; les ressources de type base de données (BLOB, DB2, Oracle …).

La différence majeure entre la couche intermédiaire et la couche inférieure est la suivante : la couche intermédiaire est à même de convertir des attributs logiques en attributs physiques (grâce au MCAT), alors que la couche inférieure s'attend à recevoir des attributs physiques dans le but de répondre à une requête liée à une ressource. Toutefois, seuls des utilisateurs privilégiés ont le droit d'utiliser les API associées à la couche inférieure.

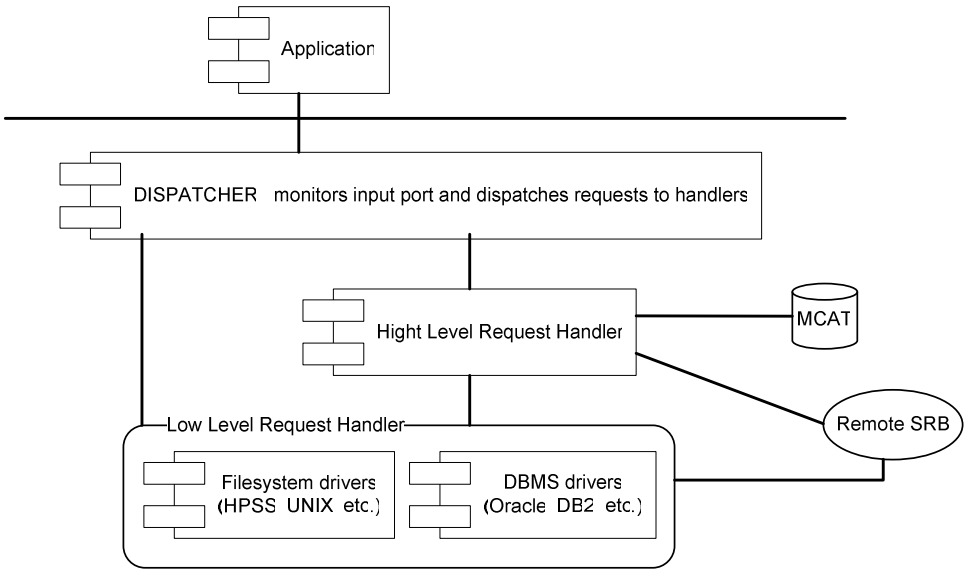


Figure 3.19 : design d'un agent SRB

3.9 API à disposition du client

Après avoir découvert l'architecture de SRB, le MCAT et les modèles de fédération, nous pouvons étudier les API à disposition des clients [SRBAPI]. Celles-ci permettent aux clients de formuler des requêtes de plusieurs types : (dé)connexion au serveur, requête / mise à jour de méta-données, création d'objets, ouvrir / lire / écrire / effacer des objets.

3.9.1 (Dé)Connexion au serveur

Nous avons ici les fonctions classiques de connexion et de déconnexion, ainsi qu'une fonction spéciale afin de nous connecter à l'aide du système de tickets. Dans le cas d'une connexion à l'aide d'un ticket, le système n'authentifie pas l'utilisateur, mais vérifie que le ticket donne droit à la requête du client.

3.9.2 Requête / mise à jour de méta-données

Ces fonctions sont destinées au management des méta-données sur les utilisateurs, groupes d'utilisateurs, collections, données et ressources. Comme nous l'avons vu dans le design d'un agent SRB, la couche intermédiaire utilise souvent le MCAT, via les fonctions décrites dans la présente section.

3.9.3 Création d'objets

Ce type de fonctions est destiné à la création d'objets dans SRB. Lors de la demande de création, SRB vérifie d'abord les informations sur les droits d'accès de l'utilisateur (grâce au MCAT). Si l'utilisateur a le droit de créer l'objet dans la collection et la ressource demandées, le système génère un chemin physique unique dans cette ressource. Ensuite, l'objet est créé dans la ressource physique et les méta-données (nom de l'objet, collection, ressource physique, utilisateur) sont enregistrées dans le MCAT. Dans le cas d'échec de création, une erreur est retournée au client.

3.9.4 Ouvrir / lire / écrire / effacer des objets

Le but de ces fonctions est d'ouvrir, fermer, lire, écrire, modifier, répliquer des objets, ainsi que de régler l'émission / suppression de tickets.

L'ouverture d'un objet demande d'interroger le MCAT afin d'obtenir les méta-données nécessaires. Par contre, lire et écrire ne demande pas d'interroger le MCAT (sauf si une trace doit être conservée dans un fichier de log). Supprimer un ficher nécessite la modification des méta-données.

3.10 Les S-Commandes

Les S-Commandes sont des lignes de commandes « Unix like », disponibles aussi bien sur Windows que sur Linux. Elles sont au nombre d'un peu moins de 70.

Des exemples de commandes sont donc Sls, Scd, Scp, …La liste complète est en annexe (Annexe C).

3.11 Les clients SRB

Outre les S-Commandes, d'autres clients SRB sont fournis par les développeurs, nous avons par exemple inQ (explorateur de fichiers SRB sous Windows), Jargon (Classes JAVA),

mySRB (client Web), le Java Admin Tool (interface d'administration en Java) et Matrix (Web service).

3.12 Extensibilité de SRB

SRB est utilisé dans le cadre d'une multitude de projets, et ce dans des domaines différents. Nous avons par exemple des projets de la NASA, des projets de physique de particules, des projets de médecine, ou des projets de neuroscience comme BIRN. Nous reparlerons de ce dernier au cours du chapitre 4.

La figure 3.20 reprend le nombre de fichiers présents dans SRB, classé par projet.

Date	5/17/02		6/30/04			1/3/06		
Project	GBs of data stored	1000's of files	GBs of data stored	1000's of files	Users with ACLs	GBs of data stored	1000's of files	Users with ACLs
Data Grid								
NSF / NVO	17,800	5,139	51,380	8,690	80	93,252	11,189	100
NSF / NPACI	1,972	1,083	17,578	4,694	380	34,452	7,235	380
Hayden	6,800	41	7,201	113	178	8,013	161	227
Pzone	438	31	812	47	49	19,674	10,627	68
NSF / LDAS-SALK	239	1	4,562	16	66	104,494	131	67
NSF / SLAC-JCSG	514	77	4,317	563	47	15,703	1,666	55
NSF / TeraGrid			80,354	685	2,962	195,012	4,071	3,267
NIH / BIRN			5,416	3,366	148	13,597	13,329	351
Digital Library								
NSF / LTER	158	3	233	6	35	236	34	36
NSF / Portal	33	5	1,745	48	384	2,620	53	460
NIH / AfCS	27	4	462	49	21	733	94	21
NSF / SIO Explorer	19	1	1,734	601	27	2,452	1,068	27
NSF / SCEC			15,246	1,737	52	153,159	3,229	73
Persistent Archive								
NARA	7	2	63	81	58	2,703	1,906	58
NSF / NSDL			2,785	20,054	119	5,205	50,586	136
UCSD Libraries			127	202	29	190	208	29
NHPRC / PAT						101	474	28
TOTAL	28 TB	6 mil	194 TB	40 mil	4,635	655 TB	106 mil	5,383

Source : San Diego SuperComputer Center

Figure 3.20 : extensibilité de SRB

Chapitre 4

Lyon Neuroimaging Database and Application (LyNDA)

Au cours de ce chapitre, nous décrirons le projet LyNDA. Nous commencerons par une mise en contexte de celui-ci (accompagnée d'une analyse de l'existant). Ensuite, nous mettrons en évidence les problèmes que le projet a résolus, ainsi que la manière d'y arriver. Enfin, nous présenterons l'application développée dans le cadre du projet. Nous conclurons par une perspective future sur le projet.

4.1 Contexte du projet

Le professeur Christian Scheiber (CERMEP, Centre d'Exploration et de Recherche Médicales par Émission de Positons), désireux de mettre en place sur Lyon un système informatique mutualisé pour le stockage et le traitement de données en neuroscience, s'est adressé au CC-IN2P3 (Centre de Calcul, Institut National de Physique Nucléaire et de Physique des Particules) voici plus de deux ans. Ce dernier ouvrait, depuis peu, ses ressources pour la recherche dans le domaine de la biologie. Son interlocuteur fut Pascal Calvat. Ensemble, ils évaluèrent les besoins du projet naissant, et décidèrent de mettre en place un ensemble de services informatiques pour satisfaire les besoins de la recherche en neuroscience.

Il s'agissait d'envoyer, à destination de SRB (mis en place au CC-IN2P3 par Jean Yves NIEF), les données produites par un appareil d'Imagerie par Résonance Magnétique (IRM) situé au CERMEP, et d'effectuer les traitements nécessaires sur ces données pour permettre aux chercheurs de les analyser.

Ce groupe de travail s'est progressivement agrandi, avec les arrivées d'Eric Boix (Ingénieur Expert) et d'Eddy Caron (maître de conférence) de l'Ecole Normale Supérieure de Lyon (ENS-LYON), qui travaillent sur le calcul parallèle. Aussi, plusieurs stagiaires ont été appelés afin de contribuer à certains aspects de ce projet ambitieux.

4.2 Architecture de la plateforme LyNDA

La figure 4.1 nous montre, de manière schématique, l'architecture mise en place dans le cadre du projet LyNDA. Nous détaillerons celle-ci d'une part lors de l'analyse de l'existant et, d'autre part, lors de la présentation de notre contribution au projet.

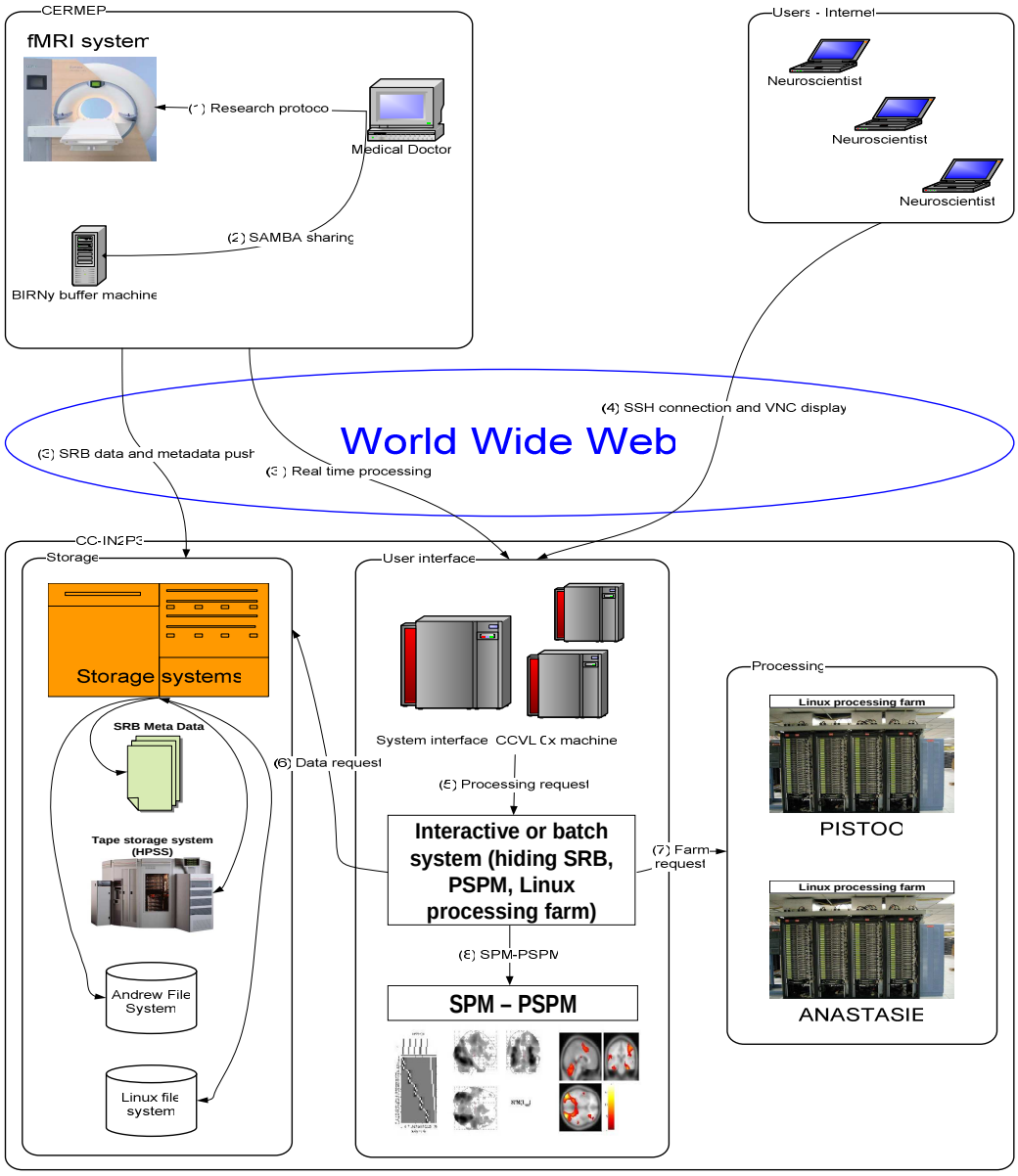


Figure 4.1 : architecture de LyNDA

4.3 Analyse de l'existant

Après avoir présenté le contexte du projet LyNDA, nous allons présenter ce qui préexistait à la contribution qui fait l'objet du présent document. Nous présenterons les moyens applicatifs à disposition des chercheurs, ainsi que l'état initial de la plateforme.

4.3.1 Etat initial de la plateforme

Certaines parties de l'architecture (présentée dans la section 4.2) étaient préexistantes à la contribution que nous présenterons plus loin. C'est le cas des composants « CERMEP », « Users - Internet », « CC-IN2P3 », « Storage (CC-IN2P3) », et Processing (CC-IN2P3) », que nous allons présenter successivement.

4.3.1.1 Composant « CERMEP »

Le Centre d'Exploration et de Recherche Médicales par Émission de Positons est un Groupement d'Intérêt Economique (GIE) dont les partenaires sont le CNRS, l'Institut National de la Santé Et de la Recherche Médicale (INSERM), les Universités scientifiques de Lyon et Grenoble, et les trois Centres Hospitalo-Universitaire de la région Rhône-Alpes : Lyon, Grenoble et Saint-Étienne. Il est situé dans l'enceinte de l'Hôpital Neuro-Cardiologique de Lyon. Le CERMEP est un centre d'imagerie médicale dans lequel s'effectue une recherche expérimentale et clinique, il dispose d'une TEP (Tomographie par Emission de Positons), d'une MEG (MagnétoEncéphaloGraphie) et d'un IRM.

Comme l'indique son nom, le composant CERMEP concerne le Centre d'Exploration et de Recherche Médicales par Émission de Positons. C'est à cet endroit que les images fonctionnelles sont acquises par l'imageur (lien (1)). Pour rappel, ces images sont acquises au regard du protocole d'acquisition.

Nous en venons ensuite au début du « tuyau » suivi par les images fonctionnelles. La première étape de celui-ci est représentée par les liens (2) et (3), et a fait l'objet d'un projet nommé BIRNy [Bon]. Le but de ce projet était de mettre en place un protocole informatique pour le transfert d'images médicales au format DICOM provenant d'un IRM situé au CERMEP, à destination d'un outil de stockage (SRB). C'est ce qui est représenté dans la figure suivante (figure 4.2) :

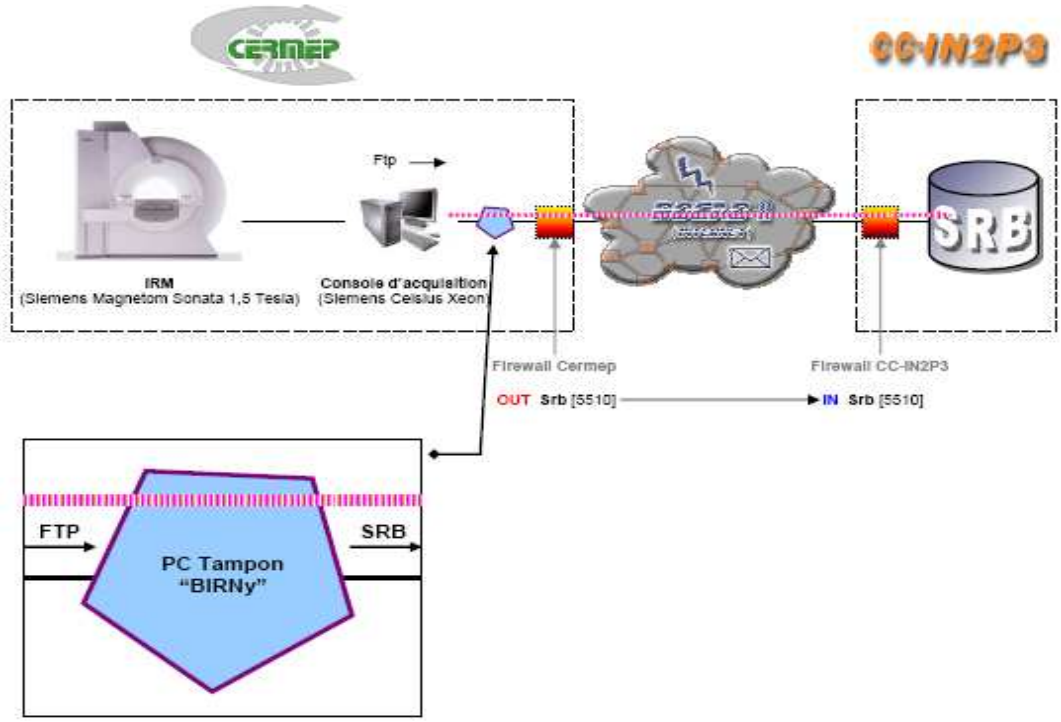


Figure 4.2 : projet BIRNy

Sans rentrer dans les détails d'implémentation du projet, nous constatons que l'envoi des images vers le serveur SRB de l'IN2P3 se fait de manière relativement originale : pour des raisons politiques (il est provisoirement interdit de transférer, à partir de la console d'acquisition, des images médicales vers l'extérieur du CERMEP), les images sont envoyées sur une machine tampon (via un partage de fichier SAMBA), et sont ensuite envoyées vers l'IN2P3. Signalons que, outre les images au format DICOM, des méta-données SRB sont également insérées. Ceci permettra bien entendu une recherche d'images sur base de leurs attributs (méta-données). Tout ceci est représenté par les liens (2) et (3) de l'architecture LyNDA.

