Christophe Thiery
ESIAL 2A - 2005/2006

# Simulation de graphes de tâches

## dans SimGrid 3

Rapport de stage de deuxième année - ESIAL

LORIA - Laboratoire Lorrain de Recherche en Informatique et ses Applications

Christophe Thiery
9 rue du docteur Roux
54500 Vandoeuvre-lès-Nancy

Ecole Supérieure d'Informatique et Applications de Lorraine
Encadrant de stage : Martin Quinson

# Simulation de graphes de tâches

## dans SimGrid 3

Rapport de stage de deuxième année - ESIAL

LORIA - Laboratoire Lorrain de Recherche en Informatique et ses Applications
BP 239 - 54506 Vandoeuvre-lès-Nancy Cedex - 03 83 59 30 00

# Avant-propos

Le stage de deuxième année à l'Esial est un stage technique, qui avait pour objectif de réaliser un travail d'implantation. Il s'inscrit dans la formation Esial en intervenant dans l'évaluation de la deuxième année. J'ai fait mon stage au Loria (Laboratoire Lorrain de Recherche en Informatique et ses Applications), au sein de l'équipe AlGorille et pendant dix semaines, du 12 juin au 18 août 2006. J'ai choisi de faire mon stage au Loria afin de découvrir le monde de la recherche, mettre mes compétences à profits et en acquérir de nouvelles en travaillant dans un domaine à la pointe de l'innovation technologique.

Ce stage a été financé par l'ARC INRIA OTaPHe. Je tiens à adresser mes remerciements les plus sincères à Martin Quinson, qui m'a proposé ce stage et l'a encadré avec beaucoup d'attention, à Arnaud Legrand et à Frédéric Suter pour leur aide précieuse et leur grande disponibilité tout au long de mon travail.

# Table des matières

# Introduction

Dans le cadre du stage de deuxième année au Loria, j'ai développé une nouvelle interface de programmation pour le simulateur SimGrid. SimGrid est un simulateur de calcul distribué reconnu, qui permet l'étude d'applications distribuées dans des environnements réalistes. Il s'avère ainsi être un outil très utile pour étudier des algorithmes distribués à très large échelle sur des plates-formes modernes comme les grilles de calcul et les systèmes pair-à-pair. Le nouveau module que j'ai développé, nommé SimDag, permet de simuler des graphes de dépendances de tâches parallèles avec SimGrid.

Dans ce rapport, je présenterai d'abord le Loria ainsi que l'équipe AlGorille. Je détaillerai ensuite le travail que j'ai effectué au cours du stage : ses enjeux et ses objectifs, avant de présenter les méthodes de travail utilisées et les résultats obtenus.

# 1 Présentation du Loria

## 1.1 Le Loria

### 1.1.1 Introduction

Le Loria est un laboratoire de recherche en informatique. Situé sur le campus universitaire scientifique de Nancy, il regroupe plus de 450 personnes parmi lesquelles des chercheurs et enseignants-chercheurs, des doctorants et post-doctorants, des ingénieurs, des techniciens et du personnel administratif.

Le Loria est une Unité Mixte de Recherche commune à plusieurs établissements (UMR 7503) :

- CNRS : Centre National de Recherche Scientifique
- INPL : Institut National Polytechnique de Lorraine
- INRIA : Institut National de Recherche en Informatique et en Automatique
- UHP : Université Henri Poincaré, Nancy 1
- Nancy 2 : Université Nancy 2

Trois missions essentielles sont réalisées par le Loria :

- la recherche fondamentale et appliquée au niveau international dans le domaine des Sciences et Technologies de l'Information et de la Communication,
- la formation en partenariat avec les universités lorraines,
- le transfert technologique par le biais de partenariats industriels et par l'aide à la création d'entreprises.

### 1.1.2 Thématiques de recherche

À travers ses 28 équipes de recherche, le Loria abrite des compétences scientifiques en algorithmique, systèmes formels, modélisation symbolique et numérique. L'activité des différentes équipes s'organise autour de six grandes thématiques de recherche aux domaines d'applications très variés :

**Calcul, simulation et visualisation à haute performance :** La physique, la chimie ou la biologie soulèvent aujourd'hui des problèmes de modélisation qui constituent de vrais défis pour l'informatique : la taille des modèles numériques à calculer, transférer et visualiser, le besoin impérieux de précision géométrique

et physique, la prise en compte de la composante temporelle et de l'interactivité dans les techniques de simulation et de visualisation. L'alliance en Lorraine du calcul scientifique, des réseaux et de la visualisation haute performance est un atout pour aborder ces défis.

