Institut Supérieur Industriel de Bruxelles
Rue Royale 150 — 1000 Bruxelles
Rue des Goujons 28 — 1070 Bruxelles
www.isib.be

Enseignement Supérieur de Type Long et de Niveau Universitaire

Haute Ecole Paul-Henri Spaak
Catégorie Technique

Déploiement de programmes en R et la plateforme Tornado à exécuter les simulations parallèles sur une grappe de calcul utilisant Sun Grid Engine

M. Mai Quang Minh NGUYEN

Travail de fin d'études

Effectué au sein de l'entreprise :
model*EAU* - Université LAVAL
Avenue de la Médecine 1065, G1V 0A6 Québec, QC
Canada

Présenté en vue de l'obtention du grade
de Master en Sciences de l'Ingénieur Industriel
en Informatique

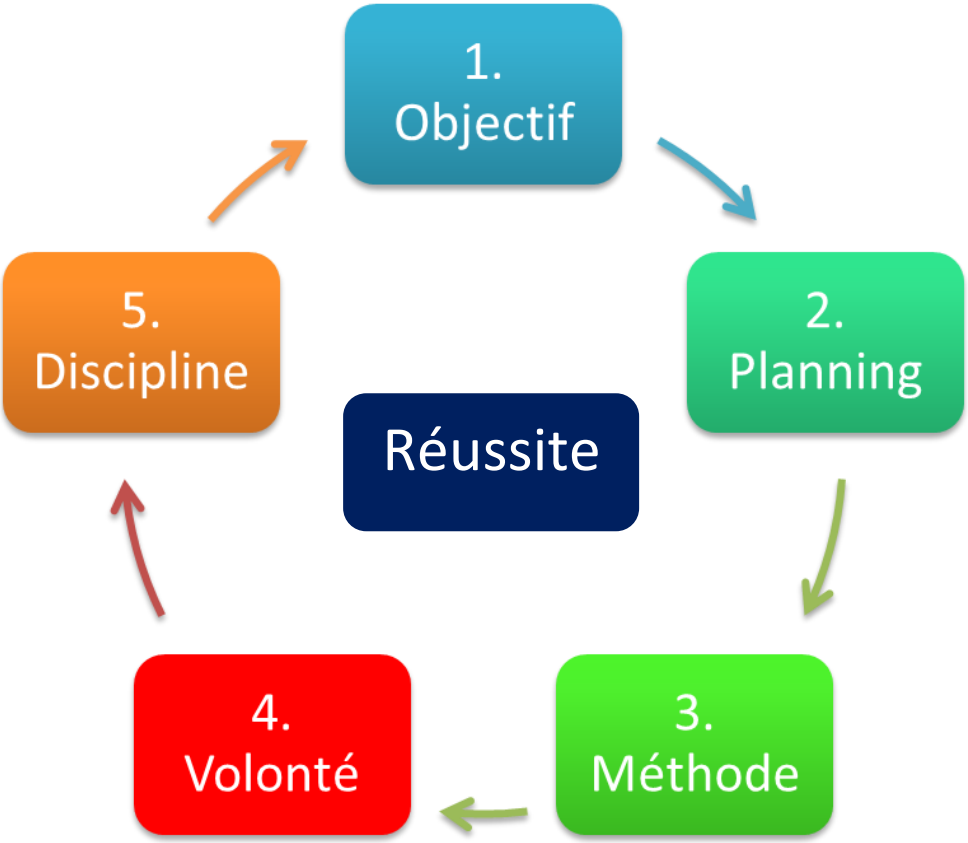
———————

Année Académique 2010-2011

———————

Numéro : ISIB-ELIN-TFE-11/04
Classification : confidentiel

———————————


1.
Objectif
2.
Planning
3.
Méthode
4.
Volonté
5.
Discipline
Réussite

Promoteurs

Prof. Dr. Ir. Peter A. VANROLLEGHEM
Chaire de recherche du Canada en modélisation de la qualité de l'eau
Département de génie civil et de génie des eaux, model*EAU*, Université Laval, Québec

Prof. Dr. Ing. Jacques TICHON
Professeur en électronique, Institut Supérieur Industriel de Bruxelles

Prof. Ir. Rudi GIOT
Professeur en informatique, électronique numérique, Institut Supérieur Industriel de Bruxelles

Résumé

La gestion environnementale et la modélisation de l'information sur la qualité de l'eau actuelles font appel à la modélisation de systèmes. Le processus de modélisation requiert une phase de simulation fonctionnelle afin de prévoir et aider à prendre des décisions adéquates pour prévenir la pollution ou gérer l'eau des rivières, des barrages. Bien entendu les prévisions ne livrent pas des valeurs précises, mais elles déterminent un espace de valeurs plausibles, la modélisation permet donc aux gestionnaires de considérer l'incertitude et de prendre de meilleures décisions.

Comme il s'agit en général des simulations très coûteuses en temps de calcul, les machines parallèles avec les algorithmes adéquats constituent le moyen de réaliser des simulations parallèles. Une grappe de calcul de haute performance (supercalculateur Colosse) est constituée par un assemblage des machines de calculs parallèles dont le nombre de processeurs dépasse 7000. Elle a été installée au sein de l'Université Laval, Québec pour rendre service à plusieurs groupes de recherche du Canada.

Ce travail de maîtrise décrit l'environnement de programmation en langage R, conçu pour créer et exécuter des simulations parallèles employant le simulateur Tornado sur le supercalculateur Colosse utilisant l'ordonnanceur Sun Grid Engine.

Le langage R est un langage de programmation libre, un excellent outil pour les calculs de statistiques et graphiques. Le noyau R fournit un compilateur R et plusieurs fonctions d'analyse statistique et de modélisation. R fournit également la possibilité de faire les calculs parallèles en utilisant le module snowfall. Ce complément de R le permet de faire des simulations parallèles de manière simple et efficace. Deux programmes ont été développés en R permettant l'exécution des simulations parallèles, l'un a été réalisé en appelant les fonctions de l'interface TornadoR du simulateur Tornado et l'autre a été implémenté avec les fonctions natives du simulateur Tornado.

Tornado est une plateforme générique de simulation utilisée par l'équipe de recherche model*EAU* de l'Université Laval pour le paramétrage et l'exécution de simulations relatives à la gestion de la qualité de l'eau. TornadoR est une interface faisant le pont entre R et Tornado. À l'intérieur du code R, il peut exécuter une simulation sur Tornado au travers de l'interface TornadoR appelant les fonctions de Tornado.

La réalisation de ce travail est divisée en 2 étapes principales :

- ✓ La première étape consiste à déployer le simulateur Tornado sur le supercalculateur Colosse en étant opérationnel sous Linux.
- ✓ La seconde étape concerne la recherche et le développement des programmes en R permettant de créer et exécuter des simulations en parallèle utilisant le simulateur Tornado.

Les résultats obtenus pour ces applications sur le supercalculateur Colosse démontrent l'efficacité de l'exécution en parallèle en nous apportant un gain énorme de temps de calcul.

Abstract

The environmental management and modeling of information on water quality calls upon the modeling of these systems. The process of modeling requires a phase of functional simulation in order to predict and help to take adequate decisions to prevent pollution or to manage the water quality of rivers, dams. Given the fact that the model outputs do not deliver precise values, but rather determine a space of plausible values, modeling allows us to consider the uncertainty and to take better decisions.

Generally, the time of calculation for water quality simulations is very long; and parallel machines with adequate algorithms therefore constitute the means to realize parallel simulations. A cluster of High Performance Computing (supercomputer Colosse) built by an assembly of machines for parallel calculations with over 7000 processors is installed at the Université Laval, Quebec to support several research groups in Canada.

This work describes the programming environment with the language R, designed to create and run parallel simulations using the Tornado simulator on the supercomputer Colosse using the Sun Grid Engine scheduler.

The R Language is an Open Source programming language, an excellent tool for statistical analysis and graphics. The kernel R provides a compiler for R and several functions for statistical analysis and modeling. R also provides the possibility to perform parallel calculations using the module snowfall. This module of R allows performing parallel simulations in a simple and effective way. Two programs were developed in R to perform parallel simulations. The first one was realized by calling functions from the TornadoR interface of the Tornado simulator and the second program was implemented with the native functions of Tornado simulator.

Tornado is a generic simulation platform used by the research team model*EAU* at Université Laval for parameter evaluation and execution of simulations for the management of water quality. TornadoR is an interface making the connection between R and Tornado. Inside the R code, it can run a simulation on the Tornado simulator through the TornadoR interface calling Tornado functions.

The realization of this work is divided into two main steps:

- ✓ The first step consists to deploy the Tornado Simulator on the supercomputer Colosse which operates under Linux.
- ✓ The second step is the research and development of 2 programs written in R to create and run parallel simulations using the Tornado simulator.

The obtained results for these applications on the supercomputer Colosse demonstrate the effectiveness of the execution in parallel bringing to us a huge gain in calculation time.

Remerciements

Tout d'abord, je souhaite adresser mes sincères remerciements au Professeur Peter Vanrolleghem pour m'avoir accepté comme étudiant stagiaire, de m'avoir accueilli si chaleureusement au sein de l'équipe de recherche modelEAU à l'Université Laval à Québec et pour l'excellent encadrement qu'il m'a offert tout au long de ce stage. J'ai beaucoup apprécié l'approche par laquelle il m'a permis de découvrir de manière générale pas à pas des activités du groupe de recherche modelEAU. Je lui suis reconnaissant pour le temps, la patience et l'énergie qu'il a investis pour me guider.

Je remercie également mes superviseurs académiques, le professeur Jacques Tichon, ainsi que le professeur Rudi Giot pour m'avoir permis d'effectuer ce stage au Canada et pour leurs encouragements et leurs conseils précieux. Je tiens à remercier l'IRISIB pour son soutien financier; ce stage n'aurait pas pu avoir lieu sans leur aide.

Mes remerciements vont à Dr. Filip Claeys pour m'avoir constamment aidé et expliqué le fonctionnement de Tornado et de m'avoir donné des conseils exceptionnels. De même, je tiens à remercier Ir. Cyril Garneau, ainsi qu'Ir. Arinala Randrianantoandro pour leurs collaborations.

Évidemment, je tiens à remercier tous les membres de l'équipe modelEAU pour leur bon accueil, leur élaboration durant tout mon stage.

Enfin, je voudrais adresser mes remerciements les plus profonds à ma famille, mes professeurs, mes amis qui m'ont accompagné tout au long de mes années d'études et qui auront ainsi contribué à l'aboutissement de ce travail.

Québec, juin 2011
Nguyen Mai Quang Minh

Table des matières

Liste des figures

Liste des tables

1
Introduction et objectifs

L'eau a gardé son importance en tant que constituant essentiel du corps humain, de par ses nombreux rôles dans le fonctionnement de l'organisme et parce qu'elle est l'élément essentiel en matière d'hygiène et de remèdes. L'eau est vie lorsqu'elle est de bonne qualité, l'eau est poison lorsqu'elle contient des contaminants. Pour ces raisons, apprendre et améliorer la qualité de l'eau afin de protéger notre source de vie sont une des missions du groupe de recherche model*EAU* du département de génie civil et de génie des eaux de l'Université Laval, Québec.

model*EAU* est un groupe de recherche autour du développement et de l'utilisation de la modélisation de la qualité de l'eau. Il focalise sur les méthodologies de collecte et d'évaluation de la qualité des données, du développement de nouveaux modèles et d'amélioration des approches de modélisation, et l'optimisation des systèmes d'eau sur base de ces modèles. L'équipe est dirigée par le Professeur Peter Vanrolleghem; elle est composée de dix étudiants gradués, quatre postdoctorats et de deux chercheurs professionnels. (En juin 2011)

Site officiel : http://modeleau.fsg.ulaval.ca/

Ce travail de fin d'études concerne le déploiement de la plateforme de simulation Tornado sur le super calculateur Colosse opérant sous Linux et le développement de deux programmes en langage R en vue de créer et d'exécuter des simulations parallèles sur Colosse. Il a été réalisé au sein du groupe model*EAU* à l'Université Laval de février à juin 2011.

La gestion environnementale et la modélisation de systèmes d'eau deviennent de plus en plus exigeantes au niveau de temps de calcul et sont donc complexes à gérer. Des outils informatiques se basent sur l'utilisation des modèles mathématiques simulant les comportements écologiques à gérer. Ils sont indispensables aux gestionnaires pour mieux comprendre le fonctionnement afin d'améliorer la qualité de l'eau et de prendre les bonnes décisions de gestion.

Nous allons voir un des outils utilisés par l'équipe model*EAU*. Il s'agit du simulateur Tornado; c'est une plateforme de simulation générique et performante permettant de définir des modèles de simulations, de changer les valeurs de paramètres et d'exécuter des expériences virtuelles sur les modèles de simulation. Tornado a été développé par la société MOSTForWATER en collaboration avec BIOMATH de l'Université de Gand, Belgique.

Pour exploiter toutes les capacités d'un outil tel que Tornado, il faut avoir une architecture de calculs de haute performance. La grappe de calcul ou bien le super calculateur Colosse qui a été installé et mis à la disposition des différentes équipes scientifiques, a augmenté et amélioré considérablement l'efficacité et la capacité de calcul pour la recherche grâce au nombre énorme de processeurs travaillant en parallèle fournissant une capacité colossale de calcul.

Le déploiement du simulateur Tornado sur Colosse est différent de celui d'un ordinateur ordinaire dans lequel les tâches sont exécutées en mode séquentiel sur un, voire plusieurs processeurs. Il fallait donc installer et configurer Tornado de manière telle qu'il travaille parallèlement avec les processeurs du Colosse. En plus, le développement des programmes en R devrait tenir compte de cette différence de Colosse pour pouvoir exploiter parallèlement ses processeurs.

Les objectifs du travail sont :

- Le déploiement du simulateur Tornado de manière à exploiter parallèlement les processeurs du Colosse en maintenant la compatibilité avec les autres modules existants et ceux à développer à l'avenir.
- Le développement des programmes en langage R utilisant le simulateur Tornado pour créer et exécuter les simulations parallèles sur Colosse.

Ce mémoire est divisé en 7 chapitres :

Chapitre 1: L'introduction et les objectifs du projet.
Chapitre 2: La grappe de calcul Colosse, ses caractéristiques ainsi que son système de gestion des tâches de calcul Sun Grid Engine.
Chapitre 3: Les bases de l'architecture et du fonctionnement de Tornado.
Chapitre 4: Le langage de programmation R et les outils permettant de faire les calculs parallèles dans l'environnement R.
Chapitre 5: La procédure pour le déploiement de la plateforme Tornado sur Colosse.
Chapitre 6: Le développement de deux solutions satisfaisantes aux objectifs et les tests ainsi que les comparaisons de celles-ci.
Chapitre 7: La conclusion du projet avec les objectifs réalisés et les perspectives.

Première partie

État de l’art

2
La grappe de calcul Colosse

Ce chapitre décrit les spécifications et l'architecture de la grappe de calcul (super calculateur) Colosse ainsi que son utilisation avec Sun Grid Engine. Elle appartient au CLUMEQ et nous allons donc commencer par découvrir ce dernier.

Les figures sont principalement reprises de [1] et [2].

2.1 CLUMEQ

Le CLUMEQ est un consortium de recherche pour le calcul scientifique de haute performance (CHP). Il regroupe L'Université McGill, l'Université Laval ainsi que l'ensemble du réseau de l'Université de Québec. Sa mission est d'offrir à ses membres une infrastructure de CHP de classe mondiale pour l'avancement des connaissances dans tous les secteurs de la recherche et d'aider les chercheurs à exploiter efficacement les infrastructures.

Le CLUMEQ fait partie de Calcul Canada, une plateforme nationale pour le CHP, qui contrôle les sept consortiums régionaux canadiens. À travers Calcul Canada, les infrastructures du CLUMEQ sont accessibles à l'ensemble des chercheurs universitaires canadiens.

Figure 2.1 Colosse : Consortium CLUMEQ

2.2 Spécifications

Colosse est une grappe de calcul [1]constituée de 960 nœuds [2]de calcul et 40 nœuds d'infrastructure. Ces nœuds sont tous constitués de 2 processeurs Intel Nehalem-EP possédant chacun 4 cœurs de traitement et 24 gigaoctets de mémoire RAM. Au total, Colosse comporte donc 8000 cœurs et 24 téraoctets de mémoire.

Tous les nœuds sont reliés par un réseau de haute performance de type Infiniband QDR dont la vitesse nominale est de 40 gigabits/sec. Les nœuds d'infrastructure sont également reliés entre eux et avec l'extérieur par un réseau de type 10-gigabit Ethernet. Parmi 40 nœuds d'infrastructure, 20 nœuds sont consacrés au système de fichiers parallèle Lustre dont la capacité utilisable atteint 1 pétaoctets.

Le système d'exploitation de Colosse est un système Linux, distribution Redhat EL5, version du kernel 2.6.18. En juin 2011, Colosse est classé 97^{ième} dans le top 500 super calculateurs mondiaux d'après www.top500.org.

2.2.1 Architecture

Colosse était le silo d'un ancien accélérateur de particules de type 'Van Der Graaff', il a été transformé en une enceinte de refroidissement de haute efficacité énergétique. Il a été complété au printemps 2009 et possède un design unique au monde. Colosse peut installer jusqu'à 56 cabinets de serveurs sur trois niveaux du bâtiment selon un aménagement cylindrique.

Les aperçus de l'architecture sont illustrés à la figure 2.2 et à la figure 2.3.

Figure 2.2 Colosse : Aperçu de l'extérieur

[1] Ou cluster, c'est un ensemble constitué de nœuds en général identiques et interchangeables administré d'une manière unique et centralisée.
[2] La plus petite unité de calcul constituant d'un cluster permettant le traitement autonome de données.

L'architecture du bâtiment est exceptionnelle incluant les éléments suivants :

- Les serveurs sont installés en cercle sur plusieurs niveaux, on obtient donc une topologie cylindrique où le cœur du cylindre correspond à une allée chaude unique et la zone annulaire périphérique correspond à une allée froide également unique.
- Les structures circulaire et verticale permettent d'éliminer les coins pour éviter les turbulences, de réduire l'accélération de l'air en maximisant la surface libre de plancher, et de minimiser la longueur des câbles du réseau.
- Des planchers de caillebotis entre chaque niveau permettent la circulation de l'air.
- Le système de refroidissement est installé au sous-sol.
- Le système d'apport d'air frais installé au-dessus du troisième niveau permet le refroidissement partiel de la grappe en utilisant l'air froid extérieur.

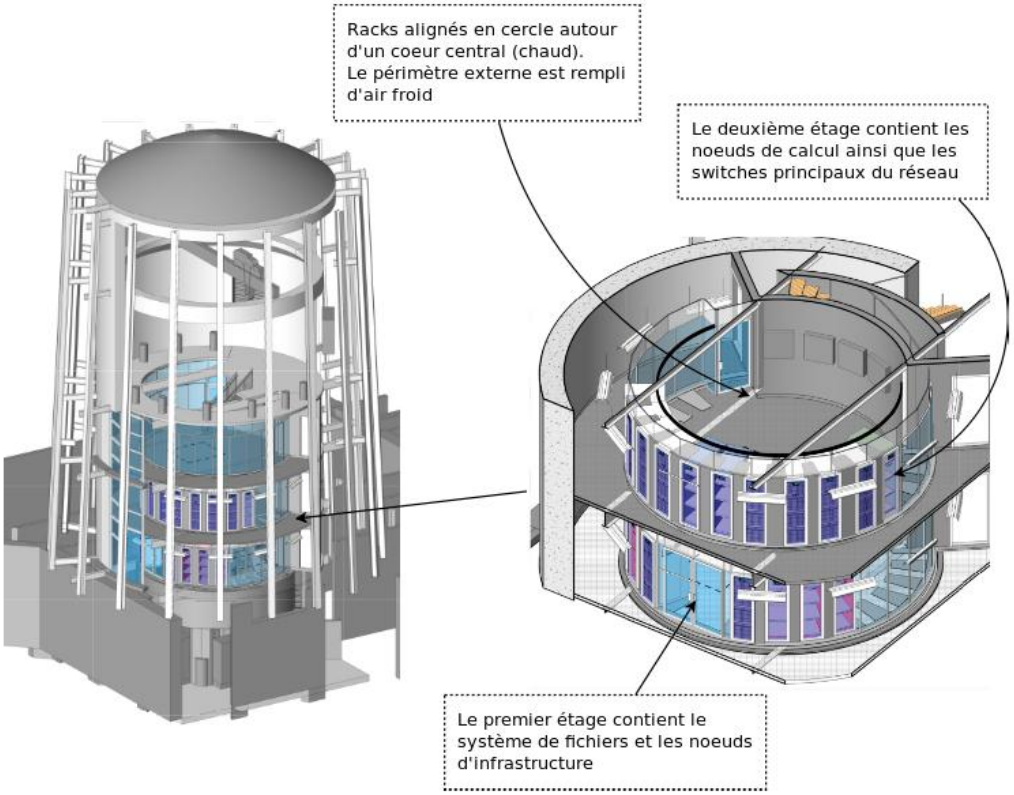


Figure 2.3 Colosse : Aperçu de l'intérieur

2.2.2 Infrastructure

D'après les informations publiées sur le site du CLUMEQ, Colosse est composé actuellement de 10 châssis de Sun Constellation C-48 Racks. Chaque châssis comporte 4 cabinets à lames et chaque cabinet peut contenir 12 lames de calcul de type Sun X6275. Chaque lame de calcul dispose 2 nœuds de calcul, composés de 2 processeurs Intel Nehalem-EP cadencés à 2.8 Ghz.

Donc, l'ensemble de 10 châssis de C-48 constitue 960 nœuds de calcul avec 2 processeurs de 4 cœurs/ processeur. Ces nœuds de calcul sont connectés et liés au système de fichier Lustre. La figure 2.4 a été prise au sein de Colosse en avril 2011 et la figure 2.5 présente un aperçu technique d'un châssis de Sun Constellation C-48.

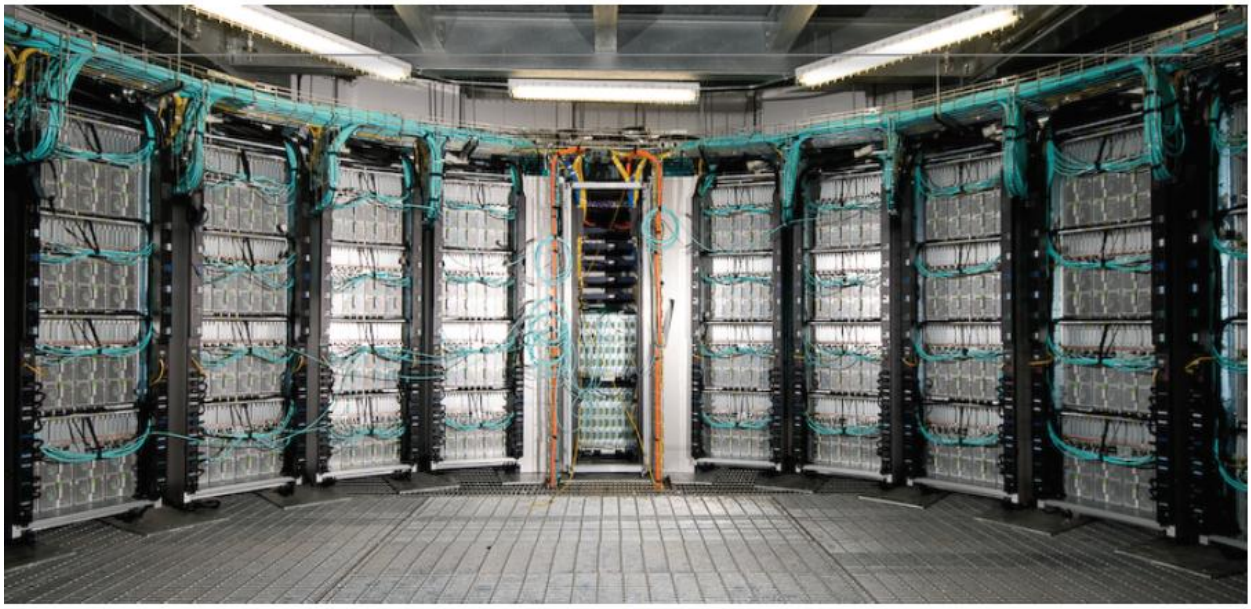


Figure 2.4 Colosse : Aperçu général de l'infrastructure

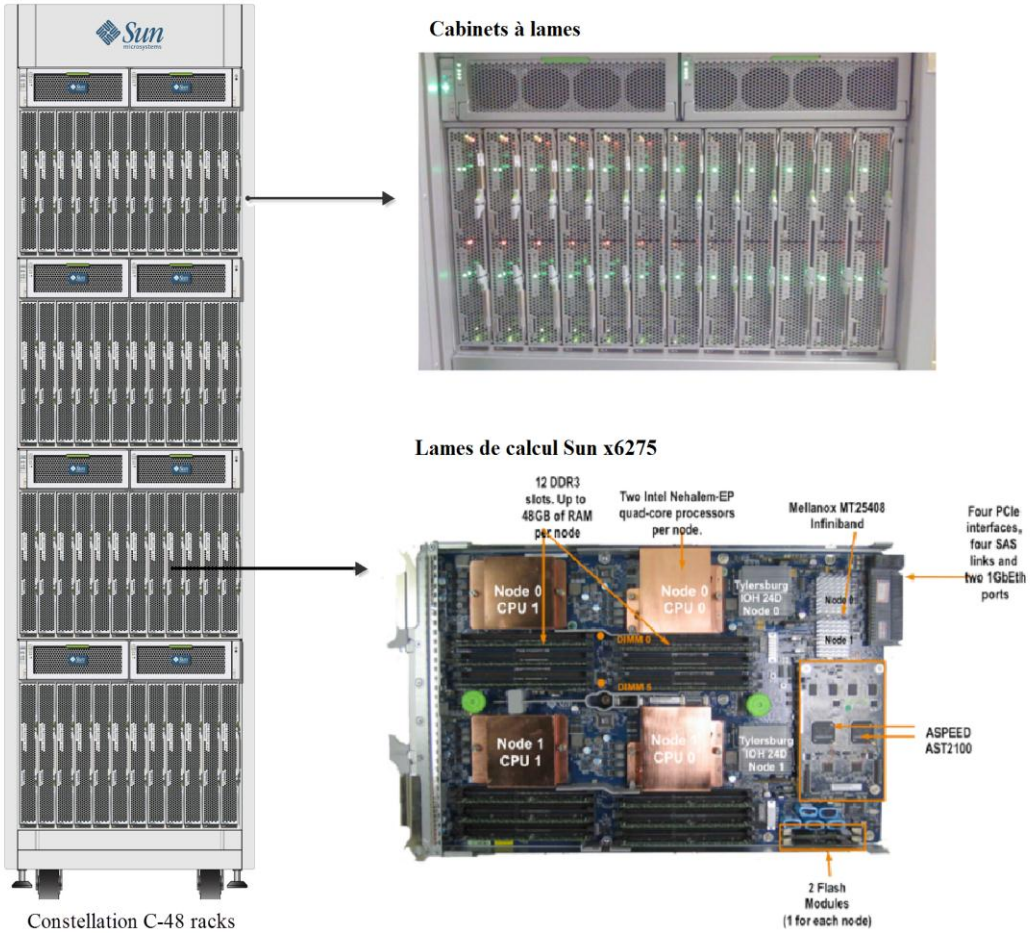


Figure 2.5 Colosse : Aperçu d'un cabinet de serveur Sun Constellation C-48, d'après [3]

L'autre particularité exceptionnelle de Colosse est son système de refroidissement installé principalement au sous-sol, sous le plancher du premier niveau. Comme illustré sur la figure 2.6, le noyau cylindrique de Colosse contient beaucoup de chaleurs produites par les cabinets de serveur de plusieurs étages. Cet air chaud est aspiré par les ventilateurs du système de refroidissement qui le fait passer dans une chambre de filtre, la chaleur est récupérée et puis transportée dans les tuyaux protégés pour chauffer le campus universitaire.