En ce qui concerne le lien (3'), nous en avons déjà discuté au chapitre 2 (Imagerie fonctionnelle par résonance magnétique, en temps réel). La stratégie temps réel est une perspective future du projet LyNDA, mais est volontairement ignorée dans un premier temps afin d'implémenter avant toute chose tout le « tuyau » suivi par les images.

4.3.1.2 Composant « Users – Internet »

Nous sommes ici dans le composant le plus simple de l'ensemble, puisqu'il s'agit des utilisateurs de la plateforme..

Toute personne désirant utiliser la plateforme devra réaliser les étapes suivantes :

- Procuration d'un compte à l'IN2P3 : cette étape sert à se voir octroyer l'accès à des machines Linux nommées CCVLI0x (CCVLIO1, CCVLIO2,…), servant de machine-interface vers les outils développés pour le projet LyNDA. En outre, il est également nécessaire d'acquérir un compte SRB sur le serveur SRB de l'IN2P3 (voir plus loin).
- Connexion sécurisée (SSH) vers la machine passerelle sur laquelle la personne est autorisée à se connecter (tests réalisés avec SSH Secure Shell et Putty).
- Importation de l'affichage distant si la personne désire utiliser les outils graphiques (tests réalisés avec VNC Viewer).

Une fois ces 3 étapes accomplies (figure 4.3), l'utilisateur est connecté à distance sur une machine lui permettant d'utiliser les ressources du centre de calcul de l'IN2P3, y compris les outils fournis par le projet LyNDA.

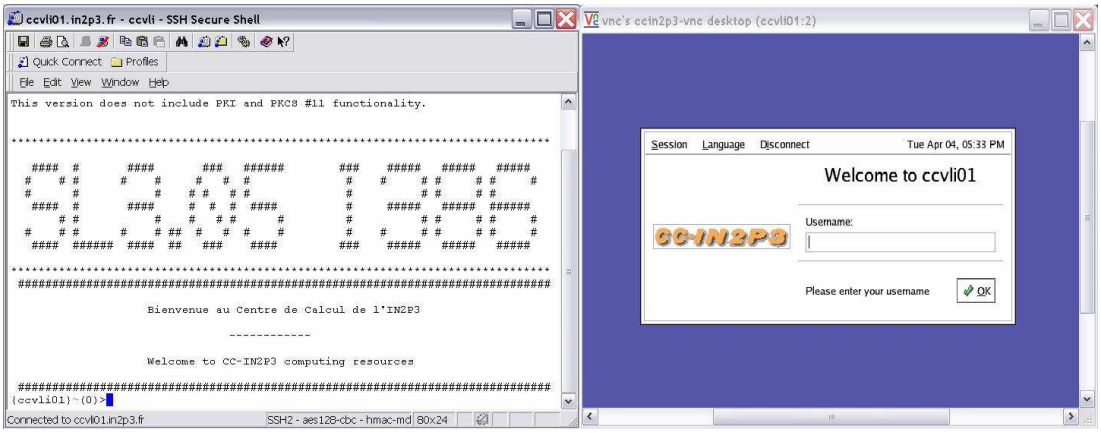


Figure 4.3 : connexion à l'IN2P3

4.3.1.3 Composant « CC-IN2P3 »

La présentation ci-dessous du centre de calcul de l'IN2P3 provient du site Internet officiel de ce dernier :

L'Institut National de Physique Nucléaire et de Physique des Particules (IN2P3), est un institut du Centre National de la Recherche Scientifique (CNRS) créé en 1971. Son champ scientifique est celui de la physique subatomique, du noyau jusqu'aux constituants élémentaires de la matière.

Le Centre de Calcul de l'Institut National de Physique Nucléaire et de Physique des Particules (CC-IN2P3) est spécialisé dans la fourniture de services informatiques nécessaires à l'analyse et à l'interprétation des processus fondamentaux de la physique subatomique. Il a

développé une expertise dans plusieurs domaines, en particulier la conception, l'installation et l'exploitation des fermes de stations pour le traitement des données (simulation, dépouillement, analyse statistique, visualisation), la conception et la mise en œuvre d'une architecture pour le stockage massif de données (bases de données, réseau de stockage, disques sécurisés, bandothèque automatisée, accès hautes performances) et la mise en place de l'infrastructure de transport des données (réseaux locaux et étendus à très haut débit). Il a également ouvert ses portes aux astrophysiciens (qui occupent aujourd'hui près de 40% des ressources informatiques du centre) et récemment, aux biologistes, dont les besoins sont proches de ceux de la physique subatomique. En parallèle, il a en outre développé une expertise dans les technologies de grille informatique (grid computing) et est devenu aujourd'hui un acteur majeur du dispositif en France.

Grâce à son expertise et à son savoir-faire technologiques, le CC-IN2P3 répond ainsi aux besoins informatiques de plus de 2500 personnes et participe à une cinquantaine d'expériences d'envergure dans les domaines de la physique des particules, de la physique des astroparticules, de la physique hadronique et de la matière nucléaire..

Aussi, il est intéressant de présenter l'infrastructure réseau dont dispose l'IN2P3 (également du site officiel de l'IN2P3) :

Les laboratoires de l'IN2P3 sont interconnectés au moyen du réseau RENATER (Réseau National de télécommunication pour la Technologie, l'Enseignement et la Recherche), équipé d'une infrastructure réseau à très haut débit et fournissant ainsi des performances optimales pour l'ensemble des chercheurs. Le CC-IN2P3 joue un rôle essentiel dans la gestion de cette infrastructure puisqu'il a la charge de la connectivité Internet de l'ensemble des laboratoires de l'institut, ainsi que de l'accès aux sites d'expériences où sont produites les données.

Le CC-IN2P3 fait aussi partie des points de présence (points sur lesquels les bénéficiaires se raccordent directement) du réseau AMPLIVA, le réseau régional à haut débit de la Région Rhône-Alpes ainsi que du Réseau Métropolitain Universitaire (RMU), et héberge LYONIX, le premier point d'échange du trafic Internet de Lyon et sa région.

En mettant en valeur un emplacement géographique central, un système de climatisation perfectionné et un personnel hautement qualifié, le CC-IN2P3 a su se placer parmi les candidats naturels à l'hébergement de ces « nœuds ».

Pour en venir au réseau interne (reliant par exemple les machines des fermes de calcul), des switchs sont installés afin de relier les ressources entre elles, celles-ci disposant de carte(s) réseau(x) offrant un débit de 100 Mbit/s ou 1000 Mbit/s.

4.3.1.4 Composant « Storage (CC-IN2P3) »

Cette partie concerne les moyens de stockage présents à l'IN2P3. Aussi, elle rappelle l'importance du troisième chapitre du présent document, nous y reviendrons un peu plus loin.

Principalement, deux niveaux de stockage de données sont utilisés au centre de calcul de l'IN2P3 :

- Stockage sur bandes magnétiques (exemple : HPSS [HPSS], figure 4.4) : ceci reste le premier média de stockage capacitif, mais son inconvénient majeur est son niveau de performance faible lors d'accès multiples. Ce média se présente sous forme de robots automatisés de marque Storagetek), eux-mêmes composés de six silos ayant chacun une capacité de 6000 cartouches (bandes magnétiques de 200Go). Actuellement, la capacité de stockage totale est de 7 Pétaoctets (soit 100000 disques de 70 Gigaoctets).

Figure 4.4 : stockage sur bandes magnétiques

- Stockage sur disques : ceci est utilisé comme complément aux bandes magnétiques, puisque son avantage est la performance des accès disques. De plus, il joue le rôle de cache pour les données stockées sur bandes magnétiques, et est sécurisé (répliqué) afin d'assurer un niveau de service 24/24h durant toute l'année. La capacité de stockage sur disque est d'un peu plus de 100 Téraoctets, répartis dans plusieurs armoires, fournies par IBM ou Hitachi (figure 4.5).

Figure 4.5 : stockage sur disques

Le système de fichiers utilisé, dans le cadre du projet LyNDA, est AFS. Le protocole NFS est également utilisé :

- AFS (Andrew File System) : c'est un système de fichiers distribués accessible à partir de l'ensemble des machines du Centre de Calcul. AFS est basé sur la technique d'un cache local sur chacune des machines du centre de calcul. AFS gère également la synchronisation entre les données locales et les serveurs de fichiers AFS.
- NFS (Network File System) : ce protocole, développé par Sun Microsystems permet le partage de fichiers en réseaux. A la différence de SAMBA (utilisé dans le cadre du projet BIRNy), NFS gère les permissions sur les fichiers, ce qui permet de l'utiliser de manière transparente (à condition que les systèmes en communication gèrent eux aussi les permissions).

Maintenant que nous avons évoqué les systèmes de fichiers et de stockage, nous allons décrire le serveur SRB présent au centre de calcul. Le serveur SRB de l'IN2P3 est installé comme le représente le schéma suivant (figure 4.6) :

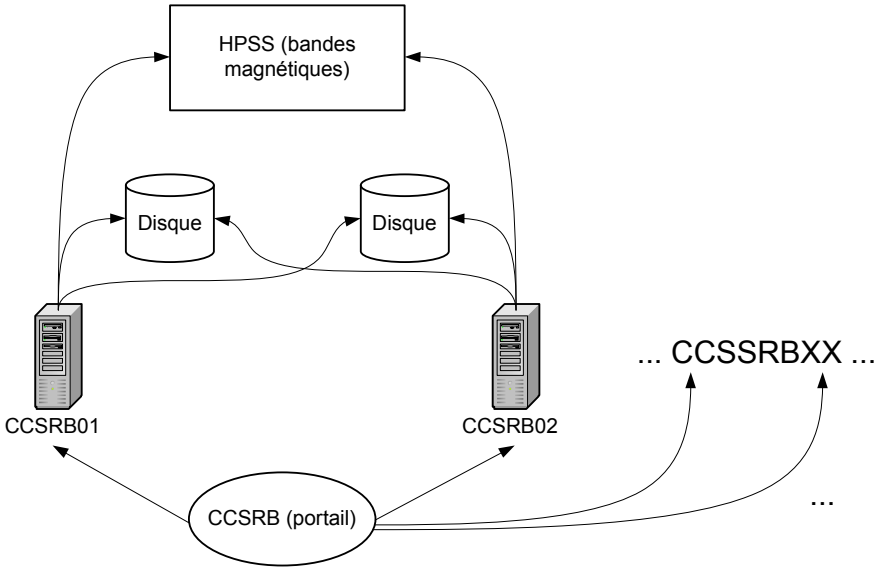


Figure 4.6 : serveur SRB de l'IN2P3

Plusieurs machines sont à disposition des utilisateurs du serveur SRB du centre de calcul, nommées CCSRBxx. Pour rappel, un démon « SRB master » et plusieurs démons « SRB agent » tournent sur chacune de ces machines.

En ce qui concerne les ressources physiques, nous savons que deux systèmes de stockage sont présents au CC-IN2P3 : le stockage sur disques (AFS, NFS) et le stockage sur bandes magnétiques (HPSS : Hight Performance Storage System, présenté lors de la section 4.3.1.4). Dans le cas du projet LyNDA, une seule ressource physique a été utilisée, il s'agit d'une ressource du type « Unix File System » (terminologie de SRB). Concrètement, il s'agit d'une ressource de stockage sur disques.

Le méta-catalogue, quant à lui, est installé dans une base de données de type Oracle.

Nous voyons donc que le modèle actuel est très simple. Il s'agit de voir comment se comportent les clients, avant de passer à l'étape suivante. Celle-ci consiste, avec l'augmentation du nombre de données, à envisager une ressource dans HPSS, avec utilisation de ressources logiques (et composites). Le disque ne sera plus qu'un cache où les fichiers se trouveront temporairement. Concrètement, il pourrait s'agir d'une ressource logique comprenant une ressource physique de type archive (cartouche), accompagnée d'au moins 2 ressources physiques de type « cache » (sur disques). Si une politique de Backup s'avère nécessaire, il faudra envisager un système de réplication. Mais répliquer des données en grands volumes sur des infrastructures comme celles du centre de calcul s'avère relativement onéreux, tout dépendra donc de l'évolution du projet.

4.3.1.5 Composant « Processing (CC-IN2P3) »

Dans un centre de calcul tel que celui de l'IN2P3, des puissances de calcul considérables sont concentrées (figure 4.7), parallèlement aux capacités de stockage évoquées au point précédent. Nous parlons ici d'une puissance de calcul de plus de 3.5 Téraflops. Celle-ci est dispensée essentiellement par une plateforme matérielle sous le système d'exploitation Scientific Linux. Les plateformes IBM/AIX et SUN/Solaris sont également représentées dans des proportions beaucoup moins importantes.

Figure 4.7 : plateformes de calcul

Les moyens de calcul du Centre sont regroupés dans plusieurs fermes de calcul et sont présentés sous forme de puissances mutualisées (les machines ne sont en effet pas dédiées à une expérience ou à un groupe spécifique, mais sont au contraire mutualisées). Ce mode de gestion permet de bénéficier d'un excellent taux de remplissage et d'une optimisation maximale des ressources. Les calculs (les codes de simulation, de traitement et d'analyse de données) s'effectuent de façon indépendante sur un processeur d'une machine.

Ces ensembles de processeurs (dont le nombre varie en fonction de la ferme utilisée) sont gérés par un ordonnanceur de tâches développé par le centre de calcul : BQS (Batch Queueing System). BQS représente un unique point d'entrée commun à tous les utilisateurs pour soumettre des tâches sur les fermes de calcul et gère toutes les plateformes supportées au Centre de Calcul. Des commandes spécifiques (API) existent donc afin de gérer les soumissions, suivis, modifications, … de jobs sur les fermes de calcul.

Deux fermes de calcul ont été utilisées dans le cadre du projet LyNDA :

- Ferme de calcul « ANASTASIE » : cette ferme est composée d'un peu moins de 1500 processeurs, dispersés sur des machines ayant des architectures hardwares différentes (ferme hétérogène). En outre, elle ne supporte que les jobs non-parallèles.

- Ferme de calcul « PISTOO » : cette ferme est composée de 17 machines, équipées notamment de deux processeurs Intel Xéon 2.8Ghz et d'une mémoire Ram de 2 Giga Octets (il s'agit donc d'une ferme homogène). Elle est spécifiquement et uniquement dédiée aux jobs parallèles.

Une première constatation est donc d'ores et déjà à faire : l'IN2P3 jouit d'une infrastructure à la pointe, dont le projet LyNDA pourra bénéficier afin de mener à bien ses objectifs.

4.3.2 Moyens applicatifs

Nous avons présenté, lors du premier chapitre, l'outil principal utilisé par les chercheurs en neuroscience : SPM2 (la version 2002 de SPM, [SPM]). Avant la mise en place du projet, c'est cet outil, et seulement celui-ci, qui était utilisable par les chercheurs.

Le Use Case suivant nous montre un cas d'utilisation classique de cette application :

Lancement d'une tâche SPM	
Utilisateur	**Système (application SPM)**
1. Manifeste son intention de traiter des images (un prétraitement ou une analyse statistique).	
	2. Demande les paramètres correspondant au type de traitement, ainsi que les images à traiter.
3. Donne les paramètres correspondant au type de traitement, ainsi que les images à traiter.	
	4. Réalise la tâche demandée.
Fin du UC	

Trois remarques sont à faire suite à ce Use Case :

- Aucun programme parallèle n'est utilisé.
- Le calcul sur les images est réalisé sur la machine sur laquelle l'utilisateur travaille.
- Les images à traiter sont stockées au moyen de systèmes de fichiers classiques (système de fichiers Linux, …).

Il apparaît que ce mode de fonctionnement, bien que limité, est assez simple puisque l'ensemble des chercheurs adopte cette façon de procéder (utilisation d'une application unique et simple : SPM).

Ce sont les limites de fonctionnement des moyens applicatifs existants (que nous venons de présenter) qui nous amène à des objectifs, que nous allons maintenant fixer. On observe en effet que :

- Du temps (précieux) peut être gagné grâce à l'utilisation d'un programme parallèle.
- Des moyens de calcul mutualisés permettraient également de gagner du temps.
- L'utilisation de SRB permettrait un partage de données entre utilisateurs.

4.4 Objectifs à atteindre

Nous avons pu constater que la méthode de travail actuelle des chercheurs en neuroscience était très simple, et que certaines perspectives d'amélioration s'offraient à nous.

Les objectifs à atteindre sont donc au nombre de trois :

- Permettre aux chercheurs de partager efficacement et facilement leurs données (leurs images médicales). Ce partage améliora d'une part la qualité de leur travail personnel, et d'autre part la qualité du travail collectif (progrès facilité dans le domaine de la neuroscience).
- Utilisation d'un programme parallèle de traitement d'images médicales. L'objectif est évident : gagner du temps !
- Utilisation des ressources de calcul mutualisées. Une fois de plus, un des objectifs et de gagner du temps (le matériel mutualisé est de très bonne qualité). En outre, cela n'oblige pas les chercheurs d'acheter leur propre matériel (qui peut s'avérer onéreux). Enfin, le fait d'utiliser du matériel commun et distant (situé au CC-IN2P3) permet de ne pas « monopoliser » son propre matériel pendant le traitement d'images (calcul très lourd !).

A cela, nous pouvons ajouter d'autres objectifs (plus généraux) que nous avons présentés au chapitre deux, et qui seront remplis au travers des 3 premiers :

- Gain de temps (ferme de calcul et application parallèle).
- Gain de sécurité (toutes les ressources sont communautarisées, sous la surveillance d'une autorité compétente, à savoir l'IN2P3).
- Stockage de larges quantités de données, de manière sécurisée. Accès aux ressources 24h/24h.
- Partage des données et de ressources (faible coût pour l'utilisateur).
- Management des données, et ce, de n'importe quel endroit disposant d'une connexion à Internet.
- Temps réel (Voir perspectives futures du projet).

Nous allons tout d'abord discuter de la manière dont nous pouvons atteindre nos objectifs. Nous discuterons ensuite de l'implémentation successive à nos choix, avant de présenter l'application finale et les perspectives futures de LyNDA.

4.5 Comment atteindre les objectifs ?

Nous devons dès à présent effectuer nos premiers choix afin d'atteindre notre triple objectif. Présentons donc les choix réalisés, objectif par objectif.