**Qualité et sûreté des logiciels :** Les problèmes de fiabilité et de sécurité des systèmes informatiques sont abordés en se focalisant sur deux objectifs : la conception de logiciels sûrs et permettant le traitement d'applications de taille réelle, et la vérification des systèmes et services critiques, embarqués ou enfouis.

**Systèmes parallèles, distribués et communicants :** Calculer sur des plates-formes hétérogènes distribuées, maîtriser le fonctionnement des réseaux hétérogènes et complexes, nécessite des recherches et des développements autour de modèles et d'architectures logicielles. Cette thématique de recherche est cruciale pour la surveillance des services à forte valeur ajoutée, d'architectures d'applications, entre autres pour la grille, les entreprises virtuelles ou l'intelligence ambiante. Les compétences du Loria portent sur la supervision, la coopération, et la distribution des programmes et des services, mais aussi sur l'étude de protocoles, la sûreté de fonctionnement, et l'évaluation de performances.

Mon stage s'inscrivait dans cette thématique.

**Modèles et algorithmes pour les sciences du vivant :** En bioinformatique, l'un des objectifs de recherche est d'expliquer comment les composants d'un système biologique interagissent pour assurer une certaine fonction ; un modèle informatique permet d'analyser, de simuler et à terme de prédire le comportement d'un système biologique ainsi que de raisonner sur les propriétés de ce système. Un autre objectif est de mieux comprendre les mécanismes d'expression de génomes à partir de leurs séquences nucléotidiques ; cela nécessite la mise au point de nouvelles méthodes algorithmiques efficaces, tenant compte d'une prolifération de données génomiques et de nouvelles connaissances biologiques. L'informatique bio-inspirée s'appuie sur la conception de systèmes multi-agents, de systèmes neuromimétiques continus ou à évènements discrets, de systèmes connexionistes de très grande taille. L'objectif est d'étudier la capacité d'apprentissage de ces

systèmes bio-inspirés, grâce à des structures et algorithmes performants alliant méthodes mathématiques et connaissances biologiques.

**Traitement de la langue naturelle et communication multimodale :** Le Loria rassemble des compétences en traitement automatique de la parole, en informatique linguistique, en interaction homme-machine, en vision, en analyse de documents, en simulation de processus perceptifs et communicatifs. Cette convergence permet de s'attaquer à l'interprétation et à la présentation de données multimodales, avec pour objectif de modéliser la perception et la cognition humaines et d'en tirer parti pour mieux communiquer et interagir avec l'utilisateur.

**Représentation et gestion des connaissances :** Si les capacités de stockage et d'organisation des systèmes d'information permettent d'envisager des masses de données gigantesques, encore faut-il se doter des moyens de les exploiter et de les partager, au travers d'outils d'indexation, de navigation, de recherche d'informations, de coopération. Cette notion soulève des problématiques nombreuses, liées au caractère très hétérogène des documents manipulés et aux critères d'organisation des données qui peuvent être très variables suivant les organisations concernées. La représentation et la gestion des connaissances s'attaquent à ces problèmes complexes, dans le cadre du Web sémantique et des entreprises virtuelles.

### 1.1.3 Le nouveau bâtiment

Le bâtiment Loria héberge la plus grande partie des activités du Loria et de l'INRIA Lorraine. Jusqu'il y a quelques mois, il était composé de deux parties : la tranche A, achevée en mars 1990, et la tranche B, qui a vu le jour en 1992. Une troisième tranche, la tranche C, vient d'être construite. L'inauguration au eu lieu le 16 juin 2006. Grâce à cette extension, le Loria dispose désormais de près de 13000 mètres carrés. Ce nouveau bâtiment permet de gérer la croissance de la recherche qui se traduit notamment par l'accueil au quotidien de plus de 500 personnes, la mise en place d'équipements spécifiques pour la réalité virtuelle et la robotique, ou l'organisation de manifestations scientifiques de plusieurs centaines de personnes.

