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 du framework Tornado sur une grappe de calcul utilisant Sun Grid Engine

M. Vincent BOUCKAERT

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 2009-2010

———————

Numéro : ISIB-ELIN-TFE-10/01
Classification : confidentiel

———————

Résumé

La modélisation de systèmes d'eau fait appel à des capacités de calcul de plus en plus importantes. Un parallélisme peut d'ailleurs être établi entre la croissance de la complexité des modèles étudiés et la croissance de la puissance de calcul des microprocesseurs. Il est donc intéressant d'implanter ces modèles et leurs expérimentations virtuelles (telles que l'optimisation, l'analyse de scénarios multiples, l'analyse de sensibilité et d'incertitude) sur les outils existants les plus performants.

Tornado est un *framework* générique utilisé par le groupe de recherche modelEAU de l'Université Laval, Québec, pour l'exécution de simulations relatives à la gestion de la qualité de l'eau. L'installation d'une grappe de calcul de haute performance (Colosse) au sein de l'Université Laval a amené à considérer le déploiement du *framework* Tornado sur un tel système. Ce travail détaille les étapes de ce déploiement et les limites d'utilisation actuelles.

L'installation de Tornado sur Colosse nécessite une série d'adaptations au niveau du code et du mécanisme de compilation. Ces adaptations sont décrites et expliquées dans ces pages. Cette installation a été suivie d'une série de tests, réalisés avec succès dans un but de validation. Les résultats de ces tests sont présentés et commentés, de même que certains problèmes persistant à l'heure actuelle.
La grappe de calcul faisant usage de l'ordonnanceur de tâches Sun Grid Engine, un outil basé dans Tornado a également été développé afin de simplifier son utilisation sur ce système, et de réaliser des simulations portant sur des modèles de grande taille dans les délais les plus brefs.

Abstract

Water systems modeling needs important and increasing computing power. A parallelism can even be established between the growth of the complexity of used models and the growth of microprocessors' calculation power. Therefore, it is interesting to implement those models and their virtual experiments (such as optimization, scenario, sensitivity and uncertainty analysis) on the most powerful calculation tools.

Tornado is a generic framework used by the research team modelEAU from Université Laval, Quebec, in order to process simulations related to water quality management. The deployment of a high performance computer cluster (Colossus) at Université Laval has lead to the consideration of installing Tornado on such a system. This work details the steps taken in order to fulfill this installation and exposes the current limits of use.

The deployment of Tornado on Colossus requires to make some adaptations in its code as well as in its compilation mechanism. Those are presented in this work. The deployment was followed by a successful series of tests, made in view of its validation. The results of these tests are presented and discussed, as are a few remaining problems.
Also, Colossus uses the Sun Grid Engine task scheduler. A Tornado-based tool has been developed in order to ease the use of Tornado on this system, which will allow to run large model simulation jobs in the shortest possible time.

En préambule à ce document, je souhaite adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce travail de fin d'études.

Je tiens tout d'abord à remercier le Professeur Peter Vanrolleghem pour m'avoir accueilli si chaleureusement au sein du groupe de recherche modelEAU à l'Université Laval à Québec, et pour m'avoir proposé un sujet aussi exceptionnel. Je lui suis reconnaissant pour le temps, la patience et l'énergie qu'il a investi pour me guider.

Je remercie aussi mes professeurs Dr. Ing. Jacques Tichon et Ir. Rudi Giot pour m'avoir permis d'effectuer ce travail de fin d'études au Canada et m'avoir mis en relation avec le groupe modelEAU, ainsi qu'à l'IRISIB pour leur soutien conséquent. Ce stage n'aurait pas pu avoir lieu sans leur aide.

Je tiens également à remercier Dr. Ing. Filip Claeys pour m'avoir constamment soutenu dans ma découverte du fonctionnement du framework Tornado, ainsi que pour m'avoir accueilli au sein de MOSTforWATER afin de me "briefer" avant mon départ. De même, je tiens à remercier Ir. Cyril Garneau pour son aide et l'intérêt particulier qu'il a porté à mon travail.

Enfin, j'adresse mes remerciements les plus profonds à ma famille et mes amis pour le soutien qu'ils m'ont donné durant la rédaction de ce mémoire, ainsi qu'à mes collègues étudiants et l'ensemble des professeurs que j'ai eu l'honneur de côtoyer à l'ISIB, sans qui ces cinq années n'auraient pas été aussi mémorables...

Table des matières

Table des figures

Liste des tableaux

Introduction et objectifs

Ce travail de fin d'études concerne le déploiement du *framework* Tornado sur une grappe de calcul utilisant Sun Grid Engine. Il a été réalisé à l'Université Laval à Québec de février à juin 2010 au sein du groupe de recherche modelEAU dirigé par le Professeur Peter Vanrolleghem.

Le *framework* Tornado est un noyau générique permettant la définition de modèles et la réalisation d'expériences virtuelles sur ces modèles. Il a été conçu initialement et est utilisé par le groupe de recherche modelEAU pour des simulations de systèmes d'eau.

L'installation récente de Colosse, grappe de calcul de haute performance [1] au sein de l'Université Laval, et la mise à disposition de celui-ci aux différents groupes scientifiques augmente considérablement les capacités de calcul disponibles pour la recherche.
Elle permet également l'exécution optimisée de grandes quantités de tâches en parallèle.

Le déploiement de Tornado sur cette grappe de calcul a donc été considéré. En effet, certaines utilisation de ce logiciel font appel à des exécutions qualifiées d'*embarrassingly parallel* : les différentes tâches ne communiquent pas ou peu et peuvent donc être exécutées entièrement en parallèle. Sur un poste classique, ce type d'exécution n'est absolument pas optimisé dans la mesure où les tâches sont exécutées en série sur un voire deux processeurs.
Dans le cadre de ce déploiement, il est intéressant de noter le parallélisme existant entre les lois de Moore et de Gujer (voir figure 1).

La loi de Moore spécifie que le nombre de transistors des microprocesseurs sur une puce de silicium double tous les deux ans (1975). Cette loi initiale a été généralisée en considérant que la puissance de calcul des

1. Classée 72ème dans le top 500 mondial des supercalculateurs, en date du premier juin 2010 (cfr . *http ://www.top500.org*)

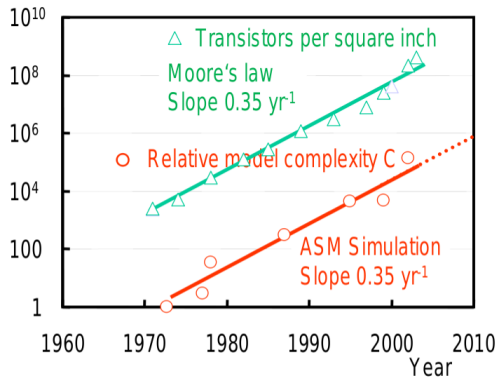

FIGURE 1 – Parallélisme entre les lois de Moore et de Gujer, d'après [9]

microprocesseurs double tous les 18 mois.
La loi de Gujer, quant à elle, estime la croissance de la complexité des modèles de simulation sur base du nombre de composants, de processus et de compartiments constituant ces modèles (voir (1))[9].

$$\text{Complexité} = \text{Nombre}_{\text{Composants}} \times \text{Nombre}_{\text{Processus}} \times \text{Nombre}_{\text{Compartiments}} \quad (1)$$

Ce parallélisme indique que la modélisation des systèmes de traitement des eaux est limitée par la performance des processeurs. L'utilisation d'un supercalculateur dans ce cadre est donc tout à fait indiquée.

L'objectif de ce travail de fin d'étude est double :

- Dans un premier temps, il vise le déploiement du *framework* Tornado sur une grappe de calcul utilisant Sun Grid Engine (SGE).
- Dans un second temps, il propose une solution permettant de simplifier la démarche nécessaire à l'utilisation de Tornado sur la grappe de calcul.

Ce mémoire est divisé en cinq chapitres, regroupés en deux parties.
La première partie consiste en l'état de l'art, et comprend trois chapitres :

- Le premier concerne Tornado. Il décrit les bases de son architecture et de son fonctionnement, deux points dont la compréhension est essentielle avant d'envisager son déploiement.

- Le second présente les caractéristiques du supercalculateur Colosse, ainsi que les outils principaux nécessaires à son utilisation.
- Le troisième introduit le logiciel Typhoon, un ordonnanceur de tâches conçu pour l'utilisation de Tornado sur un *cluster* dédié.

La deuxième partie comprend deux chapitres :

- Le premier explique la méthodologie appliquée pour le déploiement de Tornado sur le supercalculateur, ainsi que les modifications apportées au logiciel afin de le rendre compatible avec le système sur lequel il est installé.
- Le deuxième reprend le développement réalisé sur Typhoon dans le but de simplifier l'utilisation de Tornado sur Colosse.

La conclusion expose les résultats du déploiement de Tornado sur le supercalculateur et débat sur les perspectives d'avenir relatives à ce déploiement. De même, on y propose d'éventuels développements complémentaires en rapport à ce qui a été réalisé sur Typhoon.

Première partie

Etat de l'art

Chapitre 1

Tornado

Tornado est un *framework* générique conçu et maintenu par la société MOSTforWATER basée à Courtrai, Belgique. Ayant fait l'objet d'une thèse de doctorat en biologie appliquée - technologie environnementale (cfr. [2]), son utilisation principale concerne la modélisation de systèmes d'eau, notamment au sein du logiciel WEST. Sa généricité le rend toutefois applicable à bien d'autres domaines.
Le but de ce chapitre est d'expliquer les concepts importants qui entreront en jeu dans le déploiement de ce logiciel.

1.1 Principes de fonctionnement

Tornado fonctionne sur base de modèles et de fichiers de description. L'utilisateur définit un modèle à l'aide d'un langage de modélisation orienté objet (MSL ou Modelica) et le convertit, à l'aide d'un outil interne à Tornado, en code C. Ce code C peut ensuite être compilé - via une commande de Tornado faisant appel à un compilateur extérieur au programme - ce qui générera un fichier binaire exécutable, de type DLL (*Dynamic Loadable Library*) ou SO (*Shared Object*) suivant la plateforme sur laquelle la compilation est effectuée. Ce fichier binaire pourra ensuite être chargé au sein de Tornado et exécuté par celui-ci dans le cadre d'une expérience virtuelle.

Le diagramme montré à la figure 1.1 explique de manière schématique ce cheminement entre un modèle MSL/Modelica et un fichier binaire exécutable.

L'exécution d'un modèle fait également appel à des fichiers de description au format XML, qui spécifient les différents paramètres de l'expérience

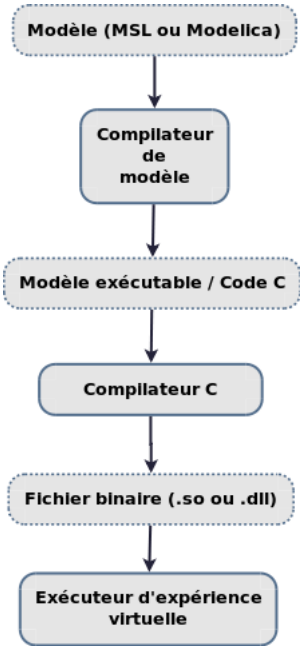

FIGURE 1.1 – Tornado : Processus de génération d'un modèle exécutable, d'après [2]

virtuelle dans laquelle le modèle sera exploité. Ces paramètres correspondent aux solveurs utilisés, aux éventuels fichiers d'entrée et de sortie, etc.

Le diagramme montré à la figure 1.2 montre de manière schématique la composition d'une expérience virtuelle.

1.1.1 Compilation d'un modèle traduit en *C*

La compilation d'un modèle traduit en *C* fait appel à un compilateur *C*, externe à Tornado (voir figure 1.3). Cet appel est réalisé par l'exécutable *tbuild* fourni par Tornado.

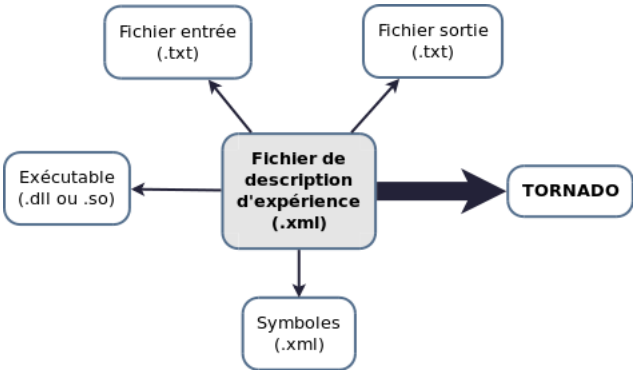

FIGURE 1.2 – Tornado : Composition d'une expérience virtuelle

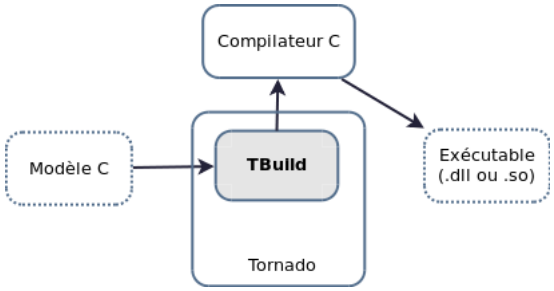

FIGURE 1.3 – Tornado : Compilation d'un modèle traduit en C

La table 1.1 reprend, pour chaque plateforme, les différents compilateurs supportés par Tornado.

TABLE 1.1 – Tornado : Compilateurs supportés

Windows	**Linux/UNIX**
Microsoft Visual C++ 6.0	Gnu C Collection
Microsoft Visual C++ 7.1	Intel C Compiler
Microsoft Visual C++ 8	
Microsoft Visual C++ 9	
Borland C Compiler 5.5	
Intel C Compiler	
Local C Compiler	

Les paramètres de compilation sont spécifiés dans le fichier *Build.cpp*, chargé d'effectuer l'appel du compilateur à partir de Tornado. Des paramètres additionnels sont également spécifiables via les options de la commande *tbuild*, chargée d'appeler le compilateur à partir de Tornado.

1.2 Architecture

Tornado dipose d'une architecture modulaire : différents modules regroupent les différentes fonctionnalités nécessaires à l'exécution du logiciel, comme repris à la figure 1.4.
Chacun de ces modules est composé d'objets compilés regroupés en librairies, utilisables par le module principal : Tornado.
Les points suivants reprennent les caractéristiques de base des modules nécessaires à la compilation de Tornado.

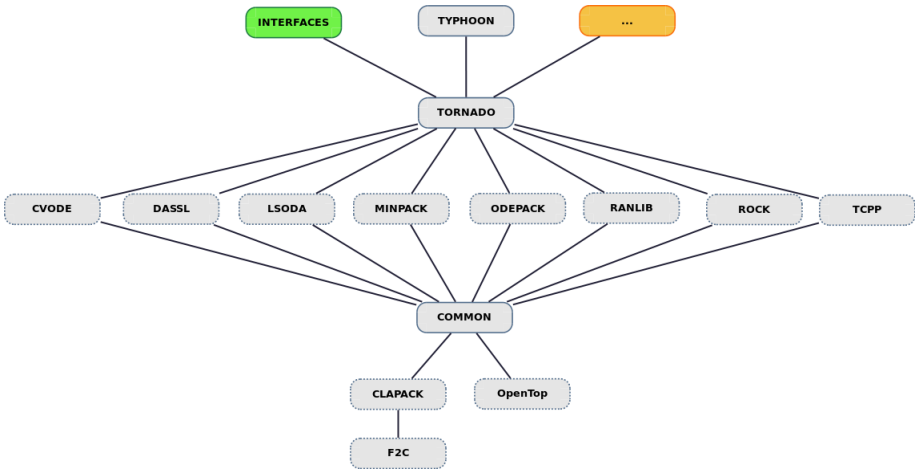

FIGURE 1.4 – Tornado : Architecture modulaire, d'après [8]

1.2.1 Tornado

Le "module" Tornado correspond au noyau de l'application. C'est à ce niveau-ci que l'on va retrouver la collection d'utilitaires exécutables faisant usage des fonctionnalités offertes dans les librairies des autres modules.
Ces utilitaires sont mis en oeuvre afin de générer des modèles exécutables suivant le processus expliqué à la figure 1.1.

1.2.2 Common

Common est un module concentrant les utilitaires de base nécessaires au fonctionnement de Tornado :

- Manipulation de fichiers,
- Utilitaires de cryptage et décryptage,
- Gestion du temps,
- Vecteurs,
- ...

1.2.3 Autres modules

Les autres modules de base composant Tornado sont :

- CVODE, une librairie de solveurs intégraux pour équations différentielles ordinaires.
- DASSL, une librairie de solveurs intégraux pour équations différentielles algébriques.
- LSODA, une librairie de solveurs intégraux pour équations différentielles ordinaires.
- MINPACK, une librairie de routines FORTRAN de résolution de systèmes d'équations non-linéaires.
- ODEPACK, une librairie de routines FORTRAN de résolution d'équations différentielles ordinaires.
- RANLIB, une librairie utilisée pour la génération de nombres aléatoires.
- ROCK, une librairie de solveurs intégraux.
- TCPP, une librairie de préprocesseurs utilisés pour la compilation des modèles.

1.2.4 CLAPACK et F2C

LAPACK est une bibliothèque logicielle dédiée à l'algèbre linéaire numérique écrite en Fortran. CLAPACK est une interface de programmation (API) destinée à établir un lien entre LAPACK et le langage C.
F2C est un programme utilisé pour convertir du code Fortran en code C.

1.2.5 OpenTop

La bibliothèque logicielle OpenTop est une bibliothèque C++ commerciale développée par Elcel Technology [1] orientée vers le développement de services web. Tornado utilise OpenTop pour la création et la synchronisation

1. http ://www.elcel.com

des threads ainsi que pour l'analyse syntaxique de documents XML.
OpenTop dispose également d'outils de gestion de ressources, de développement inter-plateformes (abstraction de plateforme), de réseau et de support pour Unicode, différents types d'I/O, ainsi que les protocoles SSL et HTTP[2].

1.3 Interfaces

Diverses interfaces permettent d'accéder à Tornado et de rendre son utilisation possible au sein de différents environnements, tels que Matlab, Java et OpenMI (voir figure 1.5).
Un programme indépendant peut donc faire appel à Tornado au travers de ces interfaces. A titre d'exemple, Matlab peut exécuter une expérience sur Tornado au travers de l'interface TornadoMEX, comme illustré à la figure 1.6.