Autour de cette chambre de filtre, il y a des tuyaux conduisant l'eau froide (5°C) de l'Université Laval ou de l'extérieur. Ces tuyaux se lient directement au système de refroidissement. À son tour, ce système produit l'air froid et cela refroidit tout l'étage du sous-sol. Vu qu'il n'y a pas d'autre sortie dans l'étage du sous-sol, l'air froid remonte en refroidissant tous les étages séparés par des planchers de caillebotis qui ont été choisis pour cette raison. Grâce à cette architecture écologique et unique au monde, Colosse a été lauréat du prix *InfoWorld Green 15 en 2010.*

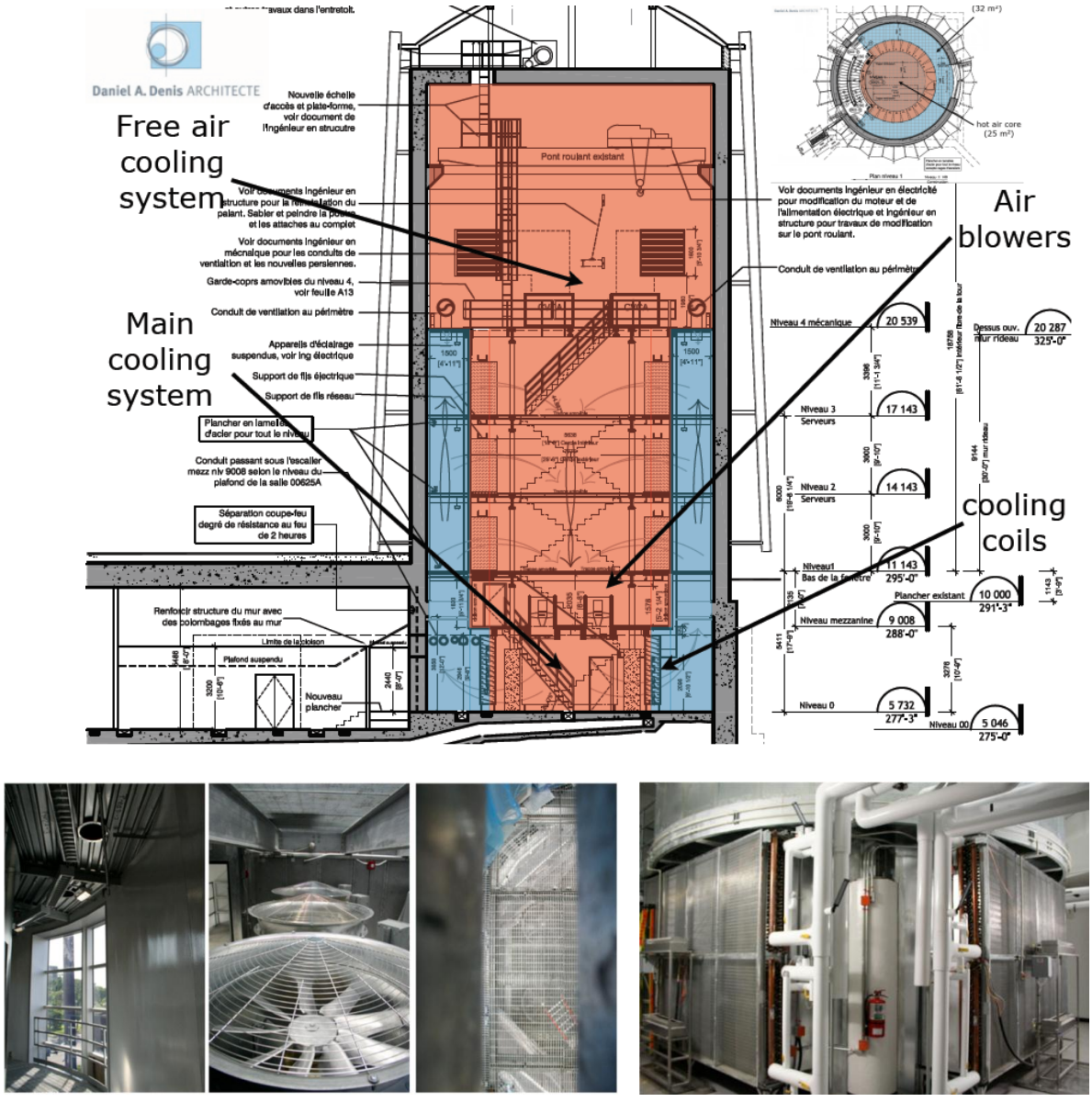


Figure 2.6 Colosse : Système de refroidissement écologique de Colosse, d'après [2]

2.2.3 Système de fichiers

Lustre est un système de fichiers parallèles distribués combinant les caractéristiques de Linux et Cluster. Il permet à plusieurs milliers de clients de travailler en même temps et de traiter plusieurs pétaoctets de données ainsi que des dizaines de gigaoctets d'entrée et de sortie par seconde. La vitesse d'écriture peut atteindre 17 gigaoctets/seconde, cela veut dire qu'en une seconde, Lustre peut écrire 4 DVD !

Ce système de fichiers est composé de trois unités principales, d'après [4] :

1. Un Meta Data Target (MDT) : l'unité qui se charge d'enregistrer les métadonnées contenant répertoires, noms de fichiers, permissions, etc. sur un serveur de métadonnées (Meta Data Server, MDS).
2. Un ou plusieurs Object Storage Server (OSS) qui permettent d'enregistrer le contenu des fichiers sur un ou plusieurs Object Storage Target (OST). OSS est configurable pour servir entre 2 et 8 OST.
3. Des clients qui se connectent au réseau pour chercher leurs données.

Toutes ces unités sont reliées au travers d'un réseau; Lustre supporte les réseaux de type Infiniband, TCP/IP en Ethernet, Myrinet, Quadrics ainsi que d'autres technologies propriétaires.

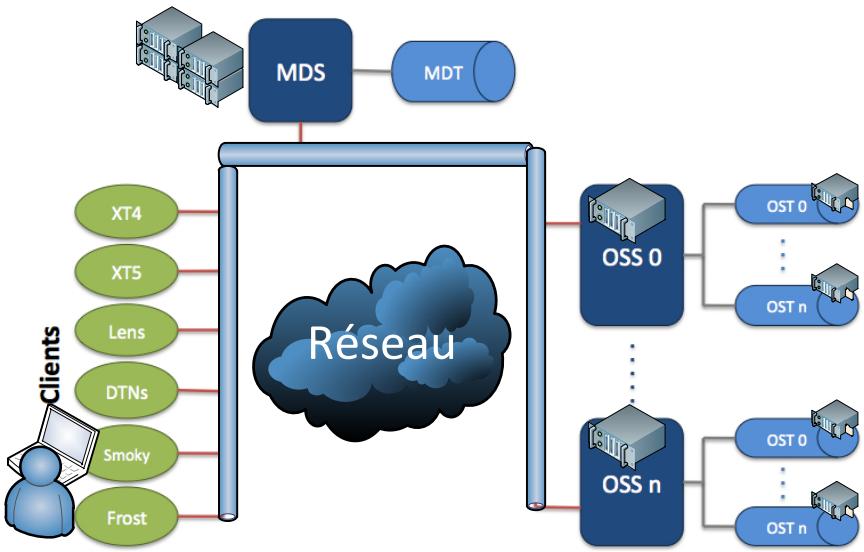


Figure 2.7 Colosse : Système de fichiers Lustre, d'après [4]

Dans le système de fichiers sur Colosse, Lustre a été configuré en 9 paires OSS et 2 MDS installés sur le serveur Sun Fire X4270 composé de 2 processeurs Intel Nehalem Quad-cœurs cadencés à 2.8GHz et 24GB Ram. En plus, chaque paire OSS est équipée de 4 aires de stockage Sun Storage J4400 disposant chacune de 24 disques durs SATA à 7200rpm configurés en RAID 6. Cette architecture est illustrée aux figures 2.7 et 2.8.

Le système de fichiers Lustre est puissant, mais il lui faut aussi un réseau robuste et dominant, c'est le réseau Infiniband de Colosse. Les 960 nœuds de calcul et 40 nœuds d'infrastructure sont reliés au système de fichiers Lustre par l'intermédiaire de 3 types de Switch.

Le premier (*QNEM IB leaf Switch*) est connecté à chaque groupe de cabinets de calcul (960 nœuds de calcul) renvoie les données au deuxième Switch (*M9 648 ports Core Switch*) qui permet l'interconnexion de l'ensemble des premiers Switch des nœuds de calcul. Ensuite, ces Core Switch distribuent les données au troisième Switch (*M2 leaf Switch 36 ports*) relié au système de fichiers Lustre comportant 40 nœuds d'infrastructure. L'ensemble de cette architecture explique la liaison entre les nœuds de calcul et le système de fichiers Lustre, il est illustré à la figure 2.8.

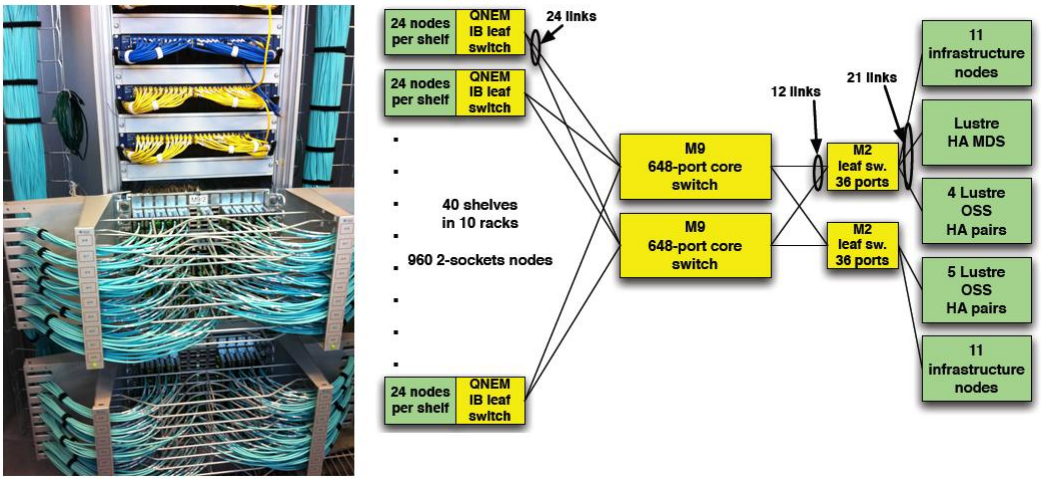
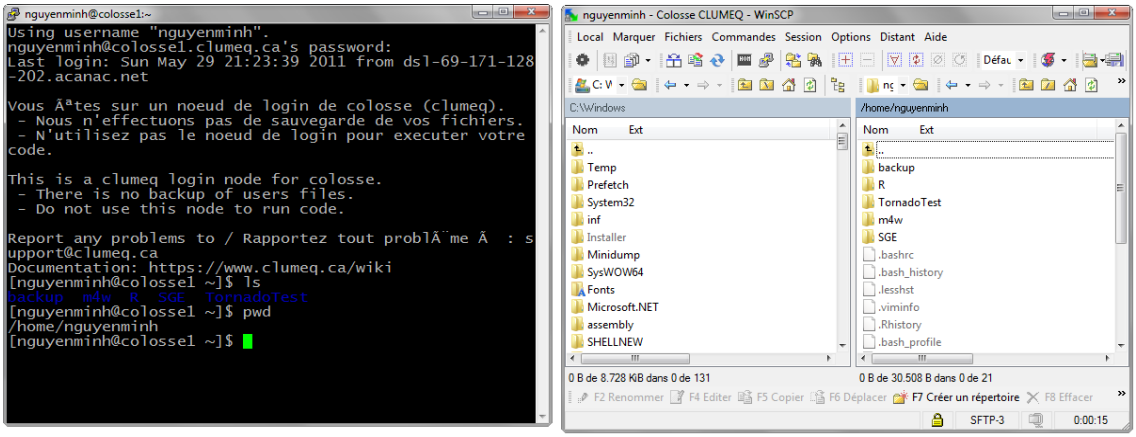


Figure 2.8 Colosse : Réseau Infiniband

2.2.4 Accès à Colosse

Nous pouvons accéder au Colosse depuis le réseau interne de l'Université Laval ou d'autres sites attachés au CLUMEQ comme les universités du Québec et également depuis Internet. Il y a 2 types d'accès distingués :

- L'accès à Colosse s'effectue via une connexion Secure Shell (SSH), on peut également manipuler les fichiers via les commandes de l'interface console.
- L'accès au système de fichiers s'effectue avec une connexion SFTP (SSH File Transfer Protocol), ce mode de connexion permet de télécharger des fichiers vers et de Colosse.



putty (Interface console) *WinSCP (Interface graphique)*

Figure 2.9 Colosse : Accès au Colosse via putty(SSH) ou WinSCP(SFTP)

Lorsque l'utilisateur se connecte au Colosse par SSH ou SFTP, il se connecte à un serveur intermédiaire Cyclops lui demandant l'authentification avant d'octroyer l'accès au Colosse *(voir chapitre 2.3.2)*. L'utilisateur peut travailler en interface console en utilisant PuttY par exemple pour exécuter différentes commandes directement sur le serveur s'il est connecté en SSH. Et il peut travailler en interface graphique utilisant le programme WinSCP par exemple s'il est connecté en SFTP. L'aperçu général de l'accès au Colosse est illustré aux figures 2.9 et 2.10.

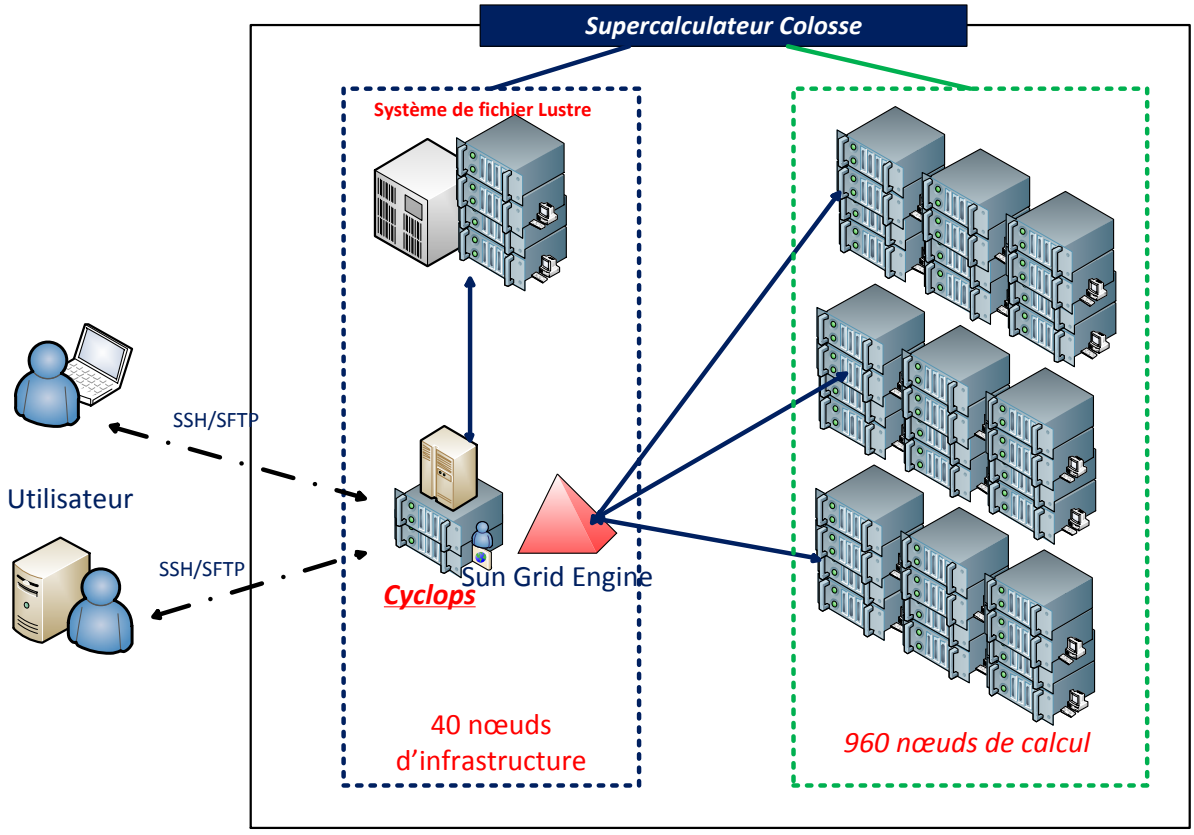


Figure 2.10 Colosse : Aperçu de l'architecture interne

Lorsque l'utilisateur est connecté au Colosse, il est redirigé vers son compte dont le chemin par défaut est */home/UserLogin.* Chaque utilisateur appartient à un groupe de recherche (projet). Pour connaître l'identifiant du groupe, il suffit d'entrer la commande *colosse-info*.

```
[user@colosse /]$ colosse-info
Your Rap ID is: yyk-770-aa
```

L'identifiant du groupe model*EAU* [*rap-id*] est : yyk-770-aa.

Chaque groupe dispose d'un espace de travail dans le système de fichiers du Colosse.

- /rap/*rap-id* : répertoire partagé du groupe où *rap-id* correspond à l'identifiant du groupe. L'espace disque initial est de 100 gigaoctets, extensible sur demande.
- /scratch/*rap-id* : répertoire partagé du groupe optimisé pour la lecture et l'écriture rapide de fichiers (stockage temporaire). Selon la politique de Colosse cet espace disque sera modifié de temps à autre.

L'utilisateur peut partager ses programmes en les mettant dans l'espace partagé du groupe.

2.3 Sun Grid Engine

2.3.1 Introduction

Sun Grid Engine (SGE) est un système de gestion des tâches de calcul sur Colosse. C'est SGE qui détermine quand et sur quels nœuds exécuter le calcul, quand arrêter les tâches en exécution selon les ressources allouées et les permissions de chaque utilisateur. Son rôle essentiel est de permettre à plusieurs utilisateurs simultanés d'avoir une utilisation efficace des ressources de Colosse, notamment les processeurs et mémoires. SGE permet aussi d'obtenir des informations sur les ressources utilisées, que ce soit pour l'ensemble des tâches, ou pour une tâche spécifique.

Tous les utilisateurs ayant accès au Colosse de CLUMEQ sont tenus à utiliser ce système d'ordonnancement pour soumettre leurs tâches de calcul. L'utilisation de SGE est donc obligatoire pour exploiter les ressources de Colosse. Il faut employer en tout le temps des commandes ***qsub*** (soumettre une tâche) et ***qstat*** (afficher la file d'attente et les informations des tâches).

SGE est un système logiciel d'origine Sun Microsystems mis à la disposition de la communauté Open Source. Les informations détaillées se trouvent à l'adresse suivante : http://gridengine.sunsource.net

2.3.2 Principes de fonctionnement

Nous sommes familiers avec des environnements graphiques interactifs où il y a des fenêtres, des menus, des boutons, etc. Mais l'environnement de Colosse est différent, nous travaillons en ligne de commande.

L'utilisateur se connecte au Colosse par une ou des connexions SSH/SFTP. Ensuite, il lui faut écrire le fichier de soumission qui décrit la tâche à exécuter et l'utilisateur soumet ce fichier à la machine Cyclops où SGE a été installé en entrant la commande ***qsub*** (*1-Submit*). Cyclops est l'ordinateur relié directement aux nœuds d'infrastructure Lustre de Colosse. Il est également le serveur d'authentification de Colosse et le système SGE a été installé sur ce dernier.

Les tâches demandées ne sont pas exécutées immédiatement, mais mises dans une file d'attente (*2-Schedule*). Et c'est seulement lorsque les ressources nécessaires sont disponibles que la tâche est mise en exécution (*3-Assign & 4-Execute*). Lorsque l'exécution de la tâche est terminée, SGE renvoie le résultat de l'exécution au système de fichiers Lustre dans le répertoire défini par l'utilisateur dans son fichier de soumission et nous allons y revenir pour récupérer les résultats (*5-Result*). Il n'y a donc pas d'interaction avec les programmes lorsque les tâches sont effectuées. Ce fonctionnement est illustré à la figure 2.11.

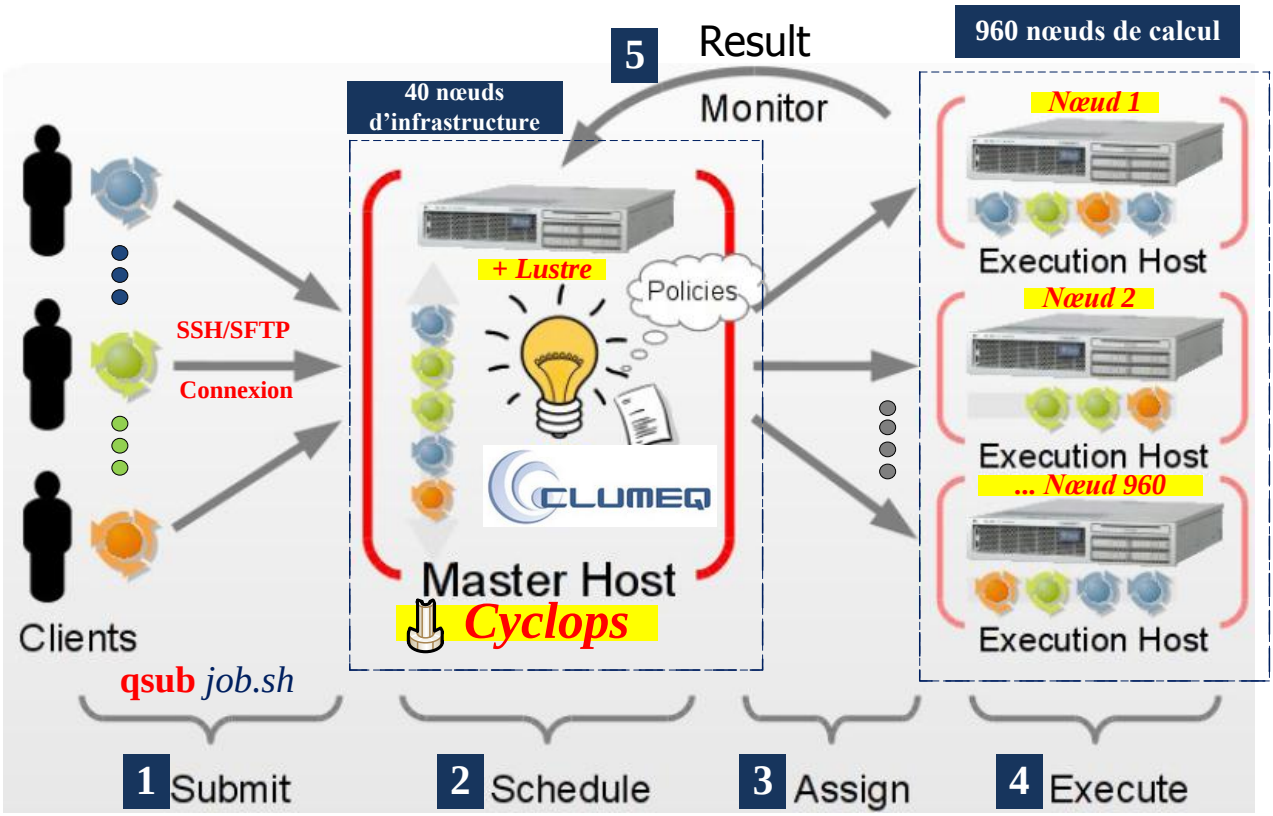


Figure 2.11 Colosse : Aperçu de l'ordonnanceur Sun Grid Engine, d'après [5]

Concernant la politique d'allocation de ressources du Colosse, afin de permettre aux centaines d'utilisateurs d'utiliser l'ordinateur en même temps, et ce dernier dispose de 960 nœuds de calcul et 40 nœuds d'infrastructure possédant chacun 8 cœurs *(voir chapitre 2.2)*, il faudrait que chaque utilisateur spécifie quelle fraction de l'ordinateur il a besoin. Cela se spécifie principalement par deux paramètres:

- Le temps d'exécution dont il a besoin pour compléter la tâche
- Le nombre de cœurs sur lesquels il veut exécuter la tâche.

Il est donc important de bien spécifier ces paramètres. S'ils sont trop élevés, on bloquera l'utilisation de la machine à d'autres utilisateurs qui pourraient en avoir besoin. S'ils sont trop petits, la tâche pourrait ne pas pouvoir s'exécuter au complet. Ce sont ces paramètres qui permettront à Colosse d'ordonnancer les tâches qui sont dans la file d'attente.

Pour l'année 2011, tous les utilisateurs de Colosse disposent d'une allocation de ressources initiale de 11 cœurs maximum. Si la tâche utilise plus de ressources que cette allocation, alors SGE va réduire la priorité de calcul et la placer dans une queue d'attente avec la priorité correspondante. Donc, la tâche risque d'attendre longtemps avant qu'elle soit exécutée.

Ainsi, si nos besoins en ressources de calcul sont effectivement plus grands que celles prévues et cela risque d'entrainer des problèmes de temps d'attente. Alors, la solution serait de faire une demande de ressources avec toutes les justifications nécessaires à envoyer au CLUMEQ.

Quant aux environnements d'exécution parallèle ou séquentielle, il y en a 2 types :

- Distributed Memory Parallel (DMP) : la tâche peut être composée de plusieurs processus ayant chacun leur espace de mémoire qui communiquent entre eux à l'aide de messages. Un exemple de ce type de programme est MPI (Message Passing Interface) qui permet d'exploiter un très grand nombre de cœurs sur l'ordinateur parallèle.

- Shared Memory Parallel (SMP) : la tâche peut être composée de plusieurs processus qui partagent une même zone de mémoire pour l'exécution parallèle, alors le programme est en mémoire partagée. Cependant, cela ne peut utiliser qu'un seul nœud de calcul, soit 8 cœurs.

Les programmes séquentiels utilisant un seul processeur peuvent également s'exécuter sur Colosse. Mais ce type de programme peut s'exécuter sur un ordinateur puissant qui fera aussi bien le travail, toutefois Colosse sera utile si on a besoin d'exécuter ce programme séquentiel un très grand nombre de fois.

2.3.3 Modules

Une série de logiciels a été installée sur Colosse sous forme de module (brique logicielle) pour charger ou décharger dans l'environnement de travail de l'utilisateur. Cette architecture modulaire permet d'avoir plusieurs versions d'une même application installées sans que celles-ci entrent en conflit afin de répondre aux besoins des utilisateurs.

Ces modules sont regroupés principalement en 5 catégories distinctes :

- Applications : Abaqus, CP2K, Flex, Geant, Octave, Python, ***R***, Wine, etc.
- Librairies : ATLAS, GotoBLAS2, Lapack, MKL, Java, etc.
- Compilateurs : **GCC**, Intel c++, Mono, Sun Studio.
- Environnement MPI : OpenMPI pour GCC, Intel et Sun Studio.
- Outils de développement: GDB, PDT, TAU

En fonction du besoin de l'utilisateur ou groupe de recherche, l'usager pourrait introduire une demande justifiée à CLUMEQ pour l'installation éventuelle des nouveaux modules.

Le tableau 2.1 reprend les commandes essentielles en rapport avec l'utilisation de modules.

Table 2.1 Colosse : Utilisation de modules

Commande	***Action***
module avail	Lister les modules disponibles
module show <NOM MODULE>	Afficher les informations sur un module
module load < NOM MODULE>	Charger un module dynamiquement
module unload < NOM MODULE>	Décharger un module
module purge	Décharger tous les modules

2.3.4 Utilisation

Pour exécuter une ou des tâche(s) sur Colosse, il faut passer par 2 étapes :

- Ecrire un fichier/script contenant la description de la tâche en spécifiant le nombre de cœurs (multiple de 8) et le temps d'exécution estimé pour cette tâche.
- Soumettre ce fichier au SGE via la commande ***qsub***

La tâche sera mise en file d'attente et lorsque SGE dispose des ressources nécessaires, la tâche sera mise en exécution. SGE va charger les modules requis (compilateurs, librairies, etc.) sur les nœuds de calcul alloués à cette tâche et se charge de l'exécution. Quand la tâche s'est terminée, SGE va renvoyer les résultats de l'exécution au système de fichiers Lustre, dans le répertoire spécifique de l'utilisateur.

