Conception et optimisation d’un système d’exécution d'une application d'imagerie médicale sur grille de calcul

Stage de Master de Recherche GEGP
Parcours Systèmes & Images
Année 2007-2008

Sorina CAMARASU
Tuteur : David SARRUT
Rapporteurs :
Hélène RATINEY, Rémy PROST

Remerciements

Je voudrais remercier particulièrement mon tuteur de stage, David Sarrut, qui m'a accompagnée tout au long de mon stage. Il a su me transmettre son dynamisme et sa motivation, ainsi que me laisser la liberté nécessaire à l'accomplissement de mes travaux, tout en gardant un œil critique et avisé. J'espère que les résultats obtenus dans le cadre de ce stage constitueront une base solide pour de futures collaborations.

Tous mes remerciements à Hugues Benoît-Cattin, grâce auquel j'ai fait la connaissance de la grille EGEE et qui m'a conseillé de m'orienter vers ce sujet de stage. Il est un professeur et un collègue remarquable.

Un grand merci pour mes collègues de bureau : Delphine Charpigny, Thomas Grenier et Rémi Flamary qui, malgré leur emploi du temps très chargé, ont trouvé le temps de lire mon rapport et de me faire des retours critiques et très constructifs.

Merci à Laurent Guigues avec lequel, David et moi avons discuté de l'architecture proposée et implémentée au cours du stage. Merci aussi à Thibault Frisson pour son aide dans l'interprétation des résultats d'incertitude et dans la relecture du rapport. Je suis aussi très reconnaissante à Fabrice Bellet qui m'a aidée maintes fois à résoudre les différents problèmes techniques.

Tous mes remerciements aussi à Jakub Moscicki, le 'père' de Ganga et DIANE qui a été très réactif et qui m'a beaucoup aidée à distance dans l'installation et l'utilisation de ces outils très performants.

Et le dernier mais pas le moindre, je voudrais remercier Olivier Basset, le responsable de ce Master, tout d'abord pour avoir mis en place cette formation que j'ai beaucoup appréciée et ensuite pour la patience et la gentillesse avec laquelle il m'a aidée dans la délimitation du sujet de stage.

Merci à tous !

Sommaire

Table des Figures

Introduction

Les grilles de calcul sont des architectures très appréciées pour leur impressionnante puissance de calcul et leurs capacités de stockage. Au-delà de ces deux atouts, les technologies de grille permettent aux utilisateurs (scientifiques, ingénieurs, médecins ...) repartis partout dans le monde mais fédérés en organisations virtuelles (Virtual Organisation – VOs) de partager facilement leurs données et/ou algorithmes. Ces quelques caractéristiques rendent les grilles particulièrement intéressantes pour la communauté médicale qui travaille avec d'importants volumes de données et des algorithmes de traitement de ces données très gourmands en termes de puissance de calcul [1].

Malgré cette correspondance évidente entre les ressources des grilles et les besoins des communautés médicales, leur mise en commun n'est pas toujours évidente. L'existence des difficultés s'explique souvent par le fait que les grilles de calcul sont des architectures hétérogènes, en plein développement et qui ne prennent pas particulièrement en compte les contraintes médicales. De plus, les applications médicales existantes n'ont pas non plus été initialement développées pour être exécutées sur des architectures spécifiques telles que les grilles.

Le sujet de ce stage de master porte sur la conception d'un système d'exécution générique optimisé pour des applications médicales sur une grille de calcul. Au cours de ce stage, nous avons décidé de mettre l'application ThIS (**Th**erapeutic **I**rridiation **S**imulator) [2] sur la grille de calcul déployée dans le cadre du projet européen EGEE (**E**nabling **G**rids for **E**-Scienc**E**) [3]. Cependant le principe et la mise en place restent généraux et applicables à d'autres applications médicales et/ou grilles de calcul.

Ce rapport commence par décrire le contexte du stage et introduire les éléments nécessaires à une bonne compréhension de la problématique et du travail réalisé. L'application ThIS, la grille EGEE, les verrous rencontrés, ainsi que les grandes lignes des solutions proposées seront présentés rapidement. Le deuxième chapitre détaillera le travail réalisé. Il s'agit dans un premier temps de faire exécuter ThIS avec succès sur la grille. Dans un deuxième temps, une analyse plus fine permettra l'amélioration du système et une exécution optimisée fondée sur le principe d'une architecture client-serveur. Enfin, une nouvelle optimisation sera mise en place par rapport au système de simulation Monte Carlo et à l'utilisation des ressources de la grille à travers un choix judicieux du critère d'arrêt de la simulation. Le troisième chapitre sera dédié au retour d'expérience et aux discussions. Ce sera l'occasion de présenter les difficultés rencontrées, d'analyser les résultats obtenus et de proposer des améliorations.

1 Contexte et objectifs

La présentation du contexte s'appuie sur les exemples concrets de l'application ThIS et la grille EGEE. Cependant, il s'agit d'une problématique commune à d'autres applications d'imagerie médicale et par conséquent d'une démarche générale, applicable à d'autre cas similaires.

1.1 Description de l'application

L'application ThIS (Therapeutic Irridiation Simulator) a été développée par David Sarrut et Laurent Guigues au sein du laboratoire CREATIS-LRMN. ThIS simule l'irradiation des tissus vivants avec des particules (protons, photons, ions de carbone…) dans le cadre du traitement du cancer (voir Figure 1). Il calcule ensuite le dépôt de dose dans les tissus du patient en fonction des paramètres d'irradiation. ThIS s'appuie sur le très connu Géant4 [4] utilisant la méthode de simulation de Monte Carlo (MC).

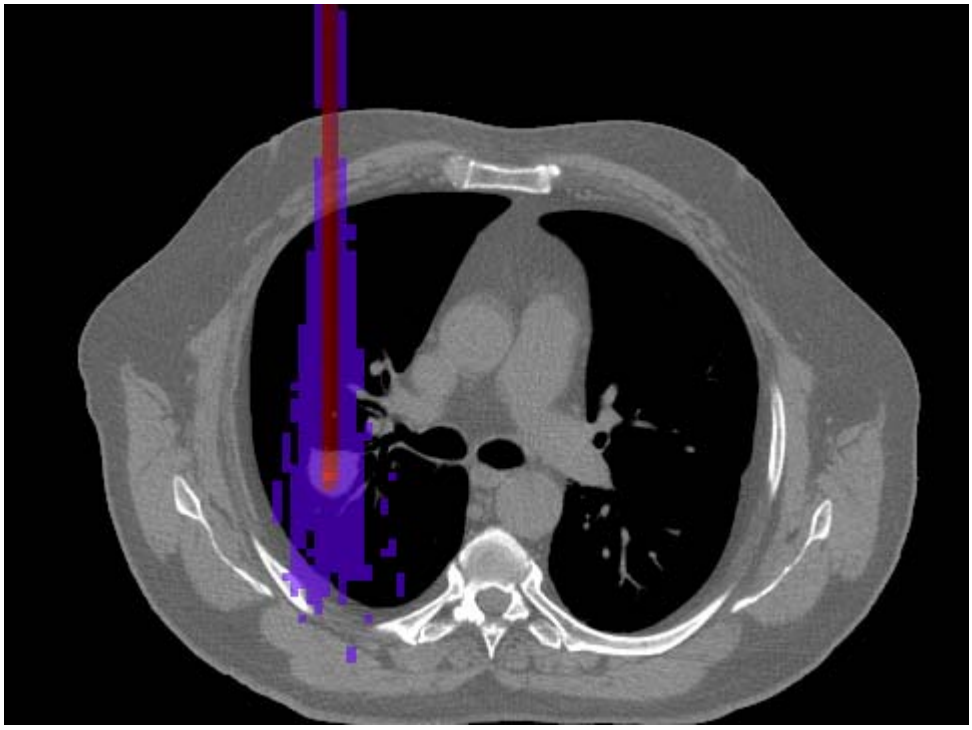

Figure 1 – Dépôt de dose obtenu avec ThIS

La méthode MC est à la base une méthode statistique pour les calculs de sommes et d'intégrales, mais également pour la résolution d'équations de divers types, les problèmes d'optimisation, de comptage... Son utilisation dans le cadre de la simulation de transport de particules à travers les tissus et dépôt de dose, est basée sur la connaissance des distributions de probabilité gouvernant l'interaction de particules (telles que les électrons ou les photons) avec la matière. L'approche MC permet ainsi de simuler des trajectoires aléatoires des particules de manière réaliste.

Pour des résultats fiables, il faut simuler un grand nombre de particules, ce qui mène à faire des calculs très importants, donc à des durées de simulation très longues. Cependant, ce temps

peut être réduit si le nombre total de particules à simuler est divisé en plusieurs sous-simulations avec moins de particules chacune. La condition à respecter pour avoir le droit de subdiviser une simulation de façon à ce que l'addition des résultats soit valide est le fait d'avoir des sous-simulations indépendantes statistiquement. L'indépendance est garantie en attribuant à chaque sous simulation une suite de nombres non-corrélés ('seed numbers') qui seront utilisés dans la simulation. Cela permet aux sous-simulations d'être exécutées en parallèle. L'exécution sera ainsi plus rapide en simulant moins de particules d'une manière concurrente sur plusieurs processeurs. Par exemple, si on veut simuler 50 M de particules, la simulation est divisée en 50 sous jobs (tâches), chacun avec 1 M de particules à simuler.

Il faut préciser que, grâce au fait que les sous-jobs soient indépendants, dans la pratique rien n'oblige à les considérer comme des sous-jobs d'une même simulation : on pourrait très bien les associer différemment. Le nom de 'sous-jobs' (ou bien sous-tâches) est utilisé simplement pour renforcer l'idée de parallélisation et de division d'un grand nombre de particules en tâches plus petites.

Par conséquent, le fait d'utiliser une architecture distribuée telle qu'une grille de calcul permettrait de réaliser ce type de parallélisation et d'optimiser soit le temps de calcul pour une qualité de simulation donnée, soit d'augmenter la qualité de simulation pour un temps donné.