Le code Matlab suivant permet d'exécuter dans Tornado l'expérience "*PredatorPrey.Simul.Exp.xml*" (située dans le répertoire "*data/PredatorPrey/*" de Tornado) et d'en sauvegarder les résultats dans la variable y :

```
y = TornadoMEX('EndValue', '$(TORNADO_DATA_PATH)/PredatorPrey/
PredatorPrey.Simul.Exp.xml', '.c1=0.1', '.pa.out_1;.ps.out_1')
```

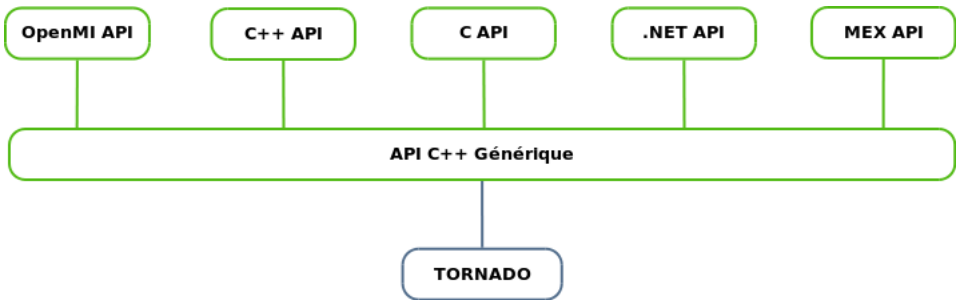

FIGURE 1.5 – Tornado : interfaces, d'après [8]

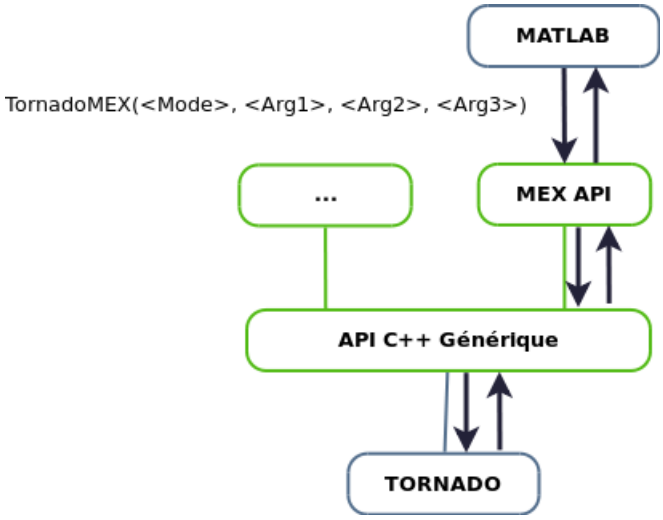

FIGURE 1.6 – Tornado : TornadoMEX

1.4 Déploiement

Cette section explique le déploiement de Tornado sur une plateforme, que ce soit via une compilation ou une installation directe.

1.4.1 Compilation de Tornado

La compilation de Tornado s'effectue en trois étapes successives :

1. Compilation du module *Common* : ses fonctionnalités étant requises par les autres modules, son installation se fait en premier.
2. Compilation des modules de solveurs, ainsi que TCPP.
3. Compilation de Tornado.

Ces différents modules sont disponibles sous forme d'archives au format *zip* reprenant le code source et différents fichiers de configuration propres aux plateformes sur lesquelles on désire compiler l'application. Pour l'ensemble des modules, l'arborescence des fichiers respecte la norme FHS (File Hierarchy Standard), tel qu'illustré à la figure 1.7.

Par rapport au reste des modules, Tornado comprend certains répertoires complémentaires dont principalement un répertoire *bin*, dans lequel se retrouve

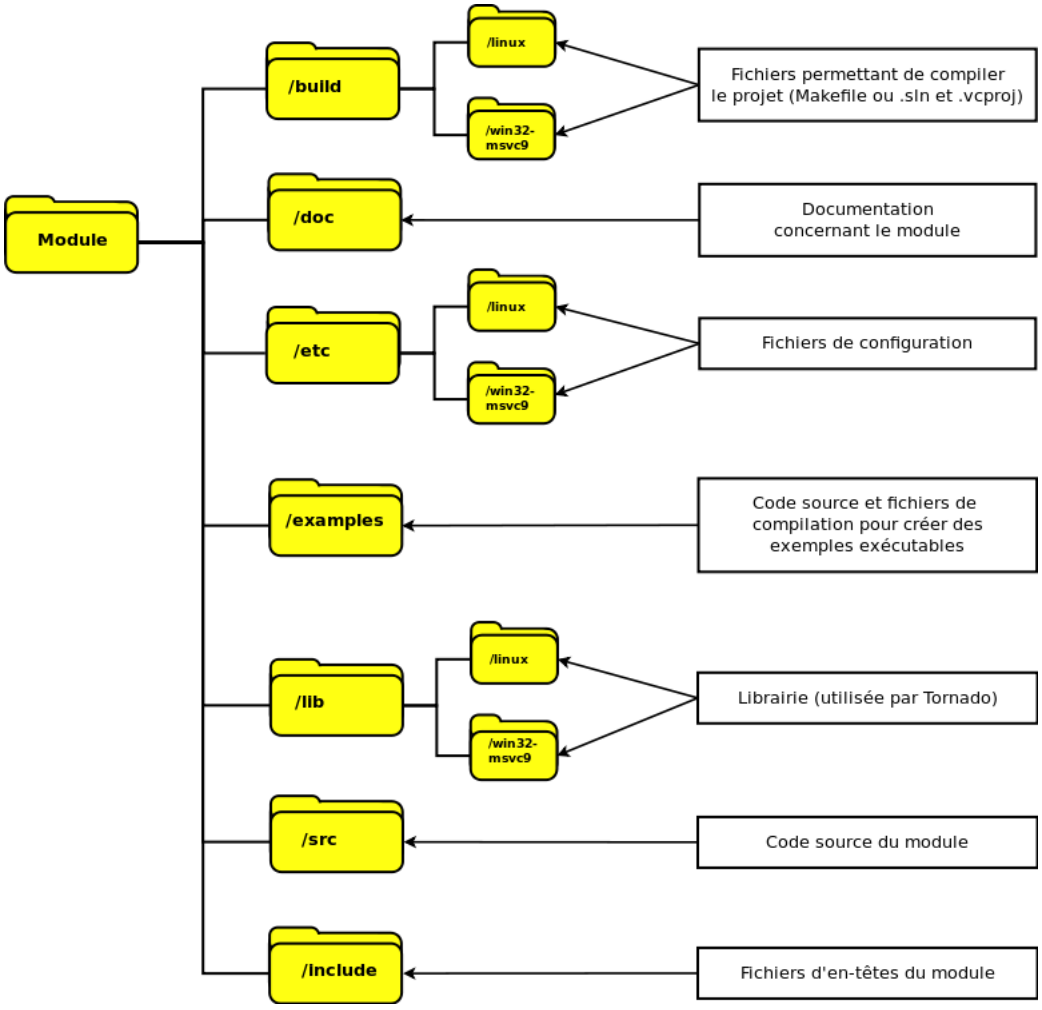

FIGURE 1.7 – Tornado : Arborescence de fichiers d'un module

l'entièreté des fichiers exécutables produits par sa compilation.

Sous Windows, Tornado est compilable à l'aide du compilateur Microsoft Visual C++ 9.0 (MSVC9), pour lequel les fichiers de projet sont disponibles dans les archives des différents modules (voir figure 1.7).
Sous Linux/UNIX, Tornado est compilable à l'aide du compilateur Gnu C Collection (GCC). Des fichiers *Makefile* sont disponibles à cet effet.

Les différents modules utilisés par Tornado sont compilés en librairies (fichiers d'extension **.lib* sous Windows et d'extension **.a* sous Linux).
La compilation de Tornado lui-même produira, en plus de librairies nécessaires à l'exécution, des bibliothèques de liens dynamiques (DLL) ou objets partagés dynamiques, suivant la plateforme (fichiers d'extension **.dll* sous Windows ou d'extension **.so* sous Linux) ainsi que des fichiers exécutables (fichiers d'extension **.exe* sous Windows).

Prérequis

Comme expliqué aux points 1.2.1 et 1.2.2, certaines librairies additionnelles sont nécessaires : CLAPACK, F2C et OpenTop. De plus, Tornado nécessite la présence de la librairie OpenSSL (librairie destinée à des utilisations en cryptographie), de flex (analyseur lexical), de Bison (compilateur de compilateur), du SDK Java (uniquement pour TornadoJNI) et de SuperPRO (compilation sous Windows uniquement).
Ces librairies et programmes étant utilisés par Tornado et les modules auxquels il fait appel, leur installation et/ou compilation est prioritaire au reste.

1.4.2 Installation

Un mécanisme d'installation directe de Tornado est disponible pour des plateformes 32 bits, que ce soit sous Windows ou sous Linux/UNIX. Il est également possible d'installer Tornado sur des plateformes Windows 64 bits.

Installation sous Windows

L'installation de Tornado sous Windows se fait à l'aide d'un fichier MSI et suit les standards d'installation propres à cette plateforme. Tornado a été installé et testé sous Windows XP, Windows Vista et Windows 7, en 32 bits et 64 bits.

Installation sous Linux/UNIX

L'installation de Tornado sous Linux/UNIX se fait à l'aide d'un script Shell nécessitant la présence du *shell sh*.
Tornado a été testé avec succès sur la distribution Linux *Debian*, ainsi que des dérivés de cette distribution (*Ubuntu* et *Xubuntu*).

1.5 Mécanismes de gestion de licence

Tornado ne fonctionne qu'avec une licence d'utilisation. Deux mécanismes sont disponibles à cet effet :

- L'utilisation d'un fichier de licence propre à la machine sur laquelle Tornado sera utilisé (ce fichier se base sur l'adresse MAC et le nom d'hôte de la machine en question).
- L'utilisation d'un serveur de licences (principe de serveur à jetons : le nombre d'instances de Tornado pouvant être exécutées simultanément est limité).

1.6 Utilisation de Tornado

Comme expliqué précédemment, Tornado permet la construction de modèles et l'exécution d'expériences virtuelles faisant appel à ces modèles.
Ceci se traduit par l'utilisation de différentes commandes disponibles au sein de Tornado.

Cette utilisation peut s'effectuer de deux façons : soit en accédant directement au noyau via une interface en lignes de commande, soit via les différentes interfaces disponibles telles que TornadoCPP, TornadoNET, TornadoMEX, etc.

1.6.1 Interface graphique

Une version disposant d'une interface graphique est disponible sous Windows : *TWin* (*Tornado for Windows*, voir figure 1.8). Cette version est basée sur l'interface *.NET* créé pour Tornado, *TornadoNET*.

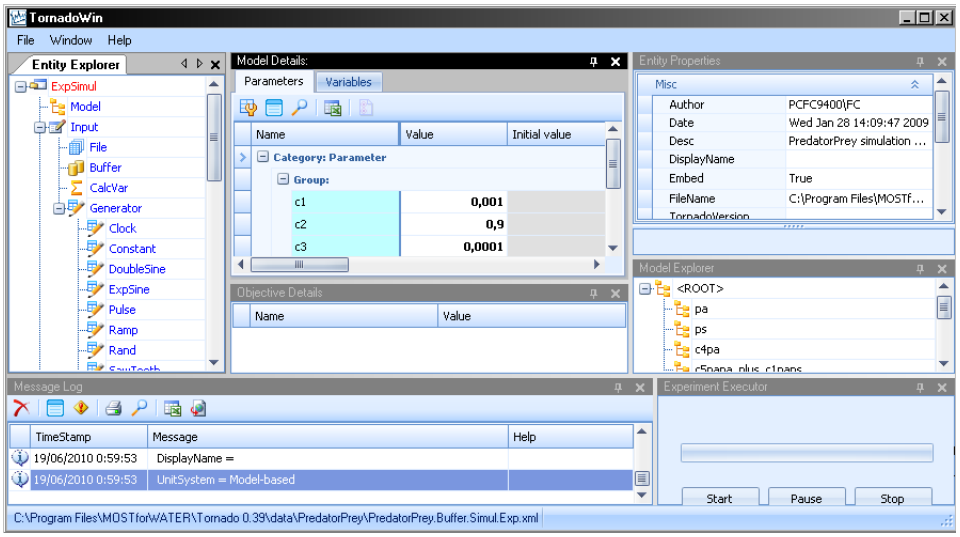

FIGURE 1.8 – Tornado : Aperçu de l'interface graphique de TWin

1.6.2 Utilisation en lignes de commande

Les commandes principales nécessaires à la réalisation des différentes étapes énoncées au début de cette section sont reprises à la table 1.2.

TABLE 1.2 – Tornado : Utilisation

Etape	**Commande**
Conversion MSL/Modelica en C	*mof2t [options] <FICHIER_MODELE>*
Génération fichier de symboles	*mof2t [options] <FICHIER_MODELE>*
Compilation modèle	*tbuild [options] <NOM_MODELE>*
Génération description d'expérience virtuelle	*tmain [options] <TYPE_EXPERIENCE> <NOM_MODELE>*
Exécution de l'expérience virtuelle	*texec [options] <FICHIER_ DESCRIPTION _EXPERIENCE>*

Chapitre 2

Le supercalculateur Colosse

Ce chapitre introduit les spécifications principales du super-calculateur Colosse ainsi que les bases de son utilisation.

Les sections 2.1 et 2.2 sont principalement reprises de [1].

2.1 CLUMEQ

Le CLUMEQ (*(Consortium Laval, Université du Québec, McGill and Eastern Quebec)*)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 du Québec.
La mission du CLUMEQ est d'offrir à ses institutions membres une infrastructure de CHP de classe mondiale, pour l'avancement des connaissances dans tous les secteurs de recherche, ainsi qu'un service d'analyse et de formation pour aider les chercheurs à exploiter efficacement ces infrastructures.
Le CLUMEQ fait partie intégrante de Calcul Canada, une plateforme nationale pour le CHP qui chapeaute les sept consortiums régionaux canadiens. À travers Calcul Canada, les infrastructures du CLUMEQ sont accessibles à l'ensemble des chercheurs universitaires canadiens.

2.2 Spécifications

Colosse est une grappe de calcul comportant 960 noeuds de calcul et 40 noeuds d'infrastructure. Ces noeuds sont tous constitués d'une paire de processeurs Intel Nehalem-EP possédant chacun quatre coeurs de traitement et 24 gigaoctets de mémoire RAM. Au total, Colosse comporte donc 8000

coeurs et 24 téraoctets de mémoire. Tous les noeuds sont reliés par une réseautique haute performance de type Infiniband (IB) Quad Data Rate (QDR) dont la vitesse nominale est de 40 gigabits par seconde. Les noeuds d'infrastructure sont également reliés entre-eux et avec le monde extérieur par une réseautique de type 10-gigabit ethernet. Parmi ces noeuds d'infrastructure, la moitié sont consacrés au système de fichiers parallèle Lustre dont la capacité utilisable atteindra 1 pétaoctets lorsqu'il sera totalement déployé. Sa capacité initiale est de 500 téraoctets.

Actuellement (juin 2010), Colosse est composé de dix châssis Sun Constellation C48. Chacun de ces châssis contient quatre cabinets pouvant contenir douze lames de calcul Sun X6275. Ces lames disposent chacune de deux noeuds de calcul, composés de deux processeurs Intel Nehalem-EP cadencés à 2.8 GHz. Ces caractéristiques sont illustrées à la figure 2.1.

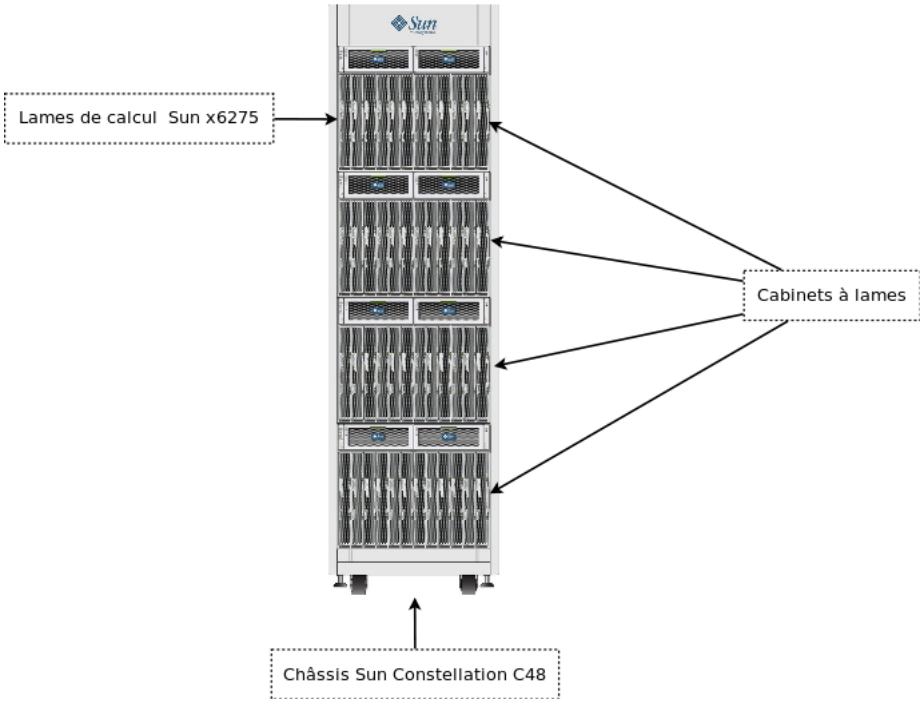

FIGURE 2.1 – Colosse : Caractéristiques techniques[6]

Chaque noeud de calcul est lié au système de fichiers Lustre et aux

noeuds d'infrastructure par l'intermédiaire de deux switches : un premier switch connecté à chaque cabinet regroupe les noeuds de calcul et renvoie les données vers un second switch (*core switch*) qui permet d'interconnecter l'ensemble des switches reliés aux noeuds de calcul.
Ces *core switches* effectuent ensuite la distribution des données sur le système Lustre.
Le schéma donné à la figure 2.2 illustre cette liaison entre les noeuds de calcul et le système de fichiers. Les détails concernant le système de fichiers Lustre sont expliqués par la suite, à la section 2.3.

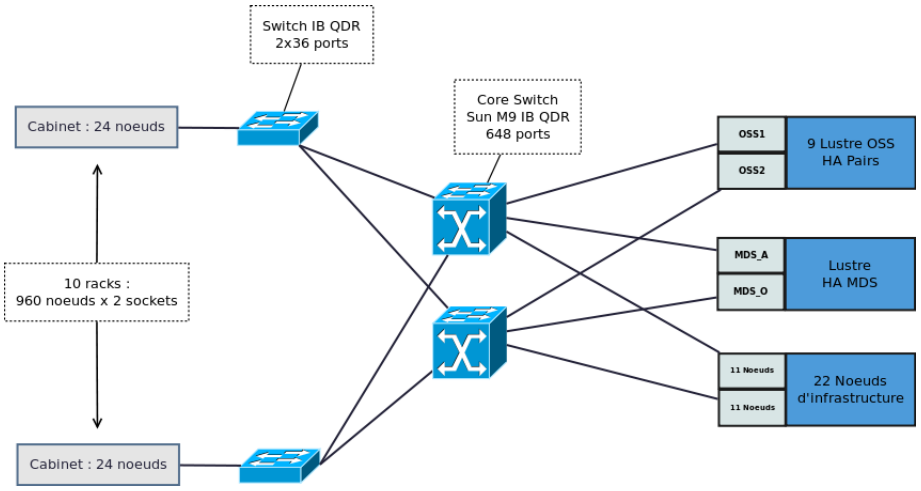

FIGURE 2.2 – Colosse : Liaison des noeuds au système de fichiers, d'après [1]

Il est aussi intéressant de considérer l'architecture particulière du bâtiment hébergeant Colosse : celui-ci est un silo initialement conçu pour contenir un accélérateur de particules (démantelé en 2006) et réaménagé pour le supercalculateur. La structure circulaire et verticale de ce silo permet une optimisation de la circulation de l'air dans le bâtiment (pas de coins, donc moins de turbulences) ainsi qu'une simplification de la solution de refroidissement qui permet une meilleure efficacité énergétique. Cette architecture est illustrée à la figure 2.3.
Une partie de la chaleur émise par l'infrastructure de Colosse, estimée à 45%, est également récupérée pour chauffer le campus universitaire.