Première étape : Le fichier de soumission et ses options

Le fichier de soumission de tâche est un script Shell contenant la description de la tâche à exécuter; il spécifie les différents paramètres obligatoires et les options que SGE doit prendre en compte lors de l'exécution de celle-ci. Dans ce fichier, chaque ligne est commencée par les caractères #$, et les options par le caractère – (tiret).

On pourrait aussi soumettre une tâche avec les options en ligne de commande, mais le fait de les écrire dans un fichier permet de garder un historique des options choisies pour vérifier ou se rappeler plus tard. La table 2.2 reprend les options essentielles à spécifier dans le fichier.

Table 2.2 Colosse : Options du fichier de soumission

Option	***Action***	***Exemple***	***Obligatoire***
N	Nom de la tâche	-N MyJob	Oui
P	rap-id du groupe/projet	-P xxx-yyy-zz	Oui
pe	Environnement parallèle	-pe default 32	Oui
l h_rt	Limite de temps d'exécution	-l h_rt=00:13:05	Oui
cwd	Exécuter la tâche depuis le répertoire courant	-cwd	Non
S	Type de Shell à utiliser	-S /bin/bash	Non
o	Nom du fichier de la sortie stdout	-o $HOME/sge/MyJob.out	Non
e	Nom du fichier de la sortie stderr	-e $HOME/sge/MyJob.err	Non
trap "" USR2	Notification de fin de tâche, à écrire à la fin du fichier		Non
-t interval:step	Définir des tâches par lot, SGE va créer une tâche avec autant de sous-tâches définies dans l'intervalle	-t 1-10:1	Non

Dans l'option –pe, on a utilisé l'environnement parallèle '*default'*, il y a 2 types de queue, SGE va choisir laquelle disponible dont les paramètres sont compatibles avec ceux de la tâche soumise sera utilisée :

- Pour les tâches moins de 24h, la queue 'short' sera utilisée et une tâche peut demander au maximum 256 cœurs.
- Pour les tâches plus de 24h, la queue 'med' sera utilisée et une tâche peut demander au maximum 128 cœurs.

Au cas où la tâche prend plus de 7 jours, la queue 'long' sera utilisée dans un autre environnement parallèle que '*default'*, cette tâche est à demander au CLUMEQ. Il y a un autre environnement comme '*test'* pour les tâches demandant 16 cœurs au maximum pour une durée maximale de 15 minutes.

Il est également possible de spécifier les différents paramètres dans ce fichier, tels que les variables d'environnement, les chargements de modules nécessaires pour l'exécution de la tâche, des conditions pour lancer une deuxième tâche éventuelle, etc.

Une des options intéressantes est la variable d'environnement $SGE_TASK_ID permettant de définir un nombre de N fois la tâche en parallèle ce qui nécessite N entrées et produit N sorties différentes (si applicable). A noter que SGE va créer les fichiers de sortie et d'erreur de l'exécution de la tâche. Ces fichiers ont la syntaxe suivante :

<NOM_DE_LA_TÂCHE>.[TYPE]<IDENTIFIANT_DE_LA_TÂCHE>

[TYPE] est le type du fichier à la sortie et prend la valeur 'e' et 'pe' pour '*error*' et la valeur 'o' et 'po' pour '*output*'.

Voici un exemple de son utilisation :

Fichier MyJobArrays.sh

```
#!/bin/bash
#$ -N JobArrays
#$ -P abc-123-yz
#$ -pe default 32
#$ -l h_rt=00:31:07
#$ -t 1-86:1

/chemin/vers/mon/programme chemin/vers/fichiers/de/donnees/Job_$SGE_TASK_ID.src
# Cela donne à l'exécution des sous tâches en parallèle
# programme  Job_1.src
# programme  Job_2.src
# programme  Job_3.src
# programme  Job_4.src
# ……..
# programme  Job_86.src
```

Figure 2.12 Colosse : Exemple du fichier de soumission au SGE

Dans cet exemple, la tâche est nommée *JobArrays*, et sera exécutée par un membre du groupe 'abc-123-yz'. Son exécution nécessite 32 cœurs pour une durée totale de 31 minutes et 7 secondes. SGE va choisir un ou des nœuds de calcul pour avoir 32 cœurs disponible afin de les allouer à cette tâche. Il exécute ensuite les tâches de 1 à 86 en parallèle. SGE va également créer les fichiers suivants en sachant que '31071986' est l'identifiant de cette tâche (par exemple):

- Les fichiers de sortie: *JobArrays.o31071986* et *JobArrays.po31071986*.
- Les fichiers d'erreur *JobArrays.e31071986* et *JobArrays.pe31071986*.

Les différents exemples de fichier de soumission se trouvent à l'annexe A.

Seconde étape : Soumettre le fichier au SGE

Lorsqu'on a le fichier de soumission, on peut le soumettre au SGE à l'aide de la commande ***qsub*** :

```
[user@colosse ~]$ qsub  <NOM_DU_FICHIER>
```

La tâche sera alors placée dans la file d'attente, sa priorité dépendra du nombre de cœurs demandés et du temps d'exécution estimé. L'utilisateur peut suivre l'évolution de la file d'attente à l'aide de la commande ***qstat*** qui affiche les tâches placées dans la file d'attente. Plusieurs options sont disponibles pour avoir plus de détails.

Table 2.3 Colosse : Options de la commande qstat

Option	*Description*
-ext	Donner une liste plus détaillée des tâches
-g c	Afficher la liste des queues, ainsi que les ressources disponibles et utilisées sur ces queues
-help	Afficher toutes les options disponibles
-j job_id	Afficher les ressources utilisées par les tâches
-u utilisateur	Afficher les tâches de l'utilisateur spécifique
-u ""*	Afficher toutes les tâches dans SGE
-f	Afficher tous les paramètres d'exécution pour toutes les tâches

L'utilisateur peut supprimer une tâche envoyée au SGE par la commande ***qdel*** avec l'identifiant de la tâche en paramètre:

```
[user@colosse ~]$ qdel  <IDENTIFIANT_DE_LA_TÂCHE>
```

Par ailleurs, pour utiliser les résultats de la tâche sur l'ordinateur local, l'utilisateur peut se connecter au Colosse en SFTP pour récupérer ces résultats à l'endroit où il l'aura spécifié.

Quelques commandes utiles sur Colosse

Table 2.4 Colosse : Commandes utiles sur Colosse

Commande	*Action*
mkdir <Dossier>	Créer un nouveau dossier vide
touch <Fichier>	Créer un nouveau fichier vide
rm –r <Dossier>	Supprimer entièrement le dossier et ses sous dossiers
rm <Fichier>	Supprimer le fichier spécifique
cp/mv source destination	Copier un fichier ou déplacer un fichier/dossier
cp –r source destination	Copier récursivement un répertoire
ls –alth	Lister tout le contenu (caché et les informations détaillées) trié sur date de création
cd <Dossier>/ cd ..	Déplacer dans un dossier ou remonter au dossier parent avec 'cd ..'
chmod a+x <Fichier>	Rendre le fichier exécutable
nano/vim <Fichier>	Visualiser et Éditer le fichier
man <Commande>	Demander le manuel de référence de la commande en question

3
La plateforme Tornado

Tornado est un logiciel générique de simulation construit pour la modélisation et l'expérimentation virtuelle avec les systèmes environnementaux complexes. Il a été développé par la société MOSTforWATER en collaboration avec BIOMATH (Department of Applied Mathematics, Biometrics and Process Control) de l'Université de Gand, Belgique. Filip Claeys, Docteur en Sciences Appliquées Biologiques en Belgique est le concepteur ainsi que l'initiateur de Tornado. Actuellement il est le chef de l'équipe de développement de WEST. La société se situe à Courtrai, Belgique. Tornado a été intégré dans la suite de logiciels WEST qui est un système de modélisation flexible et performant pour les traitements des eaux usées.

Aujourd'hui, WEST appartient à la compagnie DHI Group suite à une transaction avec MOSTforWATER dont les produits sont classés dans la chaine de produit de Mike de DHI Software et *WEST* (mikebydhi.com) est donc maintenant devenu un produit de MIKE.

Ce chapitre va expliquer l'architecture générale et les principes de fonctionnement ainsi que l'utilisation de Tornado dont la compréhension est essentielle avant d'envisager son déploiement dans le chapitre 5. La version du Tornado utilisée dans ce projet est Tornado 0.43.

3.1 Architecture

Tornado est une plateforme de simulation générique fonctionnant sous Windows et Linux. Il permet de créer et exécuter les différentes expériences virtuelles en se basant sur les modèles et fichiers de description. WEST utilise Tornado comme l'exécuteur des simulations. Tornado est construit par une architecture modulaire de différents modules et de logiciels sur lesquels le fonctionnement de Tornado est basé.

Chaque module doit être compilé pour construire les librairies et l'ensemble des modules vont structurer une bibliothèque de librairies qui sont nécessaires pour le fonctionnement du module principal Tornado. La figure 3.1 représente l'architecture et les dépendances entre les différents modules de Tornado d'après [6].

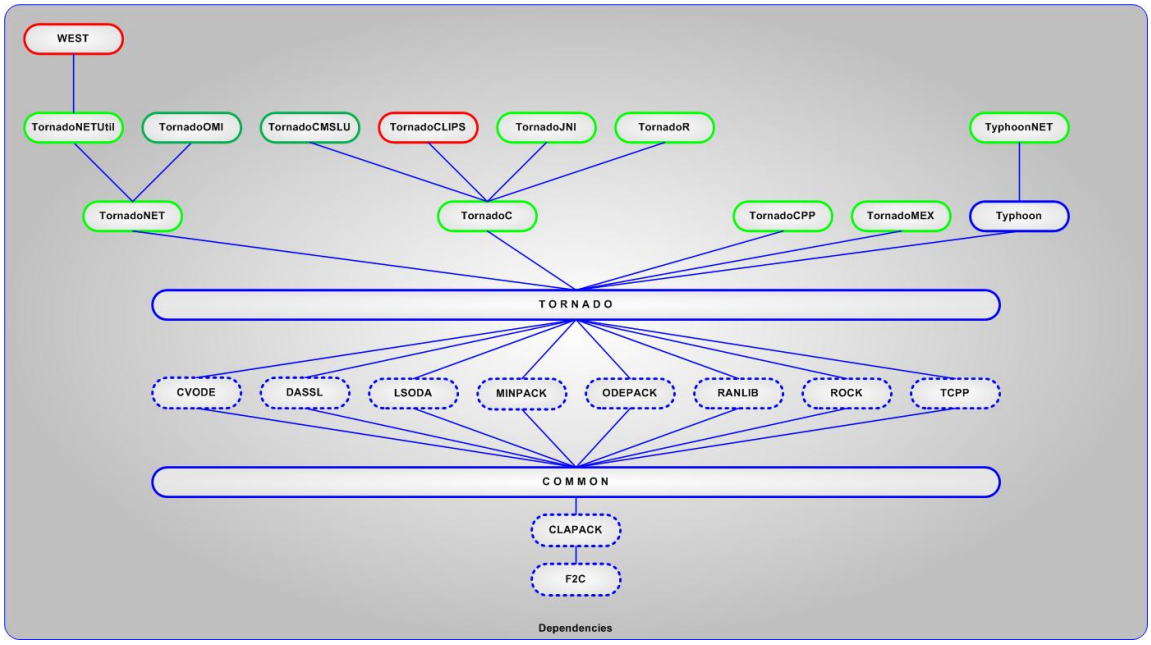


Figure 3.1 Tornado : Architecture de Tornado, d'après [6]

Par la suite, nous allons voir une brève description et l'utilité des modules nécessaires au fonctionnement de Tornado. Nous allons parcourir la figure 3.1 du bas vers le haut.

3.1.1 F2C

F2C est l'acronyme de Fortran To C étant un programme utilisé pour convertir du code Fortran en C. Ce programme est nécessaire pour le module CLAPACK. Le code source de F2C se trouve à l'adresse suivante : http://netlib.org/f2c/

3.1.2 CLAPACK

CLAPACK est une interface de programmation utilisée à établir une liaison entre le langage C et la bibliothèque LAPACK écrite en Fortran. L'ensemble CLAPACK est une bibliothèque fournissant des fonctions de l'algèbre linéaire numérique en langage C. Son code source se trouve à : http://netlib.org/clapack/

3.1.3 Common

Common est un module contenant l'ensemble d'utilitaires nécessaires au fonctionnement de Tornado tels que : Outils de cryptage et décryptage, Vecteurs, Manipulation de fichiers, etc…

3.1.4 Les solveurs

Un ensemble de solveurs a été implémenté dans Tornado pour les analyses environnementales tels que :

- CVODE : le module de solveur intégral pour les équations différentielles
- DASSL : le module de solveur intégral pour les équations différentielles algébriques
- LSODA : le module de solveur intégral pour les équations différentielles ordinaires.
- ROCK : le module de solveur intégral.

3.1.5 Les autres modules

MINPACK : ce module en FORTRAN permet de la résolution de systèmes d'équations non-linéaires.

ODEPACK : ce module également en FORTRAN permet de la résolution des équations différentielles ordinaires.

RANLIB : outil de génération de nombres aléatoires.

TCPP : ce module de préprocesseurs est utilisé pour la compilation des modèles.

3.1.6 Le noyau Tornado

Le module Tornado est le cœur, le pivot de la plateforme Tornado. Il contient un ensemble d'utilitaires appelant les fonctionnalités offertes par les autres modules. Il contient les fonctions noyau de Tornado, ainsi que la gestion de licence et les différentes librairies permettant des utilisations polyvalentes avec les fonctions noyau ou via les interfaces.

3.1.7 Les interfaces

Au-dessus du noyau Tornado, plusieurs interfaces ont été implémentées pour accéder à Tornado et rendre son utilisation possible à partir de différents environnements, telles que les applications en .NET, Matlab, R, etc. Ces différentes interfaces se basent sur le noyau générique écrit en C++ de Tornado. Elles sont présentées à la figure 3.2.

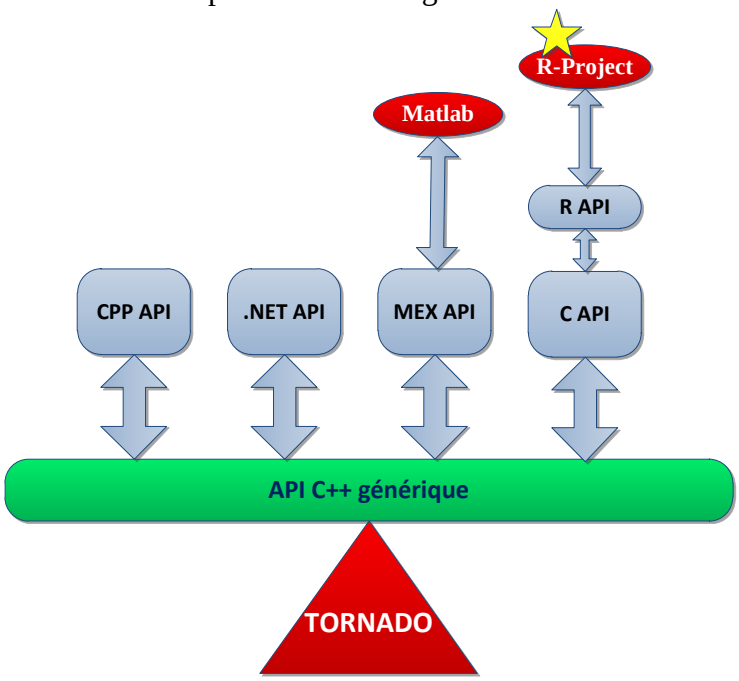


Figure 3.2 Tornado : Utilisation via des interfaces

La figure 3.2 représente un autre aperçu de dépendances entre le noyau Tornado et ses différentes interfaces par rapport à la figure 3.1.

Avec cette architecture, un utilisateur qui est familier avec son langage de programmation (.NET/Java/Matlab…) pourrait utiliser Tornado en travaillant avec l'interface correspondante de son environnement. Ce dernier va appeler les fonctions de Tornado et rendre l'utilisation totalement transparente pour l'utilisateur.

Dans ce projet, le langage R est utilisé, et donc, à partir d'un programme écrit en R on exécute une simulation d'expérience sur Tornado au travers de l'interface TornadoR. Cette interface contient les fonctions appelant les fonctions de l'interface TornadoC qui à son tour va demander les fonctions de Tornado.

Les différentes interfaces/modules se basent sur le noyau Tornado sont :

- TornadoNET : pour les utilisateurs dans l'environnement .NET. La suite de logiciels WEST se base sur cette interface.
- TornadoC : pour les utilisateurs travaillent en Java, R, CLIPS et CMSLU.
- TornadoCPP : destiné aux utilisateurs travaillant en C++.
- TornadoMEX : destiné aux usagers de Matlab.
- Typhoon : un système de tâches pour Tornado, conçu pour être utilisé sur un Cluster (un ensemble de nœuds de calcul). Typhoon n'a pas été utilisé dans Colosse car il faudrait installer Typhoon dans tous les nœuds de Colosse afin d'exploiter ces ressources. Cela n'est pas permis par Colosse avec son ordonnanceur Sun Grid Engine.

3.1.8 Gestion de licence

Tornado n'est pas gratuit. L'utilisateur doit posséder une licence de Tornado afin de le faire fonctionner. Il y a 2 méthodes pour déployer la licence de Tornado :

- L'utilisation d'un fichier de Licence propre à la machine où Tornado a été installé. L'utilisateur devrait faire une demande et l'envoyer à support@mostforwater.com contenant l'adresse MAC et le nom d'hôte de la machine sur laquelle Tornado sera utilisé. Et il faut le mettre le fichier de licence dans le répertoire suivant : *%TORNADO_ROOT_PATH%\etc\Tornado.lic.<hostname>* où TORNADO_ROOT_PATH est une variable d'environnement qui définit l'endroit d'installation de Tornado.

- L'utilisation d'un serveur de licences (principe de serveur à jetons). Le nombre d'instances de Tornado pouvant être exécutées simultanément est limité selon le contrat de service.

Dans le cadre de déploiement de Tornado sur Colosse, l'équipe model*EAU* a obtenu une licence sous fichier pour le supercalculateur Colosse.

3.2 Fonctionnement

Tornado est un logiciel générique de pointe pour les différentes 'expérimentations virtuelles' ce qui veut dire que Tornado offre une variété de types de traitements qui sont basés sur les modèles d'évaluation, tels que Simulation, Optimisation, Analyse de l'incertitude et Analyse des risques. Les modèles sont écrits dans les langages de modélisation orientés objet tels que MSL et Modelica.

À titre d'exemple, la figure 3.3 ci-dessous explique de manière schématique la procédure pour convertir un modèle écrit en langage Modelica en fichier de simulation d'expérience à exécuter ensuite par l'exécuteur Tornado en appelant les fonctions du noyau Tornado.

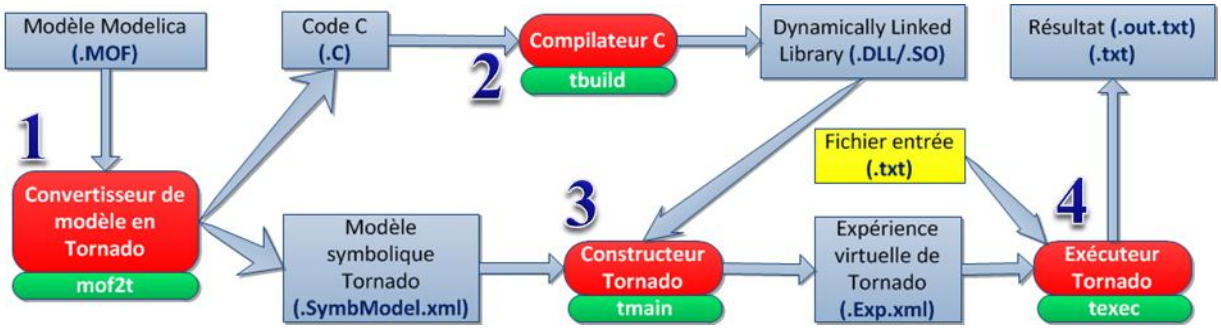


Figure 3.3 Tornado : Création des expériences virtuelles du modèle Modelica

Étape 1 : À partir d'un modèle Modelica avec l'extension MOF, l'utilisateur appel le convertisseur de MOF2T (Modelica to Tornado) à l'aide de la commande ***mof2t***, cela va générer un fichier *Exécutable* code C (l'extension *.C*) qui a été traduit du modèle Modelica et un fichier contenant le modèle symbolique Tornado (l'extension *.SymbModel.xml*)

Étape 2 : L'utilisateur appelle le compilateur C (la commande ***tbuild***) qui va compiler le Code C et produire une librairie dynamique (Dynamically-linked Library) dont l'extension est *.DLL* sous Windows et *.SO* sous Linux. Dans le cas de Colosse opérationnel sous Linux, nous aurons donc le fichier *.SO*.

Étape 3 : L'utilisateur appel le constructeur de Tornado (la commande ***tmain***) avec l'option *ExpCreateSimul* pour créer une simulation d'expérience virtuelle Tornado à partir de librairie *(.DLL/.SO)* et le modèle symbolique Tornado *(.SymbModel.xml)*. Le fichier à la sortie décrit la structure et les informations de l'expérience à exécuter en représentation XML. Son extension est donc *Simul.Exp.xml* (Simulation d'expérience virtuelle)

Étape 4 : Pour exécuter l'expérience virtuelle, l'utilisateur appelle l'exécuteur de Tornado (la commande ***texec***) pour charger le fichier XML de l'expérience virtuelle et le fichier d'entrée contenant les données comme la qualité de l'eau, la météo...(facultatif) afin d'exécuter et générer un fichier contenant le résultat de la simulation dont l'extension générale est *out.txt* et *Simul.out.txt* pour la simulation d'expérience.

3.3 Installation

Tornado fonctionne sous Windows 32/64 bits et Linux. Un mécanisme d'installation directe de Tornado pour Windows est disponible (fichier MSI, Microsoft Installer) et Tornado a été installé et testé sous Windows XP, Windows Vista et Windows 7. L'interface graphique de Tornado sous Windows est montrée à la figure 3.4.

Ainsi sous Linux, l'installation de Tornado se fait à l'aide d'un script contenant toutes les compilations nécessaires et puis l'utilisateur devrait exécuter ce script avec un shell sh. Tornado a été installé avec succès sur plusieurs distributions de Linux tels que OpenSUSE, Ubuntu, Xubuntu, et Redhat (Colosse). L'utilisateur sous Linux travaille avec Colosse en ligne de commande.

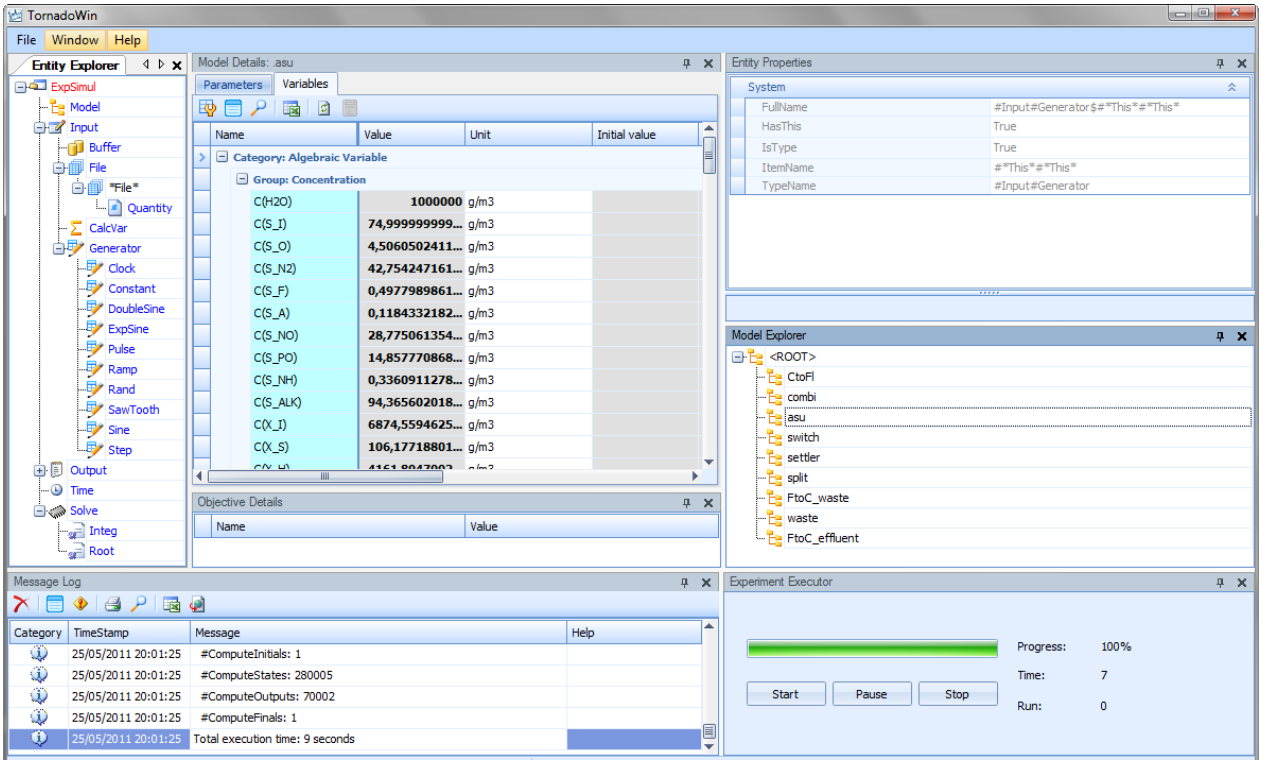


Figure 3.4 Tornado : TornadoWin

3.4 Utilisation

Tornado fonctionne sur base de modèles comme expliqué précédemment. Autrement dit, Tornado permet la création et l'exécution des expériences virtuelles à partir de ces modèles en appelant les fonctions de Tornado.

Il y a 2 manières d'utiliser les fonctions de Tornado, soit :

- Par des interfaces appelant les fonctions du noyau Tornado : à partir d'une des interfaces décrites dans le chapitre 3.1.7, l'utilisateur utilise cette interface pour accéder aux fonctions du noyau Tornado. Une version de Tornado avec l'interface graphique est disponible sous Windows. Elle est développée en langage .NET et s'est basée sur l'interface TornadoNET. Son interface est présentée à la figure 3.4.

- Par les fonctions du noyau Tornado : comme décrit dans la figure 3.3, la procédure pour convertir d'un modèle Modelica à l'expérience virtuelle nécessite des commandes pour passer et générer les fichiers au fil des étapes. Ces commandes sont reprises à la table 3.1

Table 3.1 Tornado : Convertir un modèle Modelica au Tornado Simulation

Commande	***Action***
mof2t ***Project****.mof*	Convertir le modèle Modelica en C et générer le modèle symbolique
tbuild ***Project***	Créer la librairie dynamique à l'aide du compilateur C
tmain ExpCreateSimul ***Project*** *. false*	Créer une simulation d'expérience virtuelle
texec ***Project****.Simul.Exp.xml*	Exécuter cette simulation

Afin de simplifier la procédure de conversion pour l'utilisateur, au lieu de taper 4 commandes l'une après l'autre, l'utilisateur pourrait employer la commande ***mof2simul*** qui est un simple script reprenant ces 4 commandes. Cela permettrait de faire la conversion en une seule commande. Par ailleurs, un autre script ***west2t*** a été développé, il permet de convertir un modèle constitué par les fichiers WCO et WXP (format du logiciel WEST) en simulation d'expérience virtuelle. Ces scripts sont ordinaires et se situent dans le répertoire partagé du groupe model*EAU* sur Colosse ***/rap/yyk-770-aa/bin/*** et le code source se trouve dans l'annexe B.

Il est à noter que chaque commande de Tornado utilise certain paramètre obligatoire et facultatif. Afin de savoir quels paramètres s'appliquent à chaque commande, l'option **–h** peut être utilisée pour obtenir de l'aide; par exemple :

```
texec -h
Tornado Experiment Executor (Version: 0.43)
Usage: texec [options] <XMLExpFile>+
Options:
-h,   --help                          Show this message.
-c,   --config <XMLMainFile>          Specify Tornado main spec file name.
-l,   --log <File>                    Specify log file name (use - for stdout).
-i,   --initial <String>              Specify initial values (separated by ;).
-sr,  --show_run                      Show run number.
-q,   --quiet                         Quiet mode.
…..
```

4
Le langage R

R-Project est un langage de programmation, simple et puissant. Il est de plus en plus utilisé par les étudiants, les professeurs, les chercheurs, …dans tous les domaines. Dans ce travail de fin d'études, il a été demandé de programmer en langage R et le but de ce chapitre est de présenter les caractéristiques générales et le fonctionnement de R en se basant sur [8] et [9].