4.5.1 Objectif « partage de données »

Le choix réalisé pour atteindre cet objectif a été influencé par une finalité majeure du projet LyNDA : l'entrée dans la communauté BIRN (Biomedical Informatics Research Network, [BIRN]). BIRN est une communauté de partage de données médicales à des fins de recherche (essentiellement dans le domaine de la neuroscience). A l'heure actuelle, la majorité des participants à BIRN sont situés aux Etats-Unis (50 universités et groupes de recherche), mais certains participants commencent à émerger en Europe (deux en Angleterre, un à Lyon ?).

L'architecture de BIRN repose sur la technologie que nous avons présentée au chapitre trois : Storage Resource Broker. Nous avons donc adopté celle-ci afin de faciliter la participation future à BIRN.

4.5.2 Objectif « utilisation d'un programme parallèle »

La stratégie de parallélisation s'avère cruciale dans la perspective d'imagerie médicale en temps réel. Dans un premier temps, cette stratégie doit permettre un gain de temps significatif lors du traitement des images médicales.

Lors du premier chapitre, nous avons évoqué les noms de deux programmes parallèles (traitant des images médicales) existants : Parallel SPM et SPM parallèle. Nous devons dès lors choisir un de ces deux programmes, mais lequel ?

SPM parallèle est porteur d'un désavantage qui fut décisif dans notre décision : il n'a pas été développé pour fonctionner avec la dernière version de SPM (SPM2), mais avec la version SPM99 qui fournit des résultats ayant une valeur scientifique moindre. Par contre, il a l'avantage d'avoir été testé sur un cluster à Strasbourg, et procure de bonnes performances (figure 4.8), surtout pour les étapes de normalisation et d'analyse statistique (un rapport allant jusqu'à 6 où 7 avec 12 processeurs) :

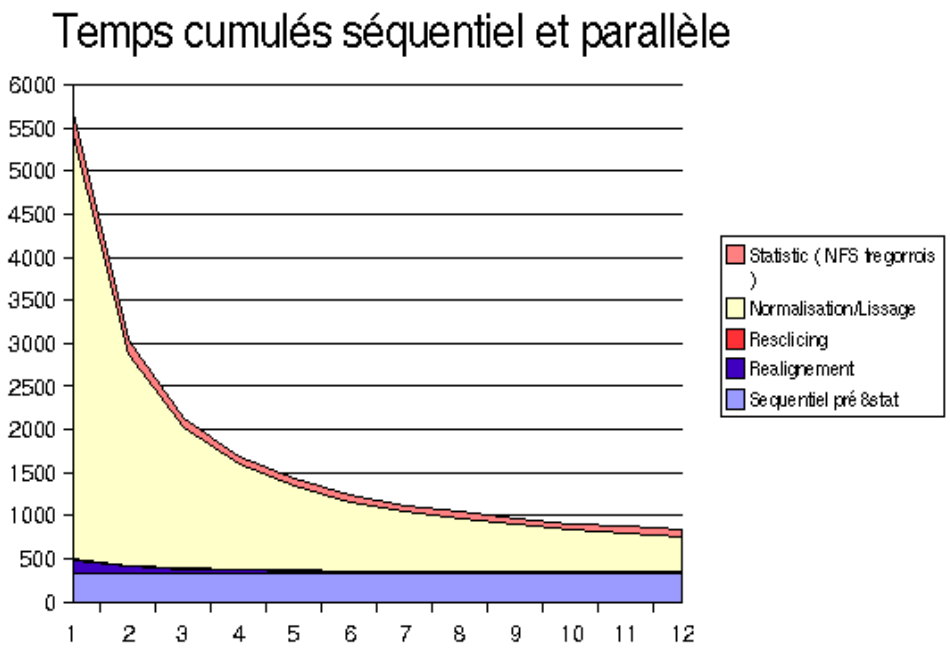


Figure 4.8 : performances de SPM parallèle

En ce qui concerne Parallel SPM (PSPM), qui constitue finalement notre choix, il jouit de l'avantage d'être développé pour fonctionner avec SPM2. Par contre, malgré sa présence dans les Toolbox officielles de SPM2, nous n'avons pas eu de garantie préalable sur ses performances[1].

Voici donc le tableau comparatif entre les 2 solutions :

Critère / Programme	Parallel SPM	SPM parallèle
Compatible SPM2 ?	Oui	Non
Langage	Matlab, C, MPI	Matlab, C, MPI
Testé ?	?	Oui
Performance ?	?	Oui

Nous verrons par la suite comment nous avons permis l'utilisation du programme Parallel SPM.

[1] Personne ne l'a testé, mis à part son concepteur.

4.5.3 Objectif « Utilisation des ressources mutualisées »

Lors de la présentation du composant « Processing (CC-IN2P3) » de l'architecture de la plateforme LyNDA, nous avons cité les noms de deux fermes de calcul (ANASTASIE et PISTOO). Nous avons bien entendu décidé d'utiliser celles-ci afin d'exécuter les traitements d'images sur ces ressources mutualisées.

Nous allons maintenant voir, au travers de l'implémentation, comment nous avons permis l'utilisation des fermes de calcul, du programme parallèle PSPM, ainsi que de la technologie SRB.

4.6 Implémentation

L'objectif de cette section est de donner quelques détails relatifs à l'implémentation d'une application destinée aux chercheurs en neuroscience : PSPMJOB. Ce nom à une double origine : d'une part le nom de l'application (open source) de base que nous avons modifié (SPMJOB), et d'autre part l'application parallèle de traitement d'images médicales (PSPM).

Le développement de l'application PSPMJOB a commencé assez simplement. Le but étant d'aider les chercheurs dans leur travail, nous sommes parti d'une application open source existante nommée SPMJOB ([SPMJOB], figure 4.9), dont les avantages principaux, par rapport au programme communément utilisé par chercheurs (SPM), sont les suivants :

- Les paramètres des traitements effectués sur les images ne doivent être rentrés qu'une fois, dans l'ordre désiré.
- Les erreurs d'encodage se corrigent facilement (retour en arrière).
- Pas besoin de tout recommencer lorsqu'un paramètre est mauvais.
- Présence de paramètres par défaut (pas besoin de tout donner).
- Processus d'exécution facilement répétables (enregistrement des descriptions de jobs dans des fichiers qu'il est possible d'écrire facilement à la main → interface batch).
- Simplification de l'interface batch de SPM.
- Remplacement presque complet de l'interface de SPM, en utilisant les mêmes fonctions.

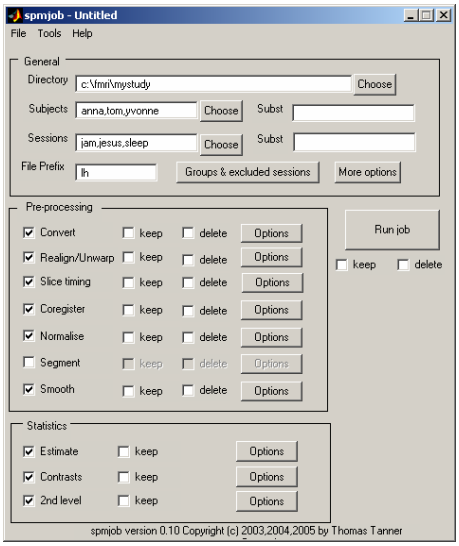


Figure 4.9 : SPMJOB (sous windows)

L'application mise en œuvre (PSPMJOB) est donc une application « patchée » sur base de SPMPJOB. Elle est écrite en MATLAB ([MATLAB], est conçue pour fonctionner sous LINUX et est sous la licence GPL ([GPL]).

Il nous faut à présent voir comment notre triple objectif de départ a été atteint au travers de l'implémentation de PSPMJOB. Nous adopterons la même démarche que lors de la section 4.5 : une discussion objectif par objectif.

4.6.1 Objectif « partage de données »

Dans l'application de départ (SPMJOB), nous savons qu'il est déjà possible de sélectionner des données réparties sur les différents disques auxquels la machine exécutant SPMJOB a accès. Dans le cas du centre de calcul, il s'agit des données partagées (sous le système de fichiers AFS) par les utilisateurs de groupe ISC (Institut des Sciences Cognitives). Mais nous voulons étendre ce mécanisme d'accès de manière plus large, en permettant l'accès aux données stockées dans le serveur SRB du centre de calcul. Comme nous l'avons déjà spécifié, la perspective future d'une entrée dans la communauté BIRN a orienté notre choix technologique vers SRB.

L'implémentation de l'accès et du partage de fichiers stockés dans SRB a donc débouché sur une première modification du code de SPMJOB, afin de permettre à l'utilisateur d'accéder à ses propres données rangées dans SRB, voir à des données partagées par d'autres utilisateurs. Concrètement, les Use Cases suivant nous montrent comment l'utilisateur doit procéder pour accéder à des données stockées via des systèmes de fichiers classiques (AFS et NFS à l'IN2P3), ainsi qu'à des données stockées dans SRB :

Accès à des données stockées via un système de fichiers classique (AFS, NFS)	
Utilisateur	**Système (application PSPMJOB)**
1. Manifeste son intention sélectionner le répertoire de son étude.	
	2. Demande le répertoire de l'étude.
3. Donne le répertoire de l'étude.	
4. Manifeste son intention sélectionner les répertoires de ses patients.	
	5. Demande les répertoires des patients.
6. Donne les répertoires des patients.	
7. Manifeste son intention de sélectionner les répertoires de ses sessions (contenant les images).	
	8. Demande les répertoires des sessions.
6. Donne les répertoires des sessions.	
Fin du UC	

Accès à des données stockées via SRB ***extends*** *Accès à des données stockées via un système de fichiers classique (AFS, NFS)*	
Utilisateur	**Système (application PSPMJOB)**
1. Manifeste son intention d'utiliser SRB.	
Idem que *Accès à des données stockées via un système de fichiers classique (AFS, NFS)* jusque la fin	
Fin du UC	

La seule différence entre les 2 Use Cases est donc que l'utilisateur doit montrer explicitement qu'il veut utiliser SRB (dans le cas du deuxième scénario).

Concrètement, l'implémentation de l'accès aux données rangées dans SRB a été faite essentiellement par l'écriture de code Java. En effet, l'équipe du San Diego SuperComputer Center propose une API Java d'accès à SRB : JARGON [JARGON]. Cette partie du travail a

donc été écrite dans le langage JAVA, ce qui n'est pas un problème puisque MATLAB supporte l'appel à des classes ou méthodes JAVA, et d'ailleurs, les interfaces de MATLAB sont elles aussi écrites en JAVA.

L'écriture de quelques classes et méthodes JAVA nous a donc permis de rendre possible le choix des données présentes dans SRB (à savoir le dossier où se trouvent les patients concernées par l'étude du chercheur (directory), les dossiers de ces patients (subjects), et enfin les dossiers contenant les images(sessions)). Remarquons que le code Java utilisant l'API Jargon permet uniquement de sélectionner les données dans SRB, mais ne permet pas de télécharger les données choisies, ni de renvoyer les résultats après les calculs. Ceci est représenté dans la diagramme de séquence suivant (cas de la sélection du dossier de l'étude, figure 4.10) :

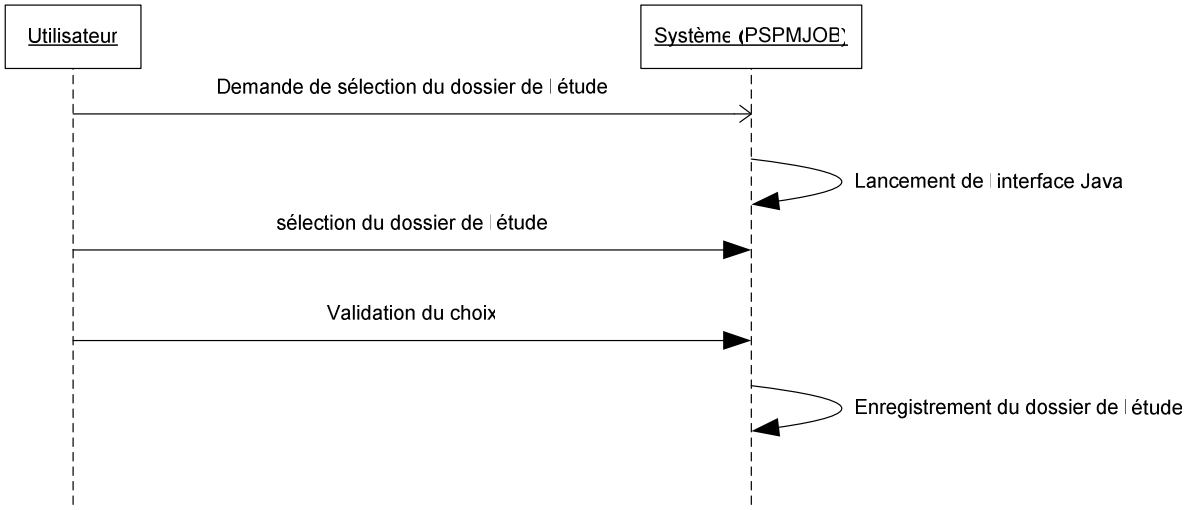


Figure 4.10 : Sélection dans SRB (Sequence Diagram)

Après la sélection des données, nous en venons à un autre problème : le téléchargement des données et le renvoi des résultats. Cet aspect a été implémenté dans PSPMJOB au travers d'appels systèmes, rendu possibles à partir de MATLAB :

- Téléchargement des images : des appels système de la commande 'Sget' SRB permettent d'aller chercher l'ensemble des images, et de les stocker temporairement (le temps des calculs) sur l'espace de stockage AFS.
- Renvoi des résultats : ici, nous utilisons des appels système de la commande 'Sput' SRB, afin d'aller ranger les résultats à l'endroit d'où proviennent les données de base.

Les interactions entre l'application PSPMJOB et le serveur SRB sont représentées dans le diagramme de séquence suivant (figure 4.11) :

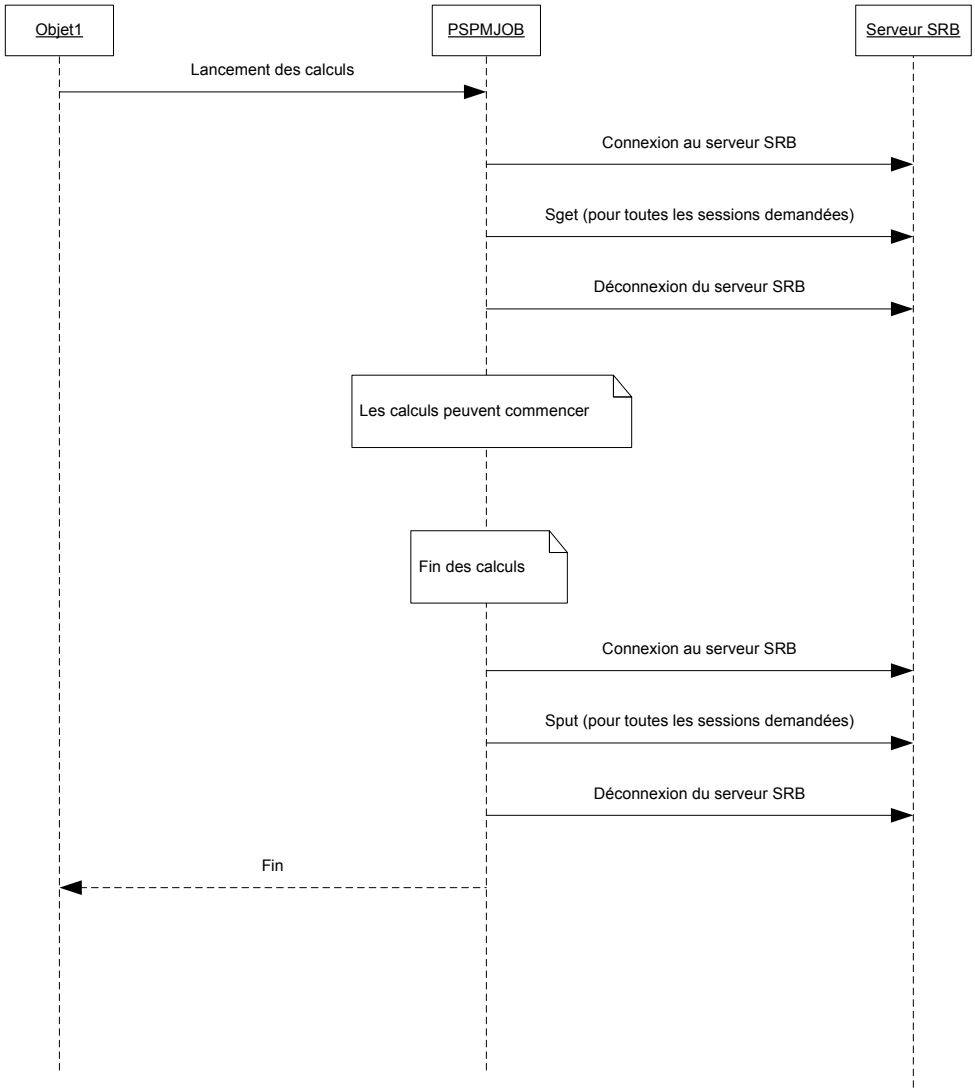


Figure 4.11 : Interactions avec le serveur SRB (Sequence Diagram)

Nous nous sommes exposés ici à un premier problème : la lenteur des commandes 'Sget' et 'Sput' lorsqu'un grand nombre de petits fichiers est concerné. Et c'est bien le cas, puisque nous manipulons des petits fichiers au format '.img' et '.hdr'.

Une première façon d'atténuer ce problème est d'utiliser l'option « bulk copy » proposée par SRB (présentée au chapitre 3, section 3.7).

Une deuxième solution est de stocker l'ensemble des images dans des fichiers Tarball (regroupement de fichiers en un seul fichier physique). Nous ne manipulons donc plus que des fichiers .tar, ce qui fait qu'il ne faut interroger qu'une seule fois lors du téléchargement, et enregistrer un seul fichier lors du renvoi des résultats (au lieu de la faire pour tous les fichiers .img et fichiers .hdr). Cette solution présente l'inconvénient qu'il est impossible de manipuler les fichiers un par un, mais cette pratique est peu courante. De plus, il est impossible de savoir quels fichiers se trouvent dans le Tarball sans le télécharger (à nouveau, il est peu probable d'être amené à le faire).

Au-delà de ces deux solutions, nous proposerons, dans la section 4.8, des optimisations d'envoi de fichiers dans SRB.