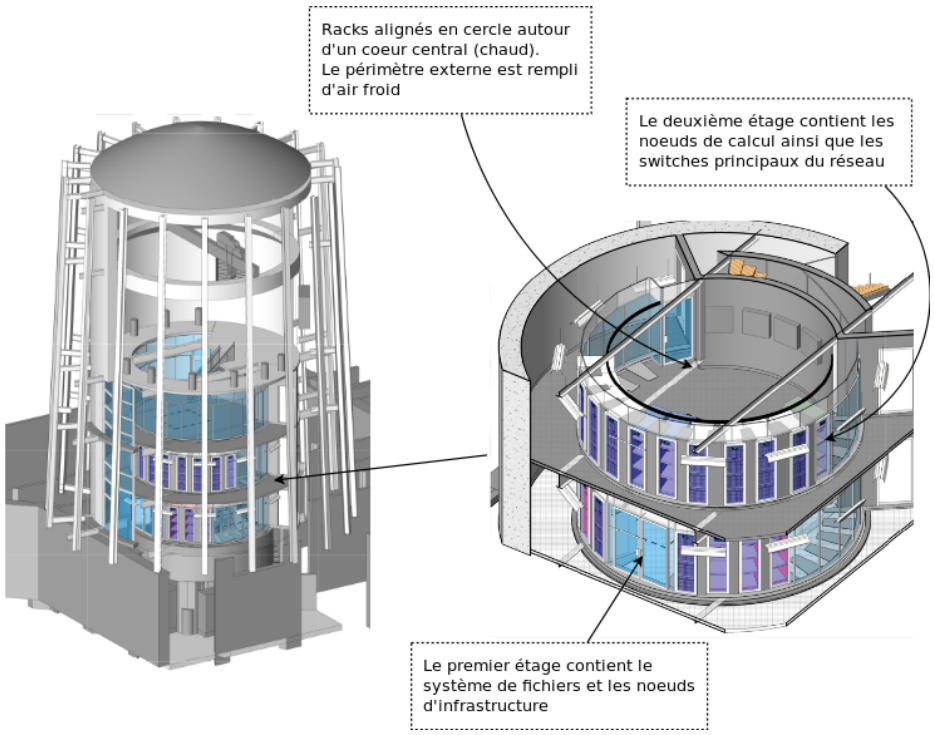

FIGURE 2.3 – Colosse : Architecture du bâtiment, d'après [4]

2.3 Système de fichiers

Colosse est équipé d'un système de fichiers parallèle distribué Lustre (combinaison de *Linux* et *Cluster*). Ce type de système de fichiers peut accepter plusieurs dizaines de milliers de systèmes clients, plusieurs petaoctets de données et des centaines de gigaoctets d'entrées et sorties par seconde.

Un système de fichiers Lustre est composé de trois unités principales :

- Une *Meta Data Target* (MDT) : unité qui se charge d'enregister les métadonnées (noms de fichiers, répertoires, permissions, etc.) sur un serveur de métadonnées (*Meta Data Server* - MDS).
- Un ou plusieurs *Object Storage Server(s)* (OSS) qui enregistrent le contenu des fichiers sur une ou plusieurs *Object Storage Target(s)* (OST). Suivant la configuration matérielle, un OSS sert entre deux et huit OSTs, chacune ciblant un système de fichiers local d'une taille pou-

vant aller jusqu'à 8 teraoctets.

- Des clients qui accèdent aux données et les utilisent.

Toutes ces unités sont liées au travers d'un réseau (Lustre est compatible avec des réseaux de type Infiniband, TCP/IP en Ethernet, Myrinet, Quadrics ainsi que d'autres technologies propriétaires).
Cette architecture est illustrée à la figure 2.4.

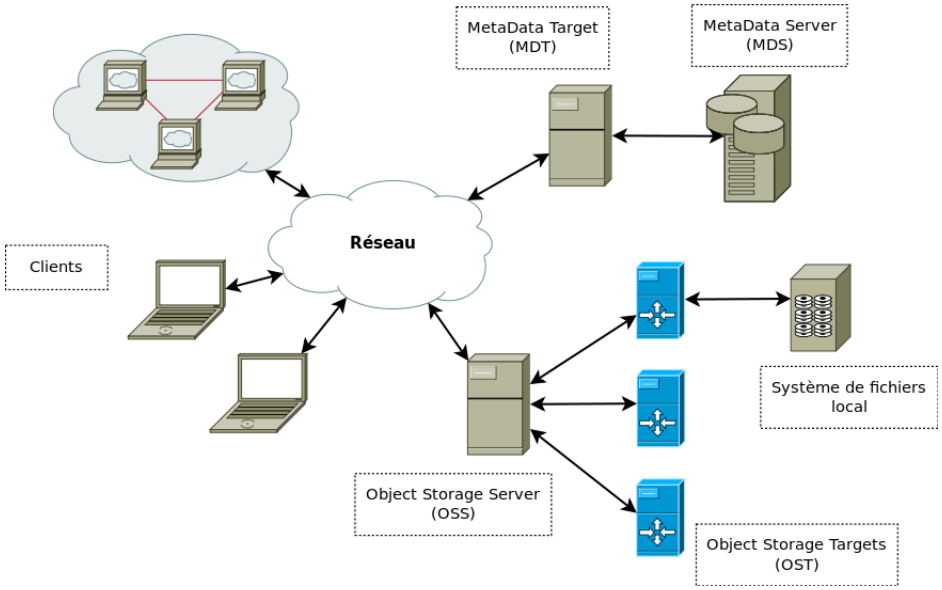

FIGURE 2.4 – Colosse : Système de fichiers Lustre

L'implémentation de ce système de fichiers sur Colosse consiste en neuf paires OSS et deux MDS installés sur des serveurs Sun Fire X4270.
Le stockage de données se fait sur des aires de stockage Sun Storage J4400 disposant chacune de 24 disques SATA à 7200rpm (quatre aires de stockage par paire d'OSS) en configuration RAID 6.

2.4 Logiciels

Colosse propose une série de logiciels à ses utilisateurs, disponibles sous forme de modules à charger au sein de son environnement.

Ces modules sont regroupés en cinq catégories distinctes :

- Applications (Flex, Octave, ...).

- Compilateurs (GCC, Intel C++ Compiler et Sun Studio).
- Librairies (Blas, Gmp, ...).
- Outils (Gnu Debug, Tau, ...).
- Environnements additionnels (Open MPI, Java, ...).

La table 2.1 reprend les commandes principales en rapport avec l'utilisation de ces modules.

TABLE 2.1 – Colosse : Utilisation de modules

Action	**Commande**
Afficher la liste des modules disponibles	*module avail*
Charger un module dans son environnement	*module load <NOM_DU_MODULE>*
Retirer un module de son environnement	*module rm <NOM_DU_MODULE>*
Afficher la liste des modules chargés dans son environnement	*module list*

2.5 Interface utilisateur

Le système d'exploitation de Colosse est un système Linux (version du kernel : 2.6.18, distribution RedHat 4.1.2-46). Au vu de l'utilisation qui est faite de ce système, l'interface utilisateur est un interpréteur de lignes de commandes.
Différents interpréteurs sont disponibles :

- Bourne Shell (*sh*),
- C-Shell (*sh*),
- Tenex C-Shell (*tcsh*).

2.6 Accès à distance

Colosse est accessible depuis le réseau interne de l'Université Laval et depuis les autres sites rattachés au CLUMEQ via une connexion 10GB Ethernet. Il est également accessible depuis Internet (voir figure 2.5).

L'accès aux commandes et aux fichiers sont clairement séparés : l'accès aux commandes s'effectue via une connexion Secure Shell (SSH), tandis que l'accès au système de fichiers s'effectue via une connexion SFTP (SSH File Transfer Protocol).

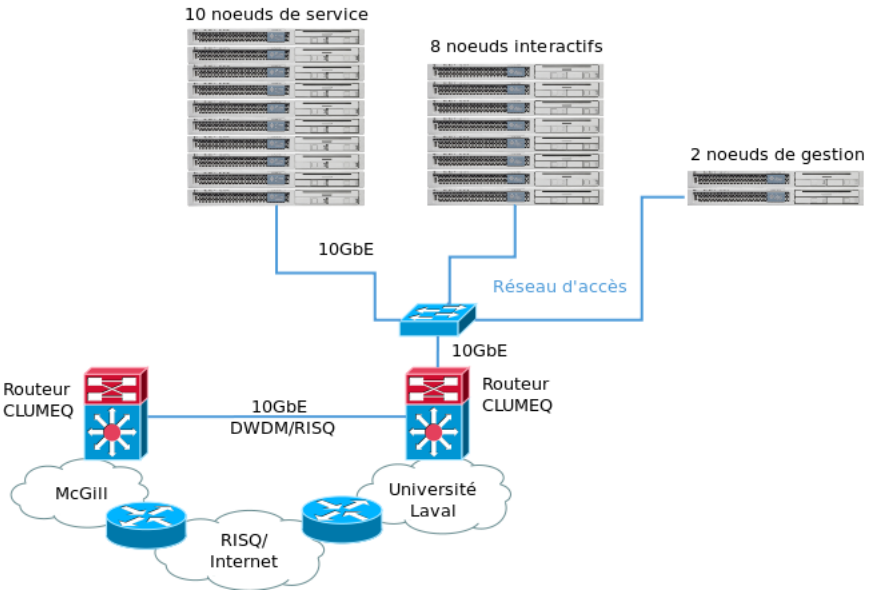

FIGURE 2.5 – Colosse : Accès à distance, d'après [4]

Lorsque l'utilisateur est connecté via l'un de ces deux protocoles, il a accès à un serveur intermédiaire, Cyclops.
Ce serveur donne accès au système de fichiers Lustre et à SGE. Si l'utilisateur est connecté en SFTP, il pourra manipuler des fichiers dans les répertoires qui lui seront rendus accessibles par ce serveur. Si l'utilisateur est connecté en SSH, il pourra exécuter différentes commandes directement sur le serveur ainsi que soumettre des tâches à SGE.
Le schéma repris à la figure 2.6 reprend brièvement cette logique.

2.6.1 Connexion SFTP

En se connectant à Colosse via un logiciel supportant le protocole de transfert de fichiers SFTP, un utilisateur a accès aux différents répertoires de son groupe de travail : son répertoire personnel, les répertoires personnels des autres membres du groupe et le répertoire principal ainsi que le répertoire de stockage temporaire du groupe.
Ce mode de connexion permet donc de télécharger des fichiers vers et de Colosse.
La figure 2.7 montre de manière schématique la structure des répertoires

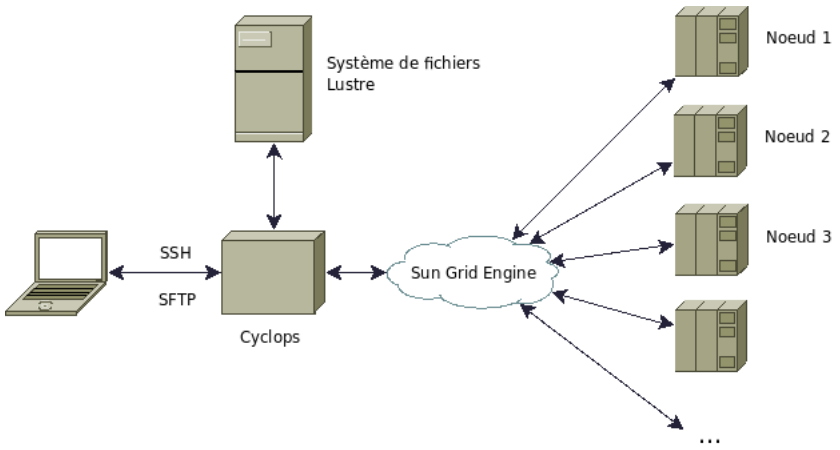

FIGURE 2.6 – Colosse : Topologie interne

principaux. La plupart des répertoires propres au système sont également accessibles en lecture ("*bin*", "*etc*", "*dev*", ...).

Le répertoire "*/home*" contient les répertoires personnels des utilisateurs, qui disposent d'un espace disque de 5 GO.
Le répertoire "*/scratch*" contient les répertoires principaux des groupes de travail, qui disposent d'un espace disque initial de 100 GO, extensible sur demande.
Le répertoire "*/rap*" contient les répertoires de stockage temporaire des groupes de travail. Ces espaces mémoire sont réinitialisés de temps à autre.

2.6.2 Connexion SSH

L'accès à Colosse via une connexion SSH permet un accès aux différentes commandes disponibles sur le système. Les utilisateurs de systèmes Microsoft Windows utiliseront donc un utilitaire tel que PuttY tandis que les utilisateurs de systèmes Linux et UNIX (MacOS inclus) utiliseront les commandes installées par défaut sur leurs systèmes.
Une fois l'utilisateur connecté, il a accès à son environnement propre ainsi qu'aux mêmes répertoires que ceux accessibles en SFTP.

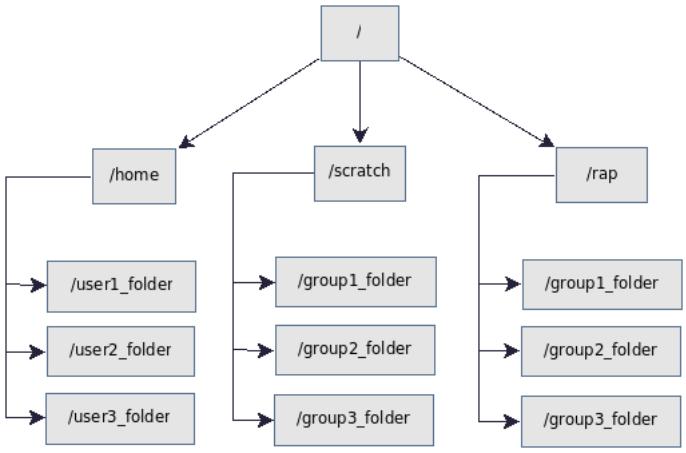

FIGURE 2.7 – Colosse : Arborescence du système de fichiers

2.7 Exécution de tâches

Colosse utilise Sun Grid Engine (SGE) comme ordonnanceur de tâches. SGE est un ordonnanceur open-source assurant l'acceptation, l'ordonnancement, la répartition et la gestion des tâches (standalone, parallèles ou interactives). Il est également responsable de la gestion des ressources (processeurs et mémoire alloués, licences logicielles et espace disque).

La soumission d'une tâche s'effectue en deux étapes :

1. Ecriture d'un script de description de la tâche.
2. Soumission de ce script à l'aide d'une commande.

Lorsque la tâche est soumise à SGE, celui-ci chargera les composants logiciels requis sur les noeuds de calcul disponibles, se chargera de l'exécution de la tâche et renverra les résultats de l'exécution au système de fichiers Lustre, dans le répertoire spécifié par l'utilisateur.

2.7.1 Script de description de la tâche

Le script de description de la tâche est un script Shell qui spécifie les différents paramètres que SGE devra prendre en compte lors de l'exécution de celle-ci. Différents exemples de scripts sont repris à l'Annexe B. La table 2.2 reprend les paramètres principaux à spécifier dans ce script.

TABLE 2.2 – Colosse : Paramètres d'une tâche

Paramètre de la tâche	**Paramètre du script**	**Paramètre obligatoire (Oui/Non)**
Nom de la tâche	*-N <NOM_DE _LA_TÂCHE>*	Oui
Indicatif du groupe de travail	*-P <INDICATIF_DU _GROUPE>*	Oui
Durée estimée pour exécuter la tâche	*-l <DUREE [h_rt = hh :mm :ss]>*	Oui
Shell à utiliser pour exécuter la tâche	*-S <EMPLACEMENT _DU_SHELL>*	Non
Nombre de coeurs à utiliser (multiple de 8)	*-pe <INDICATIF _DE_LA_FILE>*	Non
Spécification d'un fichier de sortie	*-o <FICHIER _DE_SORTIE>*	Non
Spécification d'un fichier d'erreur	*-e <FICHIER _D_ERREUR>*	Non
Utilisation du répertoire courant pour exécuter la tâche	*-cwd*	Non

Il est également possible de spécifier d'autres paramètres, tels que des variables d'environnement utilisées par le programme, des chargements de module, etc.

La soumission de ce script s'effectue à l'aide de la commande :

```
user@colosse.clumeq.ca: $ qsub <NOM_DU_SCRIPT>
```

La tâche à exécuter sera alors placée dans une file d'attente. Sa position dans cette file dépendra du nombre de coeurs demandés et de l'estimation donnée du temps d'exécution.
L'utilisateur peut suivre l'évolution de la file d'attente à l'aide de la commande *qstat*. Celle-ci affiche les données relatives aux tâches placées dans la file d'attente (voir figure 2.8).

Deux options de la commande *qstat* peuvent être soulignées :

- *qstat -u "<UTILISATEUR>"* : affiche toutes les tâches, lancées par un utilisateur défini, présentes dans la file d'attente.
- *qstat -f* : affiche tous les paramètres d'exécution pour toutes les tâches présentes dans la file d'attente.

FIGURE 2.8 – Colosse : File d'attente de SGE

Un utilisateur peut également effacer une tâche, c'est-à-dire l'enlever de la file d'attente, à l'aide de la commande :

```
user@colosse.clumeq.ca: $ qdel <TASK_ID>
```

Où *<TASK_ID>* est l'identifiant de la tâche dans la file d'attente.

2.7.2 Tâche stand-alone

L'exécution d'une tâche stand-alone ne nécessite que la spécification des paramètres repris ci-dessus.

2.7.3 Tâches parallèles

SGE offre la possibilité d'exécuter, en parallèle, un nombre n de fois la même tâche avec n entrées et sorties différentes.
Ce type d'exécution est rendu possible par le fait que SGE utilise une variable d'environnement, *$SGE_TASK_ID*, pour identifier les tâches qu'il doit

exécuter.

Pour exécuter plusieurs fois la même tâche en parallèle, il faut donc :

- Spécifier un intervalle sur lequel SGE se basera pour définir la variable *$SGE_TASK_ID*, à l'aide de l'option "-t" dans le script.
- Utiliser la variable d'environnement *$SGE_TASK_ID* dans le nom des fichiers d'entrée et de sortie (si applicable).

A titre d'exemple, la ligne suivante spécifie que la variable *$SGE_TASK_ID* variera entre 1 et 5 :

```
#$ -t 1-5
```

La ligne suivante lance une exécution de Tornado basée sur la variable *$SGE_TASK_ID* :

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

Si cette variable prend les valeurs de 1 à 5, il y aura donc cinq exécutions de Tornado lancées en parallèle, faisant appel à cinq fichiers XML différents, nommés *Job_1.Simul.Exp.xml* à *Job_5.Simul.Exp.xml*.

2.8 Récupération des données

Suivant les paramètres qui auront été spécifiés dans le script de description de tâche, différents fichiers de sortie peuvent être créés :

- Un fichier reprenant les erreurs éventuelles rencontrées lors de l'exécution de la tâche.
- Un fichier reprenant les sorties affichées par la tâche (si la tâche affiche des résultats et/ou statuts à l'écran pendant ou après son exécution).
- Les éventuels fichiers créés par la tâche.

Tous ces fichiers sont accessibles en SSH ou en SFTP, localisés là où l'utilisateur l'aura spécifié.

Il est à noter que, lors de l'exécution de tâches parallèles, des fichiers de sortie et d'erreur sont systématiquement créés. Ceux-ci ont un nom de

fichier de la forme :

$$<NOM\_DE\_LA\_TACHE>.\{type\}<IDENTIFIANT\_DE\_LA\_TACHE>$$

, où $\{type\}$ est le type de fichier produit et prend la valeur “e” (erreur) ou “o” (sortie - output).

Par exemple, l'exécution des tâches reprises à la figure 2.8 produira dix fichiers de sortie et dix fichiers d'erreur, respectivement nommés *TwoASU.o4906061* à *TwoASU.o49060610* et *TwoASU.e4906061* à *TwoASU.e49060610*.
Ces fichiers sont vides si aucune erreur n'est décelée ou aucune sortie spécifiée.