Le nouveau bâtiment facilite en effet l'organisation d'événements scientifiques tels que des conférences internationales. Il comporte un nouvel amphithéâtre de 240 places, des salles de réunion et de formation ainsi qu'un Espace Transfert facilitant la valorisation de la recherche et les échanges industriels. Par ailleurs, une salle de restauration pour le personnel et les visiteurs est disponible dans la tranche C.

Les collectivités locales (Région Lorraine, Communauté Urbaine du Grand Nancy, Conseil Général de Meurthe et Moselle) se sont fortement engagées aux côtés de l'Etat pour permettre la construction de ce nouveau bâtiment. L'université Henri Poincaré a apporté son aide financière et les autres établissements partenaires du LORIA, Nancy 2, le CNRS et l'INPL ont eux aussi soutenu le projet.

À travers cette extension, le Loria va pouvoir accroître son rayonnement régional, national comme international. Il constituera un facteur d'attractivité pour la recherche en informatique.

## 1.2 L'équipe Algorille

### 1.2.1 Sujet d'étude

Le sujet d'étude central de l'équipe AlGorille est le *"Grid computing"*, c'est-à-dire l'accès transparent et efficace à des ressources distribuées. Ces ressources sont des machines et des composants réseau qui forment ce que l'on appelle la *grille*. Cette voie de recherche est fortement liée à l'apparition des réseaux et d'Internet, et représente un des grands défis actuels de l'informatique.

L'équipe AlGorille ("Algorithmes pour la grille") travaille sur la mise en oeuvre de techniques et d'algorithmes pour faire communiquer les machines, inter-opérer les applications, allouer les ressources de la grille, améliorer la qualité des services et la sûreté des transactions. Ces travaux menés au sein de l'équipe visent à répondre à plusieurs questions, parmi lesquelles :

- Comment faire coopérer plusieurs applications distantes ?
- Comment décider de l'allocation des ressources à des tâches effectuées en parallèle ?
- Comment garantir l'absence de pertes dans les transferts d'information ?
- Comment sécuriser l'échange de données ?

Toutes ces questions constituent l'un des enjeux majeurs des technologies de l'information.

Les axes de recherche de l'équipe, détaillés ci-dessous, correspondent au premier des défis formulés par l'INRIA dans son plan stratégique : maîtriser l'infrastructure numérique en sachant programmer, calculer et communiquer sur Internet et sur des réseaux hétérogènes.

- La modélisation et la structuration d'applications complexes en terme de quantité de calculs et de données.
- Le traitement des ressources de manière transparente : ordonnancement de tâches séquentielles ou parallèles, migration de calculs, échange de données, distribution et redistribution de données.
- La validation expérimentale : reproductibilité, extensibilité et applicabilité des modèles simulés.

La méthodologie de l'équipe consiste en trois phases qui interagissent entre elles :

- La modélisation, qui permet de représenter de manière abstraite une réalité physique ou technique,
- La formulation de solutions aux problèmes modélisés lors de la première phase, basées sur cette représentation,
- La mise en oeuvre d'algorithmes pour implanter les techniques de résolution imaginées dans la deuxième phase. Cette phase valide également les modèles conçus dans la première phase.

### 1.2.2 Membres de l'équipe

L'équipe AlGorille est composée de chercheurs, d'enseignants-chercheurs, d'étudiants, d'ingénieurs et de personnel administratif.

- Jens Gustedt : Directeur de recherches INRIA ; responsable de l'équipe
- Emmanuel Jeannot : Chargé de recherche
- Frédéric Suter : Maître de conférences UHP
- Martin Quinson : Maître de conférences UHP
- Xavier Delaruelle : Ingénieur associé ; travaille sur le projet Grid 5000[1]

[1]Grid 5000 est un projet de recherche national qui vise à développer une infrastructure de grille expérimentale à large échelle : http ://www.grid5000.fr

– Pierre-Nicolas Clauss : Doctorant
– Tchimou N'Takpé : Doctorant
– Josiane Reffort : Assistante

# 2 Travail effectué

Maintenant que le Loria et l'équipe AlGorille ont été présentés, je vais détailler le travail que j'ai effectué au cours de mon stage. D'abord, je présenterai le projet SimGrid, puis j'exposerai les objectifs du stage. Ensuite, je parlerai des méthodes de travail et des outils utilisées. Enfin, je détaillerai la démarche de mon travail et je ferai un bilan du résultat obtenu.