Nous avons donc présenté notre solution de téléchargement / renvoi de fichiers images sur le serveur SRB du centre de calcul. Mais cette facette du projet est appelée à se développer dans le futur, par la participation de la communauté Lyonnaise à la communauté BIRN.

4.6.2 Objectif « utilisation d'un programme parallèle »

L'utilisation d'un programme parallèle peut s'avérer être une stratégie cruciale, surtout dans l'optique d'une imagerie fonctionnelle par résonance magnétique en temps réel. Nous avons déjà justifié notre choix de programme parallèle : Parallel SPM.

Même si l'objectif d'un logiciel parallèle est de faire gagner du temps, ce qui peut fortement intéresser les chercheurs, il nous faut éviter de changer la méthode de travail de ceux-ci.

Dans sa version actuelle, PSPM n'est pas très facile à utiliser, et nécessite l'utilisation de deux logiciels l'un à la suite de l'autre : dans la plupart des cas (réalignement, normalisation et analyse statistique), il est nécessaire d'utiliser SPM avant d'utiliser PSPM. Ceci est illustré dans le Use Case suivant (cas du réalignement) :

Réalignement d'images de manière parallèle	
Utilisateur	**Système**
1. Lance l'application SPM 2. Demande un réalignement. 4. Donne les paramètres du réalignement (nombres de patients, sessions, images, …). 6. Lance l'application PSPM. 7. Demande le réalignement. 8. Donne les images à réaligner.	 3. Demande les paramètres du réalignement. 5. Calcul la transformation à appliquer aux images.
Fin du UC	

Le Use Case nous montre très clairement que cette méthode de travail est difficilement acceptable pour l'utilisateur. Dans le cas du réalignement (le fonctionnement est presque similaire dans les autres cas), SPM sert à générer une matrice qui servira à PSPM afin d'appliquer une transformation aux images. Cette partie ne peut pas être parallélisée (impossible de diviser le travail), nous devons donc utiliser SPM2 (non parallèle) pour l'effectuer.

D'autre part, on remarque à la fin du Use Case que le calcul (application de la transformation par PSPM) n'est pas fait de manière automatique. En effet, le calcul doit être lancé manuellement, via un Script Shell dont voici un exemple :

```
#!/bin/csh

mpirun
-np $BQS_PROCNUMBER
-machinefile $BQS_PROCLISTPATH
$THRONG_DIR/local/pspm-2.0.2-beta/LINUX/bin/PSPM
-spm /afs/in2p3.fr/throng/isc/local/spm2
-pspm /afs/in2p3.fr/throng/isc/local/pspm-2.0.2-beta/PSPM_MATLAB_FILES/
-wdir /sps/isc/lhuys/etude/patient/session/ -s reslice
```

Quelques commentaires sur ce script :

- « mpirun » est une commande MPICH [MPICH], servant à lancer un programme utilisant le middleware Message Passing Interface [MPI]. Elle prend plusieurs arguments, dont :
    - « np » : le nombre de processeurs impliqués dans la tâche demandée. Ce paramètre est ici matérialisé par la variable d'environnement BQS_PROCNUMBER (initialisée automatiquement lorsqu'un job est soumis à une ferme de calcul, voir 4.6.3).
    - « machinefile » : les noms des machines impliquées dans la tâche demandée. Ce paramètre est ici matérialisé par la variable d'environnement BQS_PROCLISTPATH (initialisée automatiquement lorsqu'un job est soumis à une ferme de calcul, voir 4.6.3).
    - Le fichier binaire du programme à lancer. Dans ce cas, il s'agit de PSPM. Les arguments suivants sont des arguments du programme parallèle (PSPM).
    - « spm » (argument de PSPM) : répertoire d'installation de SPM (car des fichiers MATLAB de SPM sont nécessaires à PSPM).
    - « pspm » (argument de PSPM) : répertoire d'installation de PSPM (contenant les fichiers MATLAB de PSPM).
    - « wdir » (argument de PSPM) : répertoire contenant les images à traiter, ainsi qu'un matrice générée via l'interface de PSPM (contenant les paramètres propres à la tâche demandée, nécessaire à PSPM). Cette matrice est crée automatiquement par PSPM lorsque l'utilisateur a terminé de rentrer les paramètres dans l'interface de ce programme.

L'exécution de la commande « mpirun » a pour effet de lancer le programme parallèle (ici : PSPM). La stratégie de PSPM est celle du « maître – esclaves » : un nœud maître distribue le travail à ses esclaves, et les coordonne.

La preuve est donc faite que le fonctionnement de PSPM nécessite une adaptation. Le Use Case suivant montre la solution que nous avons implémentée dans PSPMJOB :

Réalignement d'images de manière parallèle (via PSPMJOB)	
Utilisateur	**Système**
1. Manifeste son intention d'utiliser PSPMJOB. 2. Donne le nombre de processeurs qu'il désire. 3. Etc.	
Fin du UC	

L'utilisateur n'a donc que 2 choses à faire : préciser qu'il souhaite utiliser PSPM et indiquer le nombre de processeur requis. Les étapes suivantes sont les mêmes que dans la cas d'une tâche non parallèle (donner les images, paramètres propres au type de traitement, …).

Lorsque l'utilisateur décide de lancer sa tâche, la matrice nécessaire au fonctionnement de PSPM est générée automatiquement par PSPMJOB, et la commande « mpirun » est également lancée avec les arguments nécessaires.

Au niveau performance, il s'est avéré que l'ensemble des prétraitements ne donnait pas de résultats satisfaisants via l'utilisation de PSPM. Les temps obtenus diminuant difficilement avec l'augmentation du nombre de processeurs. Le graphique suivant (figure 4.12) illustre les temps obtenus en fonction du nombre de processeurs, dans le cas d'un réalignement, d'une normalisation et du lissage de 356 images (avec les paramètres par défaut de SPM) :

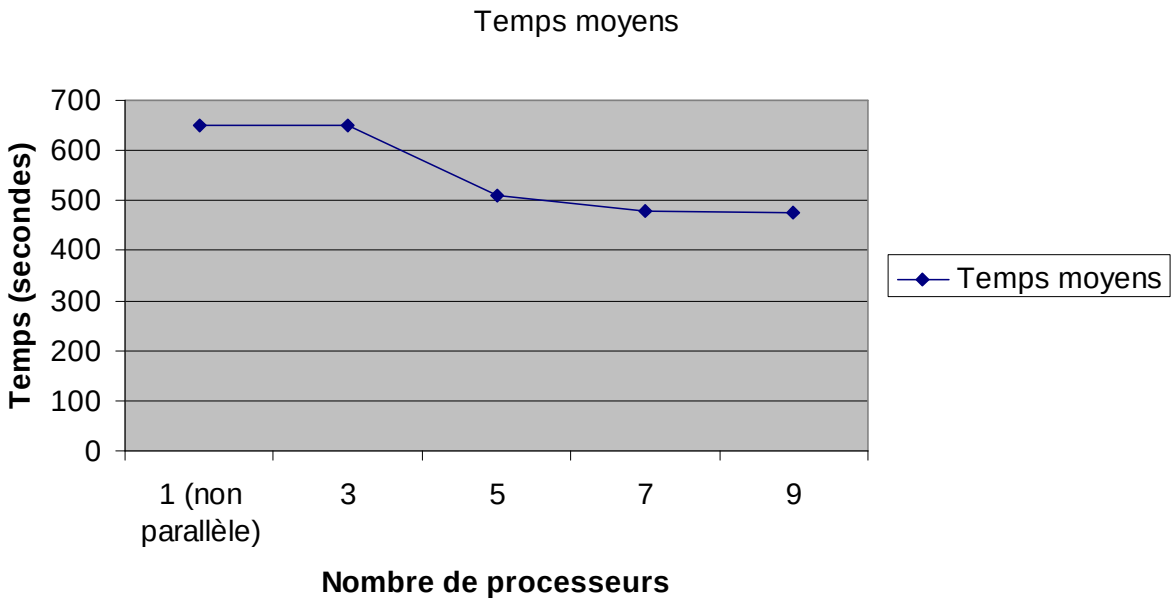


Figure 4.12 : performances de PSPM (pré-traitements)

Nous voyons que l'augmentation du nombre de processeurs n'agit pas aussi significativement que dans le cas de SPM parallèle. Ceci s'explique pour cette raison : dans la cas de la tâche que nous avons lancé (et il en est de même pour d'autres tâches impliquant des prétraitements), environ 50% du temps d'exécution des prétraitements est incompressible ; ce temps incompressible correspond aux différentes parties non réalisables de manières parallèle (génération de matrices MATLAB nécessaires à PSPM). Cette partie de la tâche pénalise donc fortement la suite, et, par conséquent, PSPM n'agit plus que sur les 50% du temps restant.

Les performances obtenues pour la partie estimation statistique sont bien meilleures (figure 4.13). Sans rentrer dans les détails du modèle statistique de nos tests (qui était

volontairement très lourd), nous avons constaté un gain de temps allant jusqu'à un facteur 6 (avec 17 processeurs !) :

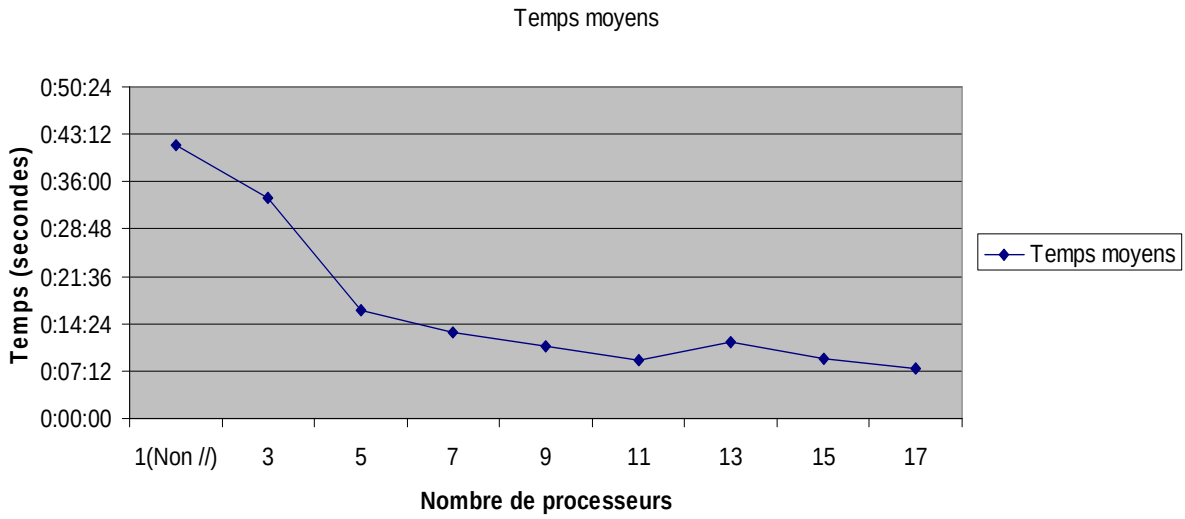


Figure 4.13 : performances de PSPM (estimation statistique)

Pour terminer l'évaluation de notre choix de programme parallèle, signalons un obstacle que nous avons rencontré lors de la mise en œuvre de PSPM : les jetons Matlab (Licences). En effet, toute machine exécutant un programme Matlab doit, au préalable, obtenir un jeton. Ceci est réalisé au CC-IN2P3 via un serveur de licences, et nous a « obligé » à acquérir de nouveaux jetons Matlab (250) afin que plusieurs utilisateurs soient à même d'utiliser l'application parallèle en même temps. (Puisqu'il faut un jeton par machine, et donc par processeur impliqué).

Nous avons donc mis en place une manière très simple d'utiliser PSPM, et évaluer les performances de ce dernier. L'utilisateur garde toutefois le choix d'utiliser l'application parallèle ou non, mais le gain du parallélisme (surtout dans le cas de l'analyse statistique) convaincra facilement les chercheurs de l'utilité d'un tel logiciel.

4.6.3 Objectif « Utilisation des ressources mutualisées »

Au cours de la section 4.3.1.5, nous avons présenté 2 fermes de calcul : une parallèle et une non parallèle. Celles-ci ont été utilisées et, sans surprise, nous avons utilisé la ferme ANASTASIE pour exécuter les tâches non parallèles et PISTOO pour exécuter les tâches parallèles.

Tout comme dans le cas de l'utilisation de PSPM, la soumission d'une tâche (appelée job) à une ferme de calcul est trop fastidieuse pour le chercheur en neuroscience. En effet, pas moins de trois fichiers sont nécessaires afin d'exécuter une tâche sur une ferme de calcul :

- Un fichier MATLAB, décrivant la tâche (MATLAB) à effectuer (prétraitements, analyse statistique). Il est facile de créer ce fichier à partir de PSPMJOB, mais il reste encore deux fichiers à écrire (trop compliqué pour le chercheur).

```
%name : lhuys_20060504220007.m

% spmjob JOB file - generated by spmjob 0.10

job = job_defaults;

…
….
….
runjob(job,'console');% Method who runs the matlab's job
exit;
```

- Un fichier Script Shell permettant de lancer MATLAB sur le premier fichier que nous avons décrit, afin d'exécuter la tâche MATLAB.

```
%name : lhuys_20060504220007_script.sh

#!/usr/local/bin/tcsh
cd /afs/in2p3.fr/throng/isc/data_spm/
%run matlab
matlab -nodisplay -r lhuys_20060504220007
```

- Un fichier Script Shell réalisant la soumission proprement dite à la ferme de calcul. Au CC-IN2P3, la soumission d'un job sur une ferme de calcul se fait via l'API Batch Queuing System [BQS]. Plusieurs commandes sont donc fournies afin de soumettre, monitorer, supprimer, … des jobs sur les fermes de calculs de l'IN2P3. La commande de soumission s'appelle « qsub ». Les paramètres pertinents dans le cadre de notre application sont les suivants :
    - « q » : la classe du job. L'IN2P3 défini plusieurs classes (types) de job, en fonction de la tranche temps allouable au job, de la mémoire virtuelle, …
    - « N » : le nom du job.
    - « M » : la mémoire physique demandée.
    - « plateform » : la plateforme désirée.

- Dans le cas d'un job parallèle, un argument supplémentaire (nommé « proc ») est nécessaire afin de donner le nombre de processeurs désiré.
- Le dernier argument est le chemin vers le Script Shell permettant de lancer la tâche désirée (dans notre cas, le Script Shell lançant MATLAB).

Sur quelle ferme de calcul le job est il soumis ? Cela dépend de la valeur de la variable d'environnement BQSCLUSTER. Si la valeur de cette dernière est « PISTOO », le job sera soumis sur la ferme PISTOO (parallèle) ; si aucune valeur n'est fixée, le job sera soumis (par défaut) sur la ferme ANASTASIE (non parallèle).

```
%name : lhuys_20060504220007_batch.sh

#!/usr/local/bin/tcsh
unsetenv BQSCLUSTER
qsub -q T -N lhuys_20060504220007 -l t=100000 -l M=1024MB -l
platform=LINUX /afs/in2p3.fr/throng/isc//lhuys_20060504220007_script.sh
```

Un exemple complet de fichiers relatifs à la soumission d'un job sur une ferme de calcul est en annexe (Annexe D).

Il est donc nécessaire de faciliter la soumission d'un job sur une ferme de calcul. A ces fins, l'utilisateur devrait juste manifester son choix d'exécuter sa tâche sur une ferme de calcul (job batch) ou sur la machine sur laquelle il exécute PSPMJOB (job interactif). L'utilisateur devrait naturellement se diriger vers le mode « Batch », puisque celui présente plusieurs avantages : pas de monopolisation de la machine de travail de l'utilisateur, gain de temps, possibilité de lancer plusieurs jobs en même temps. Par contre, l'utilisateur peut avoir le sentiment de « perdre le contrôle » de sa tâche, il se rendra compte par la suite que ce sentiment est erroné puisqu'il peut soumettre, annuler et voir l'état de ses jobs à son gré.

Aussi, le monitoring d'un job doit être rendu plus facile :

- La commande (BQS) « qcat <nom_du_job> » permet de voir l'output (impression à la sortie standard) d'un job.
- La commande (BQS) « qjob <nom_du_job> » permet de voir l'état des jobs en cours (s'ils sont en queue, en cours, …).
- La commande (BQS) « qjobp <nom_du_job> » permet de voir l'état des jobs en cours (s'ils sont en queue, en cours, …), mais de manière plus détaillée.
- La commande (BQS) « qdel <nom_du_job> » permet de supprimer un job.

Tous ces aspects seront englobés dans une interface graphique (développée en JAVA), permettant à l'utilisateur une gestion plus aisée.

Afin de profiter un maximum des ressources à dispositions (1500 processeurs dans le cas de la ferme ANASTASIE), plusieurs jobs peuvent être lancés simultanément, alors que

l'utilisateur ne fait qu'une requête. Ceci s'explique lorsque l'utilisateur lance une étude comprenant plusieurs groupes de patients. Dans ce cas précis, un job par groupe de patients sera lancé sur la ferme de calcul, chaque job étant indépendant par rapport aux autres. Par exemple, si on demande le traitement de 20 patients, regroupés en groupe de 4 patients (ces groupes étant indépendants les uns des autres), 5 jobs seront lancés sur la ferme de calcul !

Le facteur d'accélération de cette approche est difficilement mesurable. Dans le meilleur des cas, celui où l'ensemble des jobs lancés se voient tous allouer des ressources (processeurs) en même temps, le facteur d'accélération sera au mieux égal au nombre de groupes de patients (5 dans l'exemple ci-dessus). En effet, il faut rappeler que la ferme de calcul peut être hétérogène (ANASTASIE), et donc les jobs peuvent progresser à des vitesses différentes. Même si la ferme est homogène, d'autres facteurs imprévisibles comme le réseau peuvent rentrer en ligne de compte. Un dernier facteur, difficilement contrôlable, peut faire varier la vitesse d'un job : l'occupation de la machine, puisque plusieurs jobs peuvent être lancés sur une même machine (2 jobs sur une machine biprocesseur par exemple).

En résumé, l'utilisateur choisi par lui-même s'il veut exécuter (ou non) sa tâche sur une ferme de calcul. Si c'est le cas, la soumission de job(s) est effectuée à sa place et une interface graphique lui est offerte afin de monitorer ses jobs.