Chapitre 3

Typhoon

Typhoon est un ordonnanceur de tâches pour Tornado conçu pour être utilisé sur un *cluster*, c'est-à-dire sur un ensemble homogène de noeuds de calcul.
Son utilisation est destinée au cas où plusieurs instances de Tornado peuvent être exécutées simultanément, en parallèle.
Ce module est entre autre utilisé sur un *cluster* Linux (seize noeuds de calcul) du département de mathématiques appliquées, de biométrie et de contrôle de procédés de l'Université de Gand, Belgique.

3.1 Principes de fonctionnement

Typhoon fonctionne sur une base de services web permettant l'échange d'informations entre une instance maîtresse et plusieurs instances esclaves, exécutées sur les noeuds du cluster.

L'une des idées de base derrière Typhoon est d'utiliser ce logiciel comme une couche d'abstraction entre Tornado et un *cluster* : l'utilisateur qui exécute une série d'expériences virtuelles à l'aide de Typhoon ne doit pas se rendre compte que les exécutions s'effectuent à distance, sur une grappe de calcul.

La section 3.1.1, concernant l'architecture, provient principalement d'une adaptation de [7].

3.1.1 Architecture

Typhoon consiste en deux modules principaux : un module *Maître* et un module *Esclave*.
Le Maître reçoit les expériences virtuelles de Tornado, stocke celles-ci jusqu'à ce que leur exécution soit terminée, les répartit sur les noeuds de calcul disponibles et déclenche leur exécution.
Les Esclaves exécutent les expériences virtuelles et collectent les résultats demandés par l'utilisateur.

Ces deux modules sont composés de sous-modules :

- Un répartiteur, inclus dans le Maître, qui agit comme interface entre Typhoon et l'application appelante (Tornado), et qui gère les autres modules du système. Son rôle principal est d'administrer les tâches entre le moment où elles sont soumises par l'application appelante et le moment où leur exécution est terminée et que les résultats sont disponibles pour l'utilisateur.
- Un registre, assurant la gestion des Esclaves : il autorise des noeuds de calcul à être enregistrés comme tels. Ce registre est intégré dans le Maître.
- Un sélectionneur, qui choisit un poste Esclave pour exécuter la tâche suivante.
- Un vérificateur (tâche de fond), qui s'assure que tout fonctionne correctement et déclenche les actions appropriées en cas de problèmes.
- Un accepteur, inclus dans chaque Esclave, qui évalue les requêtes provenant du Maître pour l'exécution de tâches. Si une tâche est acceptée, une nouvelle instance de l'exécuteur est créée et l'exécution de la tâche est lancée.

Le transfert des entrées, sorties et états entre le Maître et les Esclaves est effectué à l'aide du protocole SOAP via l'implémentation gSOAP.
Ces principes architecturaux sont illustrés à la figure 3.1

Typhoon fournit différents mécanismes de transfert des ressources d'entrées et sorties (fichiers texte ou binaires, exécutables, etc.) entre le Maître et les Esclaves[2] :

- Enrobage (*Embedding*) : dans ce mécanisme, les contenus des ressources sont convertis en une représentation sous forme de chaîne de caractère et transférées au travers de Typhoon via SOAP. Cette approche est

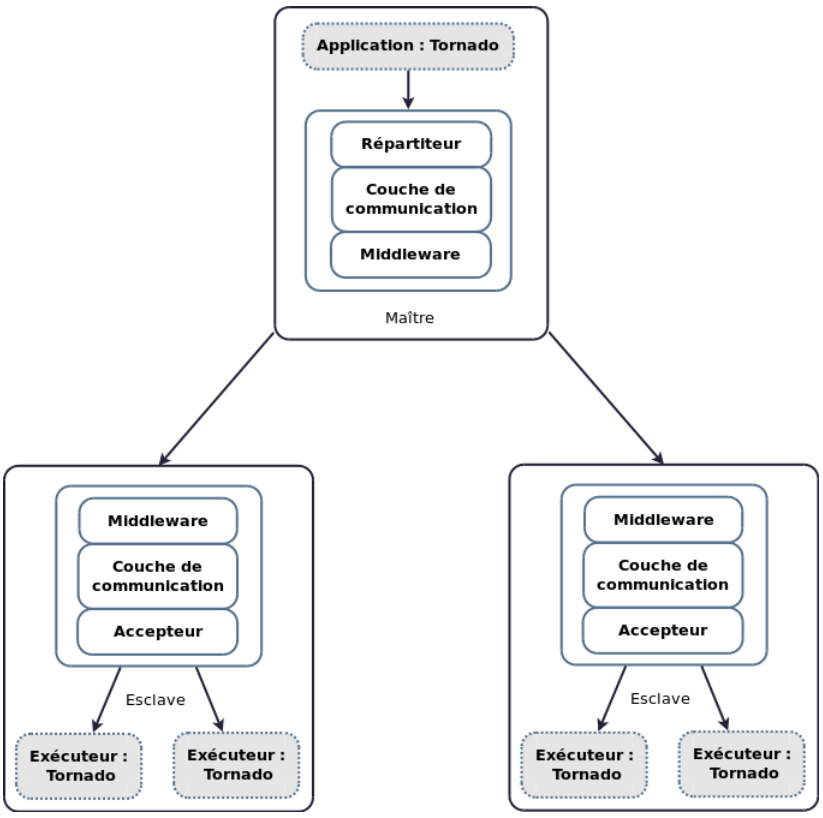

FIGURE 3.1 – Typhoon : Aperçu général, d'après [7]

pratique pour des ressources de taille limitée (<1Mo), mais relativement inefficace pour des ressources plus conséquentes. Ce mécanisme est illustré à la figure 3.2.

- Utilisation d'identifiants de fichiers ressources : dans ce mécanisme, les ressources ne sont pas enrobées. Des références vers les emplacements des ressources sur le disque local Maître ou Esclave sont transmises. L'inconvénient lié à cette approche est la nécessité d'un recours à un moyen externe pour récupérer les données sur les différents disques. L'utilisation d'un disque partagé est donc suggérée. Ce mécanisme est illustré à la figure 3.3.
- Utilisation d'identifiants HTTP de ressources : ce mécanisme est similaire au précédent, si ce n'est qu'ici, l'identifiant ne pointe pas vers un emplacement sur un disque local, mais vers une ressource rendue

disponible par un serveur Web, pouvant être téléchargée par le protocole HTTP. Ce mécanisme est illustré à la figure 3.4.

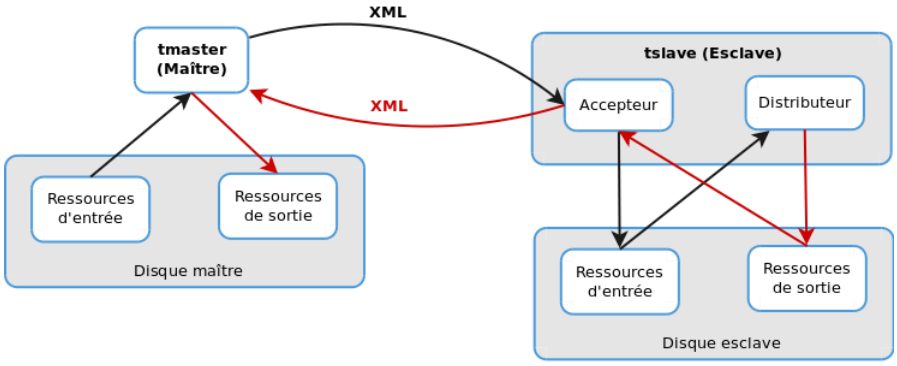

FIGURE 3.2 – Typhoon : Echange de données enrobées, d'après [2]

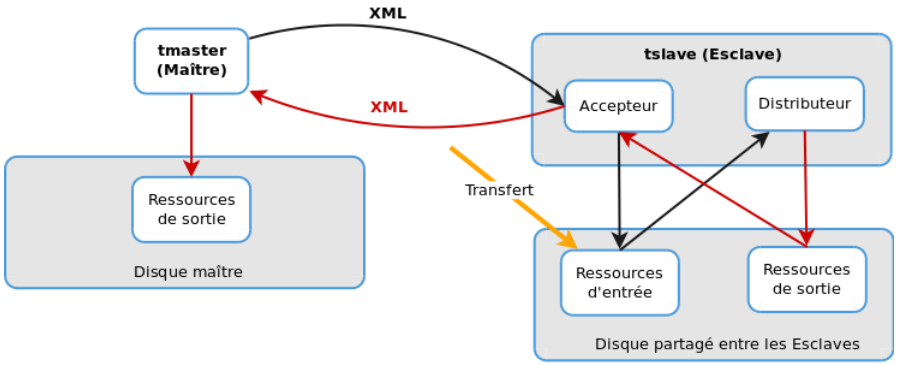

FIGURE 3.3 – Typhoon : Echange de données avec identifiants référentiels, d'après [2]

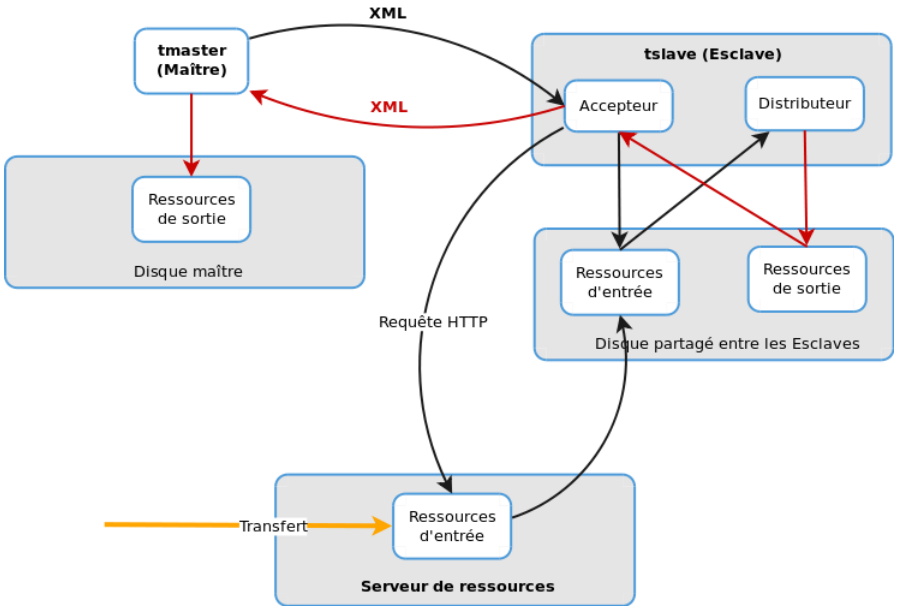

FIGURE 3.4 – Typhoon : Echange de données avec identifiants HTTP, d'après [2]

3.1.2 Définition des tâches

La définition des tâches se fait à partir d'un fichier XML ("**.Jobs.XML*", un exemple est disponible à l'Annexe D). Ce fichier établit la liaison entre le logiciel et les différentes tâches à effectuer.
Chaque tâche ("Job") s'y retrouve définie par une série de paramètres nécessaires à son exécution : nom, localisation des différents fichiers d'entrée nécessaires à son exécution (**.Exp.xml*, **.SymbModel.xml*, **.so* et **.txt* éventuel).

L'architecture de ce fichier XML est reprise à la figure 3.5.

3.1.3 Répartition des tâches

La répartition des tâches entre les différents esclaves se fait à l'aide d'un fichier XML, *Typhoon.Main.xml*, qui contient les *URL* des différents esclaves qui peuvent être utilisés.

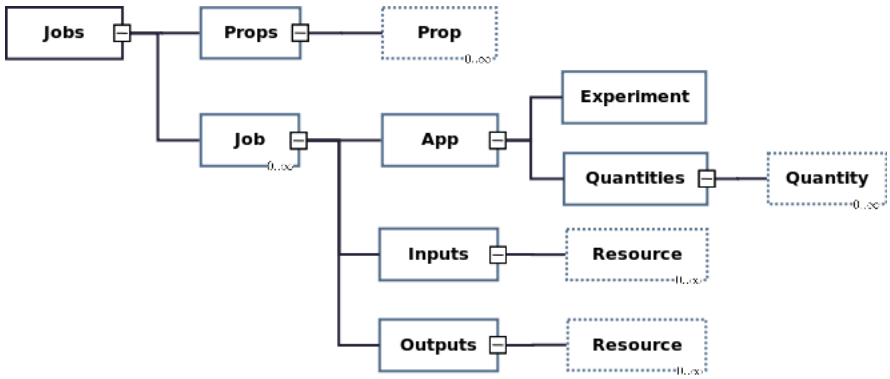

FIGURE 3.5 – Typhoon : Description des tâches, d'après [3]

3.2 Déploiement

Le déploiement de Typhoon suppose qu'une version de Tornado est déjà installée sur chaque poste utilisé (Maître et Esclaves).
Il est à noter que l'installation de Typhoon sur une plateforme Linux/UNIX nécessite la présence de l'API TornadoCPP.

3.2.1 Compilation

La compilation de Typhoon suit la même logique que celle de Tornado. Pour une compilation sous Linux/UNIX à l'aide du compilateur GCC, des fichiers de type *Makefile* sont disponibles. Pour une compilation sous Windows, on dispose des fichiers nécessaires à l'utilisation du compilateur Microsoft Visual C++.

3.2.2 Installation

L'installation de Typhoon doit s'effectuer sur les postes maîtres et esclaves. Des fichiers d'installation sont disponibles pour les systèmes Linux/UNIX et Windows.

3.3 Utilisation

L'utilisation de Typhoon se fait à l'aide d'un interpréteur de lignes de commandes.
La première étape consiste en la spécification des emplacements (URL)

des postes esclaves. Ceci s'effectue en indiquant au programme quel fichier XML de configuration utiliser (un exemple de fichier de configuration est disponible à l'Annexe E), via la commande :

```
tmaster -c <XMLFile>
```

Il est nécessaire ensuite de démarrer le service Typhoon sur les postes esclaves. Ceci s'effectue en lançant la commande suivante sur chacun de ces postes :

```
tslave
```

Ensuite, il suffit de soumettre le fichier XML de description des tâches (**.Jobs.xml*) au Maître, via la commande :

```
tmaster <XMLFile>
```

Les différentes tâches spécifiées seront alors soumises aux Esclaves et exécutées en fonction des ressources disponibles.

Il est à noter que d'autres options sont également disponibles afin de spécifier, par exemple, les numéros des ports adressés, le nombre maximum de tâches à démarrer simultanément, l'intervalle à attendre entre deux tentatives d'exécution etc.
La liste de ces différentes options est disponible au travers de la commande :

```
tmaster -h
```

De même pour les postes esclaves, les différentes options complémentaires sont disponibles au travers de la commande :

```
tslave -h
```

Enfin, les informations relatives à l'exécution des tâches sont disponibles dans différents fichiers *log*, localisés sur les différentes plateformes.

Deuxième partie

Développement et solution

Chapitre 4

Déploiement de Tornado sur Colosse

Ce chapitre explique les différentes étapes du déploiement de la version 0.39 du logiciel Tornado sur Colosse.

4.1 Adaptation à Colosse

Les versions compilées et directement installables de Tornado n'existant que pour des plateformes 32 bits, ce déploiement passe dans un premier temps par une compilation de l'application sur Colosse qui est, comme spécifié au Chapitre 2, une plateforme 64 bits.
Certaines modifications ont également été apportées dans un souci de compatibilité, que ce soit au niveau du mécanisme de compilation ou au niveau du code.

4.1.1 Emplacement d'installation

Contrairement à ce qui a été réalisé au sein de l'Université de Gand, il n'est pas possible ici d'installer Tornado sur chacun des noeuds de calcul de Colosse.
Comme expliqué au Chapitre 2, SGE charge les composants logiciels requis sur les noeuds de calcul alloués à l'exécution d'une tâche. De ce fait, Tornado ne doit être déployé qu'une seule fois, au sein du système de fichiers Lustre, avec des permissions le rendant accessible à tous les utilisateurs du groupe de travail model*EAU*. Ceux-ci pourrant alors l'exécuter sur les noeuds de

calcul en faisant appel à SGE.

Une autre solution serait de compiler Tornado en tant que module chargeable par les utilisateurs de Colosse, au même titre que les différents logiciels disponibles sur ce système. Cette option est toutefois à rejeter dans la mesure où elle ne respecte pas les clauses d'exploitation de Tornado (réservé à une utilisation au sein de modelEAU).

4.1.2 Mécanisme de licence

Comme introduit au Chapitre 1, Tornado fonctionne avec une licence. Suivant l'utilisation qui est faite du logiciel, celle-ci est soit statique et dépendante de l'adresse MAC et du nom d'hôte de la machine, soit fournie via un serveur de licences.

Lorsqu'un logiciel est exécuté à l'aide de SGE, celui-ci est chargé sur le noeud de calcul sur lequel il sera exécuté, comme montré à la figure 4.1 (pour rappel, chaque noeud de calcul dispose de 24 GO de mémoire RAM).

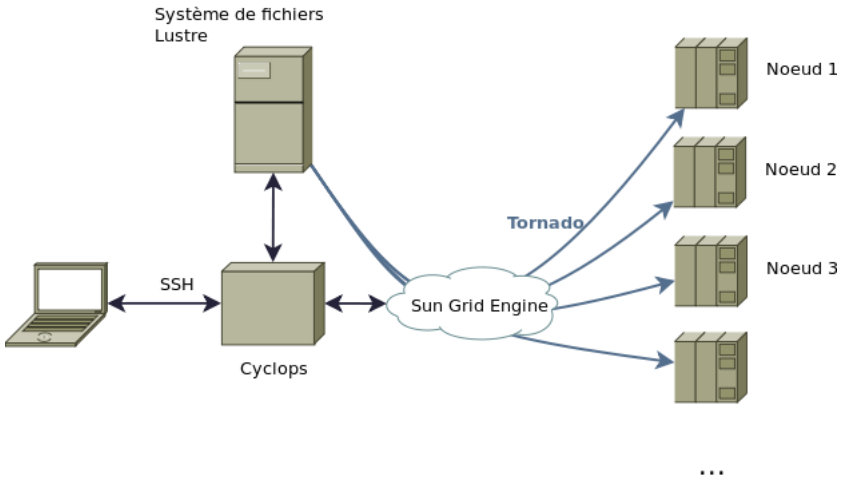

FIGURE 4.1 – Déploiement de Tornado : Chargement du logiciel sur les noeuds de calcul

Ainsi, lorsqu'une instance de Tornado sera exécutée sur un noeud de Colosse, le mécanisme de gestion de licence cherchera un fichier de licence correspondant aux caractéristiques du noeud de calcul (adresse MAC et

nom d'hôte). Ceci signifie que, si l'on désire utiliser des fichiers de licence statiques, 960 fichiers de licence seront nécessaires.
La solution la plus adéquate serait alors de passer par un serveur de licences, accessible depuis les noeuds de calcul de Colosse.

Toutefois, étant donné la nature particulière de ce déploiement, une autre solution a été adoptée : le retrait complet du mécanisme de licence. Cette solution est également utilisée sur le cluster situé à Gand.

4.1.3 Options de compilation

Une première étape consiste à déterminer quelles options supplémentaires (*flags*) sont nécessaires pour compiler Tornado sur Colosse.

Position Independent Code

Tornado est constitué d'une série de modules accessibles par un noyau. Sous Linux, ces modules sont constitués d'un ensemble d'objets compilés (fichiers d'extension **.o*) regroupés en librairies. Ces librairies seront ensuite utilisées par les instances de Tornado qui seront exécutées sur Colosse.