## 2.1 Le projet SimGrid

SimGrid est un simulateur de calcul distribué. Il est conçu pour permettre l'étude du comportement d'applications distribuées dans des environnements réalistes. Ces environnements sont les plates-formes de calcul modernes telles que les *grilles* et les systèmes pair-à-pair (P2P). Elles se caractérisent par leur échelle très importante : de plusieurs centaines à plusieurs milliers de machines. Avec le développement d'Internet, elles ont un potentiel très important, mais leur architecture reste extrêmement difficile à étudier et à réaliser, du fait de leur grande échelle et de l'hétérogénéité des ressources.

En effet, de nombreux problèmes techniques surviennent lorsque l'on veut utiliser ces plates-formes. La simple tâche de compiler un programme peut s'avérer problématique car les compilateurs et les bibliothèques varient d'un système à l'autre. De plus, n'importe quel noeud peut devenir hors d'usage à un moment donné. L'étude des problèmes théoriques posés par ces plates-formes est complexe car la mise en place d'une plate-forme d'expérimentation demande des efforts très importants. Il est donc classique d'utiliser un simulateur de plate-forme comme SimGrid pour étudier des algorithmes sur une plate-forme réaliste.

### 2.1.1 Equipe de développement

SimGrid est réalisé depuis 2000 en collaboration par Henri Casanova (Université de Hawaï, Manoa), Arnaud Legrand (Chargé de recherche CNRS, équipe Mescal de l'INRIA Rhône-Alpes) et Martin Quinson (Maître de Conférences UHP, équipe AlGorille du Loria). Henri Casanova a travaillé sur les premières versions de SimGrid. Arnaud Legrand s'occupe de la partie simulation : MSG et SURF. Martin Quinson s'occupe de XBT et de GRAS, en particulier en ce qui concerne les aspects d'exécution *in situ*.

### 2.1.2 Description du logiciel

SimGrid est un simulateur modulaire écrit en langage C, qui se découpe en plusieurs composants (voir le schéma). Les utilisateurs de SimGrid sont des programmeurs qui font appel aux fonctions fournies par l'une des trois interfaces de programmation (API) de SimGrid : MSG, GRAS ou SimDag. Chacune de ces interfaces propose des fonctions destinées à une utilisation spécifique. Selon le programme qu'il souhaite développer et la façon de représenter son programme, l'utilisateur choisira l'interface de programmation qui correspond à ses besoins.

**MSG :** API permettant de simuler des applications distribuées qui ne sont pas destinées à être mises en oeuvre dans la réalité.

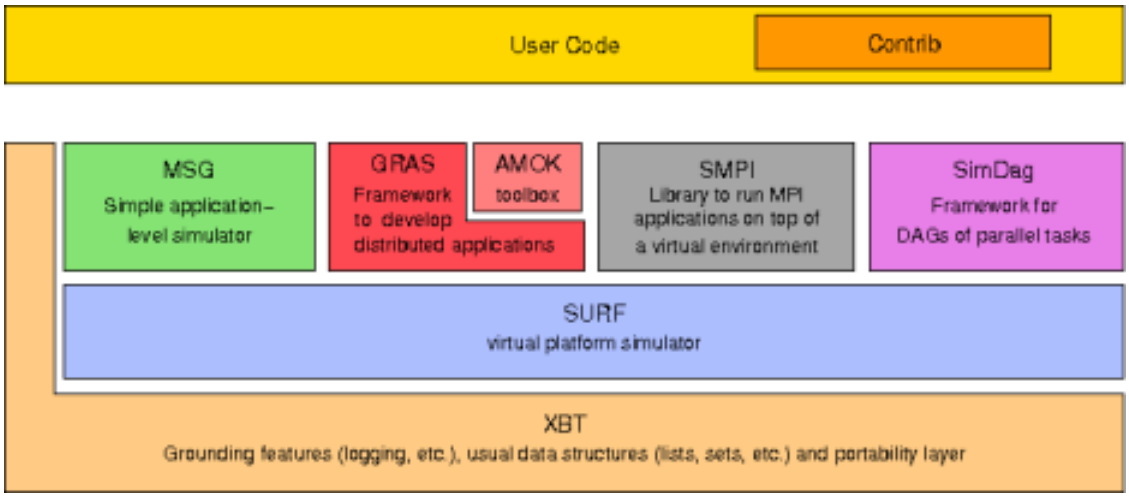
**GRAS :** API permettant à la fois de simuler ou de développer réellement des applications distribuées.