4.1 Introduction

R est un système d'analyse statistique et graphique, disponible gratuitement sous les termes de la GNU (General Public Licence) pour toutes les plateformes telles que Windows, Linux, MacOS. R a été créé par Ross Ihaka et Robert Gentleman à *l'University of Auckland*. R est un dialecte du langage S développé par John Chambers et ses collègues d'*AT&T Bell Laboratories*. Le langage S est disponible sous la forme du logiciel SPLUS commercialisé par la compagnie Insightful. Ensuite, R est devenu un logiciel libre et gratuit en 1995.

R est à la fois un langage de programmation et un logiciel de fonctions statistiques. La version de base de R contient déjà un grand nombre de fonctions statistiques et graphiques permettant, par exemple de calculer une somme, une moyenne, ou une variance ou encore de tracer un histogramme avec des données importées d'un fichier texte. R a aussi la possibilité d'exécuter des programmes stockés dans des fichiers textes. En effet, R possède :

- Un système efficace de manipulation et de stockage des données.
- Différents opérateurs pour le calcul sur tableaux.
- Plusieurs outils pour l'analyse de données et les méthodes statistiques.
- Plusieurs outils de création de graphiques pour visualiser les données.
- Un langage de programmation simple et performant comportant : conditions, boucles, expressions, moyens d'entrées et de sorties, les fonctions ordinaires et la possibilité de définir des fonctions récursives.
- ….

Plusieurs chercheurs et utilisateurs ont développé au cours des années des fonctions plus avancées qui sont disponibles à tous les utilisateurs de R. Ces fonctions sont regroupées en bibliothèques (packages/librairies) qui sont disponibles pour téléchargement sur le site du projet R : http://www.r-project.org.

Grâce à sa grande flexibilité et du fait qu'il soit multiplateforme et gratuit, R est devenu l'un des principaux programmes de calcul statistique utilisé par les chercheurs dans tous les domaines.

4.2 Installation

Dans le cas de l'installation de R sur un ordinateur ordinaire, voici les étapes à suivre :

- Aller sur l'internet à l'adresse : http://cran.r-project.org/
- Choisir la plateforme de votre système dans 'Download and Install R'
  - Windows : 2 sous-dossiers sont à votre choix, ***'base'*** contient la dernière version de R, ***'contrib'*** contient les différentes librairies contribuées par la communauté d'utilisateurs R et les anciennes versions de R.
  - Linux : choisir votre distribution de linux (Debian, Redhat, Suse, Ubuntu) et suivre les instructions pour installer R.
  - MacOS : Télécharger le fichier .pkg et suivre les instructions pour installer R.

La version R utilisée dans le cadre de ce travail est la 2.12.1 fonctionne sous Windows. Après d'avoir installé cette version sur l'ordinateur, il y a un raccourci *R [la version]* pour lancer le logiciel R.

Sur Colosse, le logiciel R version 2.10.1 a été déjà installé sous forme de module à charger dans l'environnement de l'utilisateur (*voir chapitre 2.3.3*). Pour utiliser R, il faudra que l'utilisateur charge ce module en tapant les commandes suivantes :

```
module load compilers/gcc/4.4.2
module load apps/r-2.10.1
R
```

Le compilateur Gcc est nécessaire pour le fonctionnement de R car le noyau de R est écrit en langage machine interprété qui a une syntaxe similaire au langage C mais le système lui-même est écrit en R. Toutefois, il faut avoir un compilateur comme Gcc pour faire fonctionner R sur Colosse. Après le chargement de 2 modules indispensables, il faut taper **R** pour lancer le logiciel R.

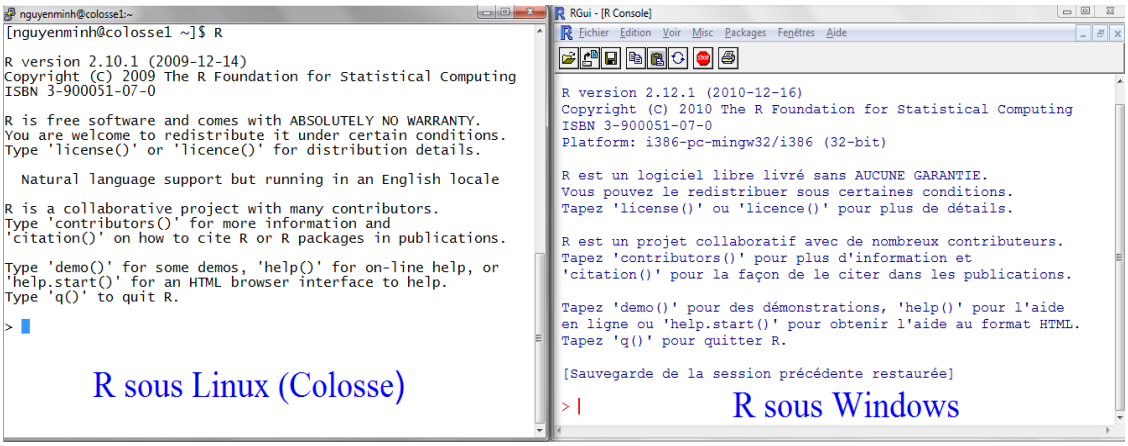


Figure 4.1 R : Environnement de travail

Nous constatons que les interfaces du logiciel R sous Windows ou Linux sont similaires (figure 4.1). Dans la fenêtre de commande, le symbole '>' signifie l'attente de commandes, alors R est prêt à exécuter les commandes. Un programme écrit sur une plateforme (Windows) est portable et fonctionnel sur une autre plateforme (Linux, MacOS) à condition que toutes les librairies utilisées dans le programme exporté soient installées et compatibles sur l'autre plateforme.

Par ailleurs, plusieurs manuels sont distribués ensemble avec l'installation de R dans le répertoire **R_HOME/doc/manual/** (R_HOME est le répertoire où R a été installé) permettent aux utilisateurs d'apprendre et pratiquer pas à pas ce langage.

4.3 Fonctionnement

D'abord, R est un langage interprété et non compilé, ce qui veut dire que les commandes tapées dans la fenêtre de commandes sont exécutées sans qu'il ait besoin de créer un programme complet comme d'autres langages de programmation tels qu'Assembleur, C, Java, etc.

Dans l'environnement de R, les variables, les données, les fonctions, etc. sont stockées dans la mémoire de l'ordinateur sous forme d'objets qui ont chacun un nom. L'utilisateur va travailler sur ces objets avec des opérateurs (arithmétiques, logiques, de comparaison, …) et des fonctions (qui sont également des objets). Une fonction en R s'écrit toujours avec les parenthèses pour être exécutée, même s'il n'y a rien entre les parenthèses (exemple **q()** pour quitter l'application). Et si l'utilisateur tape le nom de la fonction sans parenthèses, R va afficher des instructions de cette fonction. Une petite description pour comprendre le fonctionnement d'une fonction en R est représentée à la figure 4.2.

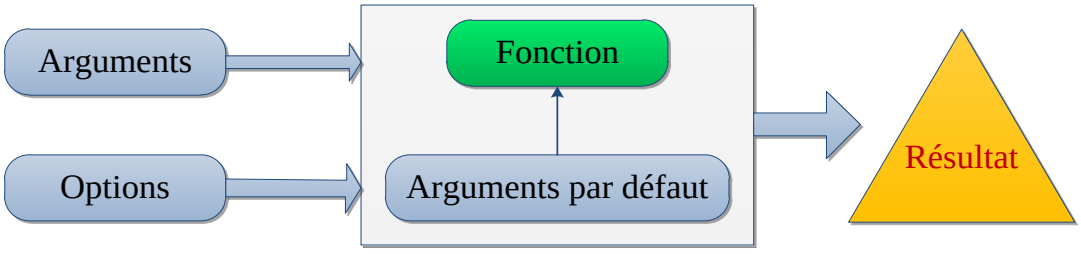


Figure 4.2 R : Une fonction en R

On constate que les arguments en jeu peuvent être des objets de type : données, formules, expression, vecteur … et c'est possible de définir certains ou tous les paramètres par défaut dans la fonction. Et l'utilisateur peut modifier ces valeurs par défaut avec les options. Ensuite, la fonction sera exécutée en fonction des arguments et générera le résultat.

Toutes les actions en R sont réalisées sur les objets présents dans la mémoire vive de l'ordinateur, donc aucun fichier temporaire n'est nécessaire. La lecture et l'écriture de fichier sont utilisées pour lire et enregistrer les données et les résultats (format PDF, graphiques, texte …). L'utilisateur exécute les fonctions au travers des commandes, et les résultats sont affichés directement à l'écran, ou stocke dans un objet, ou encore enregistre sur le disque dur (suivant le choix de l'utilisateur). Ces résultats sont également des objets et peuvent être utilisés pour faire du calcul à leur tour. L'utilisateur peut également importer les fichiers de données du disque dur

local ou d'un serveur à distance. Un aperçu du fonctionnement de R est représenté à la figure 4.3 [8]

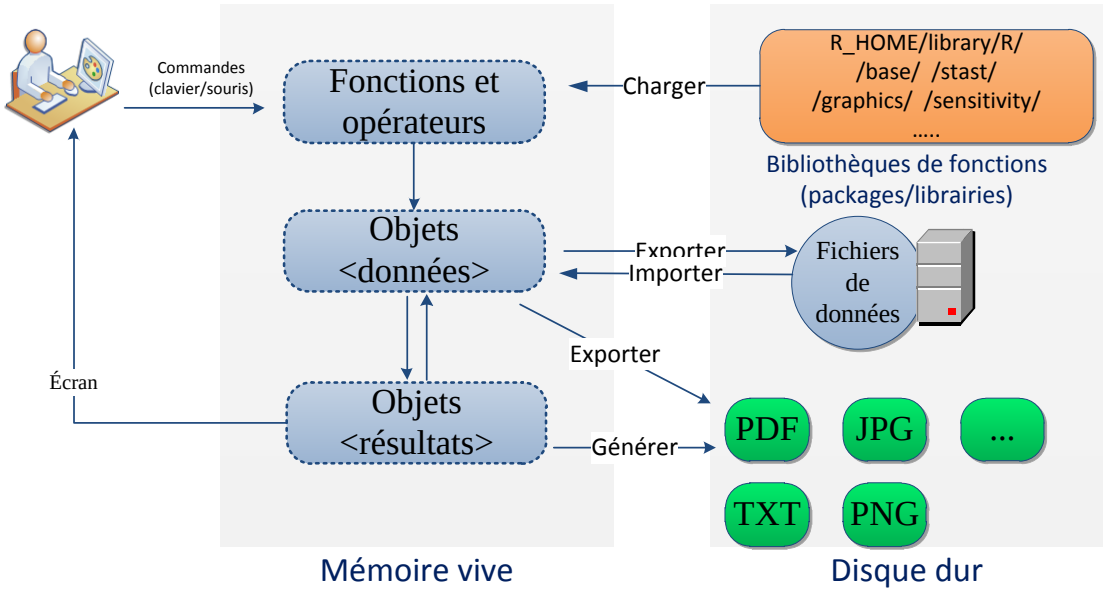


Figure 4.3 R : Aperçu de fonctionnement en R

Les fonctions disponibles dans l'environnement de R sont stockées dans une bibliothèque de fonctions qui se trouve sur le disque dur, dans le répertoire R_HOME/library (où R_HOME est le chemin complet vers le répertoire où R a été installé).

En travaillant avec R, les résultats sont affichés sur l'écran avec le numéro d'indice, par exemple :

```
> date()
[1] "Sun Jul 31 00:00:01 2011"
```

La fonction ***date()*** affiche la date du système, le chiffre [1] entre crochets indique que l'affichage commence au premier élément de la fonction date().

Le nom d'un objet en R doit obligatoirement commencer par une lettre (A-Z et a-z) et ensuite peut comporter des lettres, des chiffres (0-9), des pointes (.) et des 'underscore' (_) (tiret bas ou encore sous-tiret). R est strict avec le nom de l'objet, il distingue des majuscules et des minuscules, c'est-à-dire que les objets r et R sont des objets différents, par exemple :

```
> r = "Stage de fin d'études en "
> R <- 2011
> paste(r,R)
[1] "Stage de fin d'études en  2011"
> r <- c(1:10)
> R <- 11:20
> r * R
 [1]  11  24  39  56  75  96 119 144 171 200
>q("no")  ## quitter R sans sauvegarder la session.
```

4.4 Bibliothèques de R

Logiciel R fournit un ensemble de fonctions de base ou préprogrammées pour réaliser des calculs. Ces fonctions sont stockées dans des bibliothèques de fonctions (appelée aussi Packages ou Librairies) se trouvant à R_HOME/library avec R_HOME est le répertoire où R a été installé. Dans ce travail, R sous Windows a été installé à ***C:\Program Files\R\R-2.12.1\library*** qui est le chemin par défaut pendant l'installation du logiciel R.

Cependant sur Colosse, l'utilisateur n'a pas la permission d'installer des packages additionnels dans le répertoire où le module R a été installé sur Colosse car c'est un environnement de travail multiutilisateurs. Donc, selon les besoins de chaque usager, ce dernier pourrait installer les packages dans son répertoire personnel ou un répertoire autorisé.

Afin de centraliser tous les packages et de les partager à tous les membres de model*EAU*, ils ont été installés dans le répertoire partagé du groupe à l'adresse suivante ***/rap/yyk-770-aa/R/lib/linux/***. Ces architectures sont illustrées à la figure 4.4.

Il va falloir indiquer au logiciel R l'endroit où R doit chercher ces bibliothèques additionnelles, il faut donc créer une variable environnement incluant le chemin ci-dessous dans la session de l'utilisateur sur Colosse :

```
export  R_LIBS_USER=/rap/yyk-770-aa/R/lib/linux
```

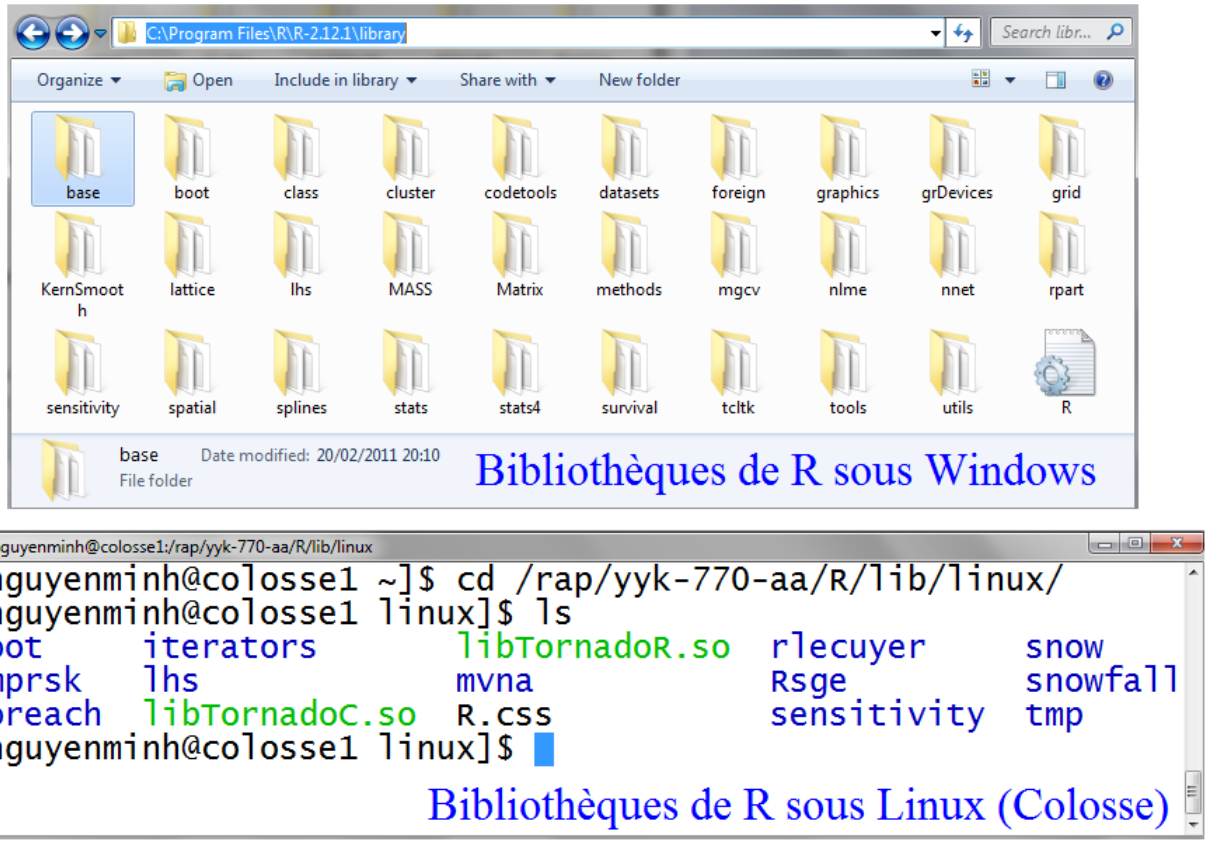


Figure 4.4 R : Des bibliothèques de R sur Windows & Linux

Dans R, la commande suivante fournit le chemin d'accès aux bibliothèques de fonctions déjà sur la machine :

```
> .Library
```

Si l'on désire utiliser une fonction d'une bibliothèque qui est installée sur la machine, mais qui n'est pas encore chargée dans l'environnement, il faut la charger. Sous Windows, on peut cliquer sur Menu Packages – Load Packages. Ou bien taper une des commandes suivantes :

```
> library("LE_NOM_DE_PACKAGE")
Exemple : library("boot","sensitivity")

> require("LE_NOM_DE_PACKAGE") ## charger le package et ses dépendances
Exemple : require("sensitivity")  ## Cette commande va charger les packages boot, sensitivity
```

On peut également apprendre les fonctions préprogrammées et la description de la bibliothèque disponible dans la machine à l'aide de la commande **"help.search"** et obtenir le manuel de l'aide sur une fonction avec la syntaxe **?NOM_DE_FONCTION**, par exemple :

```
> help.search("sensitivity")
> ?morris ## R s'ouvre une page web contenant la documentation complète et les exemples de la fonction Morris
```

Pour installer une nouvelle bibliothèque en ligne de commande, on appelle la commande suivante et choisit un serveur comportant le nom d'un pays parmi des serveurs du réseau CRAN.

```
> install.packages("LE_NOM_DE_PACKAGE")
Exemple : install.packages("sensitivity")
```

Pendant la réalisation de ce travail, les bibliothèques de fonctions suivantes ont été installées sur Colosse :

- Packages boot, lhs, sensitivity : pour faire l'analyse de la sensibilité de l'eau.
- Packages snow, snowfall, rlecuyer : pour faire les calculs parallèles.

Actuellement, il existe différentes librairies de calculs parallèles (Parallel Computing) [9] et pour réaliser des programmes qui exécutent les simulations en parallèle, **snowfall** a été choisi dû à son efficacité, sa convivialité et sa clarté dans les messages d'exécution ou d'erreurs. Il s'intègre parfaitement dans les programmes développés avec ce projet et fonctionne très bien sur Colosse. Nous pouvons trouver d'autres librairies de calcul parallèle telles que :

- snow : Support pour le calcul parallèle simple dans l'environnement R.
- doSMP : Fournit une interface parallèle pour la fonction *%dopar%* en utilisant revoIPC
- doSnow : Fournit une interface parallèle pour la fonction *% dopar%* en utilisant la libraire snow de Luc Tierney.
- Rmpi : Fournit une interface (wrapper) pour le langage MPI.

Tous ces packages sont accessibles à : http://cran.r-project.org/web/packages/ . Et sur le site de R-Project, il y a un ensemble de packages destinés aux calculs parallèles dont l'adresse est : http://cran.r-project.org/web/views/HighPerformanceComputing.html

Deuxième partie

Développement et solution

5
Déploiement de Tornado

Ce chapitre explique la démarche à suivre pour déployer la plateforme de simulation Tornado sur Colosse de telle manière que tous les membres model*EAU* puissent y accéder et l'utiliser directement. D'abord, quelques adaptations du Tornado ont été réalisées sur Colosse qui est un système de 64 bits et le programme Tornado est également exécuté en 64 bits pour exploiter toute la capacité du Colosse. Ensuite, ce chapitre présente la procédure de compilation à faire, étape par étape pour installer Tornado et finalement les tests de Tornado sur Colosse pour vérifier le bon fonctionnement.

5.1 Intégrations

Colosse opère sur la distribution de Linux, Redhat, une plateforme de 64 bits installée sur le système Cyclops qui est le serveur d'authentification et le point d'entrée du Colosse (*voir chapitre 2.2.4*). Donc il faudrait compiler le code source de la plateforme Tornado en 64bits sur Cyclops afin qu'il soit compatible avec Colosse.

Sur Colosse, tous les programmes préinstallés sont disponibles sous forme de modules (*voir chapitre 2.3.3*) afin de charger dynamiquement dans l'environnement de travail de l'utilisateur les modules nécessaires et décharger les modules dont on n'a plus besoin. Ceci est indispensable pour le bon fonctionnement des nœuds de calcul, car bien que ce sont des machines puissantes, mais ils ne contiennent aucun programme, c'est l'utilisateur qui va décrire dans le fichier de soumission (*voir chapitre 2.3.4*) de la tâche les modules nécessaires à charger aux nœuds de calculs alloués à cette tâche. Donc, Tornado serait déployé et intégré au Colosse afin d'être utilisé par les nœuds de calcul.

5.1.1 Emplacement de l'installation

Les codes sources de Tornado et ses modules dépendants sont stockés dans le système de fichiers du serveur Cyclops, à l'adresse suivante : ***/rap/yyk-770-aa/m4w***. Ce répertoire a été configuré afin qu'il soit accessible uniquement par les membres de l'équipe de recherche model*EAU*. Ces utilisateurs peuvent alors écrire dans leur fichier de soumission d'une tâche à SGE qui va exécuter Tornado sur les nœuds de calcul alloués pour cette tâche.

Pendant le déploiement de Tornado sur Colosse, il a été découvert qu'il faut définir un ensemble de variables d'environnement contenant les différents chemins de librairies, de codes sources et de fichiers de compilation de Tornado. Toutes ces variables sont définies dans un fichier qui est indispensable pour le fonctionnement du Tornado, car il précise où se trouvent les modules et les librairies dépendants de Tornado, et le programme lui-même. Ce fichier a été nommé le fichier de startup ***.m4w_settings.sh*** dont l'endroit est décrit à la figure 5.1.

Ensuite, pendant la phase d'utilisation R sur Colosse, il est également nécessaire de définir les variables d'environnement. Finalement, toutes ces variables d'environnement indiquant les répertoires des librairies nécessaires pour leur fonctionnement ont été regroupées dans ce fichier de startup. À la fin de ce fichier, les commandes ont été ajoutées pour charger les modules GCC et R dans l'environnement de travail d'utilisateur. (*Voir Figure 5.2*)

De cette manière, l'utilisateur se connecte au Colosse, et s'il désire travailler avec Tornado et R, il lui suffira d'activer ce fichier startup avec la commande suivante. Ceci est applicable pour tous les membres de model*EAU* :

```
source  /rap/yyk-770-aa/m4w/.m4w_settings.sh
```

En outre, si l'utilisateur désire automatiser cette commande, il suffira de l'ajouter à la fin du fichier $HOME/.bash_profile :

```
nano  $HOME/.bash_profile
# Ajouter cette ligne à la fin du fichier et sauvegarder. Il est appliqué dans la session suivante.
source /rap/yyk-770-aa/m4w/.m4w_settings.sh
```

De plus, pour envoyer une tâche (un job) au SGE en lui demandant d'exécuter les simulations en parallèle qui nécessitent le simulateur Tornado et le logiciel R, l'utilisateur n'a qu'à ajouter cette ligne dans son fichier de soumission de tâche. De nombreux exemples sont présentés dans l'annexe A. L'architecture des répertoires de travail sur Colosse est représentée à la figure 5.1. Dans le répertoire personnel de l'auteur (chaque utilisateur du Colosse en possède un), des projets de Tornado ont été créés qui contiennent les simulations d'expérience virtuelle (*Simul.Exp.xml*) et les codes sources en R dont l'extension est *.R*.

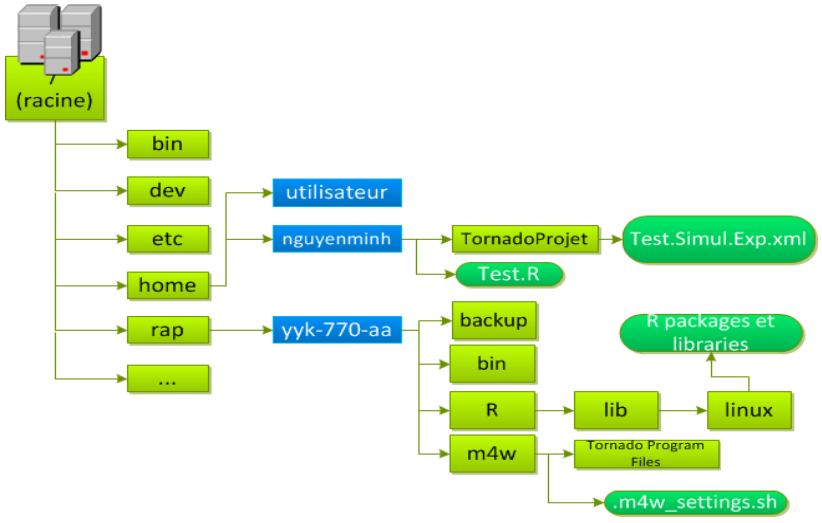


Figure 5.1 Déploiement de Tornado : Répertoires de travail sur Colosse

```
# --- R Settings ---
export R_ROOT_PATH="/rap/yyk-770-aa/R"
export R_DATA_PATH="$R_ROOT_PATH/data"
export R_LIBS_USER="$R_ROOT_PATH/lib/linux"
export R_LIBRARY_PATH="$R_ROOT_PATH/lib/linux"

# --- Tornado Settings ---
export M4W_HOME="/rap/yyk-770-aa/m4w"
export OT_ROOT_PATH="$M4W_HOME/opentop-1-5-1"
export OPENSSL_ROOT_PATH="/usr/include/openssl"
........
export TORNADO_ROOT_PATH=$M4W_HOME/Tornado
export TORNADOC_ROOT_PATH=$M4W_HOME/TornadoC
export TORNADOR_ROOT_PATH=$M4W_HOME/TornadoR
export PATH=$PATH:/rap/yyk-770-aa/bin

# Load Compilers GCC 442
module load compilers/gcc/4.4.2
# Load R-Project Module
module load apps/r-2.10.1
```

Figure 5.2 Déploiement de Tornado : Fichier startup .m4w_settings.sh

5.1.2 Options de compilation

Pour compiler les codes sources de Tornado en 64bits, il faut ajouter quelques options. Comme nous savons maintenant que Tornado se base sur des modules différents (*voir chapitre 3.1*), ils sont constitués d'un ensemble d'objets compilés (fichiers avec l'extension *.o*) et sont regroupés en librairies/modules (l'extension *.so, Shared Object*). Celles-ci sont ensuite utilisées par les instances de Tornado qui seront exécutées sur Colosse.

La compilation des Shared Objects (.so) sur une plateforme 64bits nécessite l'ajout de l'option **'-fPIC'** (Flag Position Independent Code) dans la partie CFLAG du script de compilation [10]. Ceci signifie que le code exécutable fourni par ces librairies pourra être exécuté indépendamment de sa position en mémoire. Une autre option pourra être utilisée *éventuellement* **'-fpermissive'** (Flag Permissive), elle permet d'éviter les erreurs de la compilation dues à un code non respectueux des standards.

Ces options doivent être manuellement ajoutées dans les fichiers (scripts) de paramétrage de compilation des modules constituant Tornado tels que : FC2, CLAPACK, OpenTop et d'autres modules dépendants.