La compilation de *Shared Objects* sur une plateforme 64 bits nécessite l'ajout du *CFLAG*[1] "*-fPIC*" (Position Independent Code). Ceci signifie que le code exécutable fournit par les librairies pourra être exécuté indépendamment de sa position en mémoire [11], grâce à l'utilisation d'une *Global Offset Table* (voir figure 4.3). La figure 4.2 illustre le mécanisme utilisé à l'aide d'un exemple simple.

Ce *CFLAG* devra donc être systématiquement spécifié dans les fichiers de paramétrisation de compilation des modules constituant Tornado (CLAPACK, OpenTop et F2C y compris).

4.1.4 Adaptation du code

Tout code compilé sur Colosse nécessite l'ajout du *CFLAG -fPIC*. Ceci s'applique donc également à la compilation de modèles traduits en langage *C*.

1. Variable d'environnement utilisée pour paramétrer la compilation d'un logiciel

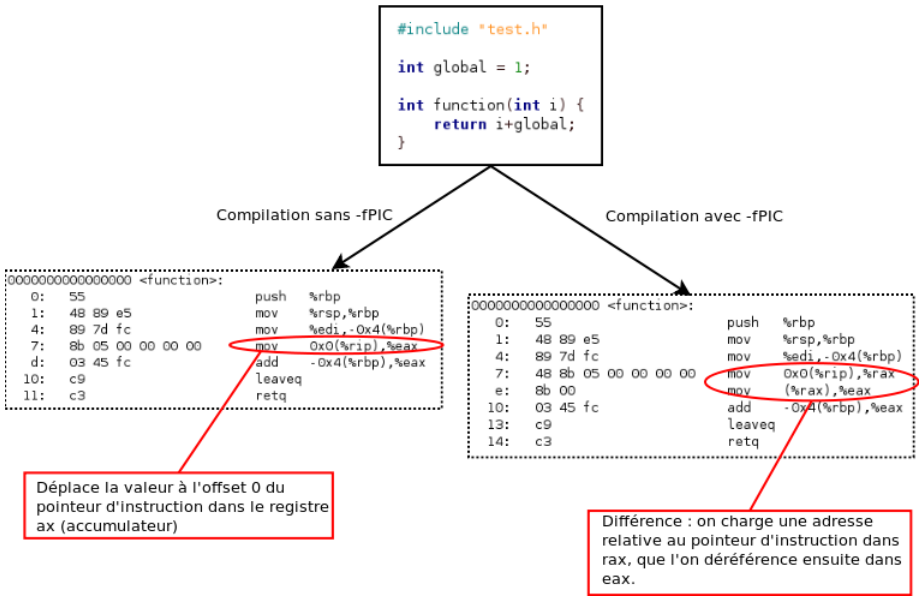

FIGURE 4.2 – Déploiement de Tornado : Position Independent Code, d'après [5]

Comme expliqué au Chapitre 1, il est possible de spécifier cette option à l'aide d'une option intégrée à la commande *tbuild*, comme suit :

```
tbuild -C -fPIC <NOM_DU_MODELE>
```

Cette option étant à spécifier systématiquement, il est toutefois plus intéressant de l'intégrer directement à la commande *tbuild*. Une modification a donc été apportée au niveau du code, dans le fichier *Build.cpp* (cfr. Annexe A).

Lors de tests préliminaires, il a également été remarqué que les librairies *libF77.a* et *libI77.a* du module F2C ne pouvaient être localisées par le programme. Ceci était dû à une erreur de syntaxe dans le fichier *Build.cpp*. La correction est visible à l'Annexe A.

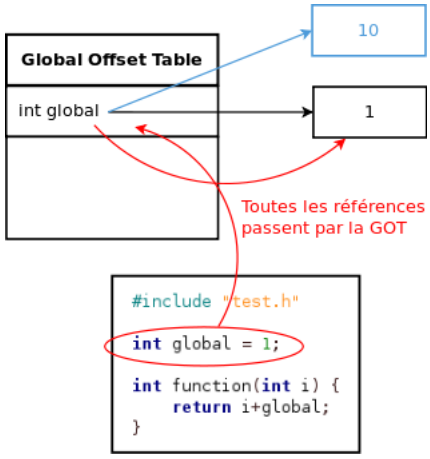

FIGURE 4.3 – Déploiement de Tornado : Global Offset Table, d'après [5]

4.2 Choix du compilateur

Comme expliqué au Chapitre 2, Colosse propose plusieurs compilateurs :
- Gnu C Compiler (GCC) : versions 4.1.2, 4.4.2 et 4.4.2+g77.
- Intel C++ Compiler (ICC) : version 11.1.059.
- Sun Studio : version 12.1.

Les instances de Tornado utilisées à l'Université de Gand ont été compilées à l'aide de GCC. Ces instances sont stables et éprouvées. Il semble donc logique d'utiliser les mêmes outils de compilation dans le but d'assurer un maximum de stabilité à la version qui sera déployée sur Colosse.

La version disponible la plus récente sera utilisée, soit GCC 4.4.2 ; la version GCC 4.4.2+g77 est rejetée car elle n'apporte aucun bénéfice, g77 étant un compilateur Fortran.

De plus, la librairie C++ standard utilisée sera celle de version 4.2.1, requise par Tornado.

4.3 Compilation

Cette section concerne la compilation des différents modules composant Tornado.

4.3.1 Modules prérequis

Comme expliqué au Chapitre 1, différents logiciels et librairies doivent être installées et/ou compilés avant de procéder à la compilation de Tornado. La table 4.1 établit la liste de ces modules en détaillant si oui ou non ils sont déjà présents sur Colosse et en reprenant la version requise pour chacun de ces modules.

TABLE 4.1 – Déploiement de Tornado : Modules prérequis

Module	**Présent sur Colosse Oui (version - emplacement)/Non**	**Version requise**
OpenSSL	0.9.8e-fips-rhel5, */usr/include/openssl*	0.9.6 ou plus
Bison	2.3, */usr/bin/bison*	Non spécifié
Flex	2.5.4, */usr/bin/flex*	Non spécifié
CLAPACK	Non	3.2.1
F2C	Non	Non spécifié
OpenTop	Non	1.5.1

CLAPACK

CLAPACK est disponible sous forme d'archive sur *Netlib*[2], un site web proposant diverses librairies, logiciels, bases de données et publications à caractère scientifique. La compilation de CLAPACK s'effectue en étapes successives :

1. Compilation des librairies F2C intégrées à CLAPACK.
2. Compilation de la librairie BLAS.
3. Compilation de CLAPACK lui-même.

La marche à suivre pour ces différentes compilations est reprise dans le fichier *README.install* disponible avec CLAPACK. Elle se résume aux lignes de commande suivantes :

2. http ://www.netlib.org

```
tar -xzvf clapack.tgz
cd CLAPACK-3.2.1
make f2clib
make blaslib
cd INSTALL
make
cd ../SRC
make
mv ../INCLUDE ../include
```

Le CFLAG "*-fPIC*" est à spécifier dans le fichier *make.inc* présent dans le répertoire racine de CLAPACK.
Il est à noter que Tornado fait appel à des fichiers d'en-tête de CLAPACK, contenus dans le répertoire *INCLUDE* du projet. Ce répertoire doit être renommé en *include* afin de respecter les standards utilisés par Tornado (autrement ce dernier ne pourra pas accéder aux fichiers dont il a besoin).

F2C

Tornado nécessite deux librairies intégrées dans F2C : *libF77.a* et *libI77.a*.
Lors de la compilation des librairies F2C intégrées au projet CLAPACK, libF77 et libI77 sont regroupées en une librairie unique : *libf2c.a*.
Deux options sont alors disponibles : soit désarchiver la librairie *libf2c.a*, en extraire les composants relatifs à *libF77.a* et *libI77.a* et réassembler ces composants en deux librairies, soit compiler ces deux librairies à l'aide des fichiers disponibles sur *Netlib*.
C'est la deuxième option qui a été choisie, pour des raisons de rapidité : elle consiste en l'exécution de deux scripts shell créant l'ensemble des fichiers nécessaires à la compilation de chacune de ces librairies, et à la compilation de chacune de ces librairies à l'aide d'un fichier *Makefile*.
Le CFLAG "*-fPIC*" est à spécifier dans chacun des fichiers *Makefile*.

OpenTop

La version 1.5.1 d'OpenTop est disponible via le site web de *MOSTforWATER*[3].
Sa compilation nécessite les étapes suivantes :

3. http ://forum.mostforwater.com/

1. Convertir l'ensemble des fichiers réguliers (fichiers contenant des données) au format UNIX.
2. Ajouter une permission d'exécution aux fichiers de configuration.
3. Exécuter le fichier *configure* afin de créer le fichier *Makefile* nécessaire à la compilation.
4. Ajouter les CFLAG's "*-fPIC*" et "*-fpermissive*" [4] dans le fichier *Makefile*.
5. Compiler OpenTop en version supportant les *Wide Character Strings*, sous forme de librairie partagée.

Ces étapes se traduisent par les lignes de commande suivantes [5] :

```
unzip opentop-1-5-1.zip
find opentop-1-5-1 -type f -exec dos2unix {} \;
cd opentop-1-5-1
chmod +x config*
./configure
###
#Ajout de -fPIC et de -fpermissive aux flags
###
nano ./buildtools/gcc_compiler_options
make release_multi_wchar_shared
```

4.3.2 Compatibilité de Tornado avec GCC 4.4.2

Depuis la version 4.3 de GCC, la plupart des fichiers d'en-tête de la librairie C++ standard n'incluent plus qu'un nombre très limité (le plus petit possible) de fichiers additionnels, comme décrit dans [10]. De ce fait, les programmes qui utilisaient, par exemple, `std::memcpy` sans inclure <`cstring`> ou `std::auto_ptr` sans inclure <`memory`> ne se compilent plus avec les versions de GCC 4.3 et ultérieures.

4. Cette option permet d'éviter des erreurs de compilation dues à un code non respectueux des standards.
5. Ces lignes de commande proviennent du forum réservé aux utilisateurs de Tornado.

TABLE 4.2 – Déploiement de Tornado : Common : En-têtes additionnels

Fichier	<cstring>	<memory>	<cstdio>
Crypto/Crypto.h	√		
Vector/Vector.cpp.inc	√		
Parser/Parser.h	√		√
Platform/Platform.h	√		
Time/Time.h	√		
Ex/Ex.h	√	√	
Interface/ICallbackMessage.h		√	

TABLE 4.3 – Déploiement de Tornado : Tornado : En-têtes additionnels

Fichier	<memory>	<climits>
Common/MSLE/ExecCalcVar.h	√	
Common/XML/XMLLicense.h		√
Common/Main/Globals.h		√

Tornado fait un usage intensif de `std::auto_ptr` ainsi que d'autres classes pour lesquelles il est nécessaire, depuis GCC 4.3, de spécifier un fichier d'en-tête additionnel. Tornado étant maintenu sur une plateforme Windows et compilé à l'aide de Microsoft Visual C++ 9, les fichiers d'en-tête requis ne sont pas spécifiés dans le code. Il est donc nécessaire de les y ajouter, tout en essayant de minimiser le nombre de fichiers impliqués.
Les tableaux 4.2 et 4.3 reprennent pour les modules concernés les fichiers ayant subi une modification et spécifient quels en-têtes ont dû être ajoutés. Le chemin d'accès spécifié est donné à partir du répertoire "*include/<NOM_DU_MODULE>*" présent dans chaque module.

4.3.3 Variables d'environnement

La compilation de Tornado nécessite la spécification d'une série de variables d'environnement, nécessaires pour localiser les différentes librairies utilisées. Ces variables d'environnement peuvent être déclarées dans le fichier *.bash_profile* (ou équivalent) de l'utilisateur.

4.3.4 Compilation des modules

La compilation de chaque module se résume aux commandes suivantes :

```
cd Racine_du_module/build/linux
source ../../etc/linux/<NOM_DU_MODULE>.conf.sh
make
```

Les fichiers **.conf.sh*, propres à chaque module, contiennent les variables d'environnement complémentaires nécessaires à leur compilation.

4.4 Tests

Les tests de validation se sont effectués en deux séries :

- Une première série de tests a été réalisée en exécution locale (sur le serveur *Cyclops*) à l'aide des fichiers d'exemple fournis au sein de chaque module composant Tornado, ainsi qu'à l'aide de tutoriels de base.
- Une deuxième série de tests a été réalisée sur les noeuds de calcul. Elle consistait en l'exécution d'expériences virtuelles portant sur différents modèles.

4.4.1 Tests en exécution locale

Les fichiers d'exemple fournis avec les modules de Tornado permettent de réaliser des tests unitaires portant sur les différentes fonctionnalités assurées par ces modules.
La figure 4.4 détaille l'arborescence des fichiers requis pour la compilation de ces exemples.

L'exécution de ces exemples a permis de déceler une erreur, au sein du module *Common>Encode*. Cette erreur concerne l'utilisation de *wchar*'s et nécessite l'ajout du *flag* `-fwide-exec-charset=utf-8` lors de sa compilation afin d'avoir une exécution dont les résultats correspondent aux attentes.

Une fois ces exemples exécutés avec succès, un deuxième test local a été effectué. Ce test a consisté en la réalisation des étapes de construction et d'exécution d'un modèle simple, basé sur les tutoriels disponibles sur le site

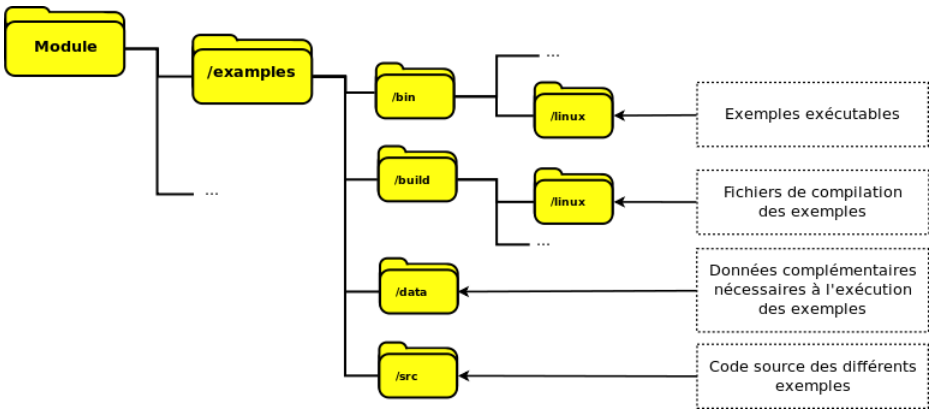

FIGURE 4.4 – Déploiement de Tornado : Arborescence des fichiers de tests unitaires

Web de développement de Tornado. En parallèle, le même modèle a été créé à l'aide d'une version de Tornado stable, et les mêmes commandes ont été exécutées, de façon à comparer les résultats obtenus.

Modèle

Le modèle est écrit à l'aide du langage *Modelica*, et sauvegardé dans un fichier, *Test.mof*. Ce fichier est ensuite traduit en langage *C* à l'aide de l'outil *mof2t* de Tornado :

```
mof2t Test.mof
```

Les fichiers obtenus sont alors :

- *Test.cP* (au lieu du résultat attendu, *Test.c*).
- *Test.SymbModel.xml* (correspond au résultat attendu).

Une comparaison est ensuite effectuée entre le contenu du fichier *Test.cP* et un fichier *Test.c* obtenu à partir du même modèle via la version stable de Tornado. Le contenu étant correct, l'erreur ne se situe qu'au niveau du nom de fichier. Celui-ci est donc renommé en *Test.c* pour la suite du test.

La compilation du modèle s'effectue à l'aide de l'outil *tbuild* de Tornado, qui fait appel au compilateur GCC :

```
tbuild Test
```

On obtient un *Shared Object*, *Test.so* (correspond au résultat attendu), qui sera exécutable au sein d'une expérience virtuelle.

Expérience virtuelle : *Simulation*

La première expérience virtuelle créée est une simulation. Celle-ci est définie à l'aide de l'outil *tmain* de Tornado :

```
tmain ExpCreateSimul Test . false
```

Le résultat obtenu correspond à ce qui est attendu : *Test.Simul.Exp.xml*, fichier XML contenant les différents paramètres définissant l'expérience virtuelle.

Exécution de la *Simulation*

Cette simulation peut être exécutée, à l'aide de l'outil *texec* de Tornado :

```
texec ./Test.Simul.Exp.xml
```

Qui produit le fichier de résultats *Test.Simul.out.txt*, correspondant également aux attentes.
Il est à noter que le chemin d'accès au fichier XML de description de l'expérience virtuelle doit être spécifié, de manière relative ou absolue. Si cela n'est pas effectué, l'exécution de l'expérience virtuelle produira une erreur, consistant en une incapacité de trouver le fichier **.so* requis.

Expérience virtuelle : *Objective Evaluation*

Une deuxième expérience virtuelle est ensuite créée, consistant cette fois-ci en une évaluation objective. Elle se construit via la commande suivante :

```
tmain ExpCreateObjEval Test.Simul.Exp.xml .
```

Ceci produit le fichier *Test.ObjEval.Exp.xml*, qui contient les paramètres nécessaires à cette autre expérience. Après ajout de certains paramètres, on rétablit ce fichier sous sa forme canonique à l'aide de la commande :

```
texp ./Test.ObjEval.Exp.xml
```

A nouveau, cette commande produit les résultats attendus : le fichier *Test.ObjEval.Exp.xml* est réarrangé par rapport aux paramètres ajoutés.

Exécution de l'*Objective Evaluation*

Cette deuxième expérience s'exécute via la commande :

```
texec ./Test.ObjEval.Exp.xml
```

Qui produit le fichier de résultats *Test.ObjEval.Simul.out.txt*. Ce fichier de résultats étant en tous points identique à celui obtenu avec la version stable de Tornado, on peut en conclure que cet outil fonctionne avec succès.

Expérience virtuelle : *Scenario*

Une troisième et dernière expérience virtuelle est finalement créée, qui consiste en une analyse de scénarios basés sur l'expérience précédente. Cette expérience virtuelle se crée via :

```
tmain ExpCreateScen Test.ObjEval.Exp.xml
```

La commande produit un fichier de description d'expérience virtuelle, *Test.Scen.Exp.xml*, comme attendu. Il est également possible de modifier certains paramètres et de rétablir une forme canonique du fichier à l'aide de la commande *texp*, comme il a été fait pour l'expérience d'évaluation objective.
Dans ce cas-ci, des paramètres ont été spécifiés afin de réaliser dix scénarios différents.

Exécution du *Scenario*

L'exécution de cette dernière expérience virtuelle est lancée via la commande :

```
texec ./Test.Scen.Exp.xml
```

Au lieu de produire les dix fichiers de sortie attendus, l'exécution s'arrête après avoir créé le deuxième. L'exécution des différents scénarios s'effectue en cascade, chaque scénario se basant sur les résultats du précédent [6]. L'erreur rencontrée ici se situe au niveau du nom de fichier produit pour la deuxième exécution, *Test.ObjEval.Sim.out.t@* au lieu de *Test.ObjEval.Sim.out.txt.2*. Ceci implique que le logiciel ne peut pas accéder au fichier de résultats requis pour la troisième exécution, et arrête donc l'exécution de l'expérience virtuelle.

Résultats

On remarque que des erreurs peuvent se produire dans les noms des fichiers créés. D'autres tests réalisés avec des noms de fichiers plus longs (quatre caractères de plus que dans le cas précédent) ont systématiquement apporté des erreurs au niveau des fichiers texte de sortie et du fichier *C* produit.
L'inconvénient est donc l'impossibilité de réaliser des expériences virtuelles de type *Scénario*.