4.7 Application PSPMJOB

Nous venons de décrire l'implémentation relative aux trois objectifs que nous nous sommes fixé. Cette implémentation a, pour rappel, donné naissance à une application nommée PSPMJOB. Celle-ci est située dans le composant « User interface (CC-IN2P3) » de notre architecture LyNDA (section 4.2).

Chaque utilisateur aura accès à PSPMJOB, après une connexion fructueuse à l'IN2P3 (voir section 4.3.1.2).

Nous allons maintenant décrire cette application au travers de captures d'écran et de commentaires sur ces dernières.

4.7.1 Interface de base

La figure 4.14 nous montre l'interface d'accueil de PSPMJOB, ainsi que les différents ajouts par rapport à l'application de départ (SPMJOB).

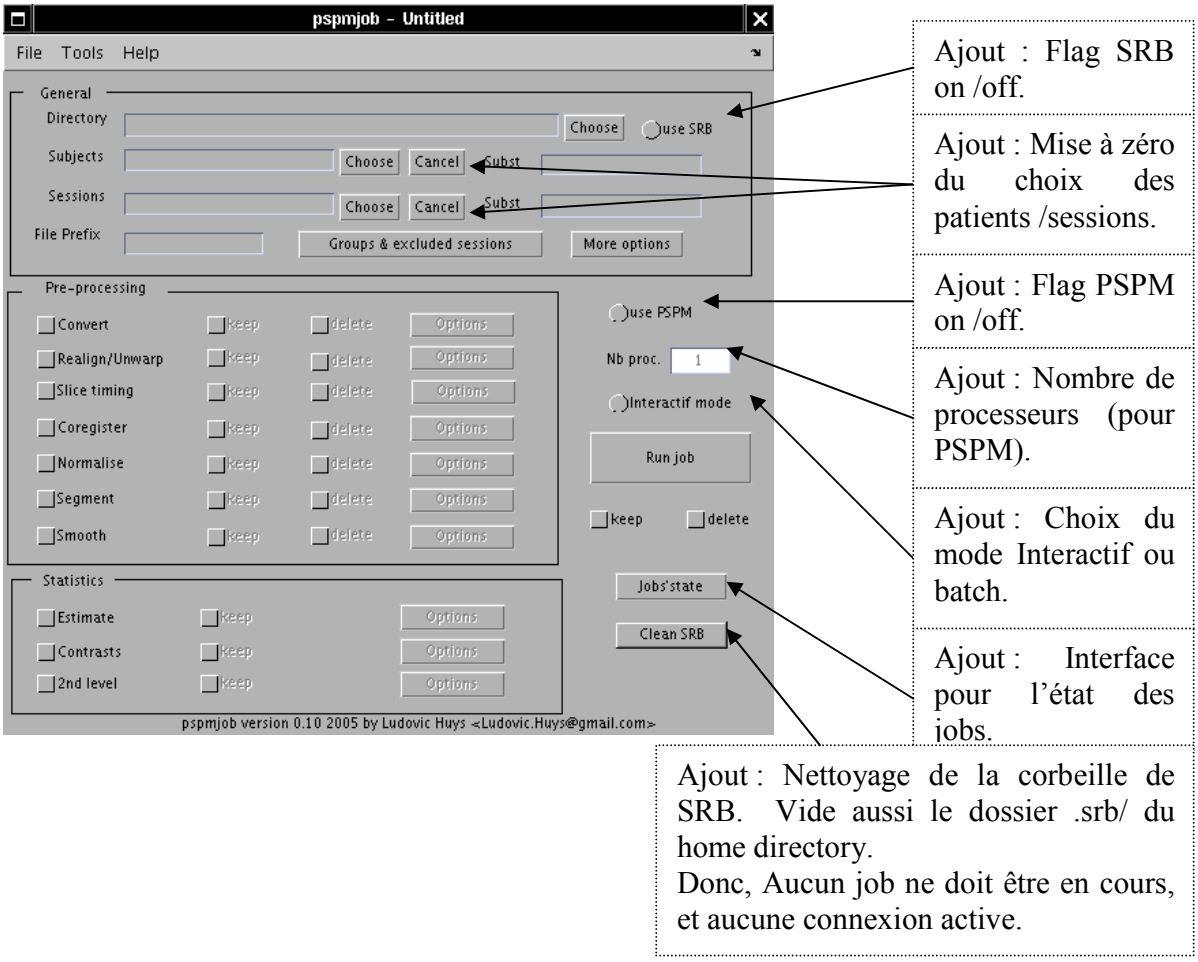


Figure 4.14 : accueil PSPMJOB

Détaillons maintenant les ajouts liés à chacun des objectifs.

4.7.2 Objectif « partage de données »

L'utilisateur a le choix d'utiliser SRB ou non (Bouton SRB on/off « use SRB »). De plus, 2 boutons de remise à zéro (« cancel ») du choix de patient(s) et session(s) ont été ajouté.

Lorsque l'utilisateur clique sur les boutons « choose » pour le dossier de l'étude, les dossiers des patients, et les dossiers des sessions, les interfaces (JAVA) suivantes apparaissent (figure 4.15) :

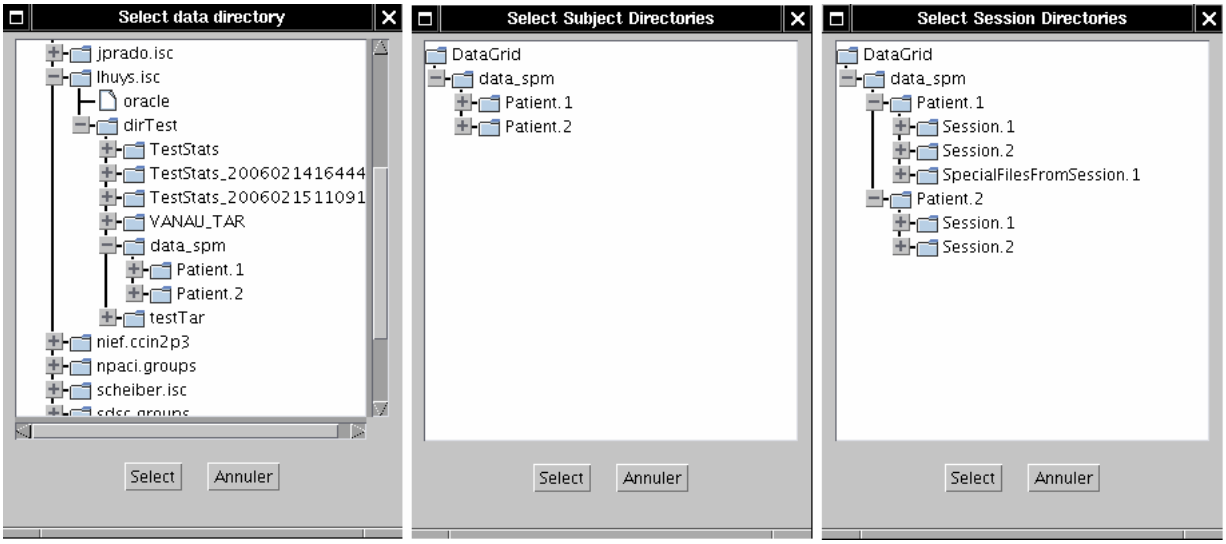


Figure 4.15 : sélection dans SRB

Enfin, un bouton a été ajouté afin de vider la corbeille de SRB (toute collection SRB qui est supprimée va dans une collection spéciale, appelée « trash »), ainsi que les fichiers temporaires créés lors des connexions vers SRB (un fichier temporaire est créé par la commande « Sinit » (contenant les informations de connexion, et n'est pas supprimé si la commande « Sexit » n'est pas utilisée).

4.7.3 Objectif « utilisation d'un programme parallèle »

A nouveau, l'utilisateur a le choix d'utiliser PSPM ou non (bouton « use PSPM »). Pour le reste, aucune modification d'interface n'est à signaler. Seul le monitoring des tâches diffère (puisque plusieurs processeurs sont impliqués).

4.7.4 Objectif « Utilisation des ressources mutualisées »

Pour rappel, l'utilisateur garde la possibilité d'exécuter ses tâches de manière interactive, mais l'utilisation de fermes de calcul est vivement conseillée.

Le monitoring des jobs se fait via l'interface (JAVA) suivante (figure 4.16) :

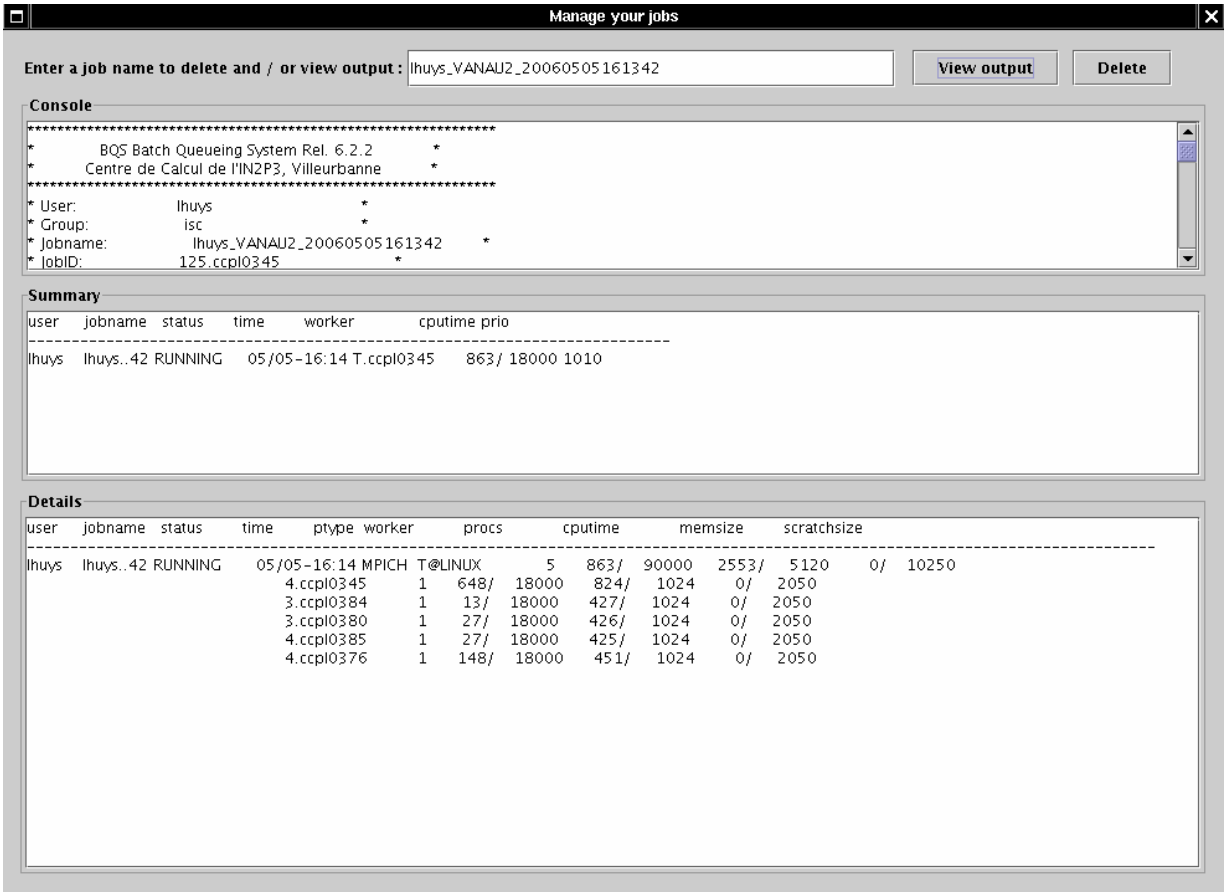


Figure 4.16 : monitoring des jobs

L'utilisateur a accès à différentes informations et fonctions via cette interface :

- L'output (impression à la sortie standard) d'un job (bouton « view output »).
- Possibilité de supprimer un job (bouton « delete »).
- Accès au statut des jobs (de manière sommaire et détaillée), fournissant certaines informations (rafraîchies après 30 secondes) dont :
    - Non d'utilisateur.
    - Non du job (générer automatiquement pas PSPMJOB).
    - Statut (dans l'exemple : Running → des machines sont allouées au job, qui peut s'exécuter).

- Temps CPU : Temps CPU consommé et demandé (par PSPMJOB). Des détails supplémentaires sur l'unité de temps utilisée au CC-IN2P3 sont disponibles en ligne ([BQS]).
- Mémoire (physique et scratch) : mémoire physique demandée (par PSPMJOB) et consommée. Idem pour la mémoire scratch (zone de mémoire locale et temporaires pouvant être utilisée par le job).

Nous avons donc présenté le « design » de PSPMJOB, conséquent au trois objectifs que nous devions remplir.

4.8 Des algorithmes d'envoi de données dans SRB

Après avoir présenté la problématique liée à la lenteur d'envoi de données dans SRB (section 4.6.1), nous allons présenter des solutions théoriques et algorithmiques permettant d'atténuer ce phénomène.

4.8.1 Nature du problème

Lorsqu'on exécute, sur un ensemble de petits fichiers (ici, des fichiers images de plus ou moins 200Ko et des fichiers headers de plus ou moins 500 Octets), les commandes SRB 'Sget' et 'Sput', il subvient un phénomène de lenteur dû à l'implémentation de l'équipe de San Diego Supercomputer Center :

- Dans le cas de la commande 'Sget', le phénomène est moins grave, puisque comme dans tout système de fichiers, les fichiers sont envoyés sur le réseau un à un. L'option « Bulk copy » réduit le problème, mais cela reste plus lent que dans le cas d'un transfert d'un gros fichier d'une taille équivalente à l'ensemble de tous les petits fichiers.
- Le cas de la commande 'Sput' est plus grave, puisque lorsqu'on envoie un ensemble de N fichiers, le serveur SRB effectue N requêtes vers le MCAT et enregistre les fichiers un par un ! (N connexions et déconnexions sur la base de données !). Ceci est donc à améliorer pour les prochaines versions de Storage Resource Broker.

Ce dernier constat (commande 'Sput') nous amène donc à 2 solutions algorithmiques développées pour palier au phénomène, avec les outils dont on dispose dans la version actuelle de SRB.

4.8.2 Solutions proposées

Les deux solutions développées sont très différentes l'une de l'autre, mais présentent des avantages / inconvénients non-négligeables. Celles-ci ont en outre été développées en Java.

La première solution consiste en l'utilisation de fichiers Tarball. La deuxième, plus compliquée, « découpe » l'ensemble des fichiers à envoyer, et les envoie de manière quasi-équitable entre ressources physiques disponibles dans SRB. Nous allons présenter ces 2 solutions.

4.8.2.1 Utilisation de fichiers Tarball

Dans cette solution, le contenu du dossier que l'utilisateur souhaite envoyer est ajouté dans un fichier Tarball ('.tar'). Ce fichier est ensuite envoyé au lieu d'envoyer l'ensemble de tous les fichiers du dossier de base. Le diagramme suivant montre le fonctionnement de l'algorithme (figure 4.17) :

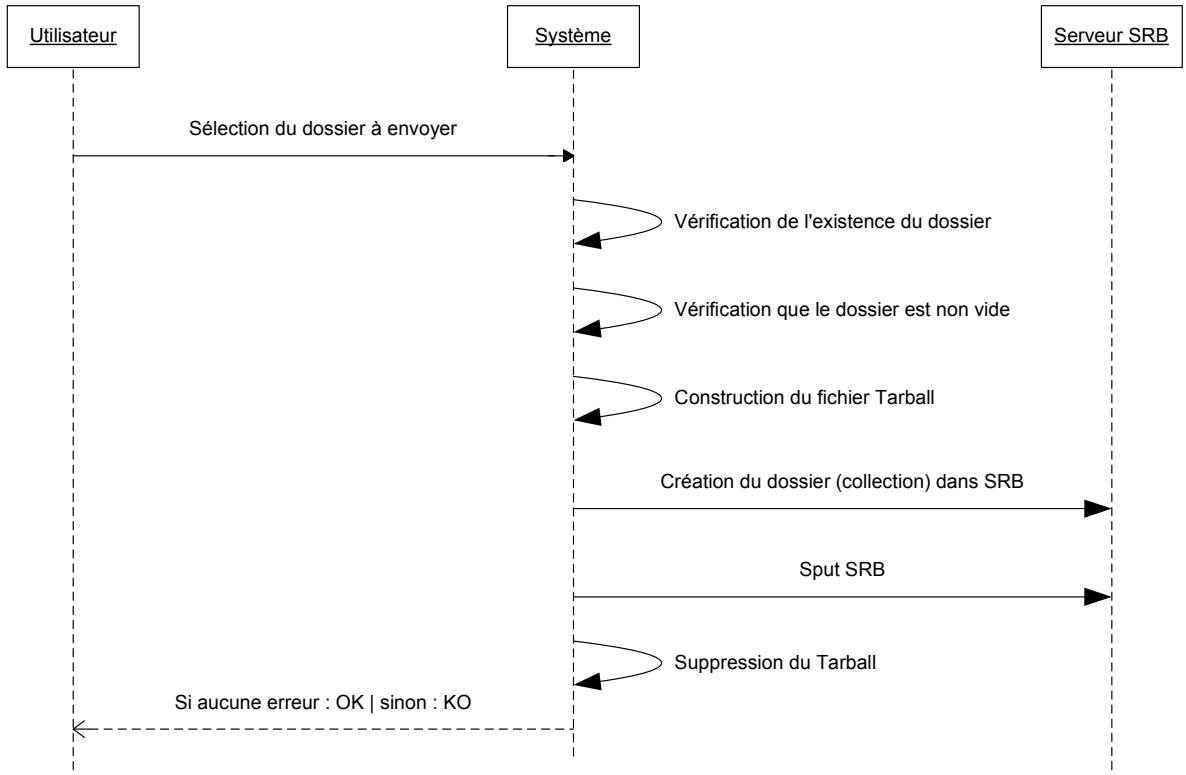


Figure 4.17 : solution par fichier Tarball

Cette solution est donc très simple et présente les avantages et inconvénients suivants :

+ Simplicité d'implémentation.

+ Rapidité.
+ Très pratique pour l'envoi de petits fichiers.
− Pas de possibilité de lister le contenu du .tar (obligation de télécharger).
− Inutile pour l'envoi de gros fichiers.
− Utilisations des ressources physiques non-optimales : tout est envoyé dans une seule ressource physique.