5.2 Procédure du déploiement de Tornado sur Colosse

Avant de déployer les différentes compilations des modules de Tornado, il est nécessaire de choisir un compilateur pour compiler les codes sources de Tornado, GCC 4.4.2 a été choisi, c'est la version plus récente sur Colosse. De plus, Tornado a été compilé et déployé avec succès avec GCC à l'Université de Gand et il semblait donc logique d'utiliser le même compilateur pour assurer le résultat de l'installation et le fonctionnement de Tornado.

Tornado se base sur différentes librairies et modules, il faut donc installer d'abord les librairies dépendantes avant d'installer le noyau Tornado. Et plusieurs librairies sont déjà présentes sur Colosse, ce sont :

- OpenSSL : version 0.9.8e-fips-rhel5 (01 Jul 2008), se trouve à ***/usr/include/openssl***, Tornado a utilisé les fonctionnalités de cryptographie de OpenSSL dans son module Common (*voir chapitre 3.1.3*) pour assurer la performance de la gestion de cryptage des librairies, d'après [7].
- Bison : version 2.3, son emplacement est **/usr/bin/bison**. Tornado l'utilise pour l'analyse lexicale et grammaticale du texte d'un modèle de haut niveau.
- Flex : version 2.5.4, son emplacement est **/usr/bin/flex**. Flex est un générateur d'analyseur lexique. Il convertit une description de haut niveau d'un certain nombre de jetons dans un code source.

La procédure du déploiement de Tornado se réalise de l'étape 1 à 7, montrée à la figure 5.3.

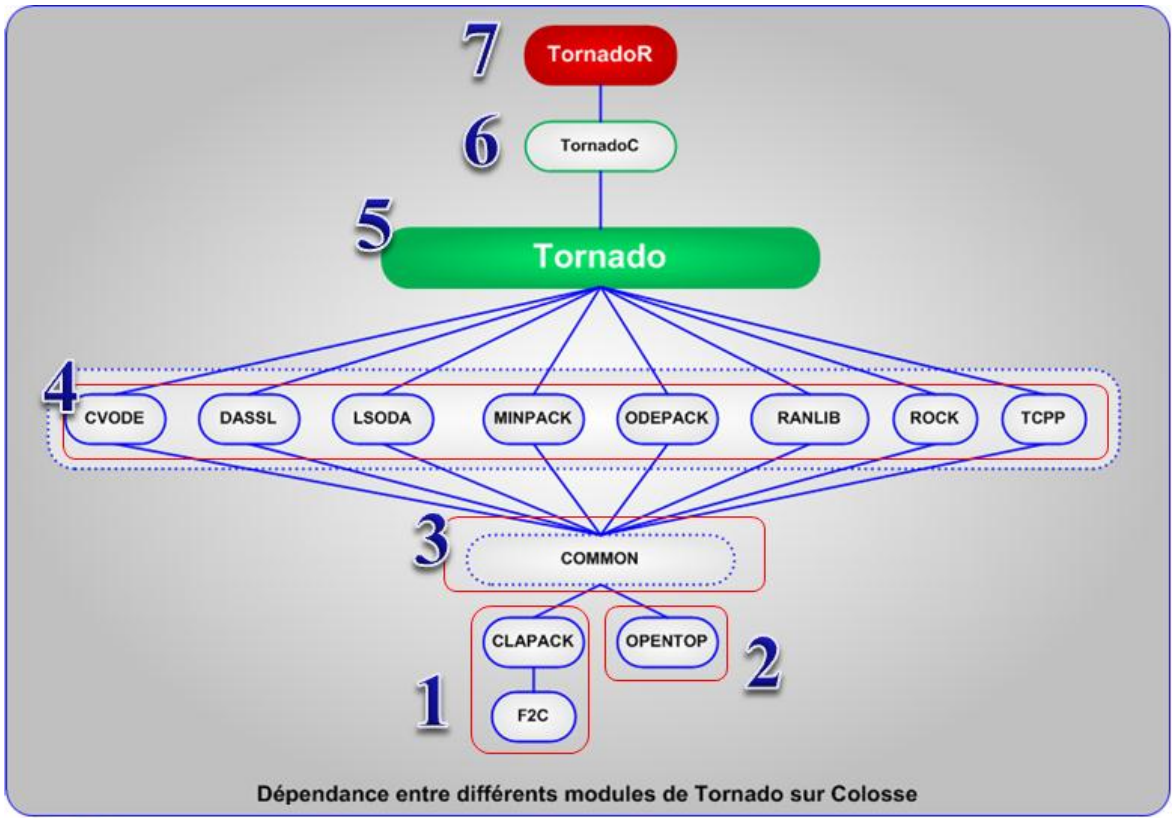


Figure 5.3 Déploiement de Tornado : Les modules de Tornado installés sur Colosse

5.2.1 F2C et CLAPACK

Tornado a besoin de 2 librairies intégrées dans F2C : libF77.a et libI77.a. L'extension .a est le fichier d'archive sous Linux. Pour créer ces 2 librairies, il faut d'abord télécharger le code source de F2C disponible sur le site de Netlib (*voir chapitre 3.1.1*). Les fichiers à télécharger sont les suivants : f2c.h, libf77 et libi77 (bundle of source C) et il faut les mettre dans un dossier nommé F2C. La création de 2 fichiers librairies se résume aux commandes suivantes (# = commentaire):

```
  cd F2C   #Contient les fichier f2c.h, libf77 et libi77
  chmod +x libf77 libi77
# Création du fichier libF77.a
  sh libf77
  cp f2c.h libF77 ; cd libF77
  nano makefile
# Ajouter –fPIC à la partie CFLAGS dans le fichier makefile
# Compiler pour créer le fichier libF77.a
  make ; cd ..

# Création du fichier libI77.a
  sh libi77
  cp f2c.h libI77 ; cd libI77
  nano makefile
# Ajouter –fPIC à la partie CFLAGS dans le fichier makefile
# Compiler pour créer le fichier libI77.a
  make ; cd ..

# Arranger les fichiers
  mkdir include ; mv f2c.h include
  mkdir lib ; mkdir lib/linux
  cp libF77/libF77.a  lib/linux ; cp libI77/libI77.a  lib/linux
```

Ensuite, nous allons compiler le module CLAPACK, son code source est également disponible sur Netlib (*voir chapitre 3.1.2*). Le fichier nécessaire clapack.tgz est à décompresser (commande ***tar***) et mettre dans la même hiérarchie de dossier F2C (ainsi pour les autres modules), le contenu après la décompression est un dossier CLAPACK-NUMERO_VERSION. On le renomme à CLAPACK, ensuite on ajoute l'option '-fPIC' dans la partie CFLAGS du fichier **make.inc** présent dans le répertoire CLAPACK et puis exécute les commandes suivantes pour le compiler :

```
  cd CLAPACK
  make f2clib ; make blaslib
  cd INSTALL ; make clean
  make
  cd ../SRC ; make clean
  make ; cd ..
  cp -R INCLUDE include
```

On obtient alors 2 fichiers blas_LINUX.a et lapack_LINUX.a. Nous poursuivons la compilation avec les commandes suivantes :

```
mkdir lib ; mkdir lib/linux
cp blas_LINUX.a    lib/linux/libblas.a ; cp lapack_LINUX.a   lib/linux/liblapack.a
cp lib/linux/libblas.a  ../F2C/lib/linux/ ; cp lib/linux/liblapack.a  ../F2C/lib/linux/
```

5.2.2 OpenTop

Le module OpenTop est une bibliothèque C++ commerciale et orientée vers les services Web, elle a été développée par Elcel Technology. Tornado utilise OpenTop pour la création et la synchronisation des threads ainsi que pour l'analyse syntaxique des documents XML. OpenTop dispose des outils de gestion de ressources, de développement inter-plateformes, de réseau et de support pour Unicode, différents types d'I/O, ainsi que les protocoles SSL et http d'après [7].

À partir de cette étape, les codes sources sont disponibles dans le Tornado Project Website [6]. Il faut avoir un compte pour accéder au projet et ses ressources. La compilation d'OpenTop se résume aux commandes suivantes (# = Commentaire):

```
# Décompresser le package opentop
  unzip opentop-1-5-1.zip
# Convertir tous les fichiers au format Unix
  find opentop-1-5-1 -type f -exec dos2unix {} \n;
  cd opentop-1-5-1
# Octroyer la permission d'exécution aux fichiers de configuration
  chmod +x config*
# Exécuter la configuration afin de créer le fichier Makefile pour la compilation
  ./configure
#Ajout les options '-fPIC' et de '-fpermissive' à COMP_SHARED_FLAGS du fichier Makefile
  nano ./buildtools/gcc_compiler_options
  make release_multi_wchar_shared
```

Le résultat de la compilation est les fichiers ***.so** (Shared Object) dans le dossier ***lib/***.

5.2.3 COMMON

À partir de cette compilation, on travaille réellement sur les codes sources de Tornado. Premièrement, la compilation de Tornado nécessite la spécification d'une série de variables d'environnement qui sont nécessaires pour localiser les différentes librairies utilisées. (*Voir chapitre 5.1.1*). Donc avant de compiler, il faut vérifier le fichier **Common/etc/linux/Common.conf.sh** contenant les localisations des différentes librairies et des options. Une faute de frappe dans ce fichier pourrait générer des erreurs à la compilation. Par ailleurs, il faut rajouter dans ce fichier l'option '-fPIC' pour compiler ce module en 64bits.

Et deuxièmement, il y a un petit problème du compilateur GCC 4.4.2 sur Colosse. En fait, à partir de la version GCC 4.3 et ultérieurs, la majorité des fichiers headers (fichiers d'en-tête) de librairie C++ n'inclut plus qu'un nombre très limité de headers et seulement les headers nécessaires, d'après [11].

Cela permet de réduire la taille de librairie mais par contre cela pose problème pour Tornado, car les codes sources de Tornado emploient intensivement des classes/fonctions :

- std ::memcpy sans écrire #include <cstring> ou <string>
- std ::auto_ptr sans écrire #include <memory>

Ainsi que les autres inclusions qui sont nécessaires pour la compilation. Elles se trouvent aux tables 5.1 et 5.2.

Et donc, pendant la compilation, ces modules avec les headers manquants ne compilent plus et génèrent les messages d'erreur à la compilation. Pour résoudre ce problème il faut analyser les messages d'erreur et ajouter manuellement les headers manquants aux fichiers nécessaires. Le tableau 5.1 reprend les fichiers ayant été modifiés et les headers ajoutés pour le module Common. Ces fichiers se trouvent dans le répertoire
Common/include/Common/<Module>

Table 5.1 Déploiement de Tornado: Headers additionnels pour le module Common

Fichier	***#include***
Crypto/Crypto.h	<string>
Manager/ManagerDLL.h	<memory>
Vector/Vector.cpp.inc	<cstring>
Parser/Parser.h	<cstring> <cstdio>
Platform.h	<cstring>
String/Convert.h	<cstring>
String/String.h	<string> / <cstring>

La compilation du module Common se résume aux commandes suivantes :

```
# Entrer dans le module Common
  cd Common
# Ajouter  '-fPIC' et '-fpermissive' + vérifier les localisations des librairies du fichier
# Common.conf.sh
  nano etc/linux/Common.conf.sh
# Appliquer les définitions des variables d'environnement dans la session d'utilisateur
  source etc/linux/Common.conf.sh
  cd build/linux
  make clean ; make depend
  make
```

À la fin de la compilation (avec succès), on obtiendra le fichier **libCommon.a** qui se trouve dans le répertoire ***Common/lib/linux***.

Note : Les fichiers *.conf.sh sont propres pour chaque module. Ils contiennent les variables d'environnement complémentaires nécessaires à leurs compilations. Donc, à chaque compilation d'un module, il faut y ajouter l'option '-fPIC' et vérifier les localisations des librairies prédéfinies dans le fichier.

5.2.4 CVODE, DASSL, LSODA, MINPACK, ODPACK, RANLIB, ROCK, TCPP

Nous passons maintenant aux compilations des solveurs et des outils pour le fonctionnement de Tornado. La procédure à suivre est exactement comme celle du module **Common**. Voici les commandes permettent de compiler un module :

```
# Entrer dans le module à compiler
  cd $NOM_DU_MODULE
# Ajouter '-fPIC -fpermissive' + vérifier les localisations des librairies dans
# $NOM_DU_MODULE.conf.sh
  nano etc/linux/$NOM_DU_MODULE.conf.sh
# Appliquer les définitions des variables d'environnement dans la session d'utilisateur
  source etc/linux/$NOM_DU_MODULE.conf.sh
  cd build/linux
  make clean ; make depend
  make
```

Après la compilation (avec succès), on obtiendra les fichiers archives avec la syntaxe ***lib$NOM_DU_MODULE.a*** dans le répertoire ***$NOM_DU_MODULE/lib/linux***. Où $NOM_DU_MODULE est utilisé pour les modules suivants : CVODE, DASSL, LSODA, MINPACK, ODPACK, RANLIB, ROCK, TCPP.

5.2.5 Tornado

Nous arrivons à la compilation du module noyau Tornado. Avant de compiler ce module, il y a une petite modification au niveau du code source. Nous constatons que dans le fonctionnement de Tornado (*voir chapitre 3.2*), l'utilisateur va appeler le compilateur **tbuild** pour créer une librairie à partir du code C. Puisque l'on veut compiler Tornado sur un système de 64 bits, il faut que ce compilateur travaille par défaut en 64 bits. Voici la modification effectuée dans le fichier ***Tornado/src/Common/Build/Build.cpp*** :

```
void CBuild::Init(
    vector<wstring>& CFlags,
    vector<wstring>& LFlags,
    vector<wstring>& Libs)
….
# Ligne 1100, où il y a F77 et I77
  else if (m_Platform == L"linux")  {
        CFlags.push_back(L"-c");
# Code ajouté
  CFlags.push_back(L"-fPIC"); // Add for SGE Usage and x64 bits
```

Si nous ne faisons pas cette modification, à chaque appel de cette fonction, il faudrait inclure l'option '-fPIC' comme décrit ci-dessous :

```
tbuild –C –fPIC <NOM_DU_MODELE_DE_SIMULATION>
```

Ensuite les ajouts de fichier d'en-têtes (headers files) aux fichiers dans la table 5.2 sont nécessaires afin d'éviter les messages d'erreur à la compilation de ce module. Ces fichiers se trouvent dans le répertoire **Tornado/src/**

Table 5.2 Déploiement de Tornado: Headers additionnels pour le module Tornado

Fichier	***#include***
Tornado/include/Tornado/Common/Main/Globals.h	<memory>
Tornado/include/Tornado/Common/Main/Main.h	<memory>
Tornado/include/Tornado/Common/MSLE/ExecCalcVar.h	<memory>
Tornado/include/Tornado/Common/MSLE/SymbModel.h	<memory>
Tornado/include/Tornado/Common/Util/License.h	<climits>
Tornado/include/Tornado/ME/MOF2T/MOF2TASTNode.h	<cstdio>
Tornado/include/Tornado/ME/MSLU/MSLUMatrix.h	<memory>
Tornado/include/Tornado/ME/MSLU/MSLUProb.h	<cstdlib> <cstring>
Tornado/include/Tornado/ME/TMSL/TMSLUtil.h	<cstring> <cstdio>
opentop-1-5-1/ot/sax/DefaultHandler.h	<memory>
Common/include/Common/Interface/ICallbackMessage.h	<memory>
Common/include/Common/Manager/Manager.h	<memory>
Common/include/Common/Props/Props.h	<memory>

Après ces interventions au code source Tornado, il est nécessaire de copier les fichiers *.a qui ont été créés après les compilations des modules dans le répertoire **Tornado/lib/linux**, avec les commandes suivantes :

```
cd Tornado/lib/linux
cp ../../../F2C/lib/linux/*.a   .
cp ../../../Common/lib/linux/*.a  .
cp ../../../RANLIB/lib/linux/*.a  .
cp libF77.a  liblibF77.a
cp libF77.a  llibF77.a
```

On aura les fichiers suivants dans le répertoire Tornado/lib/linux: libblas.a, liblapack.a, libI77.a, libF77.a, liblibF77.a, llibF77.a, libRANLIB.a, libCommon.a.

Les étapes préliminaires sont faites, nous pouvons ajouter les options '-fPIC -fpermissive' dans la partie TORNADO_CFLAGS et vérifier toutes les emplacements des librairies dépendantes. La compilation du noyau Tornado se fait avec les commandes suivantes :

```
cd Tornado/build/linux
source  ../../etc/linux/Tornado.conf.sh
make
```

Enfin, avec toutes ces préparations, Tornado doit être compilé avec succès et sans message d'erreur. Après la compilation de Tornado, les fichiers archives avec l'extension *.*a* sont créés dans le répertoire ***Tornado/bin/linux*** et les programmes exécutables et des librairies sont dans le répertoire ***Tornado/lib/linux.*** Ces fichiers sont illustrés à la figure 5.4.

```
[nguyenminh@colosse1 Tornado]$ pwd
/rap/yyk-770-aa/m4w/Tornado
[nguyenminh@colosse1 Tornado]$ ls bin/linux/
graphml2t             tgetid           tmodellib2components          TornadoSolveIntegMidpoint.so       tpalettetreelib
mof2t                 tgetinfo         tmsl                          TornadoSolveIntegMilne.so
tblocklib             tgetmac          TornadoSolveCINelder.so       TornadoSolveIntegModifiedEuler.so
tblocklib2components  tgetmetric       TornadoSolveCIRichardson.so   TornadoSolveIntegRalston.so
tblocklibmerge        ticonlib         TornadoSolveIntegAB2.so       TornadoSolveIntegRK3.so
tbuild                tinitial         TornadoSolveIntegAB3.so       TornadoSolveIntegRK45.so
tclasslib             tjobs2batch      TornadoSolveIntegAB4.so       TornadoSolveIntegRK4ASC.so
tcontainerlib         tjobs2jdl        TornadoSolveIntegAlg.so       TornadoSolveIntegRK4.so
tcontrols             tjobs2sge        TornadoSolveIntegCVODE.so     TornadoSolveIntegROCK2.so
tcreatematlab         tlayout          TornadoSolveIntegDASPK.so     TornadoSolveIntegROCK4.so
tcreatesymb           tlayout2graphml  TornadoSolveIntegDASRT.so     TornadoSolveRootBroyden.so
tcrypt                tlayout2mo       TornadoSolveIntegDASSL.so     TornadoSolveRootHybrid.so
tdiff                 tlayout2msl      TornadoSolveIntegDIRK.so      TornadoSolveScenCross.so
texec                 tlayout2msle     TornadoSolveIntegDOPRI5.so    TornadoSolveScenFixed.so
texp                  tlayout2txt      TornadoSolveIntegDOPRI853.so  TornadoSolveScenGrid.so
texp2jobs             tlicenseserver   TornadoSolveIntegEuler.so     TornadoSolveScenRandom.so
texpsimul2model       tmain            TornadoSolveIntegHeun.so      TornadoSolveScenSeq.so
tfile2batch           tmodel           TornadoSolveIntegLSODA.so     TornadoSolveSensPlain.so
tgethostname          tmodellib        TornadoSolveIntegLSODE.so     tpalettelib
[nguyenminh@colosse1 Tornado]$ ls lib/linux/
libblas.a                    libTornadoEEBVDF.a            libTornadoEEParamLib.a
libCommon.a                  libTornadoEECI.a              libTornadoEEPlot.a
libF77.a                     libTornadoEEClassification.a  libTornadoEEScen.a
libI77.a                     libTornadoEEContainerLib.a    libTornadoEEScenOptim.a
liblapack.a                  libTornadoEEControls.a        libTornadoEEScenSSRoot.
liblibF77.a                  libTornadoEEEnsemble.a        libTornadoEESens.a
libRANLIB.a                  libTornadoEEExp.a             libTornadoEESeq.a
libTornadoCommonBuild.a      libTornadoEELR.a              libTornadoEESimul.a
libTornadoCommonMain.a       libTornadoEEMC.a              libTornadoEESolve.a
libTornadoCommonMSLE.a       libTornadoEEMCOptim.a         libTornadoEESSOptim.a
libTornadoCommonProject.a    libTornadoEEObjBuffer.a       libTornadoEESSRoot.a
libTornadoCommonUtil.a       libTornadoEEObjEval.a         libTornadoEEStats.a
libTornadoCommonXML.a        libTornadoEEOptim.a           libTornadoEEXML.a
```

Figure 5.4 Déploiement de Tornado : Résultat de la compilation du module Tornado

5.2.6 TornadoC

Le noyau Tornado a été mis en place, nous installons les interfaces nécessaires pour ce projet. Nous commençons par l'interface TornadoC. Il faut également ajouter les options '-fPIC -fpermissive' dans la partie TORNADOC_CFLAGS et vérifier toutes les emplacements des librairies dépendantes dans le fichier ***TornadoC/etc/linux/TornadoC.conf.sh***. La compilation de ce module se résume aux commandes suivantes :

```
  cd TornadoC/build/linux
  source  ../../etc/linux/TornadoC.conf.sh
  make
  cd ../../..
# Copier la librairie partagée dans le répertoire des librairies de R a été construit préalablement
  cp TornadoC/lib/linux/libTornadoC.so  ../R/lib/linux
```

La librairie libTornadoC.so est créée dans le répertoire TornadoC/lib/linux. Il faudrait copier ce fichier dans le répertoire des librairies de R (*voir chapitre 4.4*).

5.2.7 TornadoR

TornadoR est le dernier module à compiler, c'est aussi le module destiné pour travailler avec le langage R. Comme toujours, avant la compilation, il faut ajouter l'option '-fPIC' dans la partie TORNADOR_CFLAGS pour créer une librairie compatible au système 64 bits et vérifier toutes les emplacements des librairies dépendantes dans le fichier ***TornadoR/etc/linux/TornadoR.conf.sh***.

```
  cd TornadoR/build/linux
  source  ../../etc/linux/TornadoR.conf.sh
  make
  cd ../../..
# Copier la librairie partagée dans le répertoire des librairies de R a été construit préalablement
  cp TornadoR/lib/linux/libTornadoR.so  ../R/lib/linux
```

Après la compilation du module TornadoR, la librairie libTornadoR.so (dans le répertoire TornadoR/lib/linux/) devrait être copiée dans le répertoire des librairies de R.

5.3 Phase de test

Les premiers tests ont été effectués sur 2 exemples du projet Tornado sur Cyclops :

- Benchmark : fournit par un collègue du groupe model*EAU*, les fichiers .WCO, .WXP et le fichier SimulInput.txt contiennent les données nécessaires pour la simulation.
- Cosines : disponible dans le dossier d'installation du Tornado avec le modèle écrit en langage Modelica avec l'extension *.mof*.

Pour construire les fichiers de simulation d'expérience virtuelle, il faut suivre le fonctionnement de Tornado (*voir chapitre 3.2*) afin d'obtenir les fichiers d'expérience virtuelle dans le cas le modèle écrit en langage Modelica. La construction des fichiers de simulation s'est effectuée à l'aide des commandes suivantes :

```
#Benchmark
# Appeler script west2t pour simplifier la construction à partir du WCO et WXP.
  west2t Benchmark
# On obtiendra le résultat de la simulation dans le fichier Benchmark.Simul.out.txt
```

```
#Cosines
# Appeler script mof2simul pour simplifier la construction à partir du modèle Modelica
  mof2simul Cosines
# On obtiendra directement la simulation Cosines.Simul.Exp.xml, et on va l'exécuter par :
  texec  ./Cosines.Simul.Exp.xml
# On obtiendra le résultat de la simulation dans le fichier Cosines.Simul.out.txt
```

Après avoir analysé les résultats obtenus des fichiers ***.Simul.Exp.xml***, le contenu est correct et les valeurs des résultats de simulation sont tout à fait plausibles.

Dans un second temps, les mêmes tests ont été effectués sur Colosses. Les 2 fichiers de soumission ont été écrits pour les exemples projet Tornado et ces 2 fichiers ont été soumis au SGE de Colosse (Exemple de fichier de soumission à l'annexe A). Après l'exécution de ces 2 jobs, on a obtenu les fichiers ***Simul.Exp.xml*** attendus et leurs résultats de simulation ***Simul.out.txt*** sont corrects par rapport aux simulations sur le serveur Cyclops.

6

Développement de programmes en R

Le contexte du programme est résumé comme suite : L'utilisateur fait l'analyse de l'importance d'un nombre de paramètres de la modélisation de la qualité de l'eau. Chacun de ces paramètres se trouve dans un espace de valeurs précises. Donc pour réaliser son travail d'analyse, l'utilisateur a créé une série de simulations différentes et chacune utilise des valeurs spécifiques des paramètres à analyser. Les valeurs de chaque paramètre sont définies dans l'espace de valeurs par l'utilisateur. Après l'exécution de toutes ces simulations avec Tornado, l'utilisateur récupère les résultats et les analyse.

L'application à développer consiste donc à recevoir les paramètres d'analyse et leurs intervalles de valeurs, tout est défini par l'utilisateur. Ensuite, elle crée un tableau qui contient un ensemble de simulations avec les valeurs différentes; chaque simulation est unique et contient les paramètres avec les valeurs différentes. Et à l'exécution, toutes ces simulations sont exécutées en parallèle. Le nombre d'exécutions en parallèle dépendra de la capacité de la machine (nœud de calcul). Finalement, l'utilisateur va chercher les résultats des simulations stockés dans un répertoire où il a défini.

Pour cela, ce chapitre explique premièrement les éléments principaux dans la recherche afin de trouver des solutions pour réaliser des programmes de calcul parallèle. Et en second temps, on verra le développement de 2 programmes en R, l'un utilise l'interface TornadoR et le package snowfall, tandis que l'autre utilise les fonctions du noyau de Tornado.

6.1 Recherche

Nous avons vu au *chapitre 3.1.7* que Tornado fournit un ensemble d'interfaces pour rendre son utilisation possible à partir des différents langage (R, Matlab, Java, etc.). Donc l'utilisateur qui était familier avec son langage de programmation peut utiliser Tornado en travaillant avec l'interface correspondante de Tornado (TornadoC, TornadoR, TornadoMEX, etc.)

Nous allons voir comment utiliser TornadoR dans ce projet et découvrir les caractéristiques de ce dernier. Ensuite, nous allons aborder la question *"Comment créer le tableau contenant les intervalles de valeurs des paramètres d'analyse ?"*, la réponse se trouve au *chapitre 6.1.2*.

6.1.1 TornadoR

Pour rappel, TornadoR est une interface faisant le pont entre le langage R-Project et le noyau Tornado. À l'intérieur du code R, on peut donc exécuter une expérience virtuelle sur Tornado au travers de l'interface TornadoR. En réalité, cette interface contient des fonctions appelant les fonctions de l'interface TornadoC qui à son tour va demander les fonctions de Tornado.

Pendant les compilations de TornadoC et TornadoR (*voir chapitres 5.2.6 et 5.2.7*), nous avons copié les librairies partagées dont l'extension est *.so* (Shared Object) dans le répertoire ***/rap/yyk-770-aa/R/lib/linux***. Les 2 fichiers ***libTornadoC.so*** et ***libTornadoR.so*** devraient être ensemble dans ce répertoire et ensuite on doit modifier le fichier startup .m4w_settings.sh (*voir chapitre 5.1.1*) afin de préciser la localisation de ces 2 fichiers. Voici la modification apportée au fichier startup ***.m4w_settings.sh*** :

```
export R_ROOT_PATH="/rap/yyk-770-aa/R"
export R_DATA_PATH="$R_ROOT_PATH/data"
export R_LIBS_USER="$R_ROOT_PATH/lib/linux"
export R_LIBRARY_PATH="$R_ROOT_PATH/lib/linux"
export LD_LIBRARY_PATH=$R_LIBRARY_PATH:$LD_LIBRARY_PATH
```

Cette modification est nécessaire pour le fonctionnement de TornadoR. Pendant le développement, il y avait un problème lors l'exécution de TornadoR avant de faire cette modification. Il est illustré à la figure 6.1.

```
[nguyenminh@colosse1 PredatorPrey]$ Rscript PredatorPrey.R
Error in dyn.load(getOption("TORNADOR_LIB")) :
  unable to load shared library '/rap/yyk-770-aa/R/lib/linux/libTornadoR.so':
  libTornadoC.so: cannot open shared object file: No such file or directory
Execution halted
[nguyenminh@colosse1 PredatorPrey]$
```

Figure 6.1 Développement en R : L'erreur de l'exécution TornadoR

Après des recherches et comme décrit à [12], l'explication de ce problème est la suivante : à l'exécution de TornadoR, celui-ci va utiliser les fonctions définies dans libTornadoR.so en appelant les fonctions définies dans libTornadoC.so (ces 2 fichiers étaient toujours ensemble). Mais la librairie libTornadoR.so ne va pas chercher libTornadoC.so dans son répertoire où elle se situe, mais plutôt dans la variable d'environnement LD_LIBRARY_PATH qui contient les localisations des librairies dynamiques.