1.2 Description d'une grille de calcul

Une grille de calcul est une infrastructure informatique destinée à mettre à la disposition des utilisateurs des ressources pour réaliser du calcul distribué et pour stocker des données. Plus concrètement, elle est constituée d'un grand nombre de machines hétérogènes et souvent délocalisées reliées par un réseau internet et rendue homogène aux utilisateurs grâce au middleware (en français, intergiciel) de grille.

Le middleware est un logiciel servant d'intermédiaire de communication entre plusieurs applications distribuées sur un réseau informatique. Du point de vue de l'utilisateur, il est comparable à un système d'exploitation plus évolué donnant accès à des services de haut niveau sur les machines de la grille. Les middlewares les plus connus dans le monde des grilles de calcul sont par exemple Condor [5], Globus [6] ou gLite (utilisé dans EGEE).

Les grilles de calcul sont aujourd'hui en plein développement. Il existe déjà de nombreuses grilles au niveau national (national Grid Initiatives ou NGI) telles que les grilles en Grande Bretagne (UK National Grid), en Allemagne (D-Grid), en Grèce (HellasGrid), en Italie (INFN Italian National Grid), au Pays-Bas (DutchGrid), ou bien au Japon (NAREGI). En France il existe depuis décembre 2007 l'Institut National des Grilles (IdG) qui a pour but de recenser des besoins concernant les grilles de calcul en France afin d'aller éventuellement vers une NGI française.

Il existe aussi des grilles de calcul de plus large échelle qui rassemblent plusieurs pays. C'est le cas par exemple de Open Science Grid (Etats-Unis et Asie), NorduGrid (pays nordiques) ou EGEE (grille européenne au départ qui s'est étendue en Asie et récemment en Afrique).

Comme précisé précédemment, dans le cadre de ce stage nous avons utilisé l'infrastructure EGEE. Dans ce qui suit nous décrirons rapidement les principes de base du fonctionnement d'une grille de calcul avec l'exemple d'EGEE. EGEE ("Enabling Grids for E-sciencE") est un

projet européen qui a mis en place une architecture de grille déployée actuellement dans le monde entier. Il s'agit d'une grille de production, c'est-à-dire une grille utilisée pour produire des résultats scientifiques 7 jours par semaine, 24h/24. EGEE détient aujourd'hui plus de 41000 processeurs et environ 5 PB (5 Million de Gigabytes) d'espace de stockage partagé par plus de 5000 utilisateurs soumettant plus de 100000 jobs concurrents tous les jours.

La Figure 2 présente la structure générale de l'infrastructure EGEE. L'interface utilisateur (User Interface – UI) est le point d'accès initial qui permet aux utilisateurs de s'authentifier (à travers des certificats et des proxys) et accéder aux ressources de la grille. A partir de l'UI, l'utilisateur peut soumettre ou annuler des tâches (qu'on appellera 'jobs' par la suite), questionner leur statut et récupérer les résultats. Ces tâches sont gérées par le Resource Broker (RB), qui prend en compte les demandes des utilisateurs et les distribue aux centres de calcul disponibles. La porte d'entrée vers chacun des centres de calcul est un 'Computing Element' (CE), qui est aussi chargé de distribuer le travail vers les nœuds de calcul du centre, appelés 'Worker Nodes' (WN) [7].

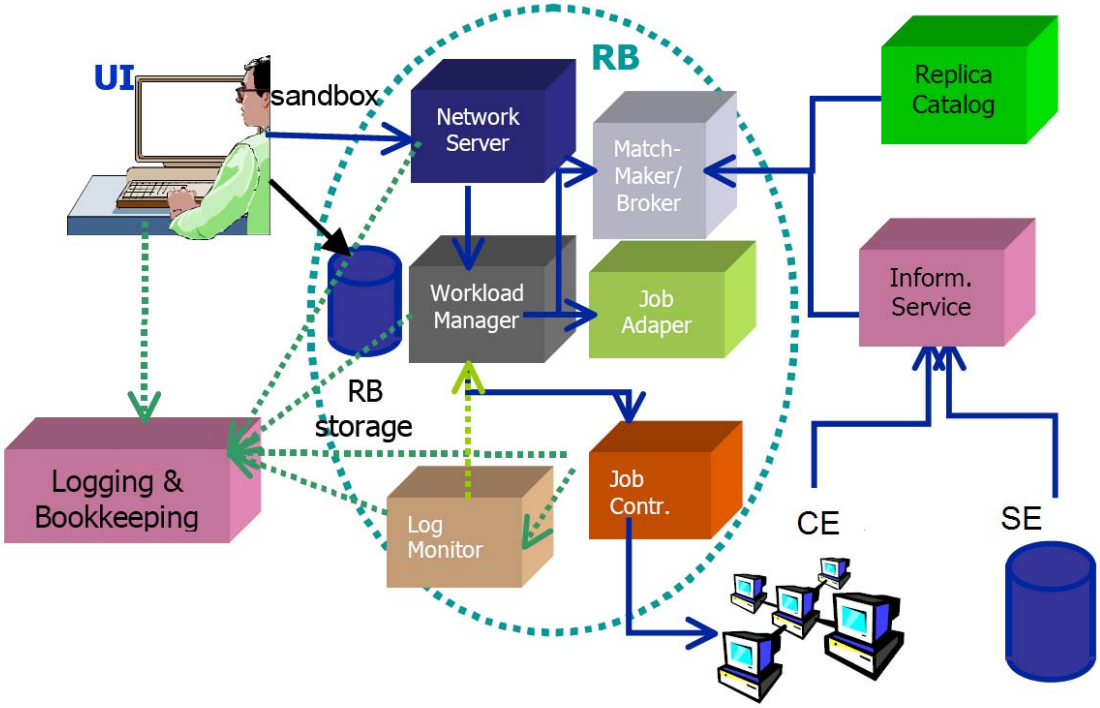

Figure 2 - Vue d'ensemble de l'infrastructure de la grille EGEE (source: [7])

Les fichiers sont stockés sur des éléments de stockage (Storage Elements – SEs) et sont enregistrés dans des catalogues de fichiers (File Catalogs). Ces catalogues permettent la localisation de fichiers (ou leur copies) qui sont distribués sur les machines de la grille. Les services de management de données (Data Management) sont responsables de la localisation, réplication et l'accès transparent aux données. Par conséquent, les utilisateurs n'ont pas besoin de savoir où se trouvent les données physiquement, mais seulement de connaitre le nom logique associé au fichier qu'ils recherchent.

Comme précisé précédemment, l'utilisateur peut soumettre ou annuler des jobs, ainsi que questionner leur statut. La vie d'un job passe donc par plusieurs phases : il est d'abord

‘Ready’ (prêt), c'est-à-dire envoyé au RB. Il sera ensuite ‘Scheduled’ (programmé), ce qui signifie qu’il a été attribué à un CE et mis dans une file d’attente. L’attente peut être plus ou moins longue [8]. Elle varie en moyenne de 3 à 10 minutes, mais elle peut aller jusqu’à 24h, cas où le job sera annulé. Une fois que le job commence à s’exécuter il devient ‘Running’. S’il s’exécute avec succès son statut sera ‘Done’, sinon il sera ‘Aborted’. Si l’utilisateur annule lui-même le job, alors il deviendra ‘Canceled’ (annulé). Ces étapes de la vie d’un job sont très importantes à connaître afin de comprendre comment on peut optimiser l’exécution d’une application sur la grille. On remarque que c’est l’état ‘Sheduled’ qui peut introduire une latence très importante et qu’une bonne piste pourrait être de profiter des jobs lancés en premier et d’éviter d’attendre les derniers.

Comme on peut le voir dans la Figure 3, leur architecture permet la parallélisation des applications de façon à obtenir un résultat final plus rapidement que si l’application était lancée sur une seule machine.

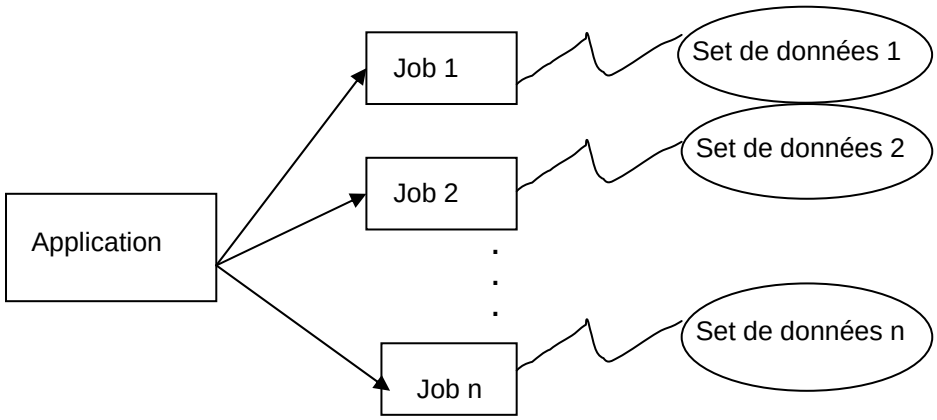

Figure 3 – Parallélisation d’application

Dans certains cas la parallélisation peut se faire au niveau des jeux de données (partie droite de la figure 3) : par exemple pour certains traitements d’image, on peut partitionner l’image et faire exécuter l’application de traitement en parallèle pour chaque partie de l’image. Dans d’autre cas il s’agit d’une parallélisation au niveau de l’application (partie gauche de la figure 3), comme par exemple pour les applications développées avec MPI ou les applications telles que ThIS, dont la parallélisation sera détaillée par la suite. Enfin, il y a des programmes qui pourront profiter des deux types de parallélisation, à la fois au niveau applicatif et au niveau des données. [9] offre plus de détails sur le partitionnement des bases de données et des applications dans le cadre de la parallélisation, ainsi qu’une vue d’ensemble sur plusieurs projets utilisant les grilles dans le domaine médical.

1.3 Verrous / Défis

Les grilles de calcul offrent des ressources de calcul et stockage importantes. Cependant, il s'agit d'architectures en pleine évolution, qui soulèvent de nouveaux problèmes.