Une extension possible de cet algorithme est d'ajouter, manuellement, des renseignements sur le contenu du Tarball. Nous parlons ici de la création de méta-données personnalisées, par la création manuelle de tables dans la base de donnée Oracle, et l'ajout d'informations sur les fichiers contenus dans les fichiers Tarball présents dans SRB. Ce mécanisme est assez fastidieux à utiliser, puisqu'il faut créer les méta-données lors de l'envoi du fichier Tarball, et effacer celles-ci en cas de suppression (mais il est impossible d'empêcher l'utilisateur d'effacer ses fichiers Tarball sans effacer les méta-données personnalisées associées…). Remarquons qu'il est trop compliqué d'utiliser les méta-données simples (faciles à insérer et effacées par le serveur en cas de suppression du fichier associé), puisqu'il est possible qu'il y ait beaucoup de fichiers dans le fichier Tarball (dès lors, comment enregistrer cette « grosse » information, si ce n'est par des méta-données personnalisées ?).

4.8.2.2 Découpage du dossier à envoyer

Cette deuxième solution est plus compliquée puisqu'elle dépend du nombre de ressources physiques à disposition (N), afin d'envoyer (plus ou moins) 1/N de la taille totale du dossier à envoyer dans chacune d'entre elles. Cette manière de procéder est illustrée dans le schéma suivant (figure 4.18) :

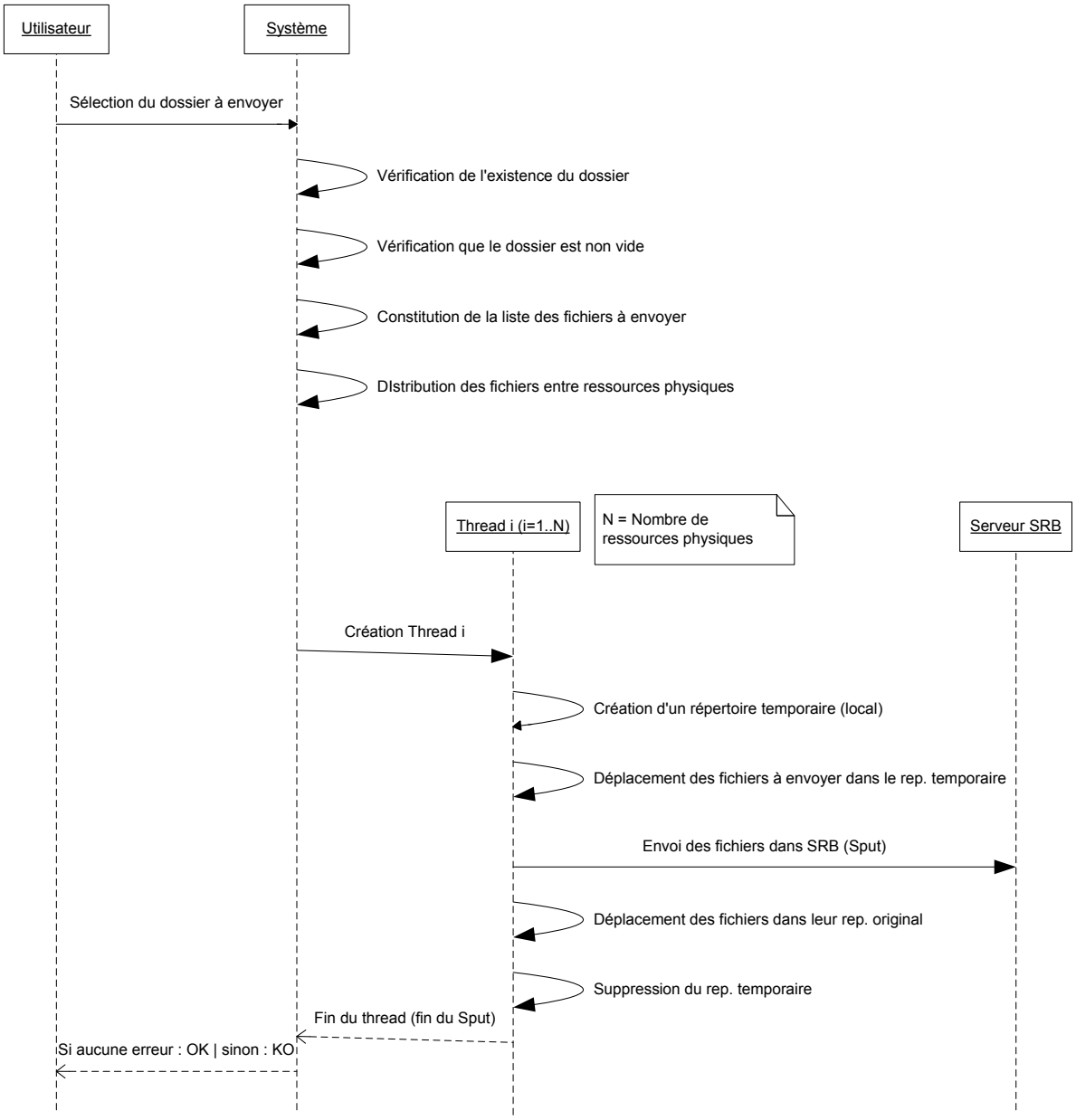


Figure 4.18 : solution par découpage du dossier à envoyer

Cette solution est donc plus compliquée et présente les avantages et inconvénients suivants :

+ Meilleure utilisation des ressources physiques (chaque ressource reçoit une quantité de fichiers plus ou moins équitable au niveau de la taille totale grâce à un algorithme de répartition très simple: le round-robin ou tourniquet).
+ On garde la possibilité de lister le contenu de la collection dans SRB.
+ Pratique aussi bien dans le cas de petits fichiers que de gros fichiers (l'algorithme de répartition se base sur la taille des fichiers pour répartir ceux-ci).
– Implémentation plus fastidieuse (répartition des fichiers, déplacements des fichiers dans des répertoires temporaires, Threads Java, …).
– Vitesse (cf. section 4.8.2.3)

Une extension possible de cette solution est une répartition pondérée des fichiers à envoyer. Par exemple, dans le cas où on dispose de 3 ressources physiques, envoyer respectivement 3 /8, 3/8, 2/8 de la taille totale dans chacune d'entre elles. Ce qui permettrait de donner une priorité aux ressources les plus rapides.

4.8.2.3 Comparaison des performances

Pour commencer, voici un tableau récapitulatif des avantages et inconvénients des deux méthodes :

Critère / Méthode	Tarball	Découpe
Simplicité d'implémentation	+	-
Rapidité	+	+/- (voir plus loin)
Type de fichiers	+ (petits fichiers) - (gros fichiers)	+
Possibilité de lister le contenu dans SRB	-	+
Utilisation des ressources physiques	-	+

En outre, ces algorithmes ont été testés dans le cadre de transfert de fichiers vers un serveur SRB installé pour les besoins de ces tests. Ce serveur SRB était doté de trois ressources physiques (importantes pour l'algorithme d'envoi par découpe du dossier à envoyer) : 2 ressources physiques situées sur le disque local de la machine où était installé le serveur, et une ressource physique sur un disque réseau (accessible par NFS, donc plus lent).

Les résultats obtenus sont repris dans le tableau et le graphique suivant (figure 4.19) :

Temps moyen (en secondes) par méthode et par fichiers envoyés				
Méthode / Fichiers envoyés	**250** fichiers (**125** .img de taille 208Ko et **125** .hdr de taille 348 Octets)	**500** fichiers (**250** .img de taille 208Ko et **250** .hdr de taille 348 Octets)	**750** fichiers (**375** .img de taille 208Ko et **375** .hdr de taille 348 Octets)	**1000** fichiers (**500** .img de taille 208Ko et **500** .hdr de taille 348 Octets)
Sput classique (avec l'option bulk copy)	230	423	573	689
Découpe du dossier à envoyer	235	420	566	683
Tarball	11	17	24	32

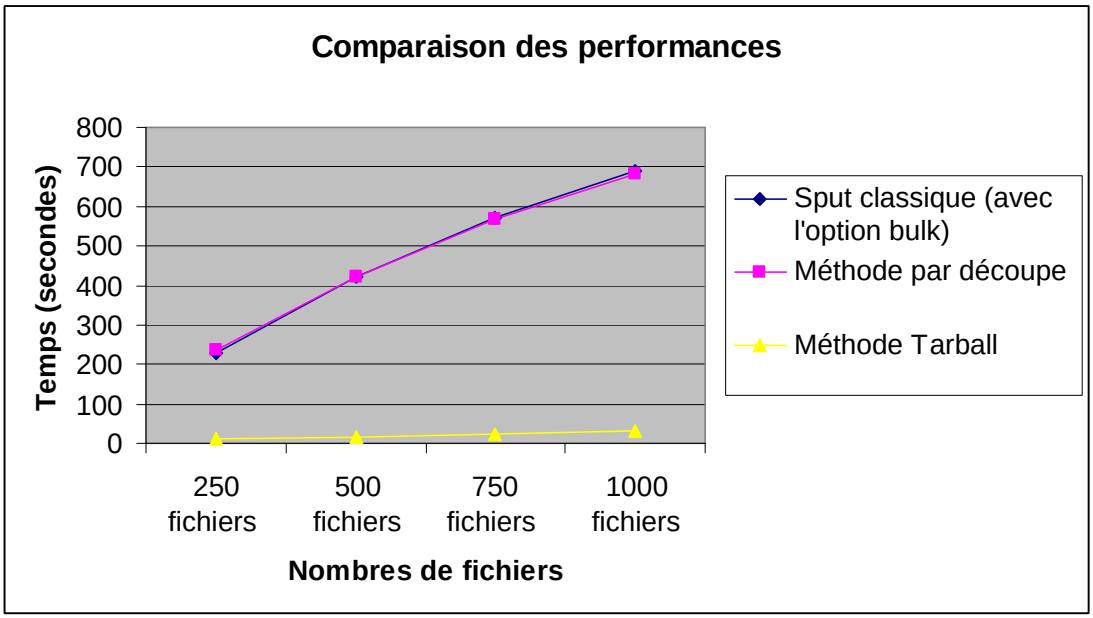


Figure 4.19 : comparaisons des performances des algorithmes d'envoi dans SRB

Nous pouvons constater que la méthode classique (Sput avec l'option bulk) et la méthode par découpe du dossier à envoyer donnent des résultats plus moins équivalents. Ceci s'explique par la raison suivante : dans les 2 cas, le « goulot d'étranglement » se situe au même endroit, à savoir l'enregistrement des fichiers dans le méta-catalogue (qui se fait un par un !). Par contre,

la méthode par fichier Tarball est très rapide, puisqu'un seul fichier est enregistré dans le MCAT (la vitesse d'exécution devient donc dépendante du réseau).

Les 2 méthodes que nous avons construites donnent donc des résultats radicalement différents, tout en présentant leurs propres avantages et inconvénients.

4.9 Perspectives futures de LyNDA

Le projet LyNDA est actuellement dans une phase de démarrage et d'ascension progressive (en terme de nombre d'utilisateurs de la plateforme). Ce projet peut se développer suivant 2 axes :

- Axe Français : à l'heure actuelle, seuls les membres de l'institut des sciences cognitives de Lyon utilisent notre plateforme. Nous savons d'ores et déjà que d'autres institutions travaillant dans le domaine de la neuroscience sont intéressées de rejoindre le projet. Ceci permettrait bien entendu d'avoir de plus larges quantités de données à disposition, remplissant encore mieux notre objectif de collaboration entre chercheurs.
- Axe International : nous avons déjà évoqué cet aspect de progression du projet, lors de la section 4.5.1. Pour rappel, il s'agit de l'entrée dans la communauté BIRN.

Aussi, rappelons un des objectifs finaux du projet LyNDA : l'imagerie fonctionnelle par résonance magnétique. Deux projets de ce type sont présentés en Annexe A et Annexe B. Les premières briques mises en place au CC-IN2P3 constituent un excellent début vers la stratégie temps réel.

Le projet LyNDA semble donc avoir un avenir prometteur, tout dépend bien entendu du nombre de chercheurs qui s'impliqueront dans celui-ci, c'est le facteur clé de la réussite d'un tel projet.

Conclusion

Nous voici arrivé au terme de notre réflexion sur notre plateforme au service de la neuroscience. Ce travail constitue un recueil d'informations sur la mise en place d'un plateforme de traitement d'images cérébrales, acquises au moyen de la résonance magnétique nucléaire. Nous avons pu constater que le trajet vers ce type de plateforme, en temps réel, n'est pas trivial.

La compréhension du domaine d'application est un élément crucial afin de fournir des outils performants et répondant aux exigences des utilisateurs (les chercheurs en neuroscience). Nous en avons présenté les bases, nécessaires à la compréhension et l'élaboration de la plateforme, au travers de définition des prétraitements et de l'analyse statistique des images, ainsi qu'une présentation des logiciels (deux parallèles et un non parallèle) qui implémentent ces fonctions.

Avant de rentrer dans les détails de notre architecture, nous en avons présenté les principes et exigences. Ceci nous a permis de prendre conscience de la palette de choix technologiques à faire par la suite.

Comme nous l'avons signalé, certaines « briques » préexistaient à notre contribution que nous avons présenté au cours du chapitre consacré au projet Lyon Neuroimaging Database and Application (LyNDA, chapitre 4). Ces briques sont, pour rappel, le transfert des images au format DICOM entre le centre d'acquisition (CERMEP) et le centre de calcul (CC-IN2P3), la connexion des utilisateurs au CC-IN2P3, ainsi que les moyens de calcul et de stockage (services rendus par le centre de calcul). Aussi, nous avons pris connaissance des « faibles » moyens applicatifs existants, ne permettant pas de partager efficacement des données médicales, d'utiliser les moyens de calcul mutualisés (fermes de calcul), et d'utiliser facilement un logiciel parallèle de traitement d'images.

Ce sont les limites de cet existant qui justifient notre contribution, axée selon trois exigences. Premièrement, nous avons mis en place un partage facile et efficace des données médicales entre chercheurs, grâce à l'utilisation de la technologie Storage Resource Broker (présentée en détails au cours du troisième chapitre). Au-delà de l'utilisation ce cette technologie, nous avons proposé deux améliorations concernant l'envoi de fichiers vers celle-ci. Deuxièmement, les utilisateurs sont maintenant à même d'utiliser un programme parallèle pour traiter leurs images, ce qui permet de gagner beaucoup de temps. Troisièmement, la soumission de jobs sur les fermes de calcul du CC-IN2P3 permet aussi d'améliorer le rendement des chercheurs (les machines de bonne qualité étant dédiées au calcul).

L’ensemble de notre contribution est matérialisé dans l’implémentation d’une application, nommée PSPMJOB, qui respecte les 3 exigences que nous avons développé.

Nous savons dès à présent que le projet LyNDA ne s’arrêtera pas à notre contribution, puisque nous savons que différentes finalités devraient être atteintes à moyen terme : l’entrée dans la communauté BIRN, la participation d’autres réseaux de chercheurs en France, et bien entendu l’aspect temps réel de l’imagerie médicale.

En se proposant de présenter les premiers pas vers ces différentes perspectives futures, nous espérons que ce travail aura pu apporter un éclairage relativement critique, permettant ainsi d’avoir une vue d’ensemble sur le domaine d’application, ainsi que sur les enjeux de la mise en place d’une plateforme telle que celle que nous avons proposé.

Annexe A

Real-Time Functional MRI Using a PC Cluster

Il s'agit ici des travaux réalisés par une équipe de recherche d'Osaka (Japon) [BMN03]. Le but principal est d'évaluer les performances d'un cluster, la tâche de celui-ci étant de traiter des images fonctionnelles en temps réel. Par temps réel, il faut entendre que les processus d'acquisition, de reconstruction, de prétraitement et d'analyse statistique doivent être exécutés en un temps plus petit que le TR (Répétition Time).

Les auteurs insistent sur le fait que plus la qualité des images augmente (donc le nombre de voxel), plus le défi du temps réel est difficile : certaines équipes de recherche ayant fait le même type de travail ne considère pas certaines étapes lourdes en calcul (comme le réalignement) afin d'arriver à des résultats favorables.

De plus, un algorithme efficace d'estimation du modèle générale linéaire (utilisé dans la partie statistique) est utilisé sur le cluster. Le but de celui-ci est d'obtenir des résultats comparables en terme de qualité à ceux obtenus par les outils tels que SPM99 ou SPM2.

Le processus mesuré en temps réel est le suivant : réalignement, lissage et analyse statistique.

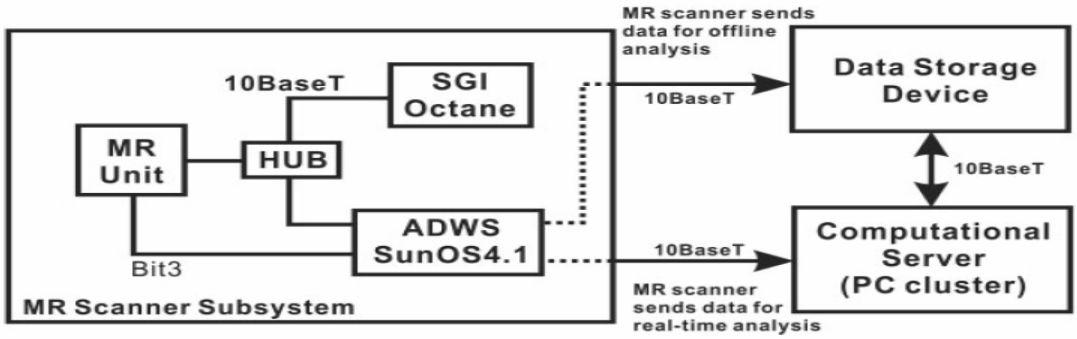
Analysons maintenant plus en détail la manière dont l'équipe a mené ses études, et les résultats obtenus.

- Système utilisé (figure A.1)

  Le système utilisé pour les recherches menées par l'équipe d'Osaka, composé d'éléments softwares et hardwares, est divisé en sous-systèmes selon la fonction de ceux-ci (le sous-système « Mr scanner subsystem » est le sous-système correspondant à l'acquisition des données, le sous-système « data storage device » stocke les données, le système « computational server » réalise les calculs). De plus, il est prévu pour fonctionner de manière offline (acquisition et stockage dans le sous-système de stockage dans le but d'une analyse future avec SPM99 ou SPM2) ou online : acquisition dans le sous-système d'acquisition, envoi direct dans le sous-système de calcul (et calcul avec notamment l'algorithme d'évaluation du modèle général linéaire), stockage des données et résultats dans le sous-système de stockage.