Donc, il est nécessaire de mettre ces 2 fichiers ensemble et définir leur localisation dans la variable LD_LIBRARY_PATH. Après cette modification, TornadoR fonctionne correctement.

6.1.2 Matrice de la fonction Morris

L'équipe model*EAU* travaille en langage R pour créer des programmes analysant l'incertitude et la sensibilité des modèles de qualité de l'eau. Elle a utilisé le package ***sensitivity*** dans le logiciel R pour travailler dans ce domaine. Dans cette bibliothèque de fonctions (*voir chapitre 4.4*), il y a une fonction s'appelle ***Morris*** qui permet de générer une matrice avec les valeurs aléatoires dans un intervalle défini avec les arguments de cette fonction.

L'une des étapes dans l'application à développer, a utilisé cette fonction pour créer une matrice dont :

- Le nombre de colonnes vaut le nombre de paramètres (NbParams) entrés par l'utilisateur.
- Le nombre de lignes vaut (NbParams+1) * nombre d'itérations (défini par l'utilisateur).

Un exemple de matrice créée par la fonction Morris est illustré à la figure 6.2.

```
# The number of Quantities with their names in the String vector (for 4 parameters, for example)
param.quantities <- c(".ASU1.M(S_O)", ".ASU1.M(S_NO)", ".ASU1.M(S_ND)", ".ASU1.M(S_NH)")
param.iterations <- 2
param.binf   <- c(4.000000,7688.3423523 ,1129.4124512, 8255.124125125)
param.bsup   <- c(5.000000, 7688.9999999, 1130.123214, 8256.231241254)
```

Matrice de Morris

ID	.ASU1.M(S_O)	.ASU1.M(S_NO)	.ASU1.M(S_ND)	.ASU1.M(S_NH)	
1	**4,053053053**	**7688,593825**	**1129,50352**	**8255,556333**	*Simulation 1*
2	4,056056056	7688,593825	1129,50352	8255,556333	*Simulation 2*
3	4,056056056	7688,5958	1129,50352	8255,556333	*Simulation 3*
4	4,056056056	7688,5958	1129,505654	8255,556333	*Simulation 4*
5	4,056056056	7688,5958	1129,505654	8255,553008	*Simulation 5*
6	**4,553553554**	**7688,684671**	**1129,873487**	**8255,380125**	*Simulation 6*
7	4,550550551	7688,684671	1129,873487	8255,380125	*Simulation 7*
8	4,550550551	7688,686646	1129,873487	8255,380125	*Simulation 8*
9	4,550550551	7688,686646	1129,875621	8255,380125	*Simulation 9*
10	4,550550551	7688,686646	1129,875621	8255,376800	*Simulation 10*

Exécutions parallèles

Figure 6.2 Développement en R : La matrice créée avec la fonction Morris

On constate qu'il y a 4 paramètres d'analyse : .ASU1.M(S_O), .ASU1.M(S_NO), .ASU1.M(S_ND), .ASU1.M(S_NH) dans la variable ***param.quantities***. Et les valeurs définies sont respectivement dans les intervalles [4,5], [7688.3423523, 7688.9999999], [1129.4124512, 1130.123214] et [8255.124125125, 8256.231241254] dans les variables ***param.binf et param.bsup***

Il y a 4 paramètres et 2 itérations dans la variable ***param.iterations;*** le nombre de lignes de matrice vaut donc 10 simulations {(4+1)*2=10} et chacune est unique, car elle comporte au moins une valeur différente que l'autre simulation. Il y aura 10 simulations à faire en parallèle (si applicable).

6.1.3 Package de R : snowfall

Snowfall est une bibliothèque de fonction écrite en langage R pour rendre plus facile la programmation parallèle sur des clusters (une grappe de calcul comme Colosse). Snowfall se base essentiellement sur le package **snow** en utilisant ses fonctionnalités du réseau et du cluster. Donc snowfall travaille également sur les différents types de connexion tels que : Socket, MPI, PVM et NetworkSpaces. Snowfall est considéré [13] comme un outil emballé de différents instruments de réseau et calcul parallèle afin de rendre son utilisation plus flexible.

Snowfall fonctionne en 2 modes : séquentiel ou parallèle, en fonction du besoin, l'utilisateur peut spécifier le mode d'exécution dans l'option de snowfall, par exemple :

```
# Charger le package snowfall dans l'environnement R
   library(snowfall)
# Définir le mode d'exécution parallèle, nombre de processeurs et le type de connexion
   sfInit(parallel=TRUE,cpus=16,type="SOCK")
# Définir une fonction qui sera exécutée en parallèle
  computing <- function(id) {…}
# Démarrer une session de calcul parallèle & exporter les données définies dans toutes les
# 'workers' créé par snowfall, chaque worker va travailler dans un core (cœur)
  sfExportAll(); sfClusterSetupRNG()
# Charger éventuellement la bibliothèque de fonctions nécessaires pour tous les workers.
  sfLibrary("PACKAGE_NAME" );
# Tous les workers exécutent la fonction computing.
# Si x = 3, il y a worker1/computing(1), worker2/computing(2), worker3/computing(3).
  sfLapply(1:x), computing);
# Fermer la session de calcul parallèle et libérer les ressources.
  sfRemoveAll();  sfStop();
```

De plus, snowfall contient différentes fonctions de calculs parallèles prédéfinies, la gestion des messages d'information et d'erreur et beaucoup d'autres outils nécessaires pour la programmation séquentielle ou parallèle. Snowfall possède également un outil qui s'appelle **sfCluster** (fonctionne en ligne de commande et interface graphique) permettant à exécuter en R les programmes parallèles utilisant le package snowfall.

Le fonctionnement de snowfall est illustré à la figure 6.3 ayant été construite selon [14].

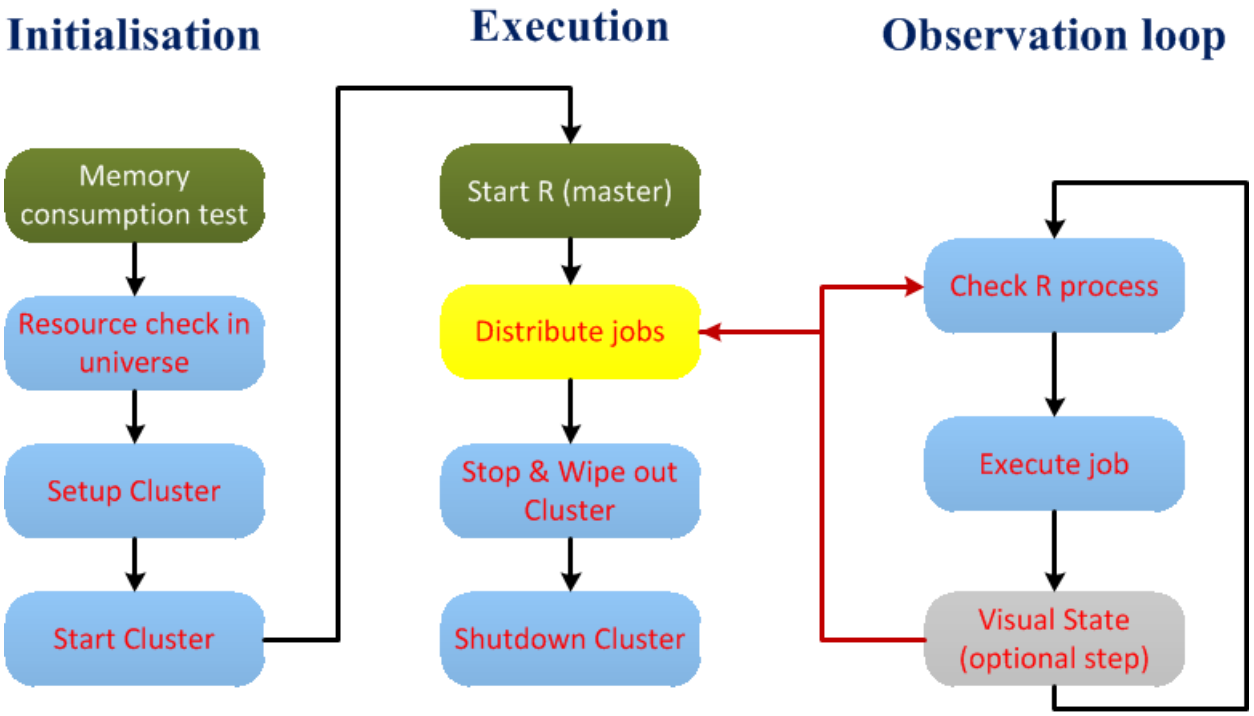


Figure 6.3 Développement en R : Fonctionnement de snowfall, d'après [14]

Le fonctionnement de snowfall est divisé en 3 étapes principales:

- Étape 1 : **Initialisation**, snowfall vérifie les ressources de la machine, et réserve ces ressources pour les travailleurs (worker) en fonction de la capacité de la machine. Si la machine possède 8 cœurs, mais on demande 16 cœurs, snowfall va utiliser réellement 8 cœurs de la machine. Ensuite, snowfall initialise et démarre la session pour l'exécution du programme R.

- Étape 2 : **Exécution**, le programme en R (Master) s'exécute et il va distribuer les tâches (jobs) aux workers (Slaves). Chaque worker exécute son job et une fois que tous les jobs sont terminés, l'exécution du programme R va fermer la session et libérer les ressources.

- Étape 3 : **Observation Loop (boucle)**, décrit le travail d'un worker slave qui a reçu des données nécessaires du programme R, il s'exécute, et s'il y a une erreur, ce job envoie un message au programme R Master, et le programme s'arrête et le message d'erreur s'affiche dans la fenêtre de l'exécution. Sinon, le job se termine et envoie également un message au programme R Master qu'il a terminé son travail.

6.2 Développement

Deux solutions en R ont été développées. Nous allons voir le fonctionnement de chacune. La première a été implémentée en utilisant TornadoR et snowfall, et la seconde a été construite en utilisant les fonctions du noyau de Tornado.

6.2.1 Solution 1 : R + TornadoR + snowfall

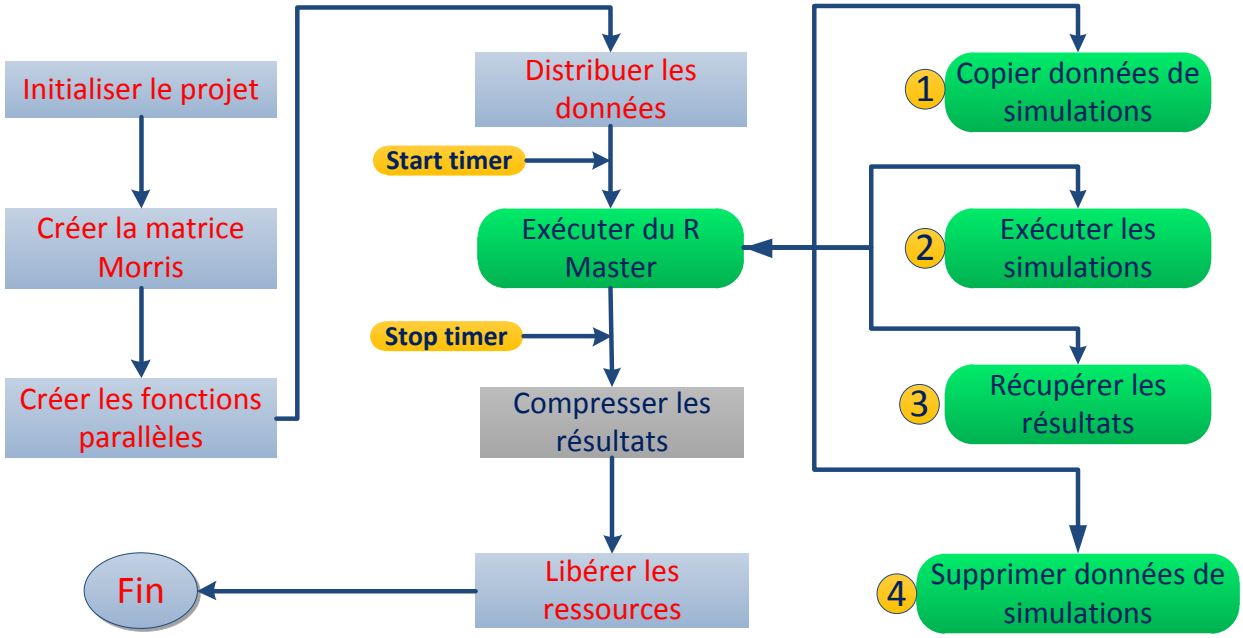


Figure 6.4 Développement en R : Fonctionnement de la solution 1 : R+TornadoR+snowfall

Comme illustré à la figure 6.4, le fonctionnement du premier programme (solution 1) est le suivant :

- Initialiser le projet : appliquer les données définies par l'utilisateur telles que : le nom du modèle de simulation, les paramètres d'analyse et leurs intervalles de valeurs ainsi que le nombre d'itérations pour construire la matrice et le nombre de CPUs souhaités à utiliser.
- Créer la matrice Morris en fonction des données définies par l'utilisateur.
- Les fonctions parallèles sont définies.
- En exécution du programme, snowfall va distribuer les données aux ressources réservées pour exécuter les simulations parallèles ('workers')
- Démarrer le compteur de temps de calcul.
- L'exécution des simulations se réalise par les appels de fonctions parallèles :
  - Copier les données de simulations en autant de nombre de simulations à exécuter.
  - Exécuter les simulations parallèles avec des données de la matrice Morris, le nombre réel d'exécutions parallèles dépendra de la capacité de la machine. À chaque simulation, le programme appelle les fonctions de l'interface TornadoR qui à son tour demande les fonctions du noyau du Tornado afin de créer une instance de Tornado pour l'exécution de cette simulation. Donc, autant de simulations parallèles, autant d'instances Tornado ont été créées et exécutées séparément.
  - Récupérer les résultats de simulation de chaque instance Tornado séparée et les regrouper dans un dossier nommé result.
  - Supprimer toutes les copies de données de simulations créées.

- Toutes les simulations terminées, le résultat est centralisé dans le dossier result, le programme arrête le compteur de temps de calcul et affiche la durée totale de l'exécution en secondes.
- Compresser le dossier result contenant les résultats.
- Libérer les ressources et afficher les messages d'informations tels que : le temps d'exécution, l'endroit des résultats.

On constate que pendant l'exécution, il y a une étape où le programme a copié les données de simulations en autant de nombre de simulations à exécuter. Cela est obligatoire en utilisant l'interface TornadoR.

En effet, les interfaces de Tornado (*voir chapitre 3.1.7*) sont classées en deux catégories :

- CPP API et .NET API sont des interfaces '*compréhensives* ' pouvant exploiter toutes les fonctionnalités offertes par le noyau générique Tornado.
- Les autres interfaces : MEX API, C API, R API (TornadoR) sont restreintes ou limitées dans le sens qu'elles ne peuvent appeler qu'une partie des fonctionnalités du noyau générique Tornado. Essentiellement, elles sont utilisées pour charger et exécuter des simulations existantes.

En plus, TornadoR s'est basée sur TornadoC qui ne peut exécuter une seule simulation à la fois, car elle est une interface expérimentale et limitée. Par conséquent, TornadoR ne peut exécuter également une seule simulation à la fois. Pour contourner cette limitation, les différentes copies ont été créées et à l'exécution des simulations, plusieurs instances TornadoR ont été créées et elles ont travaillé séparément sur les différentes copies. On a obtenu donc les différents résultats de simulations dans les différentes copies. Pour cette raison, ils sont centralisés dans le répertoire result avant de supprimer toutes ces copies.

Cette solution est très gourmande en espaces du disque dur, mais sur un système comme Colosse dont la vitesse de lecture et d'écriture sont très rapide, ce souci est négligeable. Le code source de la première solution se trouve à *l'annexe C* dans le fichier ***ParCalculs1.R***. Pour l'exécuter, il suffit de taper la commande suivante :

```
Rscript ParCalculs1.R
```

Note : Il faut charger préalablement le fichier startup .m4w_settings.sh (*voir chapitre 5.1.1*) dans l'environnement d'utilisateur avant d'exécuter la commande Rscript.

```
source /rap/yyk-770-aa/m4w/.m4w_settings.sh
```

6.2.2 Solution 2 : R + Tornado

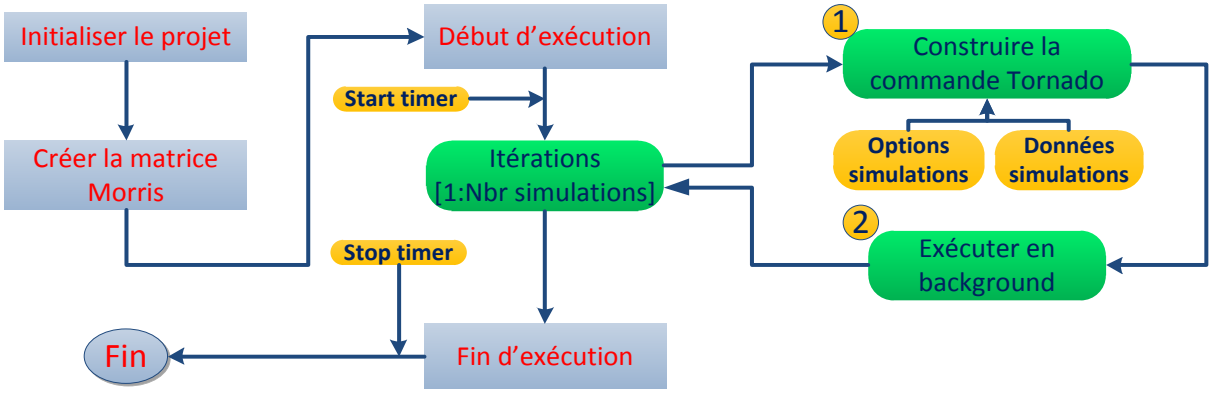


Figure 6.5 Développement en R : Fonctionnement de la solution 2 : R+Tornado

Le fonctionnement du second programme (solution 2) est illustré à la figure 6.5. :

- Initialiser le projet : L'utilisateur définit le nom du modèle de simulation, les paramètres d'analyse et leurs intervalles de valeurs ainsi que le nombre d'itérations pour construire la matrice et le nombre de CPUs souhaités à utiliser.
- Créer une matrice à l'aide de la fonction Morris en fonction des données définies.
- Début de l'exécution et démarrer le compteur de temps de calcul.
- Entrer dans la boucle d'itération
  - Construire la commande de Tornado avec les options des simulations (fichier log, le nom de fichier résultat) et les données des simulations (les valeurs des paramètres importés de la matrice Morris).
  - Exécuter cette commande en mode arrière-plan (background). Le programme appel la fonction ***texec*** du noyau Tornado qui va créer une session de Tornado avec une seule instance Tornado. Puis, la simulation s'exécute dans cette session. Les prochaines simulations arrivent et s'exécutent dans la même session avec la même instance Tornado. Donc avec une seule instance Tornado, N simulations ont été exécutées.
- Toutes les commandes de Tornado sont envoyées, et les simulations s'exécutent parallèlement. Dans ce mode d'exécution, chaque simulation est un thread (processus). À la fin d'exécution, les résultats sont stockés dans le dossier contenant des données de simulations.
- Arrêter le compteur de temps de calcul et afficher la durée totale de l'exécution et le programme libère les ressources et se termine.

On constate qu'avec une seule instance Tornado, elle pourrait traiter autant de simulations désirées, ces simulations parallèles sont les threads qui sont des processus exécutés semi-simultanément. Chaque thread est traité par un cœur (core) du processeur et il contient les différentes valeurs de paramètres de simulation ainsi que le nom de fichier de sortie distingué.

En conséquence, si nous avons N simulations, nous allons avoir N processus exécutés parallèlement (si applicable) produisant N résultats distincts. La seconde solution exploite toute la capacité du processeur vue qu'elle ne permet pas de choisir le nombre de CPUs à exécuter. Le code source de la seconde solution se trouve à *l'annexe C* dans le fichier ***ParCalculs2.R***.

6.3 Phase de test

Ces développements sont suivis d'une série de tests décrits ci-dessous. Ils sont réalisés pour vérifier le fonctionnement et comparer l'efficacité des 2 solutions.

Premièrement nous avons comparé le fonctionnement des 2 solutions, cette comparaison est reprise à la table 6.1.

Table 6.1 Développement en R: Comparaison de fonctionnement des solutions 1 et 2

Solution 1 : R + TornadoR + snowfall	***Solution 2 : R + Tornado***
1. Initialiser le projet	1. Initialiser le projet
2. Créer la matrice Morris	2. Créer la matrice Morris
3. Créer les fonctions parallèles	3. Créer la fonction Tornado
4. Copier les données pour autant de jobs à exécuter	~~Copier les données pour autant de jobs à exécuter~~
5. Exécutions parallèles avec snowfall	4. Exécutions parallèles (Thread)
6. Résultat est centralisé & compressé	5. Résultat est centralisé
7. Supprimer les copies de données	~~Supprimer les copies de données~~

Deuxièmement, une dizaine de tests a été réalisée sur les deux solutions avec un nombre de simulations proportionnel à 10 pour chaque nouveau test. Ces tests sont des simulations du modèle TwoASU dont le code source se trouve à *l'annexe C*.

La table 6.2 reprend les résultats de temps de calcul des simulations ayant été exécutés sur un nœud de calcul de 16 cœurs. La durée moyenne d'une simulation est d'environ une seconde.

Table 6.2 Développement en R: Tableau de temps de calcul des simulations

ID	***Nombre de simulations***	***Solution 1 (secondes)***	***Solution 2 (secondes)***
1	10	3,53	0,38
2	20	6,2	0,94
3	30	7,09	1,83
4	40	9,5	2,87
5	50	13,05	4,16
6	60	13,97	5,75
7	70	16,48	6,71
8	80	17,61	7,42
9	90	20,34	10,61
10	100	23,3	14,68

Un graphique de la comparaison du temps de calcul entre les deux solutions est illustré sous forme d'histogramme à la figure 6.6.

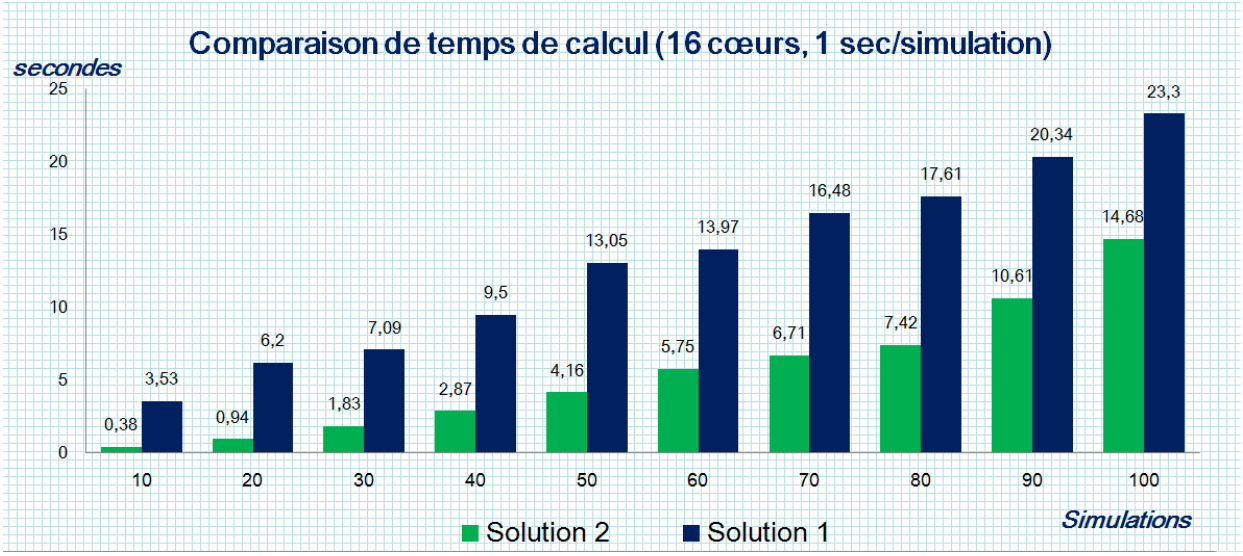


Figure 6.6 Développement en R : Comparaison de temps de calcul des solutions 1 et 2

On remarque dans tous les tests que l'exécution de la seconde solution est toujours plus rapide que la première solution. C'est à cause de la limitation de l'interface TornadoR qui fait une instance de Tornado, une seule simulation à la fois. Techniquement à chaque simulation exécutée par la première solution, chaque instance de Tornado est créée, elle s'ouvre une session, puis elle exécute la simulation dans la session et elle clôture la session à la fin de l'exécution. Et ces étapes se répètent à chaque simulation. Cela influence fortement le temps de calcul.

Table 6.3 Développement en R: Comparaison des solutions 1 et 2

Solution 1 : R + TornadoR + snowfall	***Solution 2 : R + Tornado***
+ Avantages :	**- Inconvénients :**
▪ Nombre de cœurs modifiable ▪ Le résultat a été compressé automatiquement	▪ Pas de l'option de modifier le nombre de cœurs ▪ Le résultat à compresser manuellement
- Inconvénients :	**+ Avantages :**
▪ 1 instance TornadoR, 1 processus ▪ Temps de calcul plus lent que la seconde solution	▪ 1 instance Tornado, N processus ▪ Temps de calcul plus rapide que la première solution

Nous pouvons résumer que :

- Les deux solutions répondent aux objectifs du projet et fonctionnent très bien !
- La seconde solution est plus efficace que la première, mais offre moins d'option. Cependant nous pouvons intégrer le package snowfall dans la solution 2 afin de rendre son utilisation plus flexible. Une version améliorée de la seconde solution avec l'implémentation du package snowfall se trouve à *l'annexe C*.
- L'interface TornadoR devrait être améliorée pour contrer la limitation d'une seule simulation à la fois. Et puisqu'elle est basée sur l'interface TornadoC, celle-ci devrait également être modifiée pour permettre une instance TornadoR d'avoir N exécutions de simulation. À ce moment-là, TornadoR serait très utile et beaucoup plus efficace.

7
Conclusion et perspectives

7.1 Conclusion

Le déploiement de Tornado sur Colosse a été réalisé avec succès, Tornado fonctionne donc à merveille. Deux solutions en R ont été développées, elles permettent de créer et d'exécuter les simulations parallèles en utilisant Tornado et en exploitant la capacité de calcul du superordinateur Colosse. Cela permet d'augmenter l'efficacité de travail de l'équipe model*EAU* en apportant un gain énorme de temps de calcul.

Au point de vu de l'apprentissage, j'ai appris le langage de programmation R qui était aisé à apprendre et puissant. Grâce à son extensibilité, R est capable de travailler dans plusieurs domaines tels que : le calcul parallèle, la base de données, l'analyse de la sensibilité, etc. J'apprécie également le simulateur générique Tornado avec son développement modulaire nécessitant des connaissances de très haut niveau du langage C++, ainsi que sa compilation, son fonctionnement sur les multiples plateformes et sa capacité d'exploitation sur un ensemble de nœuds de calcul.

En travaillant autonome sur mon projet, j'ai eu une grande liberté sur le choix des outils de développements, et des méthodes du travail. Ce stage m'a permis non seulement d'enrichir mes connaissances, d'acquérir plus d'expériences professionnelles, de prendre des initiatives, mais il m'a également permis d'améliorer ma méthode d'apprentissage, ma capacité à résoudre les problèmes, mon sens d'organisation et de responsabilité et de découvrir en détail pendant mes recherches, les différentes connaissances et informations afin d'aboutir au résultat escompté.