4.4.2 Tests sur la grappe de calcul

Après ces premiers résultats mitigés obtenus lors d'exécutions locales, une deuxième série de tests a été effectuée, cette fois-ci sur la grappe de calcul, en variant les paramètres fournis à SGE.

Exécution d'une expérience virtuelle sur la file mpi

Ce test a consisté en l'exécution, sur la grappe de calcul, de l'expérience virtuelle de type *Simulation* réalisée durant les tests précédents.
Le script soumis à SGE est repris à l'Annexe B.1.

6. Excepté pour le premier qui se base sur un autre fichier de résultats (provenant dans ce cas-ci de l'évaluation objective).

Le résultat de cette expérience virtuelle est un fichier au nom erronné : *Test.Simul.out.txtP*, dont le contenu est correct.

Exécution d'une expérience virtuelle en parallèle

Comme expliqué au point 2.6.3, il est possible d'exécuter plusieurs tâches en parallèle, en fonction des noeuds de calcul disponibles.
Le but de ce test est d'essayer cette possibilité mais en n'exécutant qu'une seule tâche, la même qu'au point précédent. Le script soumis à SGE est repris à l'Annexe B.2 (le fichier *Test.Simul.Exp.xml* a été renommé en *Test.1.Simul.Exp.xml* de façon à être exploitable par SGE).
Le résultat de cette expérience virtuelle est également un fichier au nom erronné : *Test.Simul.out.txtP*, dont le contenu est correct.

Exécution de plusieurs expériences virtuelles en parallèle

Ce test-ci consiste en l'exécution en parallèle de dix tâches. Celles-ci ont été fournies par Hélène Hauduc, une doctorante au Cemagref et à model*EAU*.
Le script soumis à SGE est repris à l'Annexe B.3.
Les résultats obtenus correspondent aux attentes, soient dix fichiers nommées *TwoASU_1.Simul.out.txt* à *TwoASU_2.Simul.out.txt*.

Déboguage à l'aide de Gnu Debug

Au vu des problèmes rencontrés durant ces phases de test, il a semblé intéressant de réaliser un fichier de code test reprenant les fonctions, déclarées dans Tornado, faisant usage de *wide character strings* (*wstring*s) et utilisées dans le processus de création de fichiers (code repris à l'Annexe C).
Ce code a donc été compilé de façon à être débogué à l'aide du débogueur Gnu Debug (GDB), et de multiples *breakpoints* ont été ajoutés par la suite, de façon à pouvoir vérifier les valeurs des différentes variables déclarées dans ce code.
Aucune erreur n'a été décelée dans ce code, ce qui laisse supposer que le problème se situe au niveau de la librairie C++ standard déployée sur Colosse.

Chapitre 5

Utilisation de Typhoon

Au Chapitre 2, il est expliqué que Colosse utilise l'ordonnanceur de tâches Sun Grid Engine. Ce dernier contrôle l'ensemble des noeuds de calcul et rend impossible leur utilisation au travers d'un autre logiciel. Le déploiement de Typhoon sur Colosse n'est donc pas une solution envisageable.

L'objectif initialement recherché avec le déploiement de Typhoon était de permettre aux utilisateurs de Tornado d'utiliser la grappe de calculs de façon entièrement transparente. Ceci signifie qu'un utilisateur aurait dû pouvoir, au départ de son logiciel, lancer une série de tâches en parallèle et en récupérer les résultats sans même savoir que leur exécution avait eu lieu sur Colosse.

Ce chapitre propose donc une solution intermédiaire : l'implémentation d'un outil dédié à la conversion de fichiers de description de tâches au format utilisé par Typhoon (cfr. Annexe D) en fichiers de description de tâches au format utilisé par SGE.

5.1 Aperçu général

La conversion d'un fichier de descriptions de tâches au format de Typhoon en un script de description de tâches SGE correspond en la conversion d'un fichier XML reprenant les spécifications des différentes tâches à effectuer en un fichier de script batch lançant l'exécution de ces tâches.
Les fichiers de description Typhoon permettent d'exécuter en parallèle plusieurs expériences virtuelles de types complètement différents sans relation au-

cune du point de vue “Nom de l’expérience”. Or, Colosse nécessite l’utilisation de la variable $SGE_TASK_ID dans la commande lançant l’exécution d’une tâche, ceci afin de lancer différentes tâches en parallèle. Il est donc nécessaire de séparer deux cas d’utilisation : ceux consistant en l’exécution d’expériences virtuelles pouvant être lancées simultanément car liées entre elles par un ou plusieurs caractères alphanumériques présents dans leur nom, et ceux consistant en l’exécution d’expériences virtuelles devant être lancées séquentiellement.
Deux outils de conversion de fichiers de description de tâches au format de

Typhoon existent déjà au sein de Tornado :

- *tjobs2batch*, qui permet la conversion d’un fichier de description de tâches au format de Typhoon en un script batch dont l’interprétation lance l’exécution séquentielle des différentes expériences spécifiées dans le fichier initial.
- *tjobs2jdl*, qui permet la conversion d’un fichier de description de tâches au format de Typhoon en un fichier JDL (Job Description Language, une extension de XML servant à la description des aspects d’une tâche).

5.2 Implémentation

L’outil développé est donc également implémenté au sein de Tornado, et est nommé *tjbos2sge*. Son implémentation est basée sur celle des outils déjà existants.

5.2.1 Analyse syntaxique

L’analyse syntaxique d’un document XML permet l’identification des différents éléments qui y sont repris. Cette analyse syntaxique fait ici appel à différentes fonctions implémentées au sein de la librairie OpenTop, dans les *packages sax* (Simple API for XML) et *xml*.
Une fois cette analyse syntaxique effectuée, il est possible de stocker en mémoire les différents éléments décrits dans le fichier XML et de les manipuler comme tels.

5.2.2 Détection du cas d’utilisation

Cette détection est relativement aisée : il suffit de parcourir l’ensemble des éléments et d’en comparer les noms d’expérience. Si l’unique différence

existant entre ces noms d'expérience est de type numérique, et qu'une suite logique peut être définie par ces nombres, on se retrouve dans un cas d'utilisation parallèle. Autrement, on se retrouve face à un cas d'exécution séquentielle.

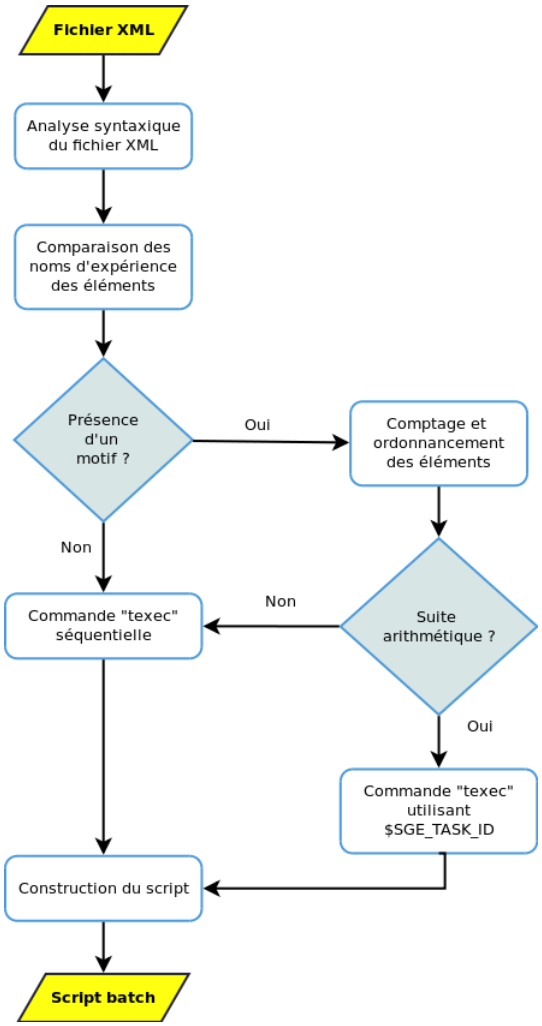

FIGURE 5.1 – Utilisation de Typhoon : Détection du cas d'utilisation

Afin de parcourir les différents éléments, il semblerait logique d'utiliser des expressions régulières. Malheureusement, il n'existe pas actuellement

d'implémentation de telles expressions au sein de la librairie C++ standard. Toutefois, certaines librairies C++, libres pour la plupart, disposent de fonctions permettant l'utilisation d'expressions régulières (*Boost*[1] et *PCRE*[2], notamment) ; il serait donc intéressant de déterminer si l'une d'entre elles pourrait être intégrée au sein de Tornado, au même registre qu'OpenTop, CLAPACK et F2C.
Le mécanisme de détection du cas d'utilisation implémenté ici suit le schéma

montré à la figure 5.1. Il suppose que le nom du fichier d'expérience a une structure du type :

<CARACTERES_ASCII> <NOMBRE> <CARACTERES_ASCII>

(où <CARACTERES_ASCII> est une chaîne de caractères non numériques). La fonction de comparaison s'assure donc que la seule différence existant entre deux noms de fichier est la partie <NOMBRE>.
Le motif est défini par une comparaison des noms de fichier d'expérience des deux premières tâches décrites dans le fichier de description au format de Typhoon. Les noms de fichier des tâches suivantes sont ensuite analysés afin de vérifier qu'ils contiennent le motif. Si cela se vérifie pour l'ensemble des tâches, un comptage est effectué pour déterminer si les parties <NOMBRE> relatives à chaque tâche constituent une suite arithmétique. Dans ce cas, le script lancera la commande *texec* en utilisant la variable d'environnement *$SGE_TASK_ID*. Si aucun motif ou suite arithmétique n'est détecté, le script lancera la commande *texec* de façon séquentielle, c'est-à-dire :

```
texec <TACHE_1> <TACHE_2> ... <TACHE_N>
```

5.2.3 Code source

Comme expliqué ci-dessus, le code source est largement basé sur celui de *tjobs2batch* et de *tjobs2jdl*. Les fichiers source suivent donc la même arborescence, comme montré à la figure 5.2.

1. http ://www.boost.org
2. http ://www.pcre.org

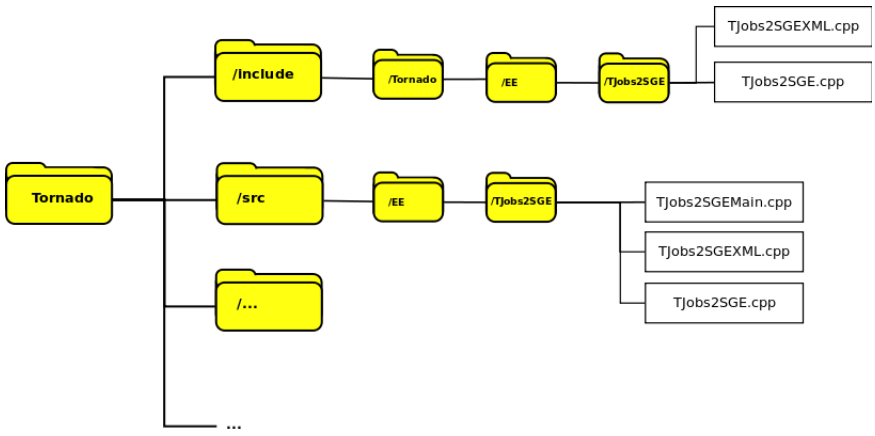

FIGURE 5.2 – Utilisation de Typhoon : Localisation du code source

Ces fichiers sont :

- *TJobs2SGEMain.cpp*, exécuté lors de l'appel de la commande *tjobs2sge*. Tout comme le reste des commandes disponibles dans Tornado, il offre un menu d'assistance basique, reprenant les options disponibles lors de son exécution.
- *TJobs2SGEXML.h*, reprenant les spécifications des fonctions relatives à l'analyse syntaxique du document XML.
- *TJobs2SGEXML.cpp*, implémentant les fonctions décrites dans TJobs2SGEXML.h.
- *TJobs2SGE.h*, définissant des fonctions utilitaires additionnelles, nécessaires pour l'analyse de motifs.
- *TJobs2SGE.cpp*, implémentant les fonctions décrites dans TJobSGE.h.

Le code de *TJobs2SGE.h* et *TJobs2SGE.cpp* est repris à l'Annexe F.

5.3 Utilisation

L'utilisation de cet outil se résume à la commande :

```
tjob2sge <Typhoon_Jobs_File>
```

qui créera donc un script, au format requis par SGE, permettant de lancer l'exécution des tâches reprises dans le fichier *<Typhoon_Jobs_File>*. La figure 5.3 montre un exemple de conversion d'un fichier de description au

format utilisé par Typhoon en un fichier de description utilisable par SGE.

Description XML utilisée par Typhoon

```
<Typhoon>
  <Jobs Version="1.0" Desc="1 job">

    <Job Name="TwoASU_1">
      <App Version="1.0" Name="Tornado.Exp">
        <Experiment Name="TwoASU_1.Simul.Exp.xml"/>
      </App>
      <Inputs>
        <Resource Name="TwoASU_1.Simul.Exp.xml" Embedded="false" URL="$(TORNADO_DATA_PATH)/TwoASU/TwoASU_1.Simul.Exp.xml"/>
        <Resource Name="TwoASU_1.$(COMMON_EXT)" Embedded="false" URL="$(TORNADO_DATA_PATH)/TwoASU/TwoASU_1.$(COMMON_EXT)"/>
        <Resource Name="TwoASU_1.SymbModel.xml" Embedded="false" URL="$(TORNADO_DATA_PATH)/TwoASU/TwoASU_1.SymbModel.xml"/>
      </Inputs>
      <Outputs>
        <Resource Name="TwoASU_1.Simul.out.txt" Embedded="true"/>
      </Outputs>
    </Job>
    <Job Name="TwoASU_2">
      <App Version="1.0" Name="Tornado.Exp">
        <Experiment Name="TwoASU_2.Simul.Exp.xml"/>
      </App>
      <Inputs>
        <Resource Name="TwoASU_2.Simul.Exp.xml" Embedded="false" URL="$(TORNADO_DATA_PATH)/TwoASU/TwoASU_2.Simul.Exp.xml"/>
        <Resource Name="TwoASU_2.$(COMMON_EXT)" Embedded="false" URL="$(TORNADO_DATA_PATH)/TwoASU/TwoASU_2.$(COMMON_EXT)"/>
        <Resource Name="TwoASU_2.SymbModel.xml" Embedded="false" URL="$(TORNADO_DATA_PATH)/TwoASU/TwoASU_2.SymbModel.xml"/>
      </Inputs>
      <Outputs>
        <Resource Name="TwoASU_2.Simul.out.txt" Embedded="true"/>
      </Outputs>
    </Job>
    <Job Name="TwoASU_3">
      <App Version="1.0" Name="Tornado.Exp">
        <Experiment Name="TwoASU_3.Simul.Exp.xml"/>
      </App>
      <Inputs>
        <Resource Name="TwoASU_3.Simul.Exp.xml" Embedded="false" URL="$(TORNADO_DATA_PATH)/TwoASU/TwoASU_3.Simul.Exp.xml"/>
        <Resource Name="TwoASU_3.$(COMMON_EXT)" Embedded="false" URL="$(TORNADO_DATA_PATH)/TwoASU/TwoASU_3.$(COMMON_EXT)"/>
        <Resource Name="TwoASU_3.SymbModel.xml" Embedded="false" URL="$(TORNADO_DATA_PATH)/TwoASU/TwoASU_3.SymbModel.xml"/>
      </Inputs>
      <Outputs>
        <Resource Name="TwoASU_3.Simul.out.txt" Embedded="true"/>
      </Outputs>
    </Job>

  </Jobs>
</Typhoon>
```

```
#!/bin/bash

#$ -N ExampleJob
#$ -P yyk-770-aa
#$ -l h_rt=00:00:10
#$ -cwd
#$ -t 1-3

source /rap/yyk-770-aa/.Tornado_Settings.sh
module load compilers/gcc/4.4.2

texec $(TORNADO_DATA_PATH)/TwoASU/TwoASU_$SGE_TASK_ID.Simul.Exp.xml
```

Script batch utilisée par SGE

FIGURE 5.3 – Utilisation de Typhoon : Exemple d'utilisation de TJobs2SGE

Il est à noter que le paramètre "estimation du temps d'exécution des tâches", requis par SGE, n'est pas définissable automatiquement par *TJobs2SGE*. Il est donc à spécifier via l'option "*-rt*" de la commande, ou à modifier par après dans le fichier produit (la valeur par défaut est fixée à 1 minute).

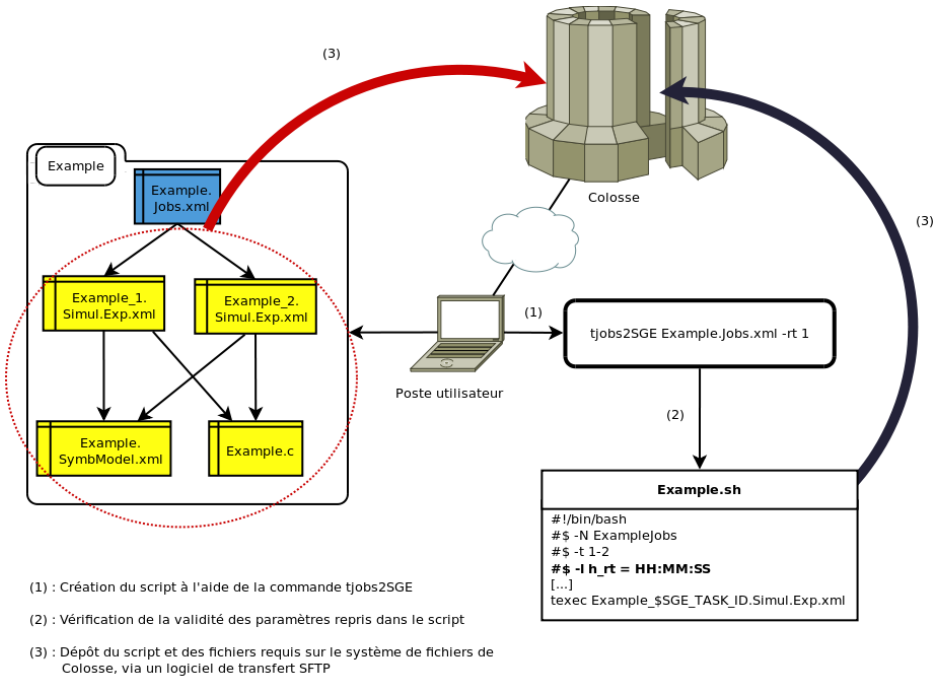

FIGURE 5.4 – Utilisation de Typhoon : Schéma d'utilisation de TJobs2SGE

Ce script doit ensuite être transféré sur Colosse, de même que les différents fichiers d'expériences virtuelles auxquels il fait référence (voir la figure 5.4 pour le principe d'utilisation).

Conclusion

L'implantation de Tornado sur Colosse a été réalisée et est fonctionnelle. Cette implantation est décrite au Chapitre 4.
De façon à faciliter son utilisation future, une attention particulière a été portée sur la transmission de connaissances. Un tutoriel (en anglais) a été rédigé expliquant pas-à-pas la procédure à suivre pour l'utilisation de Tornado sur Colosse. Ce tutoriel est disponible à l'Annexe G.
L'implantation de Tornado est d'autre part sujette à des limitations liées aux caractéristiques des différents systèmes et logiciels implantés sur Colosse ou présents dans Tornado. Des solutions ont donc été développées pour parer en partie à ces limitations.