1.3.1 Hétérogénéité

Le premier verrou est celui de la difficulté d'exécution de nouvelles applications à cause de l'hétérogénéité de la grille. Comme précisé dans la section 1.2, une grille de calcul est constituée d'un grand nombre de machines hétérogènes, ayant des caractéristiques et configurations différentes. Dans EGEE le système d'exploitation est généralement Scientific Linux (SL), mais les versions peuvent varier (SL3, SL4 ou SL5). Les programmes et bibliothèques (ou librairies logicielles), ainsi que l'architecture des machines (32 ou 64 bits) varient elles aussi d'un nœud de la grille à un autre. De plus, la configuration d'un nœud distant est généralement a priori peu connue par l'utilisateur standard et ne correspond pas forcement à la configuration locale de son PC. Tout cela mène à des incompatibilités et des erreurs d'exécution. Un problème classique est celui d'une application nécessitant une bibliothèque qui n'existe pas (ou tout simplement la version de la bibliothèque n'est pas la bonne) sur la machine sur laquelle le job a été envoyé. Une 'adaptation' au préalable de l'application et/ou le nœud sur lequel elle sera exécutée est donc indispensable dans la plupart des cas. La façon de réaliser cette 'adaptation' sera décrite dans la partie 2.1.

1.3.2 Parallélisme

Les grilles sont particulièrement utiles pour les applications qui peuvent être parallélisées, ce qui est souvent le cas pour les applications médicales. ThIS, par exemple, bénéficie d'une parallélisation 'naturelle' grâce au principe de la simulation MC, comme décrit dans la partie 1.1. L'exemple donné ici est la division d'un job initial de 50 M de particules en 50 sous jobs, chacun avec 1 M de particules à simuler qui vont s'exécuter en parallèle sur 50 machines différentes. L'exemple est facile à comprendre en tant que principe, mais la question n'est pourtant pas aussi simple. Les principales questions qui ressortent sont :

- ➔ Comment choisir la taille des sous-jobs : pourquoi utiliser 50 et non pas 100 ou bien 50 M sous-jobs, c'est-à-dire un sous-job pour chaque particule simulée ?
- ➔ Faudrait-il que tous les sous-jobs aient tous la même charge ? Etant donné qu'il y a des nœuds plus rapides que d'autres, comment distribuer cette charge ?
- ➔ Comment gérer l'ensemble des sous-jobs d'une seule tâche afin d'obtenir le résultat final ?

Le middleware de grille propose des services de base de gestion de jobs, mais il n'y a pas de moyen automatique de gestion pour les questions qu'on vient de présenter. Dans le cadre de ce travail de master, nous avons proposé une solution pour répondre à ces problèmes de manière efficace et générique (valable pour d'autres applications similaires), sera présentée dans la partie 'Travail réalisé', section 2.2.

1.3.3 Complexité de la grille

Enfin, la grille présente une complexité non négligeable, ce qui implique un temps d'apprentissage important et limite le nombre et les catégories d'utilisateurs. Il y a un grand nombre d'outils mis à disposition des utilisateurs, mais ils sont souvent mal documentés et

difficiles à utiliser. De plus, le middleware, ainsi que d'autres outils de la grille, sont toujours en cours d'évolution et le suivi de ces évolutions demande aussi du temps et peut s'avérer difficile. Par conséquent, les chercheurs ou médecins n'ayant pas de connaissance a priori dans ce domaine ont beaucoup de mal à profiter des avantages que les grilles peuvent offrir.

Le système conçu et mis en place dans le cadre de ce stage est automatisé en grande partie et donc plus facile à utiliser que les services standard de la grille. Son apprentissage devrait être rapide et pratique pour les utilisateurs tels que les stagiaires qui souhaitent lancer des simulations importantes mais qui ont un temps très limité à dédier à l'apprentissage du système de lancement des simulations. A terme, on souhaite mettre en place un portail web 'user-friendly' pour tous les utilisateurs n'ayant aucune connaissance de la grille, de façon à pouvoir l'utiliser d'une manière complètement transparente.

1.4 Objectifs du stage

L'objectif principal de ce stage est de trouver des solutions à ces verrous et proposer une méthodologie permettant une exécution optimisée des applications médicales sur des grilles de calcul. Cette optimisation se traduit principalement par un gain de temps important, mais aussi par une facilité d'utilisation. Plus concrètement, il faudra :

- ➔ Répondre au challenge de l'hétérogénéité et arriver à exécuter avec succès l'application ThIS sur les différents nœuds de la grille. En même temps, il faudra proposer une démarche plus générique à suivre dans le cas d'autres applications similaires.
- ➔ Apporter une solution innovante pour optimiser l'exécution de l'application sur la grille. Il s'agit ici de concevoir et mettre en place une architecture permettant une exécution à la fois optimisée et automatisée. Cette architecture devra répondre aux questions posées au point 1.3.3 par rapport au parallélisme et à la granularité des jobs, mais aussi résoudre les problèmes posés par les jobs qui ont échoués. De plus, à travers l'automatisation de la soumission, cette solution devra aussi pallier au problème de la complexité de la grille en étant simple d'utilisation pour les nouveaux utilisateurs.
- ➔ Proposer un critère d'arrêt de la simulation en accord avec les contraintes de la grille et les particularités de l'application. Ce critère d'arrêt devra apporter une nouvelle optimisation, car son but est d'arriver à simuler suffisamment de particules pour un résultat fiable et en même temps ne pas trop dépasser le nombre suffisant afin de ne pas perdre du temps. Il s'agira donc de trouver le bon compromis entre le gain de temps et des résultats réalistes.

Ces trois objectifs, ainsi que leur réalisation seront décrits point par point dans le chapitre suivant : « Travail réalisé ».

2 Travail réalisé

2.1 Portage de l'application sur la grille

Le ‘portage’ d’une application sur une architecture distribuée peut se définir comme l’ensemble des opérations à effectuer pour que l’application s’exécute avec succès sur cette architecture. Le portage inclut généralement la mise en place des mécanismes plus évolués pour l’optimisation et/ou la facilité d’exécution, mais passe absolument par une étape de ‘customisation’ (adaptation) de façon à rendre l’application exécutable sur la grille.

2.1.1 Objectif

Dans notre cas, le premier objectif est l’exécution avec succès de l’application ThIS sur la grille EGEE, ainsi que l’automatisation des démarches nécessaires pour une exécution réussie. La configuration des nœuds distants ne correspond généralement pas à la configuration du PC local de l’utilisateur. Par conséquent, une application qui s’exécute parfaitement sur son PC peut ne pas s’exécuter sur les nœuds de la grille. Il faut alors trouver les incompatibilités et apporter une solution de façon à rendre l’application exécutable sur la grille indépendamment de la configuration des nœuds distants.

2.1.2 Matériel et méthodes utilisés

Il n’existe pas de recette toute prête pour le portage des applications sur une grille[10], mais à travers les différents projets dans le domaine on peut trouver quelques règles et points de départ qui permettent une première approche du problème. Dans la plupart des cas, l’environnement du nœud distant nécessite un minimum de ‘personalisation’, comme par exemple la définition d’un certain nombre de variables ou le téléchargement des fichiers nécessaires à partir des serveurs connus et accessibles. Cependant, en fonction de la plateforme de grille utilisée et des droits que l’utilisateur possède, il peut y avoir un certain nombre de restrictions en ce qui concerne la reconfiguration du nœud distant. Dans ce cas, il faut ‘adapter’ l’application.

ThIS utilise le toolkit (bibliothèque logicielle) Geant4 pour la simulation du passage des particules à travers la matière. Geant4 est utilisé dans différents domaines d’activité, principalement dans la physique nucléaire et des hautes énergies, mais aussi dans la médicine ou les sciences de l’espace. ThIS utilise donc un certain nombre des bibliothèques de Geant4 qui ne sont pas forcement présentes sur les nœuds de la grille. De plus, Geant4 peut créer des bibliothèques partagées auxquelles l’exécutable ThIS sera lié.

Une bibliothèque partagée, nommée shared object (.so) sous UNIX, est un fichier de bibliothèque logicielle utilisé par un programme exécutable, mais n'en faisant pas partie. Ce fichier contient des fonctions qui pourront être appelées pendant l'exécution d'un programme, sans que celles-ci soient incluses dans l’exécutable. Ce type de bibliothèque présente l’avantage d’une réduction de taille de l’exécutable (puisque certaines parties du logiciel se situent en dehors de lui) et d’une mise à jour très simple des fonctions utiles pour toutes les applications qui les utilisent.

Le fait de lier l’exécutable vers des bibliothèques dynamiques implique le fait de pouvoir les retrouver à l’exécution. Or, la réalisation des liens se faisant sur le PC local de l’utilisateur,

rien ne garantit l'existence des mêmes bibliothèques partagées que sur le nœud distant de la grille. D'ailleurs, les tests d'exécution de ThIS sur la grille ont montré que dans la majorité des cas au moins une partie de ces fichiers n'existaient pas sur le nœud distant.

Une solution possible serait de copier avec l'exécutable l'ensemble des fichiers .so dont il a besoin pour s'exécuter proprement. Cependant, vu que l'utilisateur standard a des droits limités et ne peut pas mettre ces bibliothèques au bon endroit, il faudrait aussi rajouter dans la variable d'environnement LD_LIBRARY_PATH (qui précise au système où chercher les bibliothèques nécessaires) le chemin d'accès vers les nouvelles bibliothèques. Cette solution est assez souple du point de vue de la création de l'exécutable, mais peut poser d'autres problèmes si parmi les bibliothèques partagées nécessaires se trouvent des fichiers système qui ne correspond pas au système sur la machine distante. Par conséquent on a décidé de chercher d'autres solutions.

La solution que nous avons finalement choisi pour résoudre ce problème a été la création d'un exécutable statique, c'est-à-dire un exécutable qui n'utilise pas des bibliothèques partagées du type .so. Pour cela on a construit (build) les bibliothèques Geant4 de manière statique et au moment de la compilation (et plus précisément de la réalisation des liens) de ThIS, on a imposé l'utilisation des bibliothèques statiques uniquement (c'est-à-dire seulement des fichiers .a qui sont 'inclus' dans l'exécutable final). Cette solution présente l'inconvénient d'avoir un exécutable plus gros, mais marche très bien dans la majorité des cas. La commande 'ldd' permet de vérifier si un exécutable est statique ou dynamique (dans ce cas il pointe vers des bibliothèques dynamiques que la commande listera). Pour plus de détails techniques, l'annexe 1 présente le document réalisé dans le cadre de ce stage et publié sur le site de l'application ThIS pour la compilation statique de l'application, ainsi que de ses dépendances (bibliothèques CLHEP et Geant4).