L'équipe model*EAU* m'a permis de découvrir un autre domaine sur lequel j'ai eu peu de connaissance : la modélisation et la gestion de la qualité de l'eau. En tout de cas, mes plus grands bénéfices sont l'expérience professionnelle et l'intégration dans la société canadienne.

En fin, ma mission de ce travail de fin d'études est accomplie et tous les objectifs sont remplis. Je voudrais encore remercier Monsieur Peter Vanrolleghem, Monsieur Tichon et tous ceux que j'ai pu côtoyer dans l'équipe model*EAU*. J'ai beaucoup aimé cette équipe et mon lieu de stage et je pense que je n'aurais pas pu mieux trouver.

7.2 Perspectives

Comme décrit aux chapitres 2.2.4 et 2.3.4, le travail sur Colosse nécessite différentes étapes réalisées à l'aide des outils différents, par exemple :

- Étape 1 : Connecter au Colosse avec le logiciel Putty
- Étape 2 : Copier les données au Colosse via le logiciel WinSCP
- Étape 3 : Exécuter des commandes sur Colosse à partir de Putty, et ensuite passer par WinSCP pour récupérer le résultat sur la machine locale, …

Donc, le développement d'un outil de travail est nécessaire pour rendre l'utilisation entre l'utilisateur et le supercalculateur Colosse plus conviviale et efficace. Un prototype est présenté à la figure 7.1. Il combine différentes fonctionnalités et caractéristiques suivantes :

- Gestion de fichiers en interface graphique sur la machine de l'utilisateur et Colosse.
- Gestion d'accès au Colosse ou un serveur à distance.
- Fenêtre d'exécution des commandes sur Colosse dont l'apparence est personnalisable.
- Multiplateforme, cet outil fonctionnera sur Windows, Linux, Mac, etc.
- Conviviale et extensible.

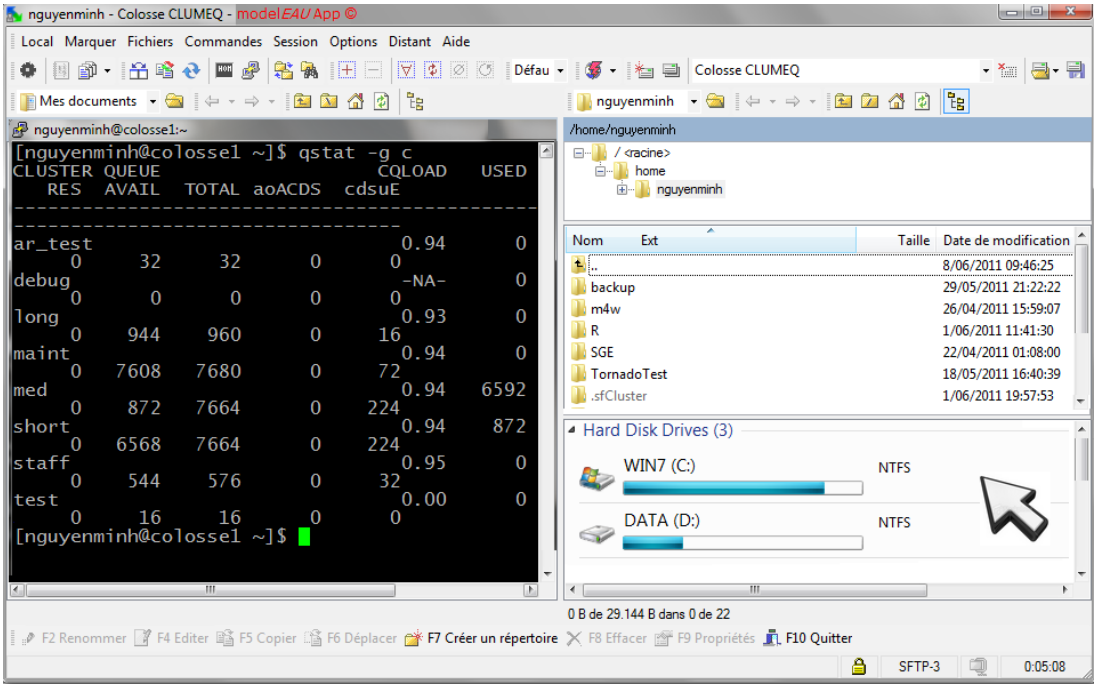


Figure 7.1 Perspective: Interface fictive du programme à développer

Et pourquoi ne pas développer un module complet en R (code, documentation, exemple, etc.) qui permet d'exécuter des différentes simulations parallèles avec des données reçues à partir d'une base de données ou d'un fichier et ce module permet à l'utilisateur de spécifier le choix de simulateur Tornado ou un autre, le type de simulation (expériences virtuelles, des équations, graphiques, physique-chimique, …). Avec les caractéristiques du langage R, cet outil serait très intéressant, il est utile pour différents types de simulations et nous apporte beaucoup de bénéfices en faisant les exécutions parallèles.

Bibliographie

[1] Parent Florent. *Lustre Deployment and early experiences*, PowerPoint presentation, CLUMEQ.CA, Lustre User Group, Aptos Canada, 2010.

[2] Parizeau Marc. *Colosse de Québec*, Présentation PowerPoint, CLUMEQ.CA, Midis-Innovation TI, 24 février 2010

[3] Sun and CLUMEQ. *Feature Story: Sun's HPC Modular Blade System Deployed in a cylindrical, 3-level silo design on the Universite Laval Campus,* http://www.sun.com/featured-articles/2009-1128/feature, 28 November 2009.

[4] Oracle. *Lustre File System, Operations Manual – Version 1.8, Part I: Lustre Architecture*, December 2010.

[5] Oracle. *Oracle Grid Engine: An overview*, White Paper, August 2010.

[6] Claeys Filip. *Tornado Project Website*, http://www2.mostforwater.com/Tornado, June 2011.

[7] Claeys Filip H.A. *A generic software framework for modelling and virtual experimentation with complex biological systems*. PhD thesis. Dept. of Applied Mathematics, Biometrics and Process Control, Ghent University, Belgium, January 2008

[8] Paradis Emmanuel. *R pour les débutants*, Institut des Sciences de l'évolution, Université Montpellier II, France, 12 septembre 2005.

[9] Schmidberger Markus. *Statistics about R Packages for Parallel Computing*, Division of Biometrics and Bioinformatics, IBE, University of Munich, Germany, February 17, 2010

[10] Drepper Ulrich. *How to write Shared Libraries*, Red Hat Inc, 19 November, 2002

[11] GCC Website. *GCC 4.3 Release Series Porting to the New Tools*, Header dependency cleanup, 25 April 2011

[12] Oracle. Linker and Libraries Guide, Chapter 4 Shared Objects, http://download.oracle.com/docs/cd/E19455-01/816-0559/6m71o2af5/index.html,2010

[13] Knaus Jochen. *Developing parallel programs using snowfall*, Institute of Medical Biometry and Medical Informatics, 04 Mars 2010.

[14] Knaus Jochen, Porzelius Christine, Binder Harald and Schwarzer Guido. *Easier Parallel Computing in R with snowfall and sfCluster*, ISSN 2073-4859, The R Journal Vol. 1/1, May 2009

Annexe

Annexe A – Exemples de SGE

Cette annexe reprend quelques exemples de fichiers de soumission de SGE utilisés pendant la phase de test des programmes en langage R et ceux de Tornado.

Tâche Unitaire : Fichier PredatorPrey.sh

```
#!/bin/bash
#$ -N TRPredator
#$ -P yyk-770-aa
#$ -pe default 16
#$ -l h_rt=00:00:10
#$ -cwd
# Charger les modules nécessaires (compilateur GCC, Environnement R, etc.)
source /rap/yyk-770-aa/m4w/.m4w_settings.sh

# La tâche s'exécute avec le programme suivant
Rscript /home/nguyenminh/R/data/PredatorPrey/PredatorPrey.R
```

Tâche unitaire avec les sous tâches parallèles : Fichier TwoASU.sh

```
#!/bin/bash
#$ -N TwoASU
#$ -P yyk-770-aa
#$ -pe default 16
#$ -l h_rt=00:01:60
#$ -cwd
#$ -t 1-100
source /rap/yyk-770-aa/m4w/.m4w_settings.sh

texec TwoASU_$SGE_TASK_ID.Simul.Exp.xml
```

Tâches parallèles : Fichier TwoASUJobs.sh

```
#!/bin/bash
#$ -N 2ASUJobs
#$ -P yyk-770-aa
#$ -pe default 16
#$ -l h_rt=120
#$ -cwd
```

```
source /rap/yyk-770-aa/m4w/.m4w_settings.sh

# Job0
texec -l "Job0.0.log.txt" -i
".Anoxic.Kla=0;.Nitrate_recycle.Q_Out2=55338;#Output#File\$#*This*#Fil
e\$FileName=0;" " Typhoon.Simul.Exp.xml" &

# Job1
texec -l "Job1.1.log.txt" -i
".Anoxic.Kla=8.08081;.Nitrate_recycle.Q_Out2=55338;#Output#File\$#*T
his*#File\$FileName=1;" "Typhoon.Simul.Exp.xml" &

# Job2
texec -l "Job2.2.log.txt" -i
".Anoxic.Kla=16.1616;.Nitrate_recycle.Q_Out2=55338;#Output#File\$#*T
his*#File\$FileName=2;" "Typhoon.Simul.Exp.xml" &

# Job3
texec -l "Job3.3.log.txt" -i
".Anoxic.Kla=24.2424;.Nitrate_recycle.Q_Out2=55338;#Output#File\$#*T
his*#File\$FileName=3;" "Typhoon.Simul.Exp.xml" &

# Job4
texec -l "Job4.4.log.txt" -i
".Anoxic.Kla=32.3232;.Nitrate_recycle.Q_Out2=55338;#Output#File\$#*T
his*#File\$FileName=4;" "Typhoon.Simul.Exp.xml" &
```

Annexe B – Scripts

Cette annexe contient les scripts simplifiant l'utilisation de Tornado sur Colosse pendant la découverte de la plateforme Tornado.

/rap/yyk-770-aa/bin/mof2simul

```
#!/bin/bash
# ----------------------------------------------------------------------------------------
# Bash Script to make Tornado Simulation directly
# with only one command from Modelica model
# Author: Nguyen Mai Quang Minh
# ----------------------------------------------------------------------------------------
echo ---- Hello $USER ----
if [ $# = 0 ]
then echo " [!] Please enter your model MOF file name without extension"
else
        read -p "File Name is correct and continue to build (y/n)?"
        case $REPLY in
```

```
        [yYoO]*)
                echo "Create C code source"
                mof2t -q $1.mof
                echo "Create executable file"
                tbuild -q $1
                echo "Create experience Simulation file"
                tmain -q ExpCreateSimul $1 . false
                echo --- Simulation Created ---
                ls *.Exp.xml
                echo "Enter texec ./" $1".Simul.Exp.xml to execute your
Simulation. Bye";;

        *)      echo "Good Bye";
        esac
fi
```

/rap/yyk-770-aa/bin/west2t

```
#!/bin/bash
# ------------------------------------------------------------------------------------------
# Bash Script to make Tornado Simulation directly
# with only one command from *.wco & *.wxp (West project Files)
# Author: Nguyen Mai Quang Minh
# ------------------------------------------------------------------------------------------
echo ---- Hello $USER -----
if [ $# = 0 ]
then echo " [!] Please enter your project name"
else
        echo "Your current path is:"$PWD;
        echo -n "Date: "; date
        echo "Your simulation Project Name is :"$1;
        read -p "Project Name is correct and continue to build (y/n)?"
        case $REPLY in
         [yYoO]*)
                wco2t -s $1.wco
                source Build.sh
                wxp2t -s -pf $1.wxp
                source Run.sh
                clear
                echo --- Simulations Results ---
                ls *.out.txt
                ;;
         *) echo "Good Bye";
        esac
fi
```

Annexe C – Codes sources des programmes en R

Cette annexe contient les codes sources des deux solutions qui ont été développées en langage R, ainsi que les autres versions qui ont été utilisées pendant la phase de test.

Exécution séquentielle : SeqCalculs.R

```
#####################################################################
# Execute Tornado simulation from R Code (Sequential Execution)
# Description : Version for Colosse (Linux)
# Author: Nguyen Mai Quang Minh
#####################################################################
  rm(list=ls(all=TRUE))
  sink('ComputingR.log', split=TRUE)
  system("rm -r result",ignore.stderr=TRUE);
  system("mkdir result",ignore.stderr=TRUE);

#*********************************************************************
# USER Section : Modify to adapt your project
#*********************************************************************
  # Your project name
  TORNADO_PROJECT <- "Benchmark"

  # The number of Quantities with their names in the String vector (for 4 parameters,
for example)
  param.quantities <- c(".ASU1.M(S_O)", ".ASU1.M(S_NO)", ".ASU1.M(S_ND)",
".ASU1.M(S_NH)")


  # Bounds of parameter ranges
  param.binf        <- c(4.000000,7688.3423523 ,1129.4124512, 8255.124125125)
  param.bsup        <- c(5.000000, 7688.9999999, 1130.123214, 8256.231241254)

  # The number of repetitions of the design
  param.repetitions <- 10
  # Rq : The row number of Matrix is equal to (quantities+1)*repetitions

#---------------------------------------------------------------
# ConcatenationOperator Function (replace paste())
#---------------------------------------------------------------
        "+" <- function(...) UseMethod("+");"+.default" <- .Primitive("+")
        "+.character" <- function(...) paste(...,sep="")

#---------------------------------------------------------------
# TORNADOR SETTINGS
#---------------------------------------------------------------
        TORNADO_MAIN <- "/rap/yyk-770-aa/m4w/Tornado/etc/Tornado.Main.xml"
```

```
	TORNADOR_LIB <- "/rap/yyk-770-aa/R/lib/linux/libTornadoR.so"
	TORNADOR_LOG <- getwd()+"/"+TORNADO_PROJECT+".R.log"
	TORNADOR_SIM <- getwd()+"/"+TORNADO_PROJECT+".Simul.Exp.xml"

#-----------------------------------------------------------------
# Load "sensitivity" package and dependencies
#-----------------------------------------------------------------
	require(sensitivity)
	step <- 0
	cat("\n-------- Step ", step<- step+1,"--------\nLoad Packages","\n")

#-----------------------------------------------------------------
# MORRIS
#-----------------------------------------------------------------
	cat("\n-------- Step ", step<- step+1,"--------\nDesign Morris Matrix","\n")
	# Matrix designed from your parameters values and construct the big matrix X
for the simulation
	mat <- morris(model = NULL, factors = param.quantities, r = param.repetitions,
	design = list(type = "oat", levels = 1000, grid.jump = 3),
	binf = param.binf, bsup = param.bsup)

	# Extract data.frame X
	Matrix	<- mat$X ; write.table(Matrix,"result/Morris_matrix.txt")

#-----------------------------------------------------------------
# Load TornadoR library and license
#-----------------------------------------------------------------
	dyn.load(TORNADOR_LIB)

	# Start Tornado & Initialization
	.C("TRInitialize", TORNADOR_LOG, TORNADO_MAIN, as.integer(1),
as.integer(0))
	cat("\n-------- Step ", step<- step+1,"--------\nTornadoR is started","\n")

	# Load Tornado Simulation Experiment
	cat("\n-------- Step ", step<- step+1,"--------\nLoading simulation","\n")
	.C("TRExpLoad", TORNADOR_SIM)

	# Start the timer
	begin<-Sys.time()

	cat("\n-------- Step ", step<- step+1,"--------\nExecute Simulations","\n")
	SimulID<-1; loop <- (length(param.quantities)+1)*param.repetitions;
	# Set the new values in the experiment
	for (line in 1:loop) {
		for (column in 1:length(param.quantities)) {
		 .C("TRExpSetInitialValue",param.quantities[column],
as.double(Matrix[line,column]))
		}
```

```
        # Change output Name
        #.C("TRExpSetInitialValue", "#Output#File$#*This*#File$FileName",
as.double(SimulID))
         .C("TRExpInitialize") # Init Experiment
         .C("TRExpRun") # Execute Experiment

          # Get back the simulation Result and put in result folder
          system("mv "+getwd()+"/"+TORNADO_PROJECT+".*out*
"+getwd()+"/result/"+TORNADO_PROJECT+".Simul."+SimulID+".out.txt")
          cat("\n > Simulation "+SimulID+": Result data in
result/"+TORNADO_PROJECT+".Simul."+SimulID+".out.txt\n")
          SimulID <- SimulID+1
        }
        # Stop the timer
        end<-Sys.time()

        # Final step : Stop Tornado
  .C("TRFinalize")
  cat("\n-------- Step ", step<- step+1,"--------\nTornadoR is stopped","\n")

        # Make the Timer calcul and save in result/durationCalcul.txt
        duration<-paste("Duration of the Calculation for  ",SimulID<-SimulID-1,"
Simulations :
",as.character(round(difftime(end,begin,units="sec"),digits=3)),"seconds")
        system("echo "+duration+" > result/durationCalcul.txt"); duration

        # zip the result in one zip file
        system("zip -rq Result_"+TORNADO_PROJECT+".zip result")
        cat("All Results Data is in the zip file : "+"Result_"+TORNADO_PROJECT+".zip")
        cat("\n\nBye :) \n\n")
```

Exécution parallèle avec la solution 1 : ParCalculs1.R

```
#####################################################################
# Execute Parallel Computing in TornadoR with snowfall extension package
# Description : Version for Colosse (Linux)
# Author: Nguyen Mai Quang Minh
#####################################################################
  rm(list=ls(all=TRUE))
  system("rm -rf result .tmp *.zip",ignore.stderr=TRUE);
  system("mkdir result",ignore.stderr=TRUE);
  sink("ParallelComputingR.log", split=TRUE)

  #-----------------------------------------------------------------
  # Initialization of snowfall
  #-----------------------------------------------------------------
  library(snowfall)

  #*****************************************************************
  # USER Section : Modify to adapt your project
  #*****************************************************************

  # Define the number of CPU in 'cpus' parameter
  sfInit(parallel=TRUE,cpus=16,type="SOCK")

  # Your project name
  TORNADO_PROJECT <- "Benchmark"

  # The number of Quantities with their names in the String vector (for 4 parameters,
for example)
  param.quantities <- c(".ASU1.M(S_O)", ".ASU1.M(S_NO)", ".ASU1.M(S_ND)",
".ASU1.M(S_NH)")

  # Bounds of parameter ranges
  param.binf        <- c(4.000000,7688.3423523 ,1129.4124512, 8255.124125125)
  param.bsup        <- c(5.000000, 7688.9999999, 1130.123214, 8256.231241254)

  # The number of repetitions of the design
  # Rq : The row number of Matrix is equal to (quantities+1)*repetitions
  param.repetitions <- 10

  #-----------------------------------------------------------------
  # DESIGN MORRIS MATRIX
  #-----------------------------------------------------------------
  require(sensitivity)
  # Matrix designed from your parameters values and construct the big matrix X for
the simulation
  mat <- morris(model = NULL, factors = param.quantities, r = param.repetitions,
  design = list(type = "oat", levels = 1000, grid.jump = 3),
  binf = param.binf, bsup = param.bsup)
```

```
# Extract data.frame X to file
Matrix       <- mat$X ;  write.table(Matrix,"result/Morris_matrix.txt")

#-----------------------------------------------------------------
# Function : ConcatenationOperator  (replace paste())
#-----------------------------------------------------------------
"+" <- function(...) UseMethod("+"); "+.default" <- .Primitive("+")
"+.character" <- function(...) paste(...,sep="")

#-----------------------------------------------------------------
# Function : Tornado Simulation
#-----------------------------------------------------------------
tornado <- function(SimulID) {

  cat("\n-------- Worker ", SimulID,"*> -------- Executing simulation .....","\n")
  #-----------------------------------------------------------------
  # TORNADOR SETTINGS
  #-----------------------------------------------------------------
  TORNADO_MAIN <- "/rap/yyk-770-aa/m4w/Tornado/etc/Tornado.Main.xml"
  TORNADOR_LIB <- "/rap/yyk-770-aa/R/lib/linux/libTornadoR.so"
  TORNADOR_LOG <- getwd()+"/result/"+TORNADO_PROJECT+"."+SimulID+".R.log"
  TORNADOR_SIM <-
getwd()+"/.tmp/worker"+SimulID+"/"+TORNADO_PROJECT+".Simul.Exp.xml"
  dyn.load(TORNADOR_LIB) # Load TornadoR library

      # Start Tornado & Initialization
      .C("TRInitialize", TORNADOR_LOG, TORNADO_MAIN, as.integer(1),
as.integer(0))

      # Load Tornado Simulation Experiment
      .C("TRExpLoad", TORNADOR_SIM)

      # Set the new values in the experiment
      for (column in 1:length(param.quantities)) {
    .C("TRExpSetInitialValue",param.quantities[column],
as.double(Matrix[SimulID,column]));
    }
  # Change output Name
  #.C("TRExpSetInitialValue", "#Output#File$#*This*#File$FileName",
as.double(SimulID))
  .C("TRExpInitialize"); # Init Experiment
  .C("TRExpRun"); #  Execute Experiment
  .C("TRFinalize"); # Finalize Experiment & Stop Tornado
  }

#-----------------------------------------------------------------
# Function :  Create Workspace for each worker
#-----------------------------------------------------------------
```

```
  system("mkdir .tmp",ignore.stderr=TRUE);
  worker <- function(ID){
      system("mkdir .tmp/worker"+ID,ignore.stderr=TRUE);
      system("cp -rf *"+TORNADO_PROJECT+"*.* *Simul*.* *.txt
.tmp/worker"+ID,ignore.stderr=TRUE);
  }

  #---------------------------------------------------------------
  # Function : Get Simulation Results
  #---------------------------------------------------------------
  getResult <- function(ID){
      system("cp  "+getwd()+"/.tmp/worker"+ID+"/"+TORNADO_PROJECT+".*out*
"+getwd()+"/result/"+TORNADO_PROJECT+".Simul."+ID+".out.txt");
      system("rm -rf "+getwd()+"/.tmp/worker"+ID);
  }

  #---------------------------------------------------------------
  # Call Function : Create Workspace for each worker
  # Start network random number generator & distribute calculation
  # copy simulation data to each worker
  #---------------------------------------------------------------
  sfExportAll(); sfClusterSetupRNG()
  sfLapply(1:((length(param.quantities)+1)*param.repetitions), worker);

  #---------------------------------------------------------------
  # Call Function : Tornado simulations
  # Load sensitivity package, then execute Simulation and make calcul time
  #---------------------------------------------------------------
  sfLibrary(sensitivity);
  start <- Sys.time();  sfLapply(1:((length(param.quantities)+1)*param.repetitions),
tornado) ;
  duration<-Sys.time()-start; system("echo "+duration+" > result/durationCalcul.txt");
  #---------------------------------------------------------------
  # Call Function : Get Simulation Results
  # Get back the simulation Result and put in result folder and zip the result in one
zip file
  #---------------------------------------------------------------
  sfLapply(1:((length(param.quantities)+1)*param.repetitions), getResult);
  #---------------------------------------------------------------
  #Stop Parallele Computing & snowfall package
  #---------------------------------------------------------------
  sfRemoveAll();  sfStop();
  #---------------------------------------------------------------
  # Zip all the results in one file
  #---------------------------------------------------------------
  TORNADOR_OUT<- ("zip -rq Result_"+TORNADO_PROJECT+".zip result");
  system(TORNADOR_OUT)
  cat("All Results Data is in the zip file : "+"Result_"+TORNADO_PROJECT+".zip")
  cat("\n\nBye :) \n\n")
```

Exécution parallèle avec la solution 2 : ParCalculs2.R

```
#####################################################################
# Execute Parallel Computing in R & Tornado running one instance Tornado
# & N processus Simulation in background ( with &)
# Description : Version for Colosse (Linux)
# Author: Nguyen Mai Quang Minh
#####################################################################
  rm(list=ls(all=TRUE))
  system("rm -rf result *.log.txt *.out.txt",ignore.stderr=TRUE);
  system("mkdir result",ignore.stderr=TRUE);
  sink("ParalleComputingR.log", split=TRUE)

  #*********************************************************************
  # USER Section : Modify to adapt your project
  #*********************************************************************

  # Your project name
  TORNADO_PROJECT <- "TwoASU"

  # The number of Quantities with their names in the String vector (for 4 parameters,
for example)
  param.quantities <- c(".Anoxic.M(S_I)",
".Anoxic.M(S_O)",".Anoxic.M(S_NO)",".Anoxic.M(X_ND)",".Aerobic.M(X_P)")

  # Bounds of parameter ranges
  param.binf        <- c(50000.0, 18.0,19000,8384,2016560)
  param.bsup        <- c(50100.0, 18.5,19001,8386,2016570)

  # The number of repetitions of the design
  # Rq : The row number of Matrix is equal to (quantities+1)*repetitions
  param.repetitions <- 100

  #---------------------------------------------------------------
  # DESIGN MORRIS MATRIX
  #---------------------------------------------------------------
  require(sensitivity)
  # Matrix designed from your parameters values and construct the big matrix X for
the simulation
  mat <- morris(model = NULL, factors = param.quantities, r = param.repetitions,
  design = list(type = "oat", levels = 1000, grid.jump = 3),
  binf = param.binf, bsup = param.bsup)

  # Extract data.frame X to file
  Matrix <- mat$X ;  write.table(Matrix,"result/Morris_matrix.txt")

  #---------------------------------------------------------------
  # Function : ConcatenationOperator  (replace paste())
  #---------------------------------------------------------------
```

```
  "+" <- function(...) UseMethod("+"); "+.default" <- .Primitive("+")
  "+.character" <- function(...) paste(...,sep="")

  #-----------------------------------------------------------------
  # Function : Tornado Simulation
  #-----------------------------------------------------------------
  loop <- (length(param.quantities)+1)*param.repetitions;
  start <- Sys.time()
 for (line in 1:loop) {
    Tornado <- ""
    Tornado <- "texec -q"
#    Tornado <- Tornado +" -l \"Job."+line+".log.txt\""
    Tornado <- Tornado +" -i \""
         for (column in 1:length(param.quantities)) {
      Tornado <- Tornado +
param.quantities[column]+"="+round(Matrix[line,column],4)+";"
     }
    Tornado <- Tornado +"#Output#File\\$#*This*#File\\$FileName="+line+";\"
\""+TORNADO_PROJECT+".Simul.Exp.xml"+"\" &"
    cat(" > Sim ",line,"\n",Tornado,"\n")
    system(Tornado)
  }
  duration <- Sys.time()-start;
  system("echo "+duration+" > result/durationCalcul.txt");
  duration
  cat("\n\nBye :) \n\n")
```

Exécution parallèle avec la solution 2 (améliorée): ParCalculs2.R

```
#####################################################################
# Execute Parallel Computing in R & Tornado running one instance Tornado
# & N Simulation ( with &) + Snowfall Package
# Description : Version for Colosse (Linux)
# Author: Nguyen Mai Quang Minh
#####################################################################
  rm(list=ls(all=TRUE))
  system("rm -rf result *.log.txt *.out.txt",ignore.stderr=TRUE);
  system("mkdir result",ignore.stderr=TRUE);
  sink("ParalleComputingR.log", split=TRUE)

  #-----------------------------------------------------------------
  # Initialization of snowfall
  #-----------------------------------------------------------------
  library(snowfall)

  #*******************************************************************
  # USER Section : Modify to adapt your project
  #*******************************************************************
  # Define the number of CPU in 'cpus' parameter
  sfInit(parallel=TRUE,cpus=16,type="SOCK")

  # Your project name
  TORNADO_PROJECT <- "TwoASU"

  # The number of Quantities with their names in the String vector (for 4 parameters,
for example)
  param.quantities <- c(".Anoxic.M(S_I)",
".Anoxic.M(S_O)",".Anoxic.M(S_NO)",".Anoxic.M(X_ND)",".Aerobic.M(X_P)")

  # Bounds of parameter ranges
  param.binf        <- c(50000.0, 18.0,19000,8384,2016560)
  param.bsup        <- c(50100.0, 18.5,19001,8386,2016570)

  # The number of repetitions of the design
  # Rq : The row number of Matrix is equal to (quantities+1)*repetitions
  param.repetitions <- 100

  #-----------------------------------------------------------------
  # DESIGN MORRIS MATRIX
  #-----------------------------------------------------------------
  require(sensitivity)
  # Matrix designed from your parameters values and construct the big matrix X for
the simulation
  mat <- morris(model = NULL, factors = param.quantities, r = param.repetitions,
  design = list(type = "oat", levels = 1000, grid.jump = 3),
  binf = param.binf, bsup = param.bsup)
```

```
  # Extract data.frame X to file
  Matrix <- mat$X ;  write.table(Matrix,"result/Morris_matrix.txt")

  #-----------------------------------------------------------------
  # Function : ConcatenationOperator  (replace paste())
  #-----------------------------------------------------------------
  "+" <- function(...) UseMethod("+"); "+.default" <- .Primitive("+")
  "+.character" <- function(...) paste(...,sep="")

  #-----------------------------------------------------------------
  # Function : Tornado Simulation
  #-----------------------------------------------------------------
 tornado <-function(SimulID){
    Tornado <- ""
    Tornado <- "texec -q"
    #Tornado <- Tornado +" -l \"Job."+SimulID+".log.txt\""
    Tornado <- Tornado +" -i \""
         for (column in 1:length(param.quantities)) {
      Tornado <- Tornado +
param.quantities[column]+"="+round(Matrix[SimulID,column],4)+";"
     }
    Tornado <- Tornado +"#Output#File\\$#*This*#File\\$FileName="+SimulID+";\"
\""+TORNADO_PROJECT+".Simul.Exp.xml"+"\" &"
    system(Tornado)
  }

  #-----------------------------------------------------------------
  # Call Function : Tornado simulations
  # Load sensitivity package, then execute Simulation and make calcul time
  #-----------------------------------------------------------------
  sfExportAll(); sfClusterSetupRNG(); sfLibrary(sensitivity);
  start <- Sys.time();  sfLapply(1:((length(param.quantities)+1)*param.repetitions),
tornado) ;
  duration<-Sys.time()-start; system("echo "+duration+" > result/durationCalcul.txt");

  #-----------------------------------------------------------------
  #Stop Parallele Computing & snowfall package
  #-----------------------------------------------------------------
  sfRemoveAll();  sfStop();

  system("echo "+duration+" > result/durationCalcul.txt");
  duration
  cat("\n\nBye :) \n\n")
```

Annexe D – Tutoriel d'utilisation des solutions développées en R

Cette annexe reprend un tutoriel expliquant pas à pas comment accéder au supercalculateur Colosse, comment configurer son environnement pour travailler avec Tornado et R afin d'employer les deux programmes qui ont été développés en R.