**AMOK :** modules qui facilitent l'utilisation de GRAS.

**SimDag :** nouvelle API de SimGrid 3 développée au cours de mon stage, permettant d'exprimer les applications sous forme de graphes de dépendances de tâches parallèles.

**SURF :** noyau de SimGrid, qui simule la plate-forme ; les différentes API (MSG, GRAS et SimDag) reposent sur SURF.

**XBT :** boîte à outils qui fournit des fonctions de bas niveau aux autres modules.




Les trois interfaces de programmation reposent sur SURF, qui est le coeur du système. SURF est apparu dans la version 3 de SimGrid, en 2005, et a permis d'améliorer sensiblement les performances du simulateur.

XBT assure la portabilité du code et fournit des fonctionnalités de base à tous les autres composants :

- Des structures de données tels que les tableaux dynamiques, les dictionnaires, les files d'attentes (FIFO) ;
- Des fonctionnalités qui facilitent la programmation et le débogage comme un mécanisme d'exceptions ou encore un système de *logs*.

### 2.1.3 Communauté

SimGrid est considéré par la communauté scientifique comme la référence dans le domaine des simulateurs de grille. En effet, la plupart des articles présentant des projets similaires se comparent à SimGrid. Il dispose d'une communauté allant au-delà du cercle des personnes participant à sa réalisation ; l'outil est en effet utilisé dans de nombreux projets et articles. SimGrid est distribué sous licence LGPL et il est utilisé dans le monde entier.

## 2.2 Objectifs du stage

### 2.2.1 Motivations

La première version de SimGrid, datant de 2000, permettait d'étudier des heuristiques d'ordonnancement de graphes de tâches parallèles. Un graphe de tâches parallèles est un graphe orienté acyclique (*DAG*) qui définit les dépendances entre les tâches à exécuter. Une tâche A dépend d'une tâche B si la tâche A doit attendre que B soit terminée avant de commencer. La première version était ainsi basée sur les DAGs de tâches parallèles.

Chaque tâche est définie par la quantité de calculs qu'elle représente (exprimée en Mégaflops) pour les tâches de calcul, et par la quantité de données à échanger (en Mégaoctets) pour les tâches de communication.

La version 2, publiée en 2002, ajoutait le module MSG. MSG ajoutait des fonctions pour simuler une application distribuée plus simplement. Le module MSG était bâti en utilisant les fonctions de la version 1, mais ce n'était pas l'idéal car il n'exploitait pas les DAGs.

La version 3, sortie en 2005, a été complètement réécrite. SURF, le nouveau noyau de simulation, est venu remplacer le système de DAG sur lequel étaient fondées les premières versions. Cela a permis d'accélérer la simulation d'un ordre de grandeur. La version 3 a également vu l'apparition du module GRAS.

Cependant, les DAGs ont donc disparu de la version 3. Or, de nombreux utilisateurs de SimGrid utilisaient la simulation de graphes de tâches. La version 3 n'offrant plus la fonctionnalité de gérer les DAGs, ces utilisateurs ne pouvaient pas profiter des améliorations de cette version, et notamment des avantages de SURF. Il devenait donc nécessaire d'implanter un nouveau module qui gère les DAGs en utilisant le noyau SURF.