Côté grille, d'autres mesures d'adaptabilité ont été nécessaires. Elles ont été réalisées à travers des scripts lancés sur les nœuds avant l'exécution de l'application ThIS. Ces scripts préparent tout l'environnement nécessaire pour l'exécution de ThIS. Ils copient l'archive contenant l'exécutable ainsi que tous les fichiers d'entrée nécessaires en local, la désarchivent, se positionnent dans le bon répertoire, lancent l'exécutable, copient les résultats sur un autre SE (Storage Element) de la grille et nettoient tout en fin d'exécution.

L'ensemble de l'exécution a été par conséquent automatisée dans une grande mesure à travers les scripts écrits :

- ➔ Un script JDL (Job Description Language), caractéristique à la grille chargé de la soumission des jobs
- ➔ Les scripts shell exécutés par le script JDL et chargés de préparer l'environnement et d'exécuter ThIS sur un des nœuds de la grille.

2.2 Parallélisation optimisée et équilibrage de charge

2.2.1 Objectif

Suite aux premiers tests réalisés, le taux des jobs annulés s'est révélé assez important. Même si on arrivait à le faire diminuer, une minorité de jobs annulés aurait un impact très fort au niveau de la simulation globale qui ne pourra pas donner de résultat final fiable. De plus, dans une grille telle qu'EGEE, les nœuds sont très hétérogènes et par conséquent certains sont plus performants et rapides que d'autres. Le temps d'une simulation complète (divisée sur plusieurs nœuds) sera donc donné par le nœud le plus lent. La parallélisation peut alors être optimisée si on arrive à donner plus de travail aux nœuds les plus performants. De cette façon, tous les nœuds, indépendamment de leurs performances, auront le même temps d'exécution mais simuleront un nombre différent de particules, ce qui améliorera le temps total de la simulation. Cette technique de parallélisation permettra en même temps un équilibrage de la charge au niveau de la grille. C'est pour ces raisons qu'une « parallélisation optimisée » s'impose. L'objectif de cette deuxième étape de mon stage a donc été la conception et l'implémentation d'un système permettant une exécution optimisée au niveau de l'application et un équilibrage de charge au niveau de la grille.

2.2.2 Architecture proposée

Initialement, si on voulait simuler un nombre P de particules (en pratique P ~ 100 Million), on envoyait N jobs (avec N ~ 100), de façon à ce que tous les jobs simulent le même nombre de particules 'P/N' et que la somme des particules simulées par tous les jobs soit égale à P. L'architecture vers laquelle on veut se diriger devrait permettre d'adapter le nombre de particules à simuler pour chaque job en fonction des performances du nœud sur lequel le job s'exécute.

La solution proposée s'appuie sur le modèle client-serveur ou bien maître-agents : les agents (clients) contactent leur maître (serveur) pour chercher du travail et pour déposer périodiquement leurs résultats. Le maître est responsable de la gestion des agents, ainsi que de la 'comptabilité' des particules déjà simulées. Quand un agent dépose ses nouveaux résultats, le maître calcule si le nombre total de particules à simuler à été atteint. En fonction du résultat, il laissera l'agent continuer la simulation ou l'arrêtera. La Figure 4 résume ce principe.

Comme on peut remarquer dans la Figure 4, les agents vont chacun à leur rythme en fonction des performances du nœud distant, du réseau ... Dans notre cas, l'agent 2 travaille plus lentement que l'agent 1 et à la fin chacun aura simulé un nombre différent de particules.

L'architecture proposée prend en compte des mesures de sauvegarde périodique des résultats, car la plateforme de la grille peut être assez imprévisible : un nœud distant ou bien la connexion vers un nœud distant peut tomber (se perdre) à tout moment. De plus, on résout le problème de la simulation incomplète à cause de quelques jobs annulés. La solution envisagée est de faire exécuter les jobs qui ont réussi un peu plus longtemps pour compenser ceux qui n'ont pas abouti.

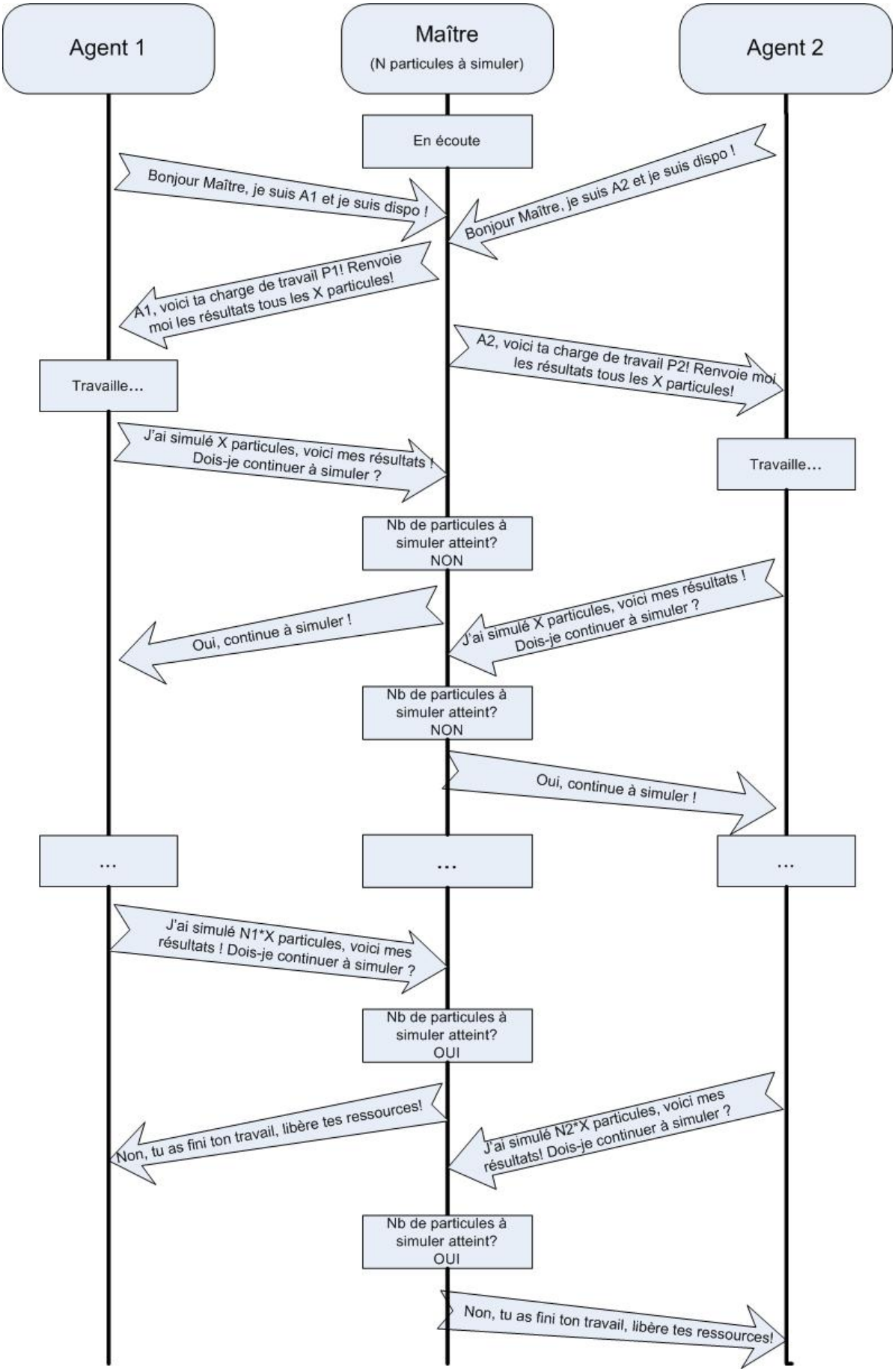

Figure 4 – Diagramme Maître-Agents

On remarque que le maître est en écoute et ce sont les agents qui doivent le contacter pour demander du travail ou transférer les résultats. Ce choix est imposé par l'architecture et les contraintes de la grille : les agents sont des jobs qui sont planifiés et envoyés sur les nœuds de la grille par le Ressource Broker. Le serveur maître n'a pas de connaissance a priori de l'adresse des machines où ces jobs sont envoyés et de toute façon, par sécurité, les nœuds n'acceptent pas des connexions entrantes (initiées pas le serveur). L'agent doit donc connaître l'adresse du maître qu'il pourra ensuite contacter. Cela est possible car le maître est lancé en premier et son adresse connue peut être transmise avec le job.

2.2.3 Matériel et méthodes utilisés

Pour l'implémentation de l'architecture conçue, nous avons décidé de s'appuyer sur des outils déjà existants. Ce choix a été fait pour deux raisons importantes : d'un côté, la complexité de la grille par rapport au temps alloué au stage et d'un autre côté la multitude des outils déjà développés dans le cadre d'EGEE. En effet, le projet EGEE a donné vie à une activité très intense autour de la grille et par conséquent à de nombreux projets très intéressants. Cependant, beaucoup de ces projets sont encore assez jeunes et peu connus à l'extérieur et même au sein des différentes communautés de la grille.

Notre choix final s'est arrêté sur deux outils développés au CERN : DIANE [11] et Ganga [12]. DIANE est un outil pour l'exécution parallèle des applications scientifiques. Ecrit en Python, il permet l'ajout de nouvelles applications (telle que ThIS) comme des modules de plugin python. Son architecture correspond parfaitement à ce que nous voulons implémenter dans notre cas : un modèle 'maître-agents' dans lequel les agents (workers) correspondent aux ressources allouées par la grille. L'allocation de ressources est indépendante de l'application elle-même et se fait d'ailleurs à travers une autre interface : Ganga. Ganga est un autre outil qui gère l'envoi des jobs sur des systèmes distribués comme la grille EGEE. Pour résumer, DIANE permet d'avoir la structure maître-agent qu'on avait conçue et Ganga permet d'envoyer les agents en tant que jobs sur la grille EGEE. Toutefois, tous les deux sont des outils génériques, que nous devons adapter afin d'intégrer notre application et les fonctionnalités de l'architecture décrite précédemment.