Les problèmes survenus sont de deux ordres :

- Un problème lié à la création de noms de fichiers, qui restreint l'utilisation de Tornado par exemple dans le cas d'utilisation de scénarios. Ce problème ne peut actuellement être contourné que manuellement, en renommant les fichiers intermédiaires avant de poursuivre les simulations ;
- L'impossibilité de déployer Typhoon sur la grappe de calcul. Ce problème a été contourné par la création d'un outil de conversion des fichiers Typhoon vers des fichiers SGE.

Futures perspectives

Des développements complémentaires pourraient s'avérer intéressants pour faciliter l'utilisation de Tornado sur la grappe de calcul Colosse.

Le premier porte sur l'identification et la résolution du problème lié aux noms de fichiers. Les recherches préliminaires effectuées laissent à penser que ce problème pourrait être lié à la version de la librairie C++ implantée sur

Colosse et utilisée par le compilateur *GCC 4.4.2*. Il pourrait aussi s'agir d'un problème lié à Tornado. Une correction de ce problème pourrait permettre de profiter de la totalité des fonctionnalités de Tornado sur Colosse.

Le second porte sur une transparence de l'accès à Colosse. Afin de simplifier la procédure d'exécution d'instances de Tornado sur Colosse, il pourrait être intéressant de développer un module complémentaire rendant possible l'accès à distance à d'autres instances de Tornado (voir figure 5.5).

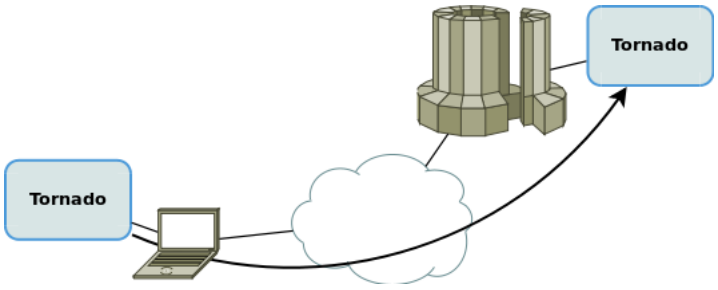

FIGURE 5.5 – Perspectives futures : Utilisation à distance

Ce module devrait fournir un support pour la connexion à un système distant sur lequel une instance de Tornado peut être exécutée (dans ce cas-ci, Colosse), ainsi qu'un outil permettant de télécharger automatiquement les fichiers créés par l'exécution distante d'expériences virtuelles.
L'intérêt de devélopper un tel module est toutefois discutable. En effet, son utilité serait de rendre transparent l'accès à un système distant, et donc d'enlever la nécessité de passer par des logiciels tiers (PuTTY, WinSCP, etc.). L'avantage que l'on en retire est une plus grande facilité de travail dans ces situations particulières. Le désavantage est un développement logiciel complexe et sur mesure (et donc pas, ou peu, portable).

Bibliographie

[1] Equipe de support Clumeq. Clumeq. http ://www.clumeq.ca, Mai 2010.

[2] Claeys F. *A Generic Software Framework for Modelling and Virtual Experimentation with Complex Environmental System.* PhD thesis, Faculty of Bioscience Engineering. Ghent University, Belgium, 2008.

[3] Claeys F., Chtepen M., Benedetti L., De Keyser W., Fritzson P., and Vanrolleghem P.A. Towards transparent distributed execution in the tornado framework. In *Environmental Application and Distributed Computing Conference (EADC), Bratislava, Slovakia*, 2006.

[4] Parent F. Calcul de haute performance à l'université laval. Présentation Powerpoint - Clumeq, Mai 2009.

[5] Wienand I. Position independent code and x86-64 libraries. http ://www.technovelty.org/code/c/amd64-pic.html, Novembre 2008.

[6] Schwartz J. Switching subjects. http ://blogs.sun.com/jonathan/entry/size_matters, Juin 2007.

[7] Chtepen M., Claeys F., Dhoedt B., Vanrolleghem P.A., and Demeester P. Computational complexity and distributed execution in water quality management. In *International Conference on Computational Science (ICCS), Atlanta, GA, USA*, 2005.

[8] MOSTforWATER. Tornado website. http ://forum.most4water.com, Mai 2010.

[9] Vanrolleghem P.A. Introduction to process modelling - biological process modelling (part 1). Summer school on modelling MBR processes, Ghent, Belgium, Juillet 2008.

[10] GCC Team. Gcc 4.3 release series : Porting to the new tools. http ://http ://gcc.gnu.org, Février 2008.

[11] Drepper U. How to write shared libraries. *Red Hat Inc.*, Août 2006.

Annexe A

Build.cpp : modifications

Le listing suivant reprend les modifications apportées à la fonction *Init* de la classe *Build*.

```
void CBuild::
Init(vector<wstring>& CFlags,
     vector<wstring>& LFlags,
     vector<wstring>& Libs)
{

[...]

  else if (m_Platform == L"linux")
  {
    //Added for SGE usage
    CFlags.push_back(L"-fPIC");

    [...]

    //Modified for SGE usage
    Libs.push_back(L"F77");
    Libs.push_back(L"I77");
  }

[...]

}
```

Annexe B

Scripts SGE

Cette annexe reprend les différents scripts utilisés durant les phases de test pour soumettre des tâches à SGE.

B.1 Tâche unitaire sur file MPI

```
#!/bin/bash

#$ -N TestJob
#$ -P yyk-770-aa

#$ -pe mpi 4

#$ -l h_rt=00:00:05

#$ -cwd

source /rap/yyk-770-aa/.Tornado_Settings.sh

module load compilers/gcc/4.4.2

texec $HOME/Test/Test.Simul.Exp.xml
```

B.2 Tâche unitaire en mode parallèle

```
#!/bin/bash
```

```
#$ -N TestJob
#$ -P yyk-770-aa

#$ -l h_rt=00:00:07

#$ -cwd

#$ -t 1-1

source /rap/yyk-770-aa/.Tornado_Settings.sh

module load compilers/gcc/4.4.2

texec $HOME/Test/Test.$SGE_TASK_ID.Simul.Exp.xml
```

B.3 Tâches parallèles

```
#!/bin/bash

#$ -N TwoASU
#$ -P yyk-770-aa
#$ -l h_rt=00:00:10

#$ -S /bin/bash
#$ -cwd

#$ -t 1-10

source /rap/yyk-770-aa/.Tornado_Settings.sh

module load compilers/gcc/4.4.2

texec $HOME/Helene_TwoASU/TwoASU_$SGE_TASK_ID.Simul
      .Exp.xml
```

Annexe C

Code de test - Utilisation de *wstrings*

Cette annexe reprend le code, dénué du contenu des fonctions, utilisé pour vérifier le processus de création de fichiers. L'ensemble de ces fonctions provient de l'espace de nom *CString* défini dans le module *Common*.

```
#include <wchar.h>
#include <ostream>
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <stdio.h>

#ifdef _WIN32
#define RIGHT_SEPARATOR L"\\"
#define WRONG_SEPARATOR L"/"
#else
#define RIGHT_SEPARATOR L"/"
#define WRONG_SEPARATOR L"\\"
#endif

using namespace std;

wstring GetEnv(const wstring& Name);

wstring Fix(const wstring& InPath, bool FixSeparators)
{
  [...]
}

bool WStringToMBS(char*& Dest, const wstring& Src)
{
```

```
  [...]
}

string WStringToString(const wstring& Src)
{
  [...]
}

bool MBSToWString(wstring& Dest, const char* Src)
{
  [...]
}

wstring StringToWString(const string& Src)
{
  [...]
}

wstring GetEnv(const wstring& Name)
{
  [...]
}

const char* ToCStr(const std::wstring& InPath)
{
  [...]
}

wstring Join(const wstring& InPath, const wstring& FileName)
{
  [...]
}

static void SerializeC(const wstring& OutputPath)
{
  [...]
}

int main(int argc_, char** argv_)
{
  SerializeC(L"./src/NomFichierDePlusVingQuatreCaracteres.txt");
}
```

Annexe D

Fichier de description de tâches au format Typhoon

Cette annexe reprend un fichier de description de tâches au format accepté par Typhoon, provenant des exemples disponibles dans le répertoire *data* de Typhoon.

```
<Typhoon>
<Jobs Version="1.0" Desc="2 jobs">

  <Job Name="Influenza">
    <App Version="1.0" Name="Tornado.Exp">
      <Experiment Name="Influenza.Simul.Exp.xml"/>
    </App>
    <Inputs>
      <Resource Name="Influenza.Simul.Exp.xml"
                Embedded="true" URL="$(TORNADO_DATA_PATH)/
________Influenza/Influenza.Simul.Exp.xml"/>
      <Resource Name="Influenza.$(COMMON_EXT)"
                Embedded="true" URL="$(TORNADO_DATA_PATH)/
________Influenza/Influenza.$(COMMON_EXT)"/>
      <Resource Name="Influenza.SymbModel.xml"
                Embedded="true" URL="$(TORNADO_DATA_PATH)/
________Influenza/Influenza.SymbModel.xml"/>
      <Resource Name="Influenza.Simul.in.txt"
                Embedded="true" URL="$(TORNADO_DATA_PATH)/
________Influenza/Influenza.Simul.in.txt"/>
    </Inputs>
    <Outputs>
      <Resource Name="Influenza.Simul.out.txt"
                Embedded="true"/>
    </Outputs>
  </Job>
```

```
    <Job Name="PredatorPrey">
      <App Version="1.0" Name="Tornado.Exp">
        <Experiment Name="PredatorPrey.Simul.Exp.xml"/>
      </App>
      <Inputs>
        <Resource Name="PredatorPrey.Simul.Exp.xml"
                  Embedded="true" URL="$(TORNADO_DATA_PATH)/
________________PredatorPrey/PredatorPrey.Simul.Exp.xml"/>
        <Resource Name="PredatorPrey.$(COMMON_EXT)"
                  Embedded="true" URL="$(TORNADO_DATA_PATH)/
________________PredatorPrey/PredatorPrey.$(COMMON_EXT)"/>
        <Resource Name="PredatorPrey.SymbModel.xml"
                  Embedded="true" URL="$(TORNADO_DATA_PATH)/
________________PredatorPrey/PredatorPrey.SymbModel.xml"/>
      </Inputs>
      <Outputs>
        <Resource Name="PredatorPrey.Simul.out.txt"
                  Embedded="true"/>
      </Outputs>
    </Job>

  </Jobs>
</Typhoon>
```

Annexe E

Fichier de configuration de Typhoon

Cette annexe reprend le fichier de configuration par défaut de Typhoon, disponible dans le répertoire *etc* du logiciel.

```
<Typhoon>
<Main Version="1.0">
  <Slaves>
    <Slave URL="http://localhost:20001"/>
    <Slave URL="http://localhost:20002"/>
  </Slaves>
</Main>
</Typhoon>
```

Annexe F

TJobs2SGE

Cette annexe reprend le code utilisé pour détecter les motifs dans les noms de fichier d'expériences virtuelles spécifiés dans le fichier de description de tâches au format de Typhoon soumis à la commande *tjobs2sge*.

F.1 TJobs2SGE.h

```
#include <string>
#include <vector>

namespace Tornado
{
  namespace SGE
  {
    bool
    CharIsNum(wchar_t& a);

    std::vector<std::wstring>
    GetPatterns(std::wstring& FileName);
  }
}
```

F.2 TJobs2SGE.cpp

```
#include "Tornado/EE/TJobs2SGE/TJobs2SGE.h"

using namespace std;
using namespace Tornado;
```

```
bool
CharIsNum(wchar_t& a)
{
  bool result = false;
  wstring check = L"0123456789";
  unsigned long i = 0;

  while (i < check.size())
  {
    if(a == check[i])
      result = true;
    ++i;
  }
  return result;
}

vector<wstring>
GetPatterns(wstring& FileName)
{
  unsigned long i = 0, j=0;
  bool next = false;
  vector<wstring> Patterns;
  wstring Temp;

  for ( i = 0; i < FileName.size(); ++i)
  {
    if( ! SGE::CharIsNum(FileName[i]))
    {
          Temp += FileName[i];
          j++;
        }
        else if( !next )
        {
          next = true;
          Patterns.push_back(Temp);
          Temp.erase();
          j = 0;
        }
  }
  Patterns.push_back(Temp);
  return Patterns;
}
```

Annexe G

Tutoriel d'utilisation

Cette annexe reprend un tutoriel expliquant comment utiliser Tornado sur le supercalculateur Colosse. Vu le caractère international de l'équipe de recherche modelEAU, celui-ci a été rédigé en anglais.

G.1 SFTP connection

Colosse's file system is accessible through an SFTP (SSH over FTP) connection.

G.1.1 SFTP softwares

This SFTP connection requires an appropriate software, like :
- WinSCP (Windows), *http ://winscp.net*,
- CoreFTP (Windows), *http ://www.coreftp.com*,
- FileZilla (Windows, Linux), *http ://filezilla-project.org*,
- gFTP (Linux), *http ://gftp.seul.org*

G.1.2 Connection settings

Different settings must be made in order to establish a proper connection. These are shown in table G.1.

G.1.3 Example with WinSCP

WinSCP is a free and open-source software that allows one to access a file server through different protocols, like SSH.

TABLE G.1 – Tutorial : SFTP Connection - SFTP connection settings

Parameter	**Value**
Host name	*colosse.clumeq.ca*
Connection port	*22*
Protocol	*SSH/SSH2*
User name	*Your Clumeq user name*
Password	*Your Clumeq password*

Figure G.1 shows how to specify the different connection settings.

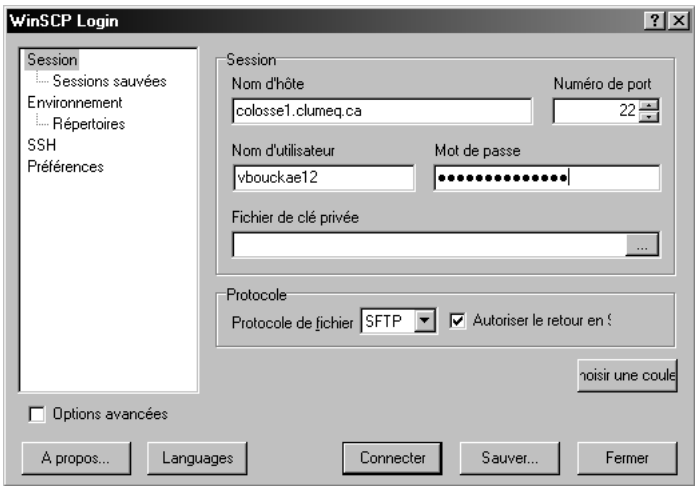

FIGURE G.1 – Tutorial : SFTP Connection - Connection settings with WinSCP

It might be useful to know that, on the first attempt, the RSA key used for the connection needs to be validated. This brings a warning pop-up, as shown in figure G.2. Don't panic and click on "Yes"... :-)

Then, assuming that your username and password are correct, you will have access to both the current directory on your PC and your home directory on Colosse as shown on figure G.3. To transfer a file from one directory to another, simply "drag and drop" it from one folder to another.

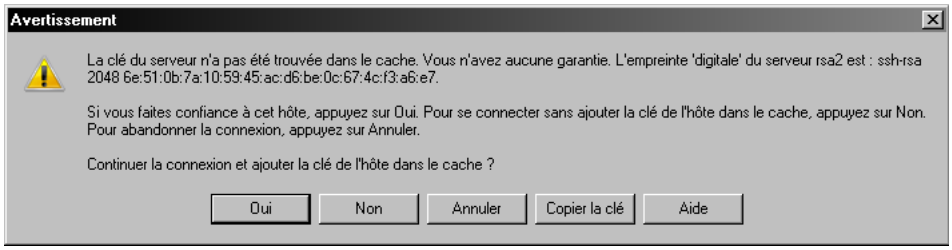

FIGURE G.2 – Tutorial : SFTP Connection - RSA key validation

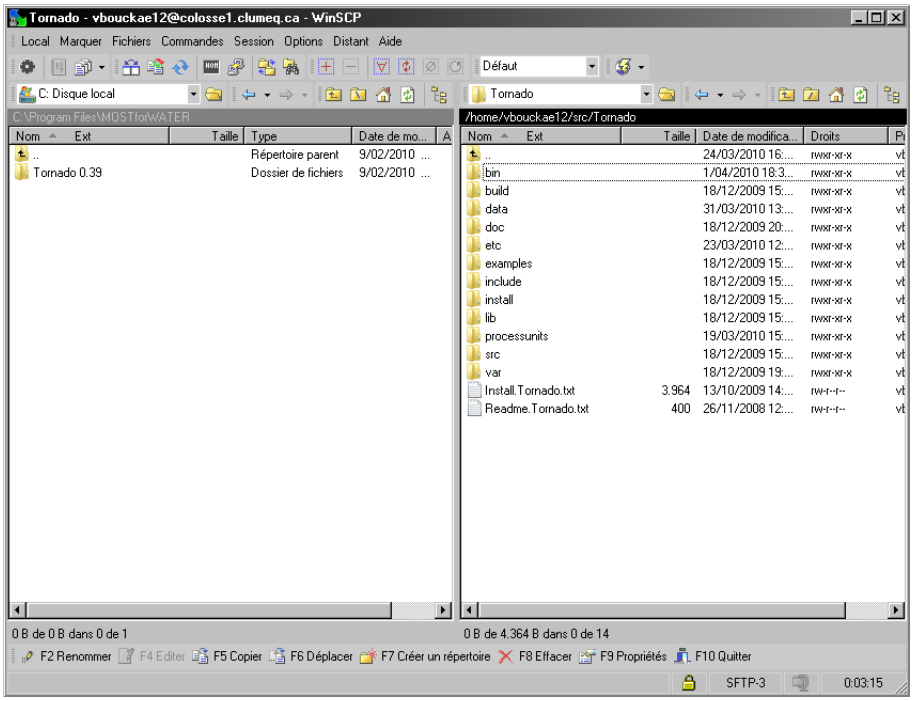

FIGURE G.3 – Tutorial : SFTP Connection - File directories with WinSCP

G.2 SSH connection

In order to access the commands available on Colosse and to be able to launch Tornado simulations, you need a software to connect yourself to Colosse's command line interface (CLI).

G.2.1 SSH software

SSH is available by default on Linux/UNIX systems. On Windows, a small program called PuTTY [1] enables one to establish an SSH connection and have access to a remote CLI.

G.2.2 Connection settings

As we did previously for the SFTP connection, it is also necessary to specify some parameters in order to establish the connection. Those parameters are the same as the ones specified in table G.1 (SFTP uses SSH for the "secure" part of the connection).

G.2.3 Example with PuTTY

Just like we had with WinSCP, we start with a window asking for some connection settings, as shown in figure G.4.
Those connection settings refer to the host name, the port number and the protocol that we'll use.
Once the settings are made, left-click on the "Open" button to access the CLI [2].

In the CLI, you will be asked to enter your user name as well as your password, as shown in figure G.5.

Finally, you have access to your environment on Colosse (see figure G.6). You start in your personal home folder, and have access to all of the available commands.

See table G.3 in appendix B for the most important commands that you might need in order to use Tornado on Colosse.