  Nous n'entrerons pas dans les détails des composants hardware. Ceux-ci sont principalement : un imageur par résonance magnétique, un réseau Ethernet (10BaseT) et un cluster de calcul de 8 machines (composé de machines classiques et pas trop onéreuses). Les machines de ce cluster sont des biprocesseurs Pentium 3 de 800 MHz, ayant une mémoire physique de 1 Giga (ce qui donne 16 nœuds de calcul).

Au niveau logiciel, les caractéristiques principales sont les suivantes : les modules écrits par l'équipe japonaise sont en C (avant MPI ([MPI],[MPICH]) comme middleware pour le calcul en mode master-slave), le système d'exploitation des machines est RedHat Linux 7.3.



Schematic diagram of the real-time analysis system for functional MRI time series. The system is composed of an MR scanner subsystem, a data storage device, and a computational server. For offline operation, the MR scanner subsystem, through the ADWS SunOS4.1 Unix workstation, sends data for storage to the data storage device, which the computational server can access for future analysis. For real-time operation, the MR scanner subsystem sends the data directly to the computational server for immediate processing. A vendor-supplied GUI running in an SGI Octane computer is used to specify the scanning parameters and other MR control parameters.

Figure A.1 - architecture de la plateforme temps réel

- Le modèle général Linéaire (GLM)

Ceci est employé en vue de l'analyse statistique. Ici, un algorithme spécifique a été développé dans l'optique d'une efficience accrue. Nous n'étudierons pas les détails liés à l'élaboration de cet algorithme.

- Acquisition des données

Deux jeux de données ont été acquis (A et B). La différence notable entre ceux-ci est la taille des matrices décrivant les volumes (64x64 en A et 128x128 en B).

- Analyse des données

Rappelons qu'un modèle général linéaire a été utilisé lors de l'analyse statistique.

Le processus en temps réel commence dès l'acquisition des données, et se termine lors de l'affichage des résultats. Les étapes intermédiaires sont le réalignement, le lissage, l'analyse statistique et l'archivage des données et résultats sur les machines de stockage.

- Résultats obtenus

Un premier tableau (figure A.2) nous montre le temps moyen de calcul (en secondes) par image, réalisé sur deux jeux de données de 130 images (sur un cluster de 16 processeurs). La différence entre ceux jeux de données est la taille de la matrice décrivant une image.

Computational Steps	Data Set A (64 × 64 × 30)		Data Set B (128 × 128 × 20)	
	Master Node	Compute Nodes	Master Node	Compute Nodes
Read data	0.023	—	0.039	—
Realignment	—	0.549	—	1.445
Spatial smoothing	—	0.140	—	0.394
Compute statistics	—	0.023	—	0.063
Cluster communication	0.080	0.322	0.223	0.659
Update SPM	0.207	—	0.231	—
Total time	0.310	1.034	0.493	2.561

Values are expressed as seconds per image volume. The master node reads data from the local hard disk, broadcasts the data to all compute nodes, receives the statistics results, and updates the displayed parametric maps. The compute nodes are responsible for realignment, spatial smoothing, and statistical analysis. Cluster communication row is the sum of tasks 2 and 4 (see text). In this case, 15 compute nodes and 1 master node were used in the analysis.

Figure A.2 - résultats des calculs temps réel

Nous voyons que le réalignement occupe plus de la moitié du temps. Le temps requis pour la communication entre les machines du cluster (envoi des images aux nœuds esclaves par le nœud maître, …) est lui aussi significatif (plus ou moins le quart du temps). Les autres opérations demandent un temps moins important. Remarquons que le temps nécessaire à l'ensemble des opérations est en-dessous du TR pour les 2 jeux de données, ce qui assure une bonne analyse en temps réel.

Une question qui nous vient naturellement est de connaître la variation du temps en fonction du nombre de nœuds (processeur) sur le cluster de calcul. Pour cela, les mêmes calculs que précédemment ont été lancés de manière offline sur un nombre variable de processeurs (figure A.3).

Data Set	No. of Compute Nodes	Realignment	Spatial Smoothing	Compute Statistics	Cluster Communication
A	15	0.549	0.140	0.023	0.322
	10	0.660	0.140	0.035	0.314
	5	1.089	0.162	0.069	0.302
	1	3.695	0.278	0.317	0.032
B	15	1.445	0.394	0.063	0.659
	10	1.782	0.380	0.093	0.504
	5	2.876	0.439	0.182	0.471
	1	10.084	0.742	0.848	0.351

Figure A.3 - résultats des calculs offline

A la lecture du tableau, nous voyons que le temps chez le nœud maître destiné à communiquer avec ses nœuds esclaves varie fortement en fonction du nombre de nœuds. De manière générale, le temps requis pour chaque opération augmente

lorsque le nombre de nœuds diminue (c'est l'inverse pour le temps de communication car la communication est moins forte quand le nombre de nœuds diminue). Aussi, il est évident que faire les calculs avec un seul nœud est rendu impossible (le temps du réalignement est déjà au-dessus du TR), mais il serait possible de le faire si les images étaient déjà prétraitées et qu'il ne restait que l'estimation statistique à faire (ou bien, les prétraitements peuvent être inclus, mais en réduisant le nombre d'images).

Mais l'augmentation du nombre de nœuds n'est bénéfique que jusqu'à un certain point : par exemple, pour le réalignement, des communications inter-nœuds sont nécessaires et sont rendues plus difficiles lorsqu'un grand nombre de nœuds est impliqué.

L'équipe japonaise a donc prouvé qu'un cluster (composé de machines peu onéreuses) est à même de réaligner, lisser et estimer le modèle statistique, sur des images de bonne qualité, en temps réel. Les paramètres importants sont le nombre et la qualité des nœuds, ainsi que la qualité de la bande passante du réseau (les technologies des grilles de calcul montrent que ceci n'est pas trop un problème).

Annexe B

A Platform for Distributed Analysis of Neuroimaging Data on Global Grids

Les recherches menées par une équipe australienne [KMLSBE05] sont sensiblement différentes de celles que nous avons analysées précédemment. Ici, le but est de mettre en place une plateforme destinée à analyser des images fonctionnelles par résonance magnétique sur la grille globale, et ce, toujours en vue de factoriser le temps nécessaire au traitement.

Les auteurs rappellent dans un premier temps que l'analyse des images requiert une puissance de calcul énorme. L'analyse pour un seul sujet peut prendre une journée, voire plus quand on sait que les chercheurs en neuroscience réalisent des études sur plusieurs sujets (même si certaines tâches similaires entre sujets peuvent être réalisées en même temps). Ce problème de puissance de calcul peut être partiellement résolu en créant des LAN (Local Area Network) concentrant des puissances de calcul de manière locale. Une autre solution est de combiner des puissances de calcul géographiquement dispersées, ce qui nécessite la distribution des données et algorithmes entre les sites impliqués, mais aussi des technologies permettant le management (partage, découverte, échange, agrégation) des ressources distribuées sur le globe (TODO ref).

D'où l'idée d'intégrer des ressources locales avec des ressources distantes, et ce, de manière interopérable avec des infrastructures telles que BIRN (TODO ref). Nous allons donc maintenant détailler la mise en place d'une architecture grille destinée au traitement d'images fonctionnelles par résonance magnétique.

- Architecture de la grille (figure B.1)

  L'objectif de la grille est bien entendu d'accroître la productivité des chercheurs, pour le moment bridé par un manque de capacité de calcul. La grille fournira un espace de calcul et une espace de stockage à un ensemble d'utilisateurs se partageant ceux-ci.

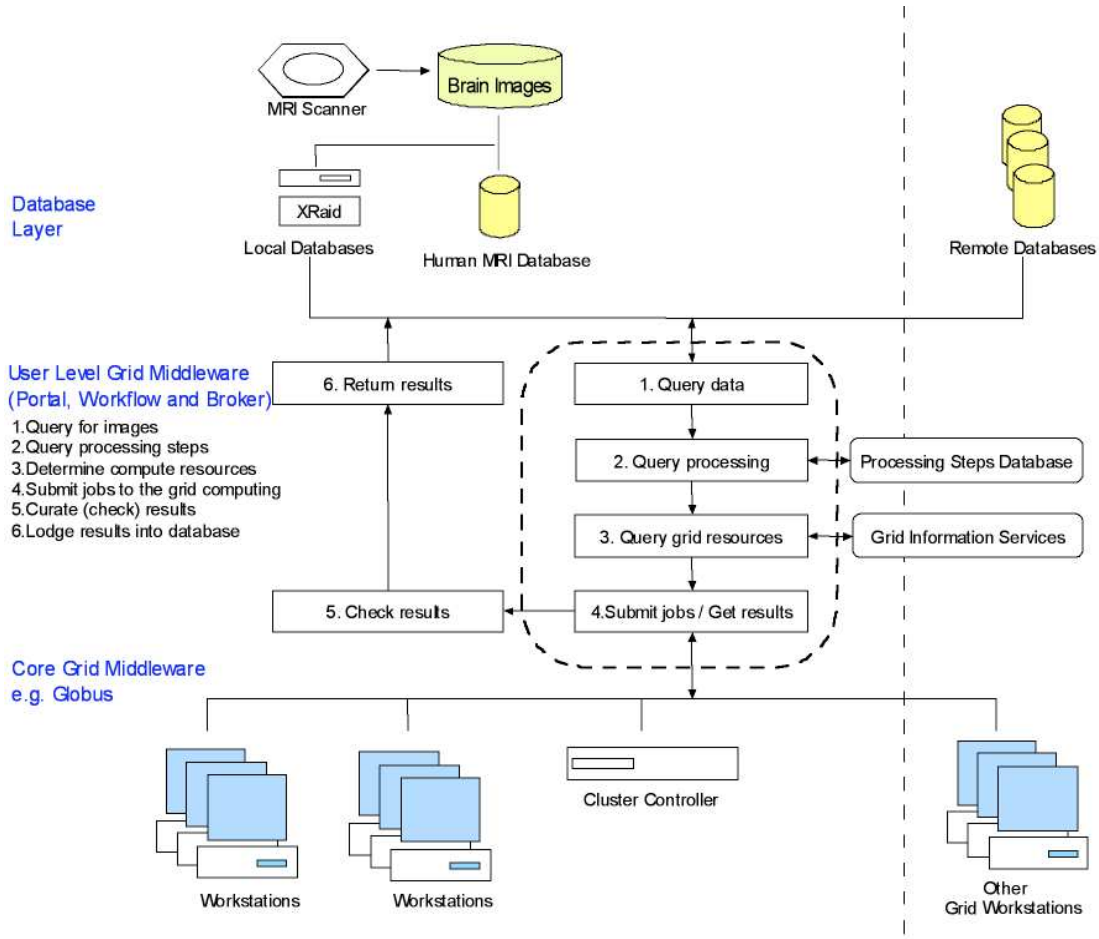


Figure B.1 - architecture de la grille

Nous voyons sur le schéma que le middleware utilisé pour la grille proprement dite est Globus (offrant le management des ressources, données, informations). Un middleware plus proche de l'utilisateur (offrant des services à celui-ci) est rajouté au-dessus de celui fourni par Globus, dont le but est de fournir des services génériques (sous formes d'API) dont l'utilisateur a besoin. La dernière couche est la couche base de données, dont la fonction est bien sûr de stocker les données des utilisateurs.

Les jobs soumis sur la grille sont décrits en utilisant XPML (XML-based Parametric Modeling Language). Les fichiers de description contiennent entre autres les données à manipuler, les traitements à effectuer, la mémoire demandée, …

Les machines utilisées sont décrites dans le tableau suivant (figure B.2) :

Grid Resource, Organisation	Cluster Details	CPU/Memory	OS	Middleware	Queue limit
Manjra Cluster, Uni. of Melbourne	16 nodes, x86	Intel P4 2.4 GHz / 512 MB	Linux RedHat 7.3	SSH-based Access, Open PBS	5
Mac Cluster; HFI	12 nodes, PowerPC	Power G5 2 x 1.8GHz/ 2 GB	Mac OS 10.3.9	SSH, Sun N1 Grid Engine 6 (SGE)	5
APAC, Canberra	154 nodes, x86	Intel P4 2.8 GHz / 1 GB	Linux RedHat 8.0	Globus Toolkit 2.4 – PBS job manager	2
Belle Grid Server, Uni. of Melbourne	Gridbus broker client node	Intel Xeon 2.8 GHz * 4 / 2 GB	Linux RedHat 8.0	Gridbus Broker 2.2	N/A

Figure B.2 - machines utilisées sur la grille

Le dernier cluster du tableau est celui d'où les expériences ont été lancées. La dernière colonne (queue limit) décrit le nombre maximum de jobs qui peut être lancé en parallèle sur une ressource particulière (dépendant de la politique locale). Si cette information n'est pas fournie, des jobs tests seront lancés pour la calculer.

- Méthode d'analyse des données (figure B.3)

Dans le but des tests, des images de 44 participants ont été acquises et converties dans le format « analyse » (format couramment utilisé). L'analyse en elle-même comprend 3 étapes que nous allons décrire de manière assez simple. La première étape consiste en l'extraction du cerveau, c'est-à-dire repérer et garder les voxels décrivant le cerveau (BET). Ensuite, l'image est enregistrée dans un espace de référence afin de normaliser celle-ci (FLIRT). La dernière étape segmente le cerveau en 3 parties (matière grise et blanche, CSF).

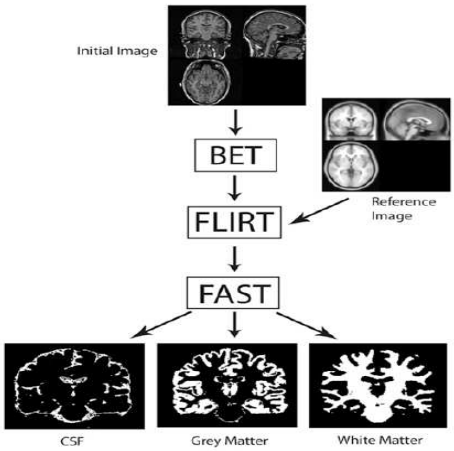


Figure B.3 - méthode d'analyse des données

- Résultats obtenus (performance de la grille)

Nous allons ici résumer brièvement les résultats obtenus sur la grille mise en place comme nous l'avons décrite ci-dessus.

La première statistique (figure B.4) concerne la capacité des différents clusters à réaliser l'ensemble du processus en fonction de la taille des images (Stage-in=début de processus ; Exec=exécution ; Stage-out=fin du processus).

Image size (MB)	Resource	Time taken (sec) for different operations to complete			
		stage-in	Exec	stage-out	Total
8.1 MB image	Manjra Cluster, Melbourne	2.19	232	1.42	235
	Mac Cluster, HFI	1.72	294	1.25	297
	APAC, Canberra	4.13	300	2.25	306
16.3 MB image	Manjra Cluster, Melbourne	3.39	278	1.75	283
	Mac Cluster, HFI	2.25	319	1.59	323
	APAC, Canberra	7.01	318	3.53	329

Figure B.4 - capacité des clusters

Ensuite nous avons le nombre de jobs soumis et terminés sur chacune des trois plateformes (en fonction du temps écoulé), ainsi que l'agrégation de l'ensemble de ces jobs (toujours en fonction du temps, figure B.5).

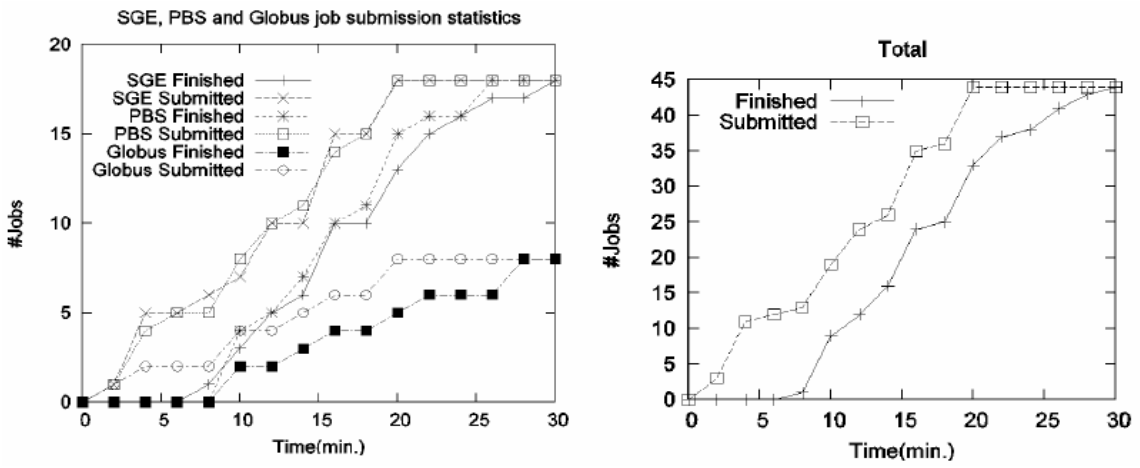


Figure B.5 - jobs par plateforme et total

Nous avons aussi le nombre de jobs terminés après 30 minutes de test (8 sur Globus avec une limite de queue égale à 2, 18 pour PBS et SGE avec des limites de queue valant 5), en figure B.6.

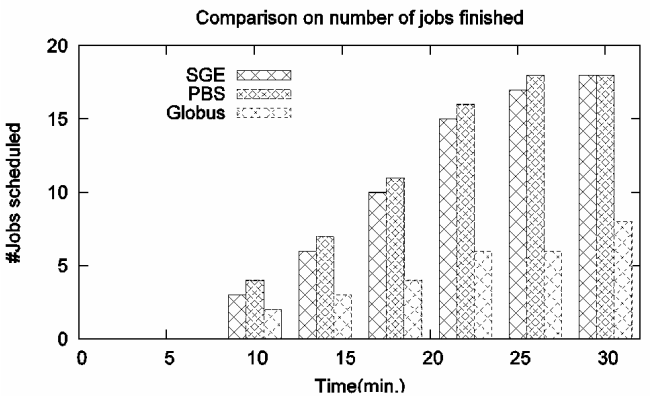


Figure B.6 - jobs terminés par plateforme à différents moments

La dernière statistique concerne le nombre de jobs ordonnancés sur les différentes plateformes, en un temps total de 30 minutes (figure B.7).