1. Comment accéder au Colosse

Comme expliqué au chapitre 2.2.4, il y a 2 types de connexion pour accéder au Colosse : SSH et SFTP. Nous utilisons la connexion SSH essentiellement pour nous connecter à la console pour exécuter les commandes et envoyer des tâches au SGE de Colosse. Et nous utilisons SFTP pour le transfert de données entre utilisateur et Colosse, cela veut dire nous pouvons envoyer les données au Colosse ou télécharger des données du Colosse sur la machine locale.

1.1. Transfert de fichiers

Connexion de type SFTP nous permet d'accéder au système de fichiers du Colosse pour télécharger les fichiers de ou vers Colosse. Nombreux logiciels permettent de faire cette connexion, tels que :

- WinSCP (Plateforme Windows) disponible à : http://winscp.net
- FileZilla (multiplateformes) disponible à : http://filezilla-project.org

Dans ce projet, logiciel WinSCP a été utilisé pour la gestion de transfert de fichiers entre la machine d'utilisateur et Colosse. Voici la démarche à suivre pour se connecter au Colosse en utilisant WinSCP, chaque étape est illustrée par les images.

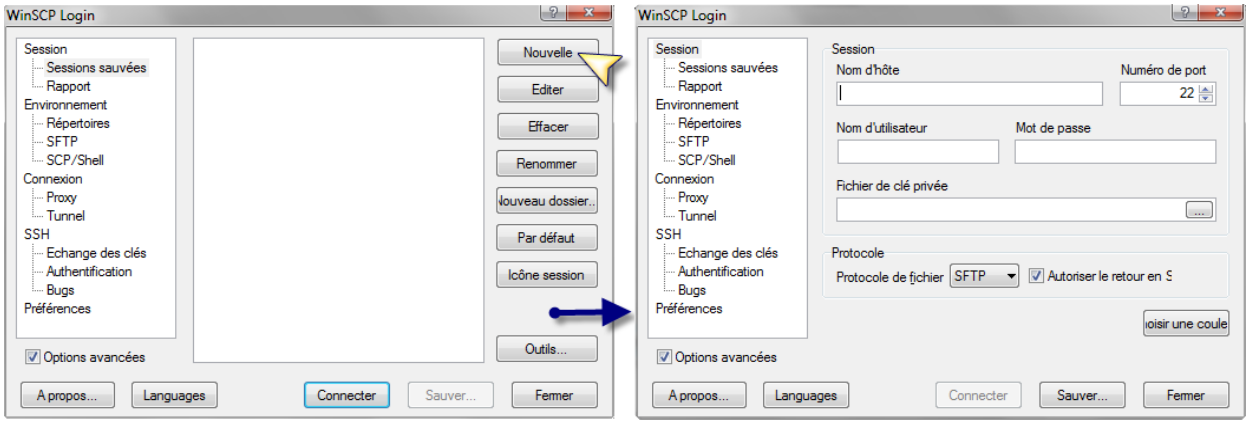


Figure Annexe D.1 Tutoriel : Nouvelle connexion SFTP avec WinSCP

Après l'exécution de l'application WinSCP, cliquez sur "Nouvelle" pour créer une nouvelle session et entrez les paramètres suivants :

- Nom d'hôte : colosse.clumeq.ca
- Numéro de port : 22
- Nom d'utilisateur & mot de passe : Vos données reçues du CLUMEQ.
- Protocole de fichier : SFTP

Laissez les autres options par défaut. Si vous souhaitez enregistrer ces données pour les prochains accès au Colosse, cliquez alors sur le bouton "Sauver".

Cliquez ensuite sur bouton "Connecter" pour accéder au système de fichiers du Colosse en SFTP. Si l'application vous demande d'ajouter une clé de sécurité, cliquez sur "Oui" pour continuer.

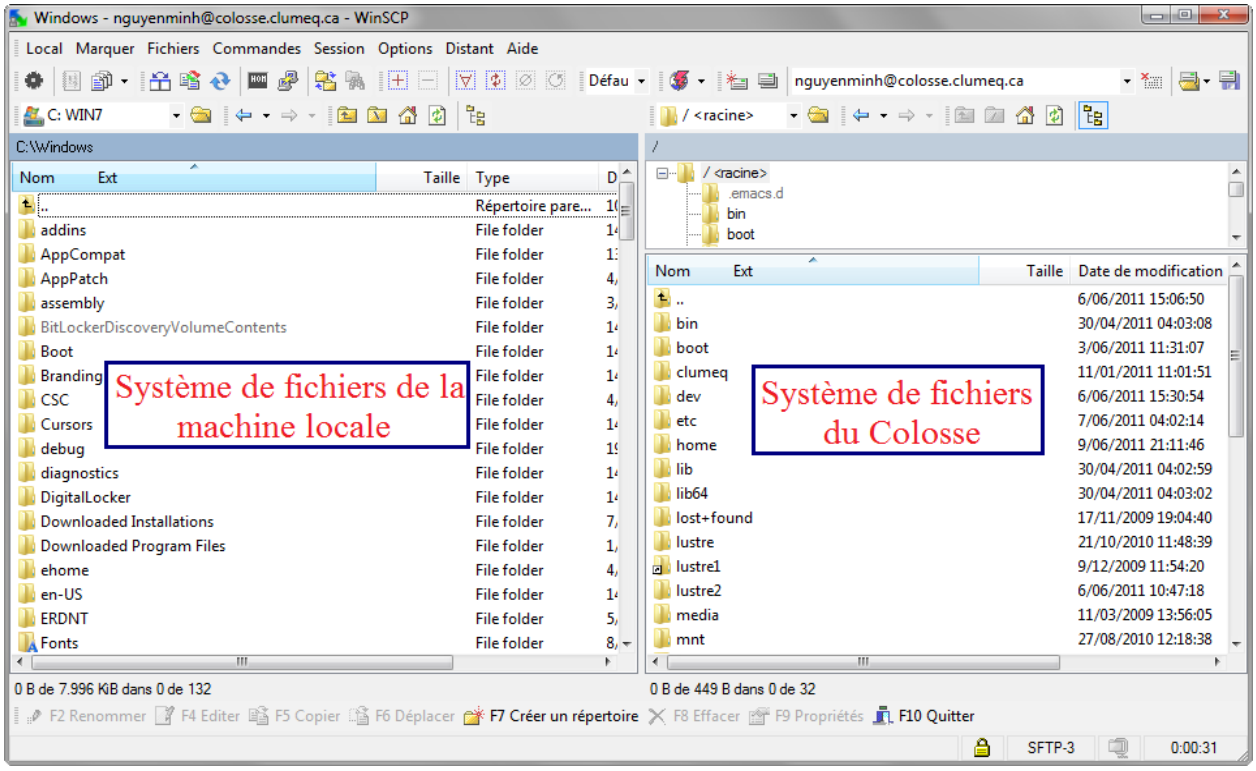


Figure Annexe D.2 Tutoriel : Interface du WinSCP

Arriver à cette interface, vous êtes connecté au Colosse ! Vous constatez que la fenêtre à gauche est le système de fichier de la machine locale (dans ce cas-ci est Windows Seven) et la fenêtre à droite est le système de fichiers du Colosse.

Les différentes opérations disponibles vous permettant de manipuler des données entre votre machine et Colosse sont utilisables avec les raccourcis suivants :

- F2 : Renommer un fichier ou dossier
- F4 : Éditer le fichier choisi, vous pouvez également changer le programme d'édition par défaut de WinSCP – notepad – avec votre programme préféré.
- F5 : Copier vos fichiers/dossiers de ou vers Colosse.
- F6 : Déplacer vos fichiers/dossiers de ou vers Colosse.
- ….

1.2. Exécuter les commandes ou envoyer des jobs sur Colosse

Une connexion de type SSH nous permet de connecter à la console du Colosse (interface en ligne de commandes) pour exécuter les commandes telles que lancer des simulations Tornado ou encore envoyer des tâches au SGE.

Pour les utilisateurs Linux, un utilitaire du système linux est la commande **ssh**, tapez la commande suivante pour accéder au Colosse.

```
$> ssh VOTRE_LOGIN@colosse.clumeq.ca
```

Vous devez ensuite entrer votre mot de passe, une fois vous êtes authentifié, vous serez redirigé vers votre répertoire personnel. S'il y a une erreur vous disant que la commande ssh n'existe pas, cela signifie que votre système linux ne dispose pas de cette commande. Vous devez l'installer, l'installation est très simple, mais en fonction de votre distribution de linux (Redhat, Ubuntu, Debian, etc.), la procédure d'installation est différente. Faites une recherche, dans Google par exemple, avec la requête suivante : " ***VOTRE_DISTRIBUTION_LINUX*** AND Install OR Tutorial AND SSH ". Vous devrez trouver la réponse.

Pour les utilisateurs Windows, l'utilitaire le plus utilisé est PuTTY, disponible gratuitement à http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html. Par la suite, nous allons utiliser ce programme pour connecter au Colosse.

Après avoir téléchargé le fichier **putty.exe** à l'adresse ci-dessus, exécutez cette application, vous devez voir l'interface suivante :

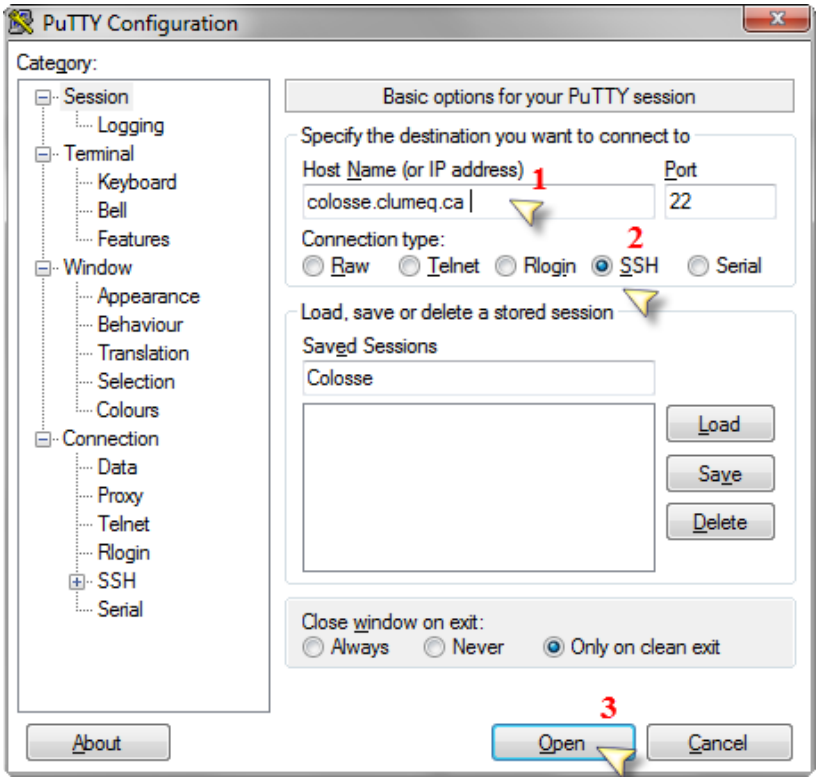


Figure Annexe D.3 Tutoriel : Nouvelle connexion SSH avec PuTTY

Entrez les paramètres suivants :

- Hostname : colosse.clumeq.ca
- Port : 22
- Connection type : SSH

Si vous désirez de sauver ces données pour les prochains accès, entrez un nom dans "Saved Sessions" et cliquez sur le bouton "Save". Sinon, passez cette étape.

Cliquez ensuite sur "Open" pour établir une connexion en SSH au Colosse.

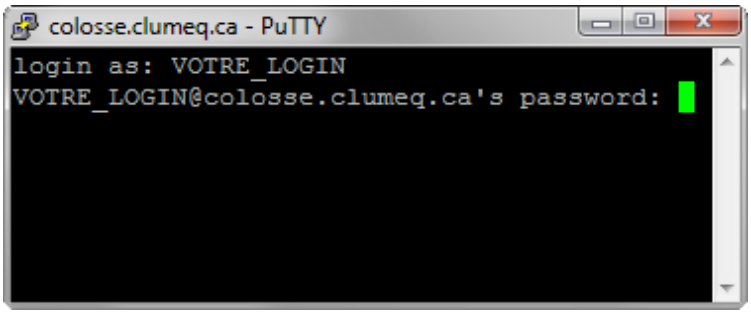


Figure Annexe D.4 Tutoriel : Authentification sur Colosse avec PuTTY

Colosse vous demander un login, entrez le vôtre et tapez "Enter". Puis Colosse vous demande votre mot de passe, tapez votre mot de passe et vous remarquez que rien ne s'affiche dans la partie "password", c'est tout à fait normal sur un système de Linux comme Colosse. Vous ne devez pas vous inquiéter, continuez à taper et faites "Enter" pour confirmer.

S'il y a une erreur dans votre login ou mot de passe, Colosse va vous demander de retaper ces données. Et après un nombre d'essais sans succès, Colosse rejette votre connexion. Et vous devez établir une nouvelle connexion.

Si vous êtes authentifié, alors vous êtes connecté au Colosse et vous devez voir la fenêtre suivante :

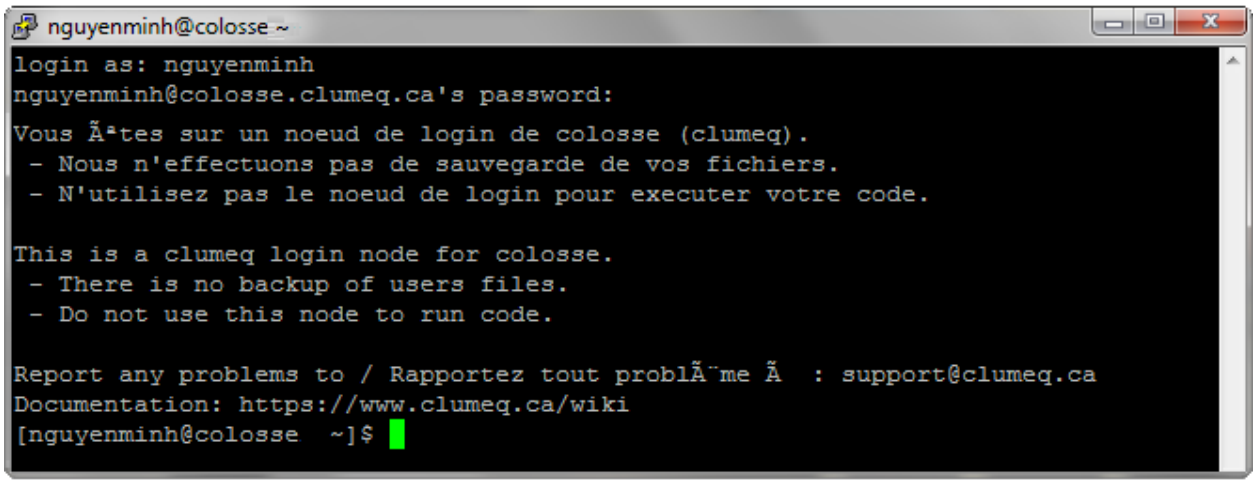


Figure Annexe D.5 Tutoriel : Interface en ligne de commande du Colosse

2. Configurer pour travailler avec Tornado et R sur Colosse

Quand vous êtes connecté au Colosse en SSH et vous désirez travailler avec Tornado et R, alors vous devez charger le fichier startup (*voir chapitre 5.1.1*). Si vous utilisez occasionnellement Tornado et R, par exemple vous voulez juste voir comment ça fonctionne Tornado ou R, alors tapez la commande suivante dans la fenêtre de PuTTY.

```
source /rap/yyk-770-aa/m4w/.m4w_settings.sh
```

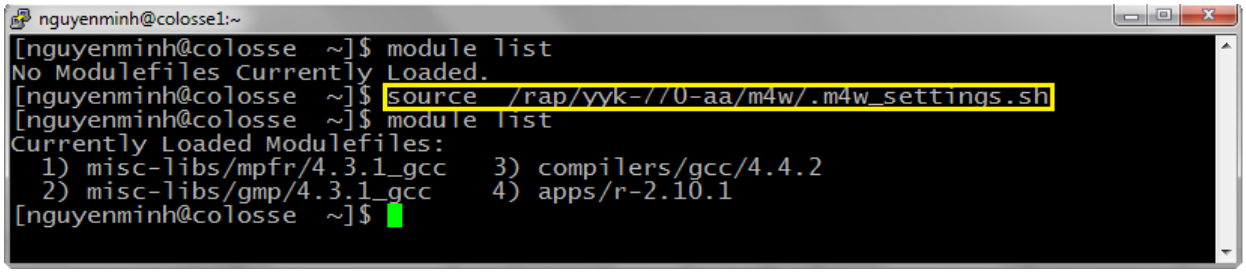


Figure Annexe D.6 Tutoriel : Charger le fichier startup .m4w_settings.sh

Nous constatons dans la figure D.6, avant de charger le fichier startup, il y a aucun module dans la session d'utilisateur (commande **module list**), et après le chargement du fichier startup, les modules tels que compilateur GCC nécessaire pour le fonctionnement de Tornado et le logiciel R ont été chargés dans la session d'utilisateur.

Si vous désirez d'automatiser cette commande, il suffirait de l'ajouter à la fin du fichier $HOME/.bash_profile, pour ce fait suivez les commandes suivantes :

```
nano $HOME/.bash_profile
# Ajouter cette ligne à la fin du fichier et sauvegarder, il est appliqué dans la session suivante
source /rap/yyk-770-aa/m4w/.m4w_settings.sh
# Quitter : clavier Ctrl + X
# Confirmer la sauvegarde : clavier Y
```

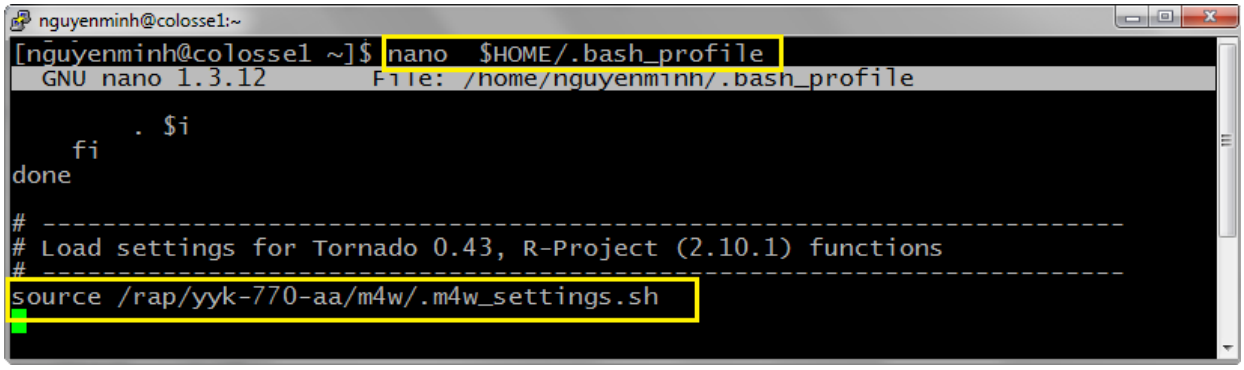


Figure Annexe D.7 Tutoriel : Automatiser le fichier startup .m4w_settings.sh

3. Comment employer les deux programmes d'exécution parallèle

Pour exécuter en parallèle les deux programmes en R de simulation, vous devez :

- Avoir un dossier qui contient tous les modèles de simulation, par exemple le fichier TwoASU.Simul.Exp.xml pour le modèle TwoASU ou bien Benchmark.Simul.Exp.xml pour le modèle Benchmark avec les fichiers de données éventuellement.
- Copier les codes de deux solutions dans ce dossier.

Ensuite, vous devez entrer les données à analyser en modifiant les valeurs des deux solutions. Voici les paramètres à adapter à votre projet :

- **cpus** : Nombre de cœurs utilisés pour l'exécution parallèle.
- **TORNADO_PROJECT** : Nom du modèle de Tornado.
- **param.quantities** : Les paramètres d'analyse.
- **param.binf + param.bsup** : Les intervalles de valeurs des paramètres.
- **param.repetitions** : Nombre d'itérations pour construire la matrice Morris. (*Voir le chapitre 6.1.2*)

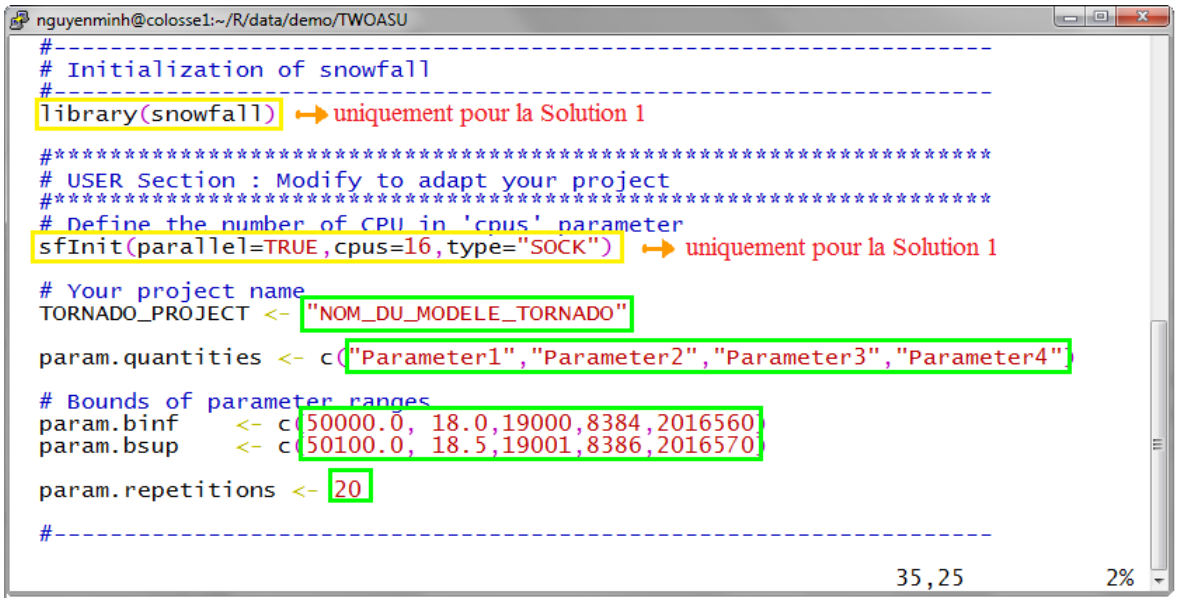


Figure Annexe D.8 Tutoriel : Les paramètres à adapter dans un nouveau projet

Vous venez de créer votre projet, il est prêt pour exécuter, si vous souhaitez exécuter localement (sur Cyclops, *voir chapitre 2.2.4*) dans le but de vérifier la validité des données entrées, alors voici les commandes pour exécuter votre projet avec la solution 1 et la solution 2

```
# La solution 1
  Rscript ParCalculs1.R
# La solution 2
  Rscript ParCalculs2.R
```

Si les simulations sont réalisées avec succès, les résultats des simulations, dont l'extension est .Simul.out.txt, seront stockés dans le dossier **result** pour la solution 1 et dans le même dossier contenant les données de simulations pour la solution 2.

Vous souhaitez exécuter les simulations sur Colosse en exploitant 256 cœurs par exemple, alors vous devez écrire un fichier de soumission (*voir chapitre 2.3.4*) dans le même dossier contenant les données de simulations et les 2 programmes en R. Un aperçu du fichier de soumission avec les paramètres à adapter selon votre projet est présenté à la figure D.9.

```
#!/bin/bash
#$ -N NOM_DE_LA_TACHE
#$ -P IDENTIFIANT_DU_GROUPE_DE_RECHERCHE
#$ -pe default NOMBRE_DE_COEURS
#$ -l h_rt=HEURES:MINUTES:SECONDES

#$ -cwd
# Les modules nécessaires seront chargés dans les noeuds de calculs alloués
source /rap/yyk-770-aa/m4w/.m4w_settings.sh

# Commandes pour exécuter
# Appeler le programme exécutant les simulations parallèles
# NOM_DU_PROGRAMME_EN_R est soit ParCalculs1.R, soit ParCalculs2.R
Rscript NOM_DU_PROGRAMME_EN_R
```

Figure Annexe D.9 Tutoriel : Les paramètres à adapter dans le fichier de soumission (SGE)

Dans ce fichier de soumission, les paramètres suivants sont obligatoires

- NOM_DE_LA_TACHE : Comment vous appelez votre tâche ?
- IDENTIFIANT_DU_GROUPE_DE_RECHECHE : Votre RAP ID.
- NOMBRE_DE_COEURS : un nombre multiple de 8.
- HEURES:MINUTES:SECONDES : Le temps estimé de l'exécution.
- NOM_DU_PROGRAMME_EN_R : ParCalculs1.R ou ParCalculs2.R

Après avoir adapté ces paramètres en fonction de votre calcul, nommez ce fichier, par exemple : MyJob.sh. Et finalement envoyez cette tâche au système de gestion des tâches Sun Grid Engine (SGE) du Colosse à l'aide de la commande **qsub** (voir chapitre 2.3.4, seconde étape), ensuite vous pouvez utiliser la commande **sge** pour vérifier l'état actuel de votre tâche.

```
qsub MyJob.sh
sge
```

En général, cette tâche ne sera pas exécutée immédiatement, mais mise dans une file d'attente. Et c'est seulement lorsque les ressources nécessaires sont disponibles que la tâche sera mise en exécution par SGE. Lorsque l'exécution de la tâche est terminée, SGE renverra le résultat de l'exécution au répertoire où le fichier de soumission a été créé et vous pouvez y aller récupérer ces résultats.