Comme décrit dans la Figure 5, DIANE doit être installée sur la machine serveur (sur laquelle est lancé le Maître) et sur l'interface utilisateur (UI) à partir de laquelle on a accès à la grille EGEE pour soumettre les jobs. Techniquement, le Maître peut être lancé sur l'UI aussi, mais pour des raisons de sécurité et pour une meilleure visibilité, nous avons préféré utiliser une autre machine. Sur l'UI, il doit y avoir DIANE et Ganga à la fois : DIANE pour créer les agents et Ganga pour les soumettre en tant que jobs sur la grille EGEE. Le Maître crée au lancement un fichier d'identification dans lequel il précise son adresse (adresse IP et port d'écoute). Ce fichier doit être copié sur l'UI et ensuite envoyé sur les nœuds distants (WN = Worker Node) de façon à ce que les agents sachent où contacter le Maître.

En pratique, pendant le stage, nous avons utilisé l'UI glite.unice.fr (qui se trouve au laboratoire I3S à Sophia Antipolis) pour installer Ganga et DIANE pour les agents. Le serveur DIANE a été installé et exécuté au laboratoire CREATIS sur mon PC. La machine sur laquelle s'exécute le Maître, qui fait office de serveur, doit pouvoir accepter des connexions de l'extérieur. Pour des questions de sécurité, une demande spéciale d'ouverture de port TCP a été faite pour l'adresse IP en question au niveau du service informatique de l'INSA (raison pour laquelle on a mis le serveur sur une machine autre que l'UI).

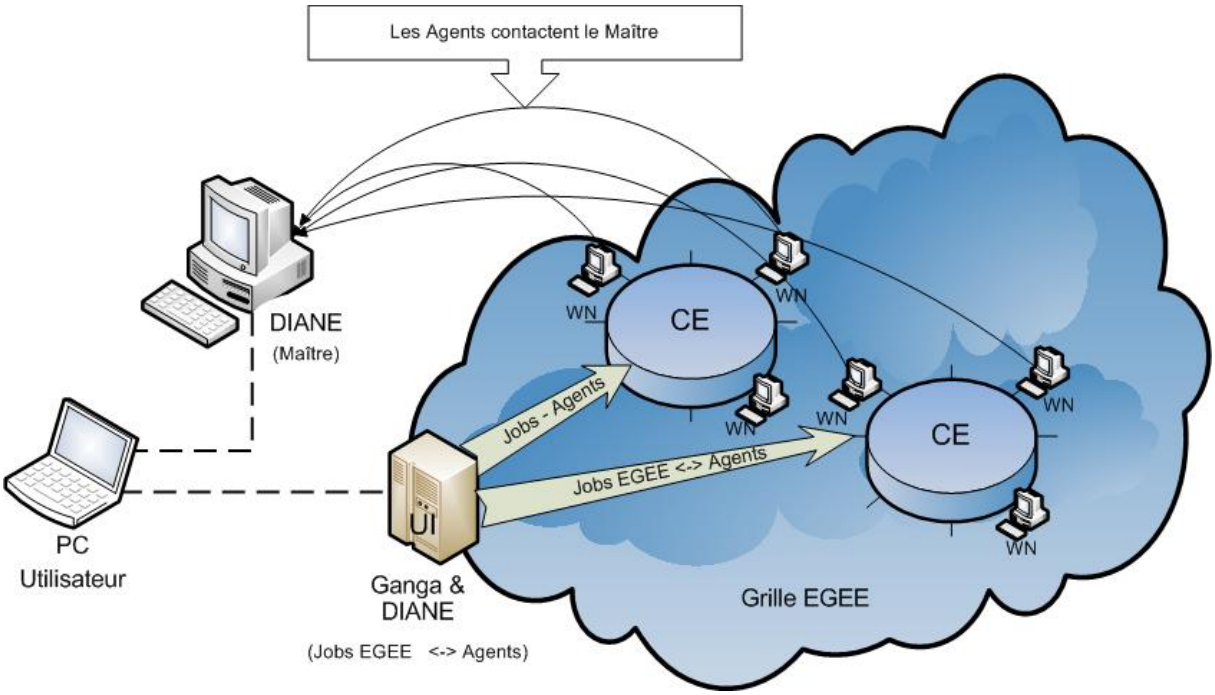

Figure 5 – Vue d'ensemble de l'architecture proposée

Le système d'exploitation par défaut pour Ganga et DIANE est Scientific Linux, c'est-à-dire le système déployé sur les nœuds de la grille. Sur la machine serveur (maître) on a aussi installé Scientific Linux 4.5. Ganga et DIANE ont été développés en Python et ils nécessitent pour leur utilisation l'installation de Python 2.3. La communication entre les clients (agents) et le serveur (maître) s'appuie sur OmniORB, une implémentation de la norme CORBA (Common Object Request Broker Architecture). CORBA est né dans les années 1990 du besoin de faire communiquer ensemble des applications en environnement hétérogène et représente aujourd'hui un nom de référence dans tout ce qui est architecture distribuée et hétérogénéité. Le package omniORB nécessaire est présent dans les packages DIANE.

2.2.4 Solution mise en place

Sur le serveur nous avons installé DIANE. DIANE possède un manager d'application (Application Manager) qui est chargé de la création et la gestion des agents. Comme précisé dans la section 1.1, pour que la division d'une simulation ThIS sur plusieurs nœuds soit correcte il faut attribuer à chaque agent un 'seed number' différent et indépendant des autres. Par conséquent, on a choisit de créer à travers le manager d'application DIANE autant d'agents que de fichiers de 'seed' donnés en entrée.

Le comportement des agents, ainsi que leur interaction avec le maître doit être défini (codé en python) dans un fichier qu'on appellera ThIS.py. Ici sont définies plusieurs classes, telle que la classe THISWorker qui contient les fonctions que l'agent exécutera. Tout d'abord, il se charge de la personnalisation de l'environnement sur le nœud distant : téléchargement des fichiers d'input nécessaires, création de répertoire, désarchivage, définition des variables d'environnement nécessaires. Il s'agit des opérations précédemment exécutées à travers les scripts shell écrits lors du portage initial de ThIS sur EGEE.

Une fois cette étape finie, l'agent lancera l'application ThIS en tant que sub-processus. Cela est nécessaire pour que l'agent garde la main sur le nœud tant que l'application s'exécute afin de pouvoir contacter le maître et transférer les résultats.

Pour cette architecture, ThIS a été enrichi avec une nouvelle fonctionnalité : celle de pouvoir lancer un script périodiquement si on le précise dans un de ces fichiers de configuration. Cette fonctionnalité est utilisée pour signaler à l'agent que de nouveaux fichiers de résultat sont prêts. Plus précisément, après avoir mis à jour ces résultats, ThIS lance un script qui va simplement créer un nouveau fichier 'OUTPUT_READY'. L'agent va alors regarder si ce fichier existe. Si c'est le cas, alors il archive les résultats temporaires et les transfère sur le serveur. Il efface ensuite le fichier 'OUTPUT_READY', qui sera recréé dès que des nouveaux résultats seront prêts. L'écriture des résultats se fait toute les n particules, où n est défini par l'utilisateur dans les fichiers de configuration de ThIS.

2.2.5 Tests et résultats

Afin d'évaluer l'efficacité de notre solution, nous comparons le temps de simulation et la fiabilité obtenus avec notre approche par rapport à l'approche existante précédemment sur la grille EGEE. Les tests ont été effectués en partant du principe qu'on veut simuler 20000000 (20M) particules à travers 100 jobs, avec donc une moyenne de 200000 particules par sous-job :

- ➔ La méthode classique consiste à envoyer 100 jobs indépendants à travers les commandes spécifiques au middleware EGEE.
- ➔ Avec la nouvelle architecture, le maître a aussi à sa disposition 100 agents indépendants lancés avec Ganga. Les résultats sont sauvegardés sur le nœud distant tous les 50.000 particules et transférés sur le serveur toutes les 100.000 particules simulées.

Les tests ont été réalisés par paquets de 100 sous-jobs pour chacune des méthodes. Les deux méthodes ont été testées simultanément pour qu'elles soient validées dans les mêmes conditions de charge de la grille. Les simulations ont été répétées à deux moments différents (début et fin août) afin de prendre en compte les éventuelles différences de charge.

Le tableau ci-dessous présente une comparaison des résultats obtenus avec la nouvelle architecture par rapport à la méthode classique de soumission de jobs sur EGEE :

	Echecs		Succes	Résultat final	Remarques
Approche classique	12% (annulés, proxy expiré)	10% (erreurs d'exécution)	78%	78%	
Nouvelle architecture	15%		65% (3% repris)	100%	20% des jobs n'ont pas eu de travail

Figure 6 - Tableau des résultats

Les résultats montrent qu'il y a des échecs dans les deux cas. Dans le cas classique, les échecs (22% en moyenne) sont dus à moitié au fait que le job est resté dans une file d'attente plus d'une journée, temps maximal de vie du proxy créé par l'utilisateur afin d'être autorisé à envoyer des jobs. Ce proxy est renouvelable, mais le job envoyé et non exécuté à temps est perdu. De toute façon, dans la majorité des cas on ne voudra pas attendre plus de 24h pour

qu’un job commence à s’exécuter. L’autre moitié des échecs provient des erreurs d’exécution qui sont souvent dues à des incompatibilités avec le nœud distant. Il s’agit souvent des nœuds d’un même CE configuré différemment et pour lequel la phase de ‘personnalisation’ n’as pas suffisamment bien marché. Dans le cas de la nouvelle architecture, on a en moyenne 15% d’échecs qui proviennent souvent des incompatibilités entre Ganga (ou bien entre notre application) et le nœud distant.