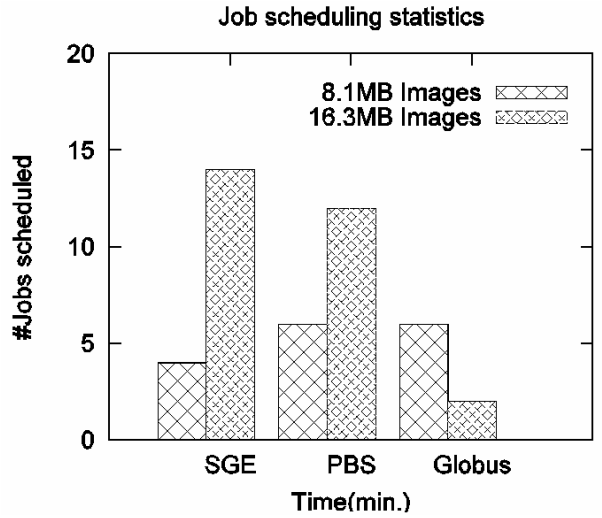


Figure B.7 - jobs ordonnancés par plateforme

Cette étude a donc démontré qu'il était possible de mettre en place une architecture de type grille (permettant le partage des ressources) au service de la neuroscience. Les performances obtenues varient notamment en fonction du middleware utilisé, de la taille maximale des queues de jobs, où des types de machines. L'architecture pourrait être étendue avec des mécanismes d'authentification, services web, …

Annexe C

Listes des S-Commandes

Basic Operations

Sput, Sget, Sls, Smv, Srm, Scp, Sln , Scat, Sappend

Collections

Scd, Spwd, Smkdir, Srmdir, SgetColl, SmodColl

Metadata

SgetD, SmodD, SgetR, SmodR, SgetT, SgetU, SmodE, Smeta, Sufmeta, Sannotate, Squery

Bulk Operations

Sbload, Sbregister, Sbunload, Spullmeta, Spushmeta

Tickets

Sticket, Srmticket, Stls, Stcat

Containers

Slscont, Smkcont, Srmcont, Sreplcont, Ssyncont

SRB Information

Serror, Shelp, Sattrs, Stoken

Session Administration

Sinit, Sexit, Senv, Spasswd, Sauth, Schdefres, Schhost

Data Administration

Sphymove, Sreplicate, Sbkupsrb, Srmtrash, Ssyncd, Sstage, Srsync, Schksum, Schmod

MCAT Administration

Singestgroup, Singesttoken, Singestuser, SmodifyUser, Sregister, Sregisterlocation, Sregisterresource, SdelValue

Advanced Commands

Spcommand, Ssh

Federation

Szone, Szonesync.pl

Annexe D

Exemple de fichiers relatifs à l'exécution d'un job

- Fichier Matlab, décrivant la tâche (Matlab) à effectuer

```
% lhuys_20060514194742.m

% pspmjob JOB file for batch submission - generated by pspmjob 0.10

% Modify java classpath and matlab path

javaaddpath /afs/in2p3.fr/home/throng/isc/local/pspmjob_java/
addpath /afs/in2p3.fr/home/throng/isc/local/lyddia/pspmjob/
addpath /afs/in2p3.fr/home/throng/isc/local/spm2
addpath /afs/in2p3.fr/home/throng/isc/local/pspm-2.0.2-
beta_debug/PSPM_MATLAB_FILES/

job = job_defaults;

% the directory of the data. The path should be absolute.
% pspmjob expects a directory structure as follows:
% required: a subdirectory for each subject with subdirectories
% for each session and an anatomy directory for the anatomy scans.
%   <subject1>/
%     anatomy/
%     <session1>/
%     <session2>/
%     ...
%   <subject2>/
%   ...
% optional subdirectories are: 'onsets','durations',
%     'params','weights','regressors' (see *_load below)

job.directory = '/sps/isc/scratch_pspmjob/lhuys_20060514194742';

% a name or description for the job
% default: empty or filename
job.name = 'lhuys_20060514194742';

% comma-separated directory names of the subjects
% the names may contain format strings (see job.subject_subst)
job.subjects = 'p1,p2';
```

```
% comma-separated directory names of the sessions
% the names may contain format strings (see job.subject_subst)
job.sessions = 's1';

% Realign images
job.realign.enabled = 1;
% Registration quality: Quality versus speed trade-off.
% Highest quality (1) gives most precise results, whereas lower qualities
% gives faster realignment.
% The idea is that some voxels contribute little to the estimation of
% the realignment parameters. This parameter is involved in selecting
% the number of voxels that are used.
% default: 0.75
job.realign.estimate.quality = 0.001000;
% Interpolation method for parameter estimation:
% 0: Nearest Neighbour
% 1: Trilinear
% 2: 2nd Degree B-Spline
% 3: 3rd Degree B-Spline
% 4: 4th Degree B-Spline
% 5: 5th Degree B-Spline
% 6: 6th Degree B-Spline
% 7: 7th Degree B-Spline
% Inf: Fourier Interpolation
% default: 2
job.realign.estimate.interp = 1;
% 0: no images
% 1: images 2..n
% 2: all images (1..n)
% default: 2
job.realign.reslice.which = 0;

% Rormalize images
job.normalise.enabled = 1;
% One or more templates. The image(s) will be fitted (in the least
% squares sense) to the optimum linear combination of these templates.
% String or cell array of strings.
% default: fullfile(spm('Dir'),'templates','T1.mnc')
job.normalise.template = fullfile(spm('Dir'),'templates','EPI.mnc');
% whether to reslice the scans
% default: 1
job.normalise.reslice.which = 0;

% Smooth images
job.smooth.enabled = 1;
% scalar FWHM for isotropic, row vector of 3 values for anisotropic
smoothing.
% for multiple subsequent smoothing specify each FWHM in a row.
% default: []
job.smooth.fwhm = 6;
```

```
% We have to download srb files
job.isSrb = 1;
job.srbdirectory = '/neuro/home/lhuys.isc/dirTest/testTar';

% Use PSPM
job.isParallel = 1;

% Run the job
runjob(job,'console');

% Exit Matlab
exit;
```

- Fichier Script Shell permettant de lancer MATLAB sur le premier fichier, afin d'exécuter la tâche MATLAB

```
% lhuys_20060514194742_script.sh

#!/usr/local/bin/tcsh
cd /sps/isc/scratch_pspmjob/lhuys_20060514194742/
matlab -nodisplay -r lhuys_20060514194742
```

- Fichier Script Shell réalisant la soumission proprement dite à la ferme de calcul

```
% lhuys_20060514194742_batch.sh

#!/usr/local/bin/tcsh

%submit on pistoo farm
setenv BQSCLUSTER pistoo

qsub -q T -N lhuys_20060514194742 -l t=6000 -l M=1024MB -l platform=LINUX
-l proc=5 -l ptype=MPICH
/sps/isc/scratch_pspmjob/lhuys_20060514194742/lhuys_20060514194742_script.sh
```

- Fichier output fourni par BQS

```
% lhuys_20060514194742.o0015676

************************************************************
*             BQS Batch Queueing System Rel. 6.2.2         *
*          Centre de Calcul de l'IN2P3, Villeurbanne       *
************************************************************
* User:                    lhuys                           *
* Group:                   isc                             *
* Jobname:                 lhuys_20060514194742            *
* JobID:                   109.ccpl0381                    *
* Class:                   T                               *
* Worker:                  ccpl0381.in2p3.fr               *
* Operating system:        Linux 2.4.21-27.0.2.ELSDRsmp    *
************************************************************
* Queued on:               05/14/2006-19:47:45             *
* Eligible since:          05/14/2006-19:47:45             *
* Starting on:             05/14/2006-19:48:43             *
************************************************************

Prologue has set TMPBATCH to /scratch/lhuys109.ccpl0381
Warning:
  MATLAB is starting without a display, using internal event queue.
  You will not be able to display graphics on the screen.

                            < M A T L A B >
                Copyright 1984-2006 The MathWorks, Inc.
                       Version 7.2.0.294 (R2006a)
                            January 27, 2006

  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.

mkdir: cannot create directory
`/sps/isc/scratch_pspmjob/lhuys_20060514194742': File exists
downloading SRB files
You are using the prod version of the SRB utilities.
rmdir: `/sps/isc/scratch_pspmjob/lhuys_20060514194742/p1/': Directory not
empty
rmdir: `/sps/isc/scratch_pspmjob/lhuys_20060514194742/p2/': Directory not
empty
end downloading SRB files
job started: 14-May-2006 19:50:48
Realignment:
 subject 1: p1
begin : pspm phase for reslice
end : pspm phase for reslice
```

```
subject 2: p2
begin : pspm phase for reslice
end : pspm phase for reslice
Normalisation:
 subject 1: p1
Smoothing by 0 & 8mm..
Coarse Affine Registration..
Fine Affine Registration..
3D CT Norm...
 iteration  1:  FWHM =  36.48 Var = 240.424
 iteration  2:  FWHM =  15.72 Var = 2.55326
 iteration  3:  FWHM =  13.52 Var = 0.933259
 iteration  4:  FWHM =  12.45 Var = 0.569305
 iteration  5:  FWHM =  11.89 Var = 0.441277
 iteration  6:  FWHM =  11.64 Var = 0.391112
 iteration  7:  FWHM =  11.51 Var = 0.368105
 iteration  8:  FWHM =  11.45 Var = 0.357497
 iteration  9:  FWHM =  11.44 Var = 0.353553
 iteration 10:  FWHM =  11.43 Var = 0.351672
 iteration 11:  FWHM =  11.42 Var = 0.350626
 iteration 12:  FWHM =  11.42 Var = 0.350267
 iteration 13:  FWHM =  11.42 Var = 0.350192
 iteration 14:  FWHM =  11.41 Var = 0.349061
 iteration 15:  FWHM =  11.41 Var = 0.349061
 iteration 16:  FWHM =  11.41 Var = 0.348393
Saving Parameters..
begin : pspm phase for normalize
end : pspm phase for normalize
 subject 2: p2
Smoothing by 0 & 8mm..
Coarse Affine Registration..
Fine Affine Registration..
3D CT Norm...
 iteration  1:  FWHM =  36.48 Var = 240.424
 iteration  2:  FWHM =  15.72 Var = 2.55326
 iteration  3:  FWHM =  13.52 Var = 0.933259
 iteration  4:  FWHM =  12.45 Var = 0.569305
 iteration  5:  FWHM =  11.89 Var = 0.441277
 iteration  6:  FWHM =  11.64 Var = 0.391112
 iteration  7:  FWHM =  11.51 Var = 0.368105
 iteration  8:  FWHM =  11.45 Var = 0.357497
 iteration  9:  FWHM =  11.44 Var = 0.353553
 iteration 10:  FWHM =  11.43 Var = 0.351672
 iteration 11:  FWHM =  11.42 Var = 0.350626
 iteration 12:  FWHM =  11.42 Var = 0.350267
 iteration 13:  FWHM =  11.42 Var = 0.350192
 iteration 14:  FWHM =  11.41 Var = 0.349061
 iteration 15:  FWHM =  11.41 Var = 0.349061
 iteration 16:  FWHM =  11.41 Var = 0.348393
 Saving Parameters..
```

```
begin : pspm phase for normalize
end : pspm phase for normalize
Smoothing:
 subject 1: p1
begin : pspm phase for smooth
end : pspm phase for smooth
 subject 2: p2
begin : pspm phase for smooth
end : pspm phase for smooth
uploading SRB files
You are using the prod version of the SRB utilities.
end uploading SRB files
job finished successfully
job finished: 14-May-2006 20:23:07
elapsed time: 00:32:19


*****************************************************************
* User:                    lhuys                                *
* Group:                   isc                                  *
* Jobname:                 lhuys_20060514194742                 *
* JobID:                   109.ccpl0381                         *
* Class:                   T                                    *
* Worker:                  2.ccpl0381.in2p3.fr                  *
* Operating system:        Linux 2.4.21-27.0.2.ELSDRsmp         *
*****************************************************************
* Queued on:               05/14/2006-19:47:45                  *
* Eligible since:          05/14/2006-19:47:45                  *
* Started on:              05/14/2006-19:48:42                  *
* Ended on:                05/14/2006-20:23:12                  *
* with status:             ENDED                                *
*****************************************************************
* Parallel type:           MPICH                                *
* Number of processors:    5                                    *
* MPPM:                    N                                    *
*****************************************************************
* Elapsed time:            0:34:30                              *
* CPU total real:          0:14:56                              *
*     total normalized:    3:21:36 (time limit: 1:40:00)        *
*     system real:         0:00:10                              *
* SPOOL (stdout&err):      15 KB                                *
* SCRATCH:                 0 MB                                 *
* VIRTUAL STORAGE:         2479 MB                              *
* XTAGED:                  0 MB                                 *
* CPU Rate Raw (CPU/elaps):08 %                                 *
*          Corrected:      15 %                                 *
* IO Rate (xtage/CPU):     0 KB/second                          *
*****************************************************************
```

Bibliographie

[Ba98] Chaitanya Baru. Archiving Metadata. SDSC. 1998.

[Ba982] Chaitanya Baru. Managing very large scientific data collections. SDSC. 1998.

[BABAR] BaBar. http://www.slac.stanford.edu/BFROOT/

[BIRN] Biomedical Informatics Research Network. http://nbirn.net/.

[BMN03] Epifanio Bagarinao, Kayako Mastuo et Toshiharu Nakai. Real-Time Functional MRI Using a PC Cluster. Osaka – Japon. 2003.

[BMRW] Chaitanya Baru, Reagan Moore, Arcot Rajsekar et Michael Wan. The SDSC Storage Resource Broker. SDSC. 2005.

[Bon] Gabriel Bondaz. Transfert et Stockage d'images médicales au format DICOM vers l'outil SRB (Projet BIRNy). CCIN2P3 (Lyon). 2005.

[BQS] BQS – Batch Queueing System. IN2P3-Lyon. http://cc.in2p3.fr/rubrique351.html.

[BR98] Chaitanya Baru et Arcot Rajsekar. A Hierarchical Access Control Scheme for Digital Libraries, Conférence of Digital Libraries. Pittsburgh. 1998.

[CFLMMRW96] Chaitanya Baru, Richard Frost, Joe Lopez, Richard Marciano, Reagan Moore, Arcot Rajasekar et Mike Wan. Metadata design for a massive data analysis system. San SDSC.1996.

[CMS] CMS – Compact Muon Spectrometer. http://cmsinfo.cern.ch/Welcome.html/.

[Cun05] Columbia University 2005. About functional MRI (General). http://www.fmri.org/fmri.htm.

[DCM] Dublin Core Metadata. http://purl.org/metadata/dublin_core. 1997.

[DMV99] Vincent Denolin, Thierry Metens et Philippe Van Ham. Imagerie Fonctionnelle par RMN. Systèmes Logiques et Numériques. 1999.

[DIC] DICOM, Digital Imaging and COmmunications in Medecine. http://medical.nema.org/.

[Fri03] Karl J Friston. Experimental design and Statistical Parametric Mapping. The Wellcome Dept. of Cognitive Neurology. University College London. 2003.

[GPL] GPL – General Public License. http://www.gnu.org/copyleft/gpl.html.

[GSI] GSI – Globus Security Infrastructure. http://www.globus.org/Security/.

[HPSS] HPSS – High Performance Storage System. http://www.hpss-collaboration.org/hpss/index.jsp.

[JARGON] JARGON - Java Api for Real Grids On Networks. http://www.sdsc.edu/srb/jargon/.

[KMLSBE05] Scott Kolbe, Tianchi Ma, Wei Liu, Wee Siong Soh, Rajkumar Buyya et Gary Egan. A Platform for Distributed Analysis of Neuroimaging Data on Global Grids. The University of Melbourne. 2005.

[MATLAB] MATLAB. http://www.mathworks.com/.

[Mo05] Reagan Moore. Building Preservation Environments with Data Grid Technology. SDSC. 2005.

[Mo052] Reagan Moore. Persistent Collections. SDSC. 2005.

[MPI] Message Passing Interface. http://www-unix.mcs.anl.gov/mpi/.

[MPICH] Message Passing Interface, une implémentation,. http://www-unix.mcs.anl.gov/mpi/mpich/index.htm

[MRW04] Reagan Moore, Arcot Rajsekar et Michael Wan. Data Grids, Digital Libraries and Persistent Archives: An Integrated Approach to Publishing, Sharing and Archiving Data. 2004.

[NAR] NARA Persistent Archive Prototype. http://www.sdsc.edu/NARA/Publications.html.

[NAS] NASA Information Power Grid (IPG) is a high-performance computing and data grid. http://www.ipg.nasa.gov/.

[NPACI] NPACI Data Intensive Computing Environment thrust area. http://www.npaci.edu/DICE/.

[PDB] PDB - Protein Data Bank. http://www.rcsb.org/pdb/.

[PERL] PERL - Practical Extraction and Report Language. http://www.perl.com.

[Ra02] Arcot Rajasekar. Replicated Data Management using SRB. GGF-4 Toronto. 2002.

[RWMJK02] Arcot Rajasekar, Michael Wan, Reagan Moore, Arun Jagatheesan et George Kremenek. Real Experiences with Data Grids - Case studies in using the SRB. SDSC. 2002.

[RWMS04] Arcot Rajsekar, Michael Wan, Reagan W. Moore et Wayne Schroeder. Data Grid Federation. San Diego Supercomputer Center. 2004.

[SDSC] San Diego Supercomputer Center. http://www.sdsc.edu/

[SIO] SIOExplorer. http://nsdl.sdsc.edu/.

[SPM] SPM - Statistical Parametric Mapping. http://www.fil.ion.ucl.ac.uk/spm.

[SPMJOB] SPMJOB. http://spmjob.ffii.org/.

[SPMP] SPM parallele. http://spm.parallel.free.fr/

[SRB] SRB – Storage Resource Broker. http://www.sdsc.edu/srb.

[SRBAPI] SRB client API. http://www.sdsc.edu/srb/index.php/Client_API.

[Thi03] Bertrand Thirion. Analyse de données d'IRM fonctionnelle: statistiques, information et dynamique. Thèse, TSI, Télécom Paris. 2003.

[WRMA03] Michael Wan, Arcot Rajasekar, Reagan Moore et Phil Andrews. A Simple Mass Storage System for the SRB Data Grid. SDSC. 2003.

[WS05] Michael Wan, Wayne Schroeder. SDSC SRB Core Technology. SDSC. 2005.