### 2.2.2 Cahier des charges

Le sujet de mon stage était donc de développer ce nouveau module nommé SimDag. Voici la liste des fonctions de SimDag, que je devais donc implanter :

```
// Fonctions qui donnent des informations sur les liens réseau
void*       SD_link_get_data(SD_link_t link);
void        SD_link_set_data(SD_link_t link, void *data);
```

```
const char* SD_link_get_name(SD_link_t link);
double      SD_link_get_capacity(SD_link_t link);
double      SD_link_get_current_bandwidth(SD_link_t link);
double      SD_link_get_current_latency(SD_link_t link);

// Fonctions qui donnent des informations sur les stations
// de travail
SD_workstation_t  SD_workstation_get_by_name(const char *name);
SD_workstation_t* SD_workstation_get_list(void);
int         SD_workstation_get_number(void);
void        SD_workstation_set_data(SD_workstation_t workstation,
                void *data);
void *      SD_workstation_get_data(SD_workstation_t workstation);
const char* SD_workstation_get_name(SD_workstation_t workstation);
double      SD_workstation_get_power(SD_workstation_t workstation);
double      SD_workstation_get_available_power(
                SD_workstation_t workstation);

// Fonctions qui donnent des informations sur les routes
SD_link_t*  SD_workstation_route_get_list(SD_workstation_t src,
                SD_workstation_t dst);
int         SD_workstation_route_get_size(SD_workstation_t src,
                SD_workstation_t dst);

// Fonctions qui manipulent les tâches
SD_task_t   SD_task_create(const char *name, void *data,
                double amount);
int         SD_task_schedule(SD_task_t task, int workstation_nb,
                SD_workstation_t **workstation_list,
                double *computation_amount,
                double *communication_amount, double rate);

void*       SD_task_get_data(SD_task_t task);
void        SD_task_set_data(SD_task_t task, void *data);
```

```
const char* SD_task_get_name(SD_task_t task);
double      SD_task_get_amount(SD_task_t task);
double      SD_task_get_remaining_amount(SD_task_t task);
void        SD_task_dependency_add(const char *name, void *data,
                SD_task_t src, SD_task_t dst);
void        SD_task_dependency_remove(SD_task_t src,
                SD_task_t dst);
e_SD_task_state_t SD_task_state_get(SD_task_t task);
void        SD_task_watch(SD_task_t task, e_SD_task_state_t state);
void        SD_task_unwatch(SD_task_t task,
                e_SD_task_state_t state);

void        SD_task_unschedule(SD_task_t task);

SG_task_t   *SD_simulate(double how_long);
```

Pour avoir des informations détaillées sur le rôle de chaque fonction, consultez la documentation de SimDag : `http://simgrid.gforge.inria.fr/doc/group__SD__API.html`. Un exemple de programme utilisant SimDag est donné en annexe.

Les fonctions de SimDag se classent en trois catégories principales :

- les fonctions relatives aux stations de travail (*workstations*),
- les fonctions relatives aux liens réseau (*links*),
- les fonctions relatives aux tâches (*tasks*).

Une station de travail représente une ressource capable d'exécuter une tâche de calcul. Elle est caractérisée par un nom et une puissance de calcul.

Les liens réseau permettent de relier les stations de travail entre elles. Chaque lien peut être vu comme une ressource capable d'exécuter une tâche de communication. Un lien est caractérisé par son nom, sa bande passante (quantité d'informations qui peuvent circuler sur le lien en une seconde) et sa latence (délai entre le moment où une information est envoyée et le moment où elle est reçue par le destinataire). Deux stations de travail peut être reliées par une série de liens appelée route.

Les stations de travail et les liens constituent l'environnement sur lequel on travaille ; cet environnement est ce que l'on appelle la plate-forme ou encore la

grille.

Outre le travail d'implantation qui consistait à programmer ces fonctions, le sujet du stage demandait également que j'établisse la documentation de SimDag, afin de permettre aux futurs utilisateurs de savoir comment utiliser ces fonctions. Un autre objectif était également de créer des tests d'assurance qualité, destinés à vérifier que SimDag donne bien les résultats attendus, mais aussi de comparer ses performances aux versions précédentes de SimGrid.

## 2.3 Méthodes de travail

Dans cette section, je vais expliquer brièvement l'organisation et les méthodes de travail que nous avons adoptées pour mon stage.

### 2.3.1 Travail en équipe

SimGrid étant développé conjointement au Loria et à l'INRIA Rhône-Alpes, l'équipe de développement est répartie sur les deux sites. La communication avec l'équipe a été un élément primordial pour la réussite de mon stage. J'ai pu poser beaucoup de questions à Arnaud Legrand et Martin Quinson par courrier électronique et messagerie instantanée.

Ils ont également pu suivre mon travail de près. En effet, le projet SimGrid utilise le système de développement collaboratif CVS afin de gérer facilement le travail en équipe. Grâce à CVS, à chaque fois que quelqu'un fait une modification sur SimGrid, tous les membres de l'équipe de développement reçoivent un courrier électronique qui détaille les changements. Ainsi, toute l'équipe pouvait suivre mes travaux à chaque étape de développement et me signaler immédiatement ce qui n'allait pas, ou me proposer d'autres manières de procéder.