On remarque que dans le cas classique, si on ne prévoit pas plus de particules à simuler que nécessaire, on n’arrivera jamais au résultat final. On obtiendra en moyenne seulement 78% du résultat attendu. Avec la nouvelle architecture, même si seulement 65% des agents travaillent, on obtiendra bien le résultat complet. Il s’agit de 65% des jobs seulement car les jobs les plus rapides prennent la charge des jobs qui n’ont pas encore commencé à s’exécuter. On a donc 20% des agents qui, une fois qui seront en mesure de s’exécuter, n’auront plus rien à faire ! Cela ne constitue aucun problème pour les ressources de la grille, car ils ne monopoliseront pas inutilement le nœud. Dès qu’ils contactent le maître pour demander leur travail ils sauront qu’ils peuvent libérer les ressources.

Le premier avantage de notre méthode montré par ces résultats est d’avoir des résultants complets sans devoir prévoir plus que nécessaire. Le deuxième avantage, qui est moins visible à première vue mais qu’on a recherché dès le départ, est une importante optimisation du temps d’exécution et donc de récupération du résultat. Dans le cas classique, si on ne prévoit pas plus de jobs que nécessaire, on a de fortes chances d’attendre 24h jusqu’à ce que des jobs soient annulés. On n’a donc aucune garantie du temps maximum à attendre. Avec la nouvelle solution mise en place le temps d’une simulation globale telle que décrite précédemment (20M particules et 100 agents) a été de 1h45 en moyenne. Sur un PC Intel Duo Core à 2.4 GHz, une simulation de 20M de particules avec les mêmes paramètres que celle lancée sur la grille prend environ 8h30. L’exécution se réalise donc presque 5 fois plus vite. De plus, le grand avantage de la grille est qu’on peut lancer plusieurs simulations en même temps sans perdre en rapidité, ce qui n’est pas possible sur un seul PC.

En pratique, dans la majorité des cas, on voudra simuler plus de 20M de particules pour un patient (voir section 2.3). Il sera donc d’autant plus intéressant de réaliser ces calculs sur la grille et le plus rapidement possible.

2.3 Critère d'arrêt adapté aux simulations MC et à l'architecture de la grille

2.3.1 Problématique

On rappelle que ThIS simule l'irradiation des tissus vivants avec des particules (protons, photons, carbone …) dans le cadre du traitement du cancer. La tumeur doit être détruite grâce à l'irradiation, en faisant attention en même temps à ne pas endommager les tissus sains. Afin d'irradier sélectivement les tumeurs, plusieurs faisceaux sont utilisés, comme montré dans la Figure 7.

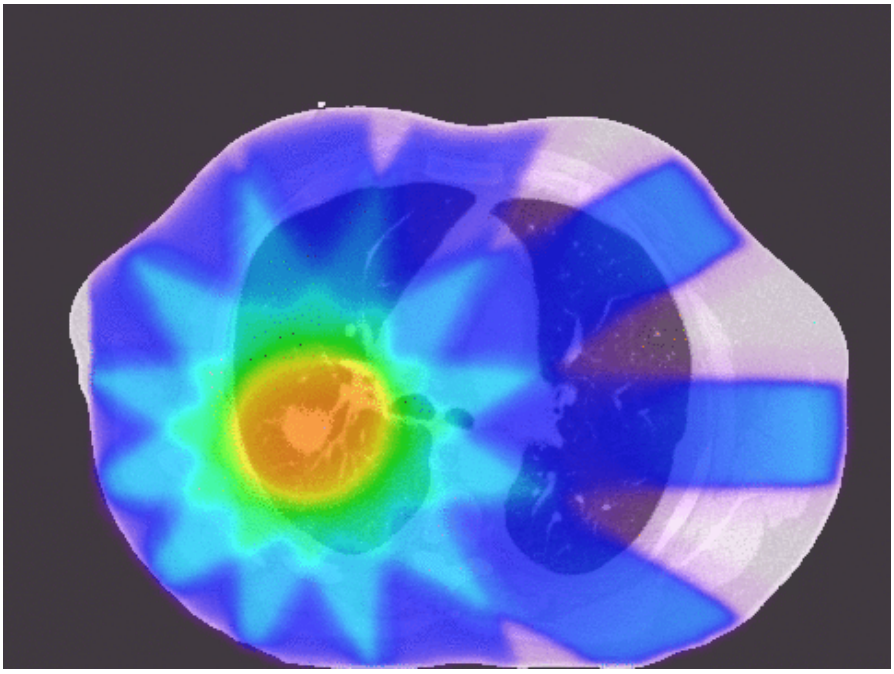

Figure 7 - Faisceaux convergeant sur la tumeur

Le but est d'irradier la tumeur avec la dose la plus grande possible, tout en limitant la dose déposée dans les tissus sains [13]. Le problème n'est pas simple à résoudre et c'est pour cela que des simulateurs comme ThIS sont utilisés pour la planification du traitement. Ces simulateurs doivent proposer un bon compromis entre leur précision et durée de simulation, car les plus précis (basés sur les technique MC) sont souvent extrêmement longs. Leur précision est généralement déterminée à l'aide des incertitudes statistiques.

Comme présenté dès le début, ThIS utilise la méthode MC, qui est une méthode de simulation statistique. Par conséquent, le dépôt de dose calculé est sujet à une incertitude statistique. Grâce au théorème centrale limite, il a été montré dans [14] que cette incertitude statistique est proportionnelle à 1/N, où N est le nombre de particules simulées. L'incertitude décroit donc avec le nombre de particules simulées, mais on n'a pas la connaissance à priori du nombre de particules à simuler pour un résultat fiable.

2.3.2 Objectif

A présent le nombre de particules à simuler est choisit grâce à l'expérience que l'utilisateur détient de l'application. L'objectif de cette dernière partie de mon stage a été de proposer une méthode automatique de détection du nombre de particules nécessaires pour une simulation fiable. Il s'agit donc d'un critère d'arrêt adapté à la fois aux simulations MC et aux contraintes de la grille. Cela permettrait aux nouveaux utilisateurs de réaliser des simulations en un temps optimum dès le départ sans avoir à passer du temps et relancer plusieurs simulations simplement pour trouver ce nombre empiriquement.

2.3.3 Matériel et méthodes utilisés

Pour la réalisation de cet objectif, deux outils ont été utilisés :

- ➔ CLITK (http://www.creatis.insa-lyon.fr/rio/CLITK). Clitk est une bibliothèque développée au laboratoire, qui permet de réaliser facilement des opérations sur les images obtenues avec ThIS (addition des résultats pour obtenir le résultat final, multiplication…), ainsi que d'implémenter le calcul d'incertitude.
- ➔ VV (http://www.creatis.insa-lyon.fr/rio/vv). VV est un outil de visualisation crée dans le but de faciliter la visualisation spatio-temporelle des images 2D, 2D+t, 3D, 3D+t. Il permet aussi la comparaison des images par transparence ou fusion, ce qui a été utilisé dans le cadre du stage pour la fusion des cartes de dépôt de dose et l'image d'entrée (thorax avec tumeur).

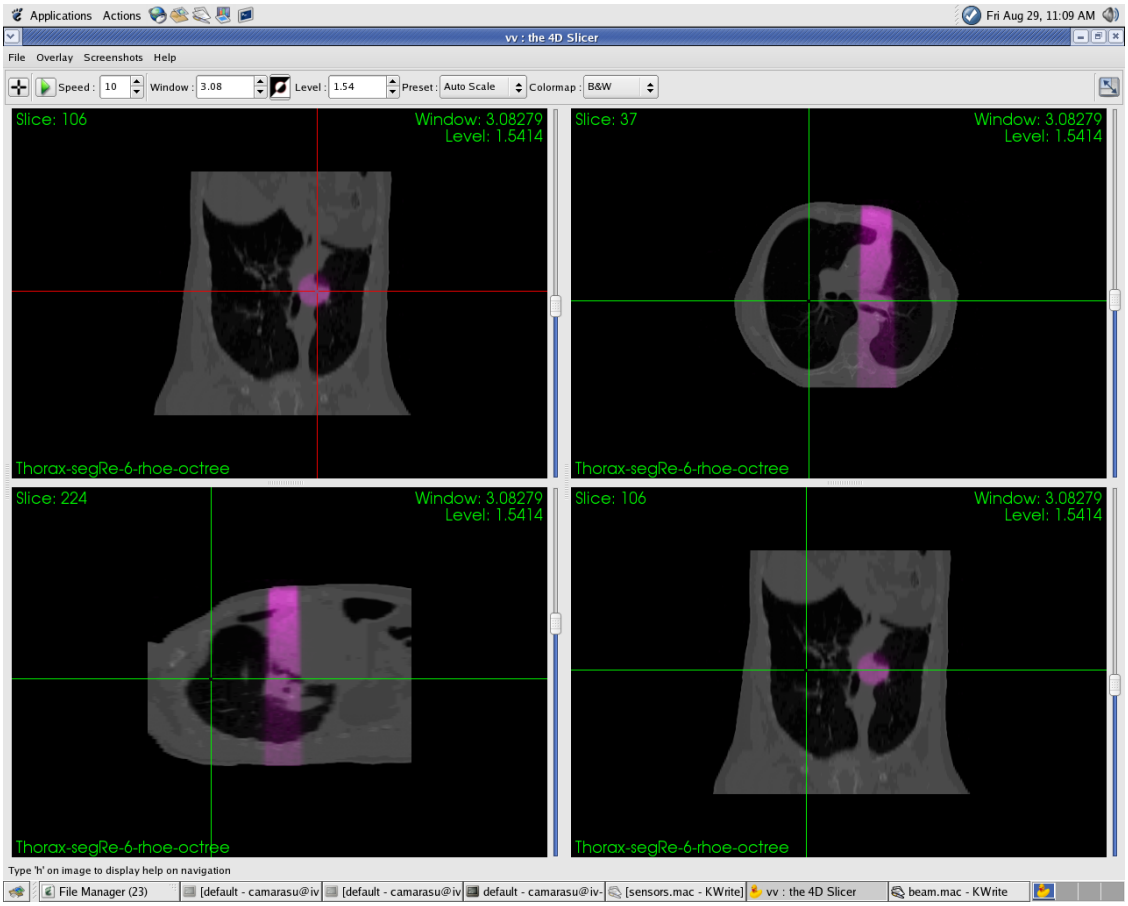

Figure 8 - Dépôt de dose superposé à l'image du thorax

Lors d'une simulation, N événements indépendants seront simulés. Il s'agit de l'irradiation des tissus avec des particules d'une certaine énergie, de façon à ce que dans chaque voxel une certaine dose soit déposée. La Figure 8 présente le résultat obtenu (le dépôt de dose) superposé à l'image d'entrée (le thorax) et visualisé avec VV.