1. Available for free at *http ://www.chiark.greenend.org.uk/ sgtatham/putty/download.html*

2. Notice : at the first attempt, there will again be a warning pop-up regardind the RSA key. Again, just press "Yes".

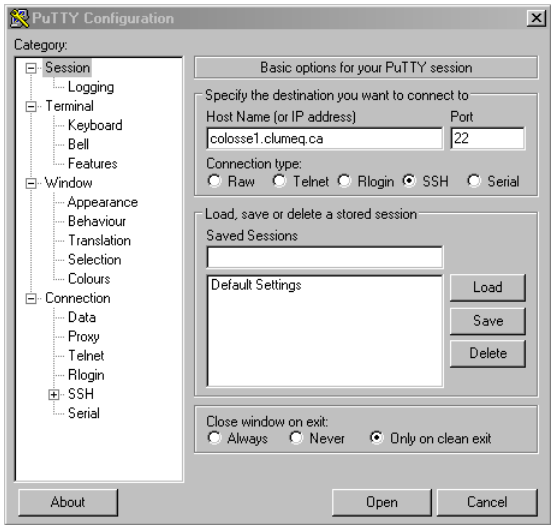

FIGURE G.4 – Tutorial : SSH Connection - Connection settings with PuTTY

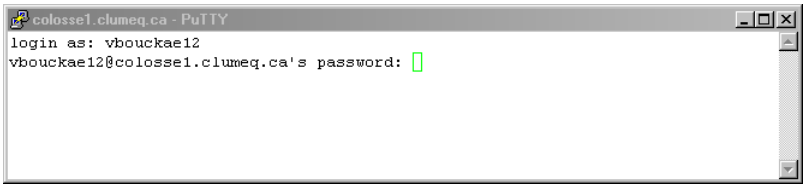

FIGURE G.5 – Tutorial : SSH Connection : Access to CLI with PuTTY

G.3 Running Tornado

In order to run a Tornado virtual experiment (VE) on Colosse, you will need to :

1. Update your *.bash_profile* file.
2. Upload the necessary files.
3. Build the model.
4. Execute Tornado.

Please note that a complete example is available in appendix C.

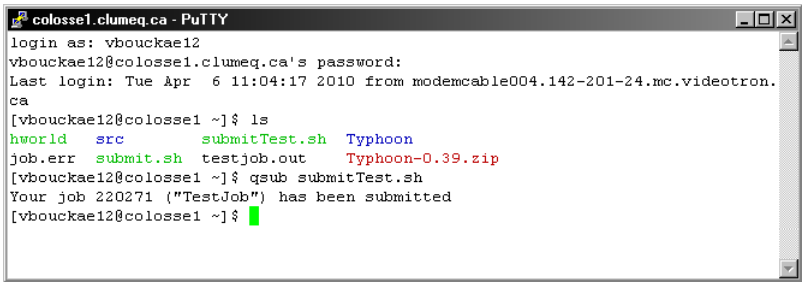

FIGURE G.6 – Tutorial : SSH Connection - Command line interface

G.3.1 Update your .bash_profile

Your *.bash_profile* file specifies all the settings required by your work environment (e.g. it will locate where the Tornado executables are so that you can use them).
To update this file, you can either edit it using a basic text editor directly within the CLI (*nano* or *vi*) or download it with your SFTP software, make the update on your computer and upload it back.

What you need to change

Your *.bash_profile* file needs to contain the following information (to be added after the "*# User specific environment and startup programs*" line) :

```
export TORNADO_HOME="/rap/yyk-770-aa/M4W"

PATH=$PATH:.:$HOME/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.:$HOME/lib

export LD_LIBRARY_PATH="$TORNADO_HOME/src/opentop-1-5-1/lib:
       $LD_LIBRARY_PATH"

export COMMON_PLATFORM="linux"
export COMMON_EXT="so"

export TORNADO_ROOT_PATH="$TORNADO_HOME/src/Tornado"
export TORNADO_DATA_PATH="$TORNADO_ROOT_PATH/data"

export OT_ROOT_PATH="$TORNADO_HOME/src/opentop-1-5-1/ot"
```

```
export OPENSSL_ROOT_PATH="/usr/include/openssl"
export F2C_ROOT_PATH="$TORNADO_HOME/src/F2C"
export CLAPACK_ROOT_PATH="$TORNADO_HOME/src/CLAPACK"
export COMMON_ROOT_PATH="$TORNADO_HOME/src/Common"
export CVODE_ROOT_PATH="$TORNADO_HOME/src/CVODE"
export DASSL_ROOT_PATH="$TORNADO_HOME/src/DASSL"
export LSODA_ROOT_PATH="$TORNADO_HOME/src/LSODA"
export MINPACK_ROOT_PATH="$TORNADO_HOME/src/MINPACK"
export ODEPACK_ROOT_PATH="$TORNADO_HOME/src/ODEPACK"
export RANLIB_ROOT_PATH="$TORNADO_HOME/src/RANLIB"
export ROCK_ROOT_PATH="$TORNADO_HOME/src/ROCK"
export TCPP_ROOT_PATH="$TORNADO_HOME/src/TCPP"

export PATH="$TORNADO_ROOT_PATH/bin/linux:$PATH"
```

Update within the CLI

You can update your *.bash_profile* file within the CLI, using the *nano* program (simple text editor).
To do so, type the following command when you are in your home directory :

```
nano .bash_profile
```

Edit your file, and press "Ctrl-X" to exit (type "Y" to save the changes made).

Update using SFTP

Simply download your *.bash_profile* file with your SFTP software (if you don't see it in your directory, select an option enabling you to see the hidden files), edit it using *Notepad* or similar, and upload it back to your directory on Colosse.

G.3.2 Uploading files

By necessary files, we mean :
- The model translated in C language ("**.c*" file).
- The "**.SymbModel.xml*" file.
- The "**.Exp.xml*" file.
- The input file(s) (if applicable).

Those files must be uploaded in a folder on Colosse, using an SFTP software as discussed in section G.1.

G.3.3 Building the model

To build the model, your first need is a C/C++ compiler. We use the GCC-4.4.2 compiler, that you load in your environment with the following command :

```
module load compilers/gcc/4.4.2
```

Then, you can use the "*tbuild*" Tornado command to compile your model.

G.3.4 Executing Tornado

The execution of Tornado on Colosse's nodes requires a job description script (bash script). This script will tell Colosse how many nodes you would like to use, which queue you want your job to be put on, where Tornado's executables are located and so on.
A script example is available in appendix A.
This script can be written on your computer, using *Notepad* or similar, and then uploaded on Colosse using your SFTP software. It can also be written directly on Colosse using *nano* or another text editor.
Table G.2 lists the most important options and parameters that you need to specify in your submission script.
Other options are available. Check on *https ://support.clumeq.ca* for further

TABLE G.2 – Tutorial : Running Tornado : Important options and parameters for the job description script

Option/Parameter	**Action**	**Compulsory (Yes/No)**
#$ -N *Job_Name*	Specifies the job name	Yes
#$ -P *Group_ID*	Specifies the group (project) ID	Yes
#$ -l *h_rt=hh :mm :ss*	Specifies the estimated run time for your job	Yes
#$ -pe *default|host|hosts* *Nb_Slots*	Specifies that you want to use a parallel environment with *Nb_Slots* slots	No No
#$ -cwd	Specifies that the current directory is the one in which the execution takes place (e.g. the one used for inputs and outputs)	
#$ -t *a-b*	Defines a range (first number must be strictly positive) that will be used if you choose to execute a given number $(b - a)$ of jobs in parallel	No

details.

Once the script is written and uploaded on Colosse, you need to submit it to Colosse's job scheduler with the following command :

```
qsub [Your_Script.sh]
```

Executing an array of tasks

To execute an array of tasks in parallel, you need to specify a range of numbers that will be used to define a range of tasks. This is done with the "*-t*" option :

```
#$ -t a-b
```

Then, you need to use the environment variable called *$SGE_TASK_ID* in your executable call. This variable will match the range you defined.

For example, let's assume you wish to run three simulations at the same time, named respectively *Example.1.Simul.Exp.xml*, *Example.2.Simul.Exp.xml* and *Example.3.Simul.Exp.xml*.
Therefore, you need to define a range going from 1 to 3 :

```
#$ -t 1-3
```

Then, when calling *texec*, you will write :

```
texec Path/To/Your/Experiment/Files/Example.$SGE_TASK_ID.Simul
      .Exp.xml
```

And... That's it !

Executing a single task

Due to some bug issues, executing a single task needs a little more than just specifying the compulsory options and launching *texec*.
You will need to execute your single task as if it were in a range of parallel tasks (so just define a range going from 1 to 1 and do as above).

G.4 Example of a job description script

```
#!/bin/bash

#$ -N JobArrayTest
#$ -P yyk-770-aa

###Specifies that the expected run time is of 1 second
#$ -l h_rt=00:00:01

#$ -S /bin/bash
#$ -cwd

###Specifies that jobs will range from 1 to 3
#$ -t 1-3

source /rap/yyk-770-aa/.Tornado_Settings.sh
module load compilers/gcc/4.4.2

texec $TORNADO_DATA_PATH/HeleneSimul/TwoASU_$SGE_TASK_ID.Simul
      .Exp.xml
```

G.5 Useful UNIX commands

Table G.3 lists some useful UNIX commands that you might need while working on Colosse.

TABLE G.3 – Tutorial : Useful UNIX commands

Command	**Action**
ls	Lists the current directory
cd <*Path*>	Opens the directory specified by *Path*
rm <File_Name>	Erases specified file
cp <Path/File_Name> <Dest_Path/Dest_File>	Copies a file from a specified location to another one.
mv <Path/File_Name> <Dest_Path/Dest_File>	Moves a file from a specified location to another one.
zip <Dest_File> <File_Name>	Compresses a file in an ZIP archive. Interesting thing is that youo can compress many files at once using "*" in the file name like `zip archive.zip *.out.txt`
module load <*Module_Name*>	Loads *Module_Name* in your environment
nano <*File*>	Opens *File* in a basic text editor
qsub <*File.sh*>	Submits a job defined by *File.sh*
qstat <-options>	Prints information regarding the jobs currently submitted to the queue
exit	Disconnects you from the CLI

G.6 Example of a Tornado use on Colosse

This example is made for Windows users and assumes that Tornado, PuTTY and WinSCP are installed and running. It also assumes that some files were produced by Tornado, on the user's machine : Example{n}.c, Example{n}.SymbModel.xml and Example{n}.Simul.Exp.xml, where {n} ranges from 1 to 3.

G.6.1 SFTP upload

The first thing to do is to upload all files in the appropriate directory on Colosse. First you need to connect yourself with WinSCP, as shown in figure G.1.
Then you can access your folders on Colosse, and upload the required files, as shown in figure G.7.

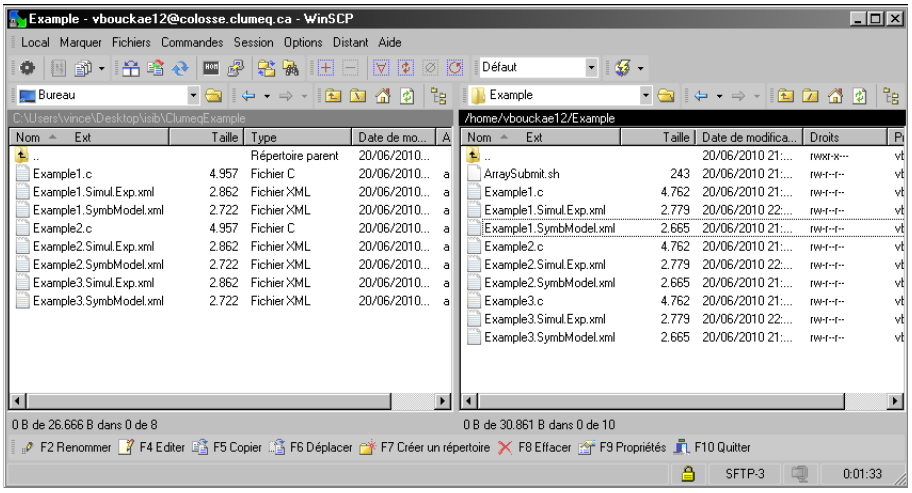

FIGURE G.7 – Tutorial : Uploading files with WinSCP

G.6.2 Building the model

Now that your files are on Colosse, you need to work with them. In order to do that, your first need is to establish an SSH connection with PuTTY, as we've seen in section G.2 (see figures G.4 and G.5).
Once connected, you have to get to the files you just uploaded. Those are located in the */Example/* folder in your home repository. To open this folder, use the command `cd <folder>`, as shown in figure G.8. Then you can see the contents of the folder with the command `ls`.

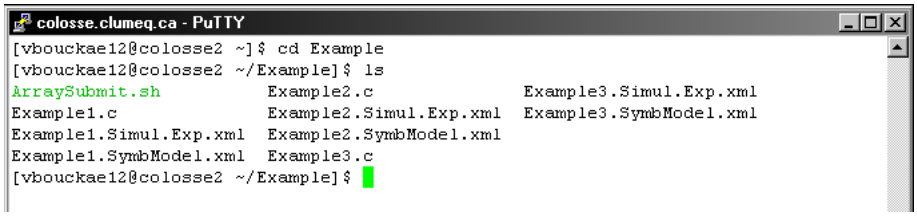

FIGURE G.8 – Tutorial : Accessing the folder with PuTTY

To build the model, you need to :

- Load the GCC 4.4.2 compiler : this is done with the command `module load compilers/gcc/4.4.2`.

- Execute the `tbuild` command from Tornado.

Those steps are illustrated in figure G.9.

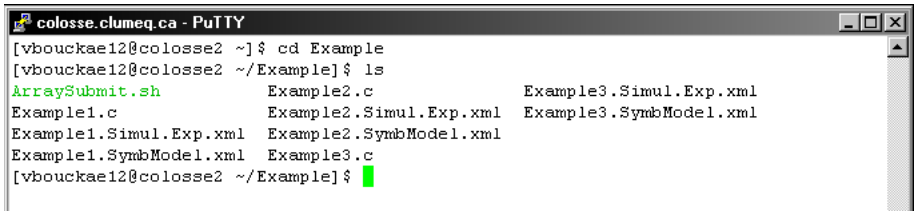

FIGURE G.9 – Tutorial : Building your models

As we have three different models, we have to execute this command three times :

```
tbuild Example1
tbuild Example2
tbuild Example3
```

Now we have everything we need to start our simulations : the executable models, the SymbModel files and the Simul files.

G.6.3 The job description script

The script we will submit to SGE will run (or try to) the three simulations at the same time, on different nodes (depending on what's available). It is given here below (note that the lines starting with "###" are comments) :

```
#!/bin/bash

###NAME OF THE JOB###
#$ -N ExampleJob

###PROJECT (modelEAU) ID###
#$ -P yyk-770-aa

###EXPECTED RUNTIME###
#$ -l h_rt=00:00:15
```

```
###SHELL USED TO PARSE THIS SCRIPT###
#$ -S /bin/bash

###ALL THE OUTPUTS WILL BE CREATED IN THE CURRENT DIRECTORY###
#$ -cwd

###Specifies that there will be three jobs running in parallel###
#$ -t 1-3

###LOADS THE ENVIRONMENT VARIABLES NEEDED###
source /rap/yyk-770-aa/.Tornado_Settings.sh

###LOADS THE COMPILER WE NEED###
module load compilers/gcc/4.4.2

###LAUNCHES THE EXECUTION###
texec $HOME/Example/Example$SGE_TASK_ID.Simul.Exp.xml
```

In our example, this script was created under Windows and uploaded on Colosse with WinSCP, along with the simulations files. As it was created on Windows, there might be some "unexpected characters" problems. That is, characters added due to the shift from one system to another.
In order to remove those possible additional characters, use the command `dos2unix` :

```
[vbouckae12@colosse2 ~/Example]$ dos2unix ArraySubmit.sh
```

G.6.4 Launching the simulations

Now we can launch our simulations. This is done by submitting the script to SGE, as shown in figure G.10.

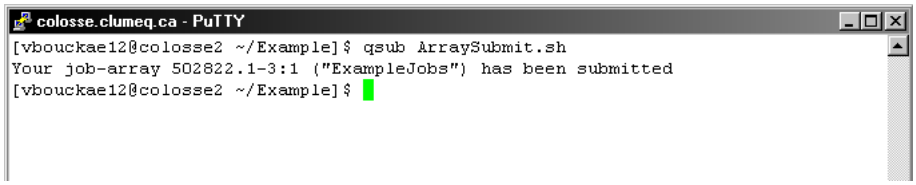

FIGURE G.10 – Tutorial : Launching the simulations

G.6.5 Monitoring the status of your jobs

You can keep a close watch to the status of your jobs with the command qstat :

```
[vbouckae12@colosse2 ~/Example] qstat -u "vbouckae12"
```

This command can be interpretated as "show the queue status of jobs belonging to user vbouckae12". If you want to see the status for all the jobs running at a certain time on Colossus, simply change the <username> part to "*" :

```
[vbouckae12@colosse2 ~/Example] qstat -u "*"
```

The status can be something like "qw" (in queue, waiting), "Eqw" (error in your job, but still in queue and waiting - will be deleted soon) or "r" (running). Other possibilities also exist [3].

G.6.6 Retrieving the data

Then, the only thing left to do is to retrieve the data produced. Go back to your WinSCP window, and simply move your output files to the folder where you'd like to save them (see figure G.11.

3. For a complete list, have a look on :
https ://www.nbcr.net/pub/wiki/index.php ?title=Understanding_SGE_job_status

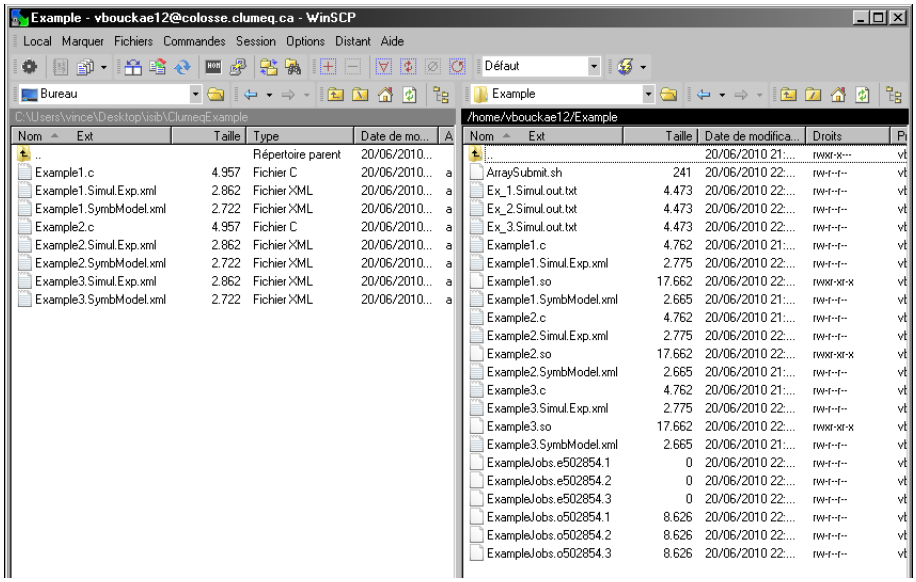

FIGURE G.11 – Tutorial : Retrieving the data

You may notice that there are some files named *Example1.o502854.1* and so on. Those are log files containing the errors encountered during the execution (Example1.**e**502854.1, for instance) and the screen outputs thrown by the execution (Example1.**o**502854.1).

Also, as it might be more convenient to transfer smaller files, you can use the *zip* command to build archives. In our case, if we want to build an archive containing all our "**.out.txt*" files (*Ex_1.Simul.out.txt*, *Ex_2.Simul.out.txt* and *Ex_3.Simul.out.txt*), we'll use it this way :

```
zip results.zip *.out.txt
```

Therefore, the file *results.zip* will contain all of our results.