Au cours d'une simulation complète, dans un voxel k se déposera la dose D_k, qui représente la somme des doses $d_{k,i}$ déposées aux instants i, où i est un événement indépendant parmi les N. Comme montré dans [15], l'incertitude statistique U_k sera alors donnée par :

$$U_k = \sqrt{\frac{1}{N-1} * \left(\frac{\sum_{i=1}^{N} d_{k,i}^2}{N} - \left(\frac{\sum_{i=1}^{N} d_{k,i}}{N} \right)^2 \right)} \quad (1)$$

ThIS calcule et sauvegarde (si précisé dans le fichier de configuration) à chaque fois deux cartes de dose : la carte contenant les D_k (c'est-à-dire la dose totale déposée dans chaque pixel) et la carte contenant la somme des carrées des doses, qu'on va noter S_k (comme « Squared »). Cette carte est importante et doit être écrite lors de l'exécution du programme, car sinon on ne peut pas la retrouver seulement à partir de la carte des D_k (le carré de la somme des doses est différent du carré de la somme !). La relation ci-dessus devient alors :

$$U_k = \sqrt{\frac{1}{N-1} * \left(\frac{S_k}{N} - \left(\frac{D_k}{N} \right)^2 \right)} = \frac{1}{N} * \sqrt{\frac{S_k * N - D_k^2}{N-1}}, \quad (2)$$

Ce qui représente l'incertitude statistique absolue pour un voxel k. Comme présenté dans [15], pour une interprétation plus intuitive on utilisera l'incertitude relative exprimée par :

$$\varepsilon_k = 100 * \frac{U_k}{d_k}, \quad (3)$$

Où d_k représente la quantité moyenne de la dose dans le voxel k ($d_k=D_k/N$) et ε_k représente le pourcentage relatif d'incertitude par voxel. Cette valeur est plus facile à comprendre car elle décroit si D_k augmente, ce qui paraît naturel : plus on simule et plus la dose déposée est importante, plus le résultat est certain et donc l'incertitude est moindre (car elle est proportionnelle à $1/ D_k$).

Si les différentes références ([15], [16]) semblent être d'accord sur la valeur du seuil d'incertitude à prendre en compte (2%), il y a plusieurs propositions concernant la manière de le calculer et de le prendre en compte. Parmi les différentes méthodes on peut citer le calcul du seuil sur l'ensemble des voxels, sur le voxel avec D_{min} ou D_{max} ou bien sur une certaine région d'intérêt (ROI).

2.3.4 Solution proposée

Il a été montré dans [15] que le fait d'appliquer le seuil (de 2%) d'incertitude aux points D_{min} et D_{max} seulement ne représente pas une solution suffisamment fiable. De même, le fait de l'appliquer sur la totalité de l'image n'est pas forcement une bonne solution, car il peut y avoir des régions non-concernées par le traitement et donc avec un dépôt de dose inexistant ou très faible (ce qui implique une incertitude relative maximale). Pour pallier à ce problème on peut imaginer de prendre en compte seulement les voxels avec un dépôt de dose au moins égale à 50% de la dose maximale D_{max}. Cependant, même si cette variante peut donner une bonne mesure globale de l'incertitude, elle n'est pas adaptée aux contraintes imposées par la grille et l'architecture proposée précédemment.

Pour calculer cette mesure en temps réel (en fonction des résultats des différents nœuds) on a besoin des cartes de dose de tous les nœuds. Cela implique donc que ces cartes doivent être copiées périodiquement sur le serveur pour que le maître puisse calculer l'incertitude et déterminer si la simulation doit continuer ou pas. Or, la taille d'une carte de dose peut aller, en fonction de la taille de l'image d'entrée, jusqu'à 50 Mo. On peut considérer qu'il y a en moyenne 100 agents pour une même simulation et que chaque agent a au moins une carte de dose et une carte des doses au carré qu'il doit transférer sur le serveur environ toutes les 5 minutes. Cela peut donc représenter 10 Go de données à transférer toutes les 5 minutes pour un seul utilisateur ! Cela représente donc une charge beaucoup trop importante pour la grille. C'est pour cette raison qu'on a décidé de s'orienter vers une solution qui prendrait en compte plutôt des régions d'intérêt (ROI), ce qui pourrait réduire considérablement la taille des fichiers à transférer.

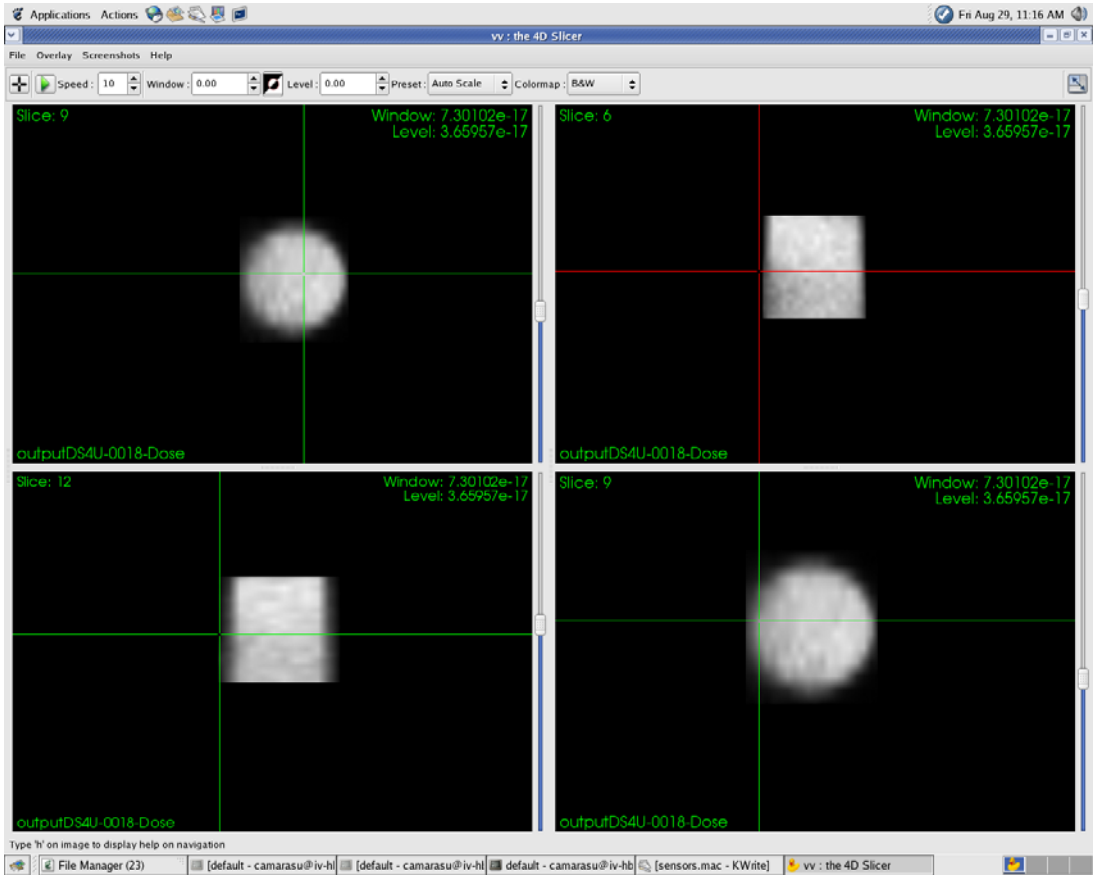

Figure 9 - Dose déposée dans la ROI

Les ROI pour le calcul de l'incertitude sont d'ailleurs proposés dans d'autres ouvrages aussi. [16] étudie la le choix de la ROI par rapport au volume cible à traiter ou PTV (Planning Target Volume) et éventuellement aux contraintes des organes pouvant être endommagés par le traitement ou OAR (Organs At Risk). Afin de ne pas introduire une trop grande charge de travail pour délimitation du ROI et afin de limiter la taille du fichier à transférer, on a décidé de choisir comme ROI la région centrale du volume cible à traiter (la tumeur).

Le choix de la région peut se faire visuellement grâce à l'outil de visualisation VV. Une fois le centre et la taille de la ROI choisis, ThIS permet de les spécifier dans un de ses fichiers de configuration. Cela permettra de sauvegarder les résultats de la simulation pour cette région dans des fichiers autres que les fichiers pour l'image globale. De cette façon, les agents transféreront à intervalle régulier seulement ces fichiers plus petits à partir desquels le maître réalisera le calcul de l'incertitude. Les fichiers complets seront transférés pendant la simulation pour des raisons de sécurité (pour ne pas les perdre si le nœud tombe), mais à intervalle plus important et bien sûr à la fin de la simulation pour le résultat final. La Figure 9 présente la dose obtenue dans la ROI et visualisée avec VV.

Il faut préciser que nous avons aussi envisagé de faire le calcul d'incertitude sur chaque nœud pour éviter le transfert de données sur le serveur uniquement pour ce calcul. Cependant, cette solution n'est pas possible, car l'incertitude ne peut pas être calculée localement. Comme la relation (2) le montre, l'incertitude U_k est le résultat d'une racine carrée qui dépend de la somme de tous les N événements. Or, un nœud simule seulement une partie de ces N événements et pour ce calcul il faut avoir accès à la totalité. C'est pour cette raison que les résultats temporaires doivent être centralisés avant de calculer l'incertitude.

2.3.5 Tests et résultats

Il faut rappeler que cette étude est surtout une étude qualitative dans la mesure où on veut déterminer si la solution proposée est fiable. Des tests beaucoup plus conséquents devront être réalisés et interprétés avec l'aide d'un physicien des particules expert en dosimétrie pour valider notre approche.

Les tests ont été réalisés en faisant varier le nombre de particules, tout en utilisant les mêmes images d'entrée et les mêmes propriétés du faisceau de protons. L'incertitude a été calculée à partir de la formule (3) sur des régions différentes de l'image et dans deux cas de figure : avec et sans prendre en compte les voxels dans lesquels il n'y a pas de dose déposée. Les résultats sont résumés dans la Figure 10.

Nb particules	Img entiere sauf voxels où Dk=0	Img entiere	ROI	ROI sauf voxels où Dk=0
1 000 000	82%	93%	22%	22%
5 000 000	64%	80%	10%	10%
7 000 000	61%	78%	9%	9%
21 000 000	42%	64%	5%	5%
50 000 000	29%	55%	3%	3%

Figure 10 - Résultats des incertitudes

Les résultats obtenus sont cohérents et correspondent globalement aux résultats attendus. L'incertitude relative décroit avec le nombre de particules simulées (et donc avec la dose déposée) et elle reste très grande pour moins de 20M particules.

On remarque qu'une incertitude calculée sur l'ensemble de l'image (que ça soit sans ou avec les voxels sans dose) est une mesure trop globale qui n'est pas très significative. Le fait de choisir une ROI est donc d'autant plus justifiée. La ROI choisie près de la tumeur donne des informations intéressantes et les valeurs des incertitudes sont proches de celles présentées dans la littérature [15]. On remarque aussi que dans cette configuration, le fait de prendre ou pas en considération les pixels avec une dose nulle n'est plus important, car on est dans une région près de la tumeur où tous les pixels reçoivent une dose importante.

Les résultats montrent donc que notre solution est fiable, mais des tests quantitatifs devront être conduits pour une analyse plus fine de ce critère, ainsi que pour mieux déterminer le choix de la ROI.

3 Discussion et retour d'expérience

3.1 Difficultés rencontrées

Une première difficulté importante a été la maitrise, dans un temps relativement court, des problématiques et des domaines d'activité très diversifiés : des grilles de calcul, de la simulation MC et de l'imagerie médicale, des architectures maître-agents, de la statistique, de la programmation et de la configuration Linux... Cependant, malgré les difficultés, cette diversité a été pour moi un vrai plus car elle a apporté une dimension très intéressante à ce stage.

Le côté pratique du stage et la mise en place des solutions nouvelles ont représenté une source importante de difficultés techniques. La soumission des jobs sur la grille a constitué une première étape difficile non seulement à cause de la nouveauté qu'elle représentait pour moi, mais surtout à cause de la multitude d'erreurs qui peuvent intervenir indépendamment de notre action : impossibilité de créer le proxy pour l'authentification, RB (Ressource Broker) en panne temporairement… La compilation statique ou la configuration d'un serveur (et du firewall) sous Linux ont représenté bien d'autres défis techniques intéressants.

3.2 Analyse globale des résultats

Le système d'exécution optimisé conçu et mis en place lors de ce stage a été testé et comparé à la solution de base disponible dans le cas général. Les résultats obtenus montrent les performances de la solution proposée, qui présente une optimisation importante par rapport à l'existant. L'optimisation concerne plusieurs aspects :

- ➔ Il s'agit tout d'abord du temps de calcul, qui est réduit considérablement. Cette réduction du temps de calcul a été réalisée essentiellement grâce à une optimisation de la répartition du travail en fonction de la rapidité des jobs, mais aussi grâce au critère d'arrêt proposé. Cette diminution du temps d'exécution et l'obtention d'un résultat plus rapide est une contrainte importante dans l'adoption des applications telle que ThIS dans le système médical.
- ➔ Dans un deuxième temps, on peut parler d'optimisation dans la facilité d'utilisation. La solution accessible à travers les services standards de la grille présente une complexité trop importante pour qu'elle soit facilement adoptée par des nouveaux utilisateurs. La solution proposée dans le cadre du stage garantit une automatisation importante, étant donc plus facile à utiliser une fois mise en place.

Ces résultats sont très prometteurs et ouvrent la voie vers des nouvelles perspectives.

3.3 Perspectives

On a vu que l'architecture maître-agent est performante, mais elle peut être améliorée. Le degré d'automatisation peut être augmenté pour une utilisation plus facile. A terme il serait souhaitable d'intégrer ce système dans un portail web très facile et intuitif à utiliser par les chercheurs et médecins. De plus, le système doit encore être amélioré pour être plus facilement configurable.

Concernant la solution proposée pour la condition d'arrêt, elle a été testée seulement indépendamment, c'est-à-dire qu'elle n'a pas encore été intégrée dans le système de soumission maître-agents. A court-terme, une série de tests plus évolués permettra une appréciation plus quantitative de cette condition d'arrêt. A plus long-terme, une amélioration importante serait la détection automatique de la ROI à sélectionner pour le calcul de l'incertitude.

4 Conclusion

Ce stage de master a constitué un travail extrêmement intéressant. La diversité des sujets abordés, leur nouveauté, ainsi que les gens avec lesquels j'ai eu la chance de travailler m'ont donné une motivation très forte qui m'a aidée à toujours travailler avec beaucoup d'intérêt.

Les solutions proposées suite au travail de recherche effectué se sont montrées appropriées et efficaces. Des améliorations seront bien sûr désirables et je souhaite fortement continuer afin de transformer ce projet en un outil performant et utilisé par le plus grand nombre d'utilisateurs. La grille apporte une scalabilité (la possibilité d'accepter un grand nombre d'utilisateurs en même temps sans détérioration de performances) très importante et appréciée dans ces cas d'utilisation. A terme on envisage de créer une plateforme virtuelle de radiologie [10] qui regrouperait un nombre important de choix d'applications médicales capables de s'exécuter sur des machines distantes (très probablement une grille de calcul, mais aussi des ressources isolées) de manière transparente à l'utilisateur.

Les grilles de calcul sont des technologies nouvelles et puissantes qui sauront s'imposer à plus grande échelle dans l'avenir. Comme on l'a vu à travers ce travail, leur adaptation n'est pas encore parfaite pour tous les domaines d'activités, mais il y a déjà une activité importante partout dans le monde pour trouver des solutions à ces problèmes.

Je suis très contente d'avoir la possibilité de travailler sur des problématiques aussi intéressantes que les grilles de calcul et l'imagerie médicale et surtout de contribuer à leur approche et leur futur.

5 Bibliographie

[1] J. Montagnat, et al., "Medical images simulation, storage, and processing on the European DataGrid Testbed," *Journal of Grid Computing (JGC)*, vol. 2, 2004.

[2] ThIS Homepage, http://www.creatis.insa-lyon.fr/rio/ThIS

[3] EGEE Homepage, http://www.eu-egee.org/

[4] J. Allison, "Geant4 - A Simulation Toolkit," *Nuclear Physics News*, vol. 17, 2007.

[5] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency and Computation: Practice and Experience*, vol. 17, pp. 323-356, 2005.

[6] Globus Homepage, http://www.globus.org/toolkit/

[7] F. Chollet and E. Fede, "Accessing EGEE/LCG Grid infrastructure from LAPP," 2007.

[8] T. Glatard, D. Lingrand, J. Montagnat, and M. Riveill, "Impact of the execution context on Grid job performances," in *International Workshop on Context-Awareness and Mobility in Grid Computing (WCAMG'07)*, Rio de Janeiro, 2007.

[9] S. Camarasu, H. Benoit-Cattin, J. Montagnat, and D. Racoceanu, "Grids for Content-Based Medical Image Indexing and Retrieval," in *ICT4Health*, Manila, 2008.

[10] S. Camarasu, et al., "Towards a Virtual Radiological Platform Based on a Grid Infrastructure," in *MICCAI*, New York, 2008.

[11] J. T. Moscicki, "DIANE - Distributed Analysis Environment for GRID-enabled Simulation and Analysis of Physics Data," in *NSS IEEE* NSS IEEE, 2004.

[12] A. Maier, et al., "Ganga - an optimiser and front-end for Grid job submission," in *Second EGEE User Forum* Manchester, 2007.

[13] U. Amaldi, "HADRONTHERAPY IN THE WORLD."

[14] I. J. Chetty, et al., "AAPM Task Group Report No. 105: Monte Carlo-based treatment planning," *Med. Phys.*, vol. 34 2007.

[15] I. J. Chetty, et al., "REPORTING AND ANALYZING STATISTICAL UNCERTAINTIES IN MONTE CARLO–BASED TREATMENT PLANNING," *Int. J. Radiation Oncology Biol. Phys.*, vol. 65, pp. 1249–1259, 2006.

[16] B. Vanderstraeten, et al., "EVALUATION OF UNCERTAINTY-BASED STOPPING CRITERIA FOR MONTE CARLO CALCULATIONS OF INTENSITY-MODULATED RADIOTHERAPY AND ARC THERAPY PATIENT DOSE DISTRIBUTIONS," *Int. J. Radiation Oncology Biol. Phys.*, vol. 69, pp. 628–637, 2007.

6 Annexes

Annexe 1

ThIS Static Build (http://www.creatis.insa-lyon.fr/rio/ThIS/StaticBuild)

This document describes a step by step procedure for building a static version of ThIS.

1. In order to be able to link ThIS only to static libraries, you will first need to build CLHEP and Geant4 statically.

a.For **CLHEP** use the option disable-shared when configuring the CLHEP build like this:

- *./configure* ***--disable-shared*** *--prefix /home/CLHEP*

b. For **Geant4** do not change the default option of building only static libraries. To do this answer 'no' to the following question asked when configuring your build:

- *By default 'static' (.a) libraries are built.*
- ***Do you want to build 'shared' (.so) libraries? [n]***

If you need help with installing Geant4 and CLHEP you may have a look at http://geant4.slac.stanford.edu/installation/ for detailed and well structured instructions.

2. Link **ThIS** statically by adding ***'LDFLAGS+=-static'*** at the end of the ThIS GNUmakefile.

After compilation (*make release=1*), if you want to check that the executable does not use shared libraries, you can use the 'ldd' command which will print out for you the names of the needed shared libraries:

- *ldd this* (where this=executable name)

If you get no answer means you successfully linked ThIS statically.