Nous avons également fait quelques réunions afin de faire le point et de discuter de la suite du stage. Enfin, je faisais chaque soir un petit compte-rendu qui expliquait ce que j'avais fait dans la journée, les problèmes que j'avais rencontré et leurs solutions. Ainsi, Arnaud et Martin savaient exactement où j'en étais chaque jour. Ces multiples modes de communications nous ont permis d'être efficaces.

### 2.3.2 Technologies utilisés

Durant ce stage, j'ai travaillé sous Linux avec mon ordinateur portable personnel. Bien entendu, le principal langage que j'ai utilisé était le C puisque SimGrid est écrit en C. Je connaissais déjà le langage C qui est enseigné à l'ESIAL en première année. Mais le travail que j'ai effectué pendant le stage m'a permis de perfectionner mes connaissances en C et d'utiliser des outils d'aide à la compilation et au débogage. J'ai ainsi découvert les outils autoconf et automake, qui permettent de générer automatiquement des scripts de configuration et des Makefile en fonction l'environnement utilisé. J'ai également appris à utiliser Valgrind, un outil de vérification de la mémoire allouée par un programme C ou C++. Grâce à Valgrind, j'ai pu détecter et corriger des fuites mémoire dans SimDag, ce qui a permis d'améliorer sensiblement les performances.

Par ailleurs, à l'occasion de mon stage, j'ai appris à utiliser Doxygen. Il s'agit d'un outil permettant de générer la documentation d'un projet sous forme de pages HTML, à partir des informations situées dans les commentaires du code source et dans des fichiers de documentation.

J'ai également appris le langage Perl, qui m'a servi à mettre à jour un script de conversion de fichiers de plates-formes. Enfin, j'ai appris à utiliser LaTeX pour réaliser ce rapport de stage.

## 2.4 Tâches effectuées

### 2.4.1 Découverte de SimGrid

Les premiers jours de mon stage ont été consacrés à la découverte de SimGrid. Cette première étape consistait à installer et à prendre en main le logiciel, comprendre l'organisation des différents modules et leurs interactions. En particulier, j'ai étudié le fonctionnement de MSG afin de m'en inspirer pour développer SimDag.

### 2.4.2 Implantation de SimDag

Le développement de SimDag en lui-même a pris environ trois semaines. J'ai commencé par développer les fonctions les plus évidentes, puis je me suis occupé petit à petit des fonctions plus compliquées.

Les fonctions liées aux stations de travail et aux liens ont été les plus rapides à développer, car SURF prend déjà en charge les données de la plate-forme.

En revanche, les fonctions qui concernent les tâches ont demandé plus de travail car c'était à SimDag de gérer les dépendances entre les tâches. SURF gère les tâches parallèles mais ne prend pas en compte de dépendances entre elles Une tâche parallèle est une quantité de calcul et de communication à exécuter. L'utilisateur de SimDag place les tâches sur les stations de travail de son choix, en fonction de l'algorithme qu'il met en oeuvre.

SimDag apporte la notion de dépendance entre les tâches. Définir une dépendance entre deux tâches A et B signifie que la tâche A doit être terminée avant que la tâche B puisse commencer. La liste des dépendances des tâches constitue ce que l'on appelle le *graphe de dépendances de tâches* (DAG).

Par ailleurs, les tâches de SimDag sont des tâches parallèles, c'est-à-dire qu'une tâche peut se décomposer en petites quantités de calcul qui s'exécutent en même temps sur des stations de travail différentes, et en quantités de communication qui circulent sur les liens réseau entre ces stations.

### 2.4.3 Documentation

Une fois le module SimDag terminé, il était important de rédiger sa documentation afin de renseigner les futurs utilisateurs sur son fonctionnement. La documentation de SimDag, tout comme celle des autres modules, est disponible (en Anglais) sur le site de SimGrid : `http://simgrid.gforge.inria.fr/doc/group__SD__API.html` Elle s'organise en plusieurs pages web : types de données, stations de travail, liens, tâches, dépendances des tâches et simulation. Chacune de ces pages détaille une liste de fonctions que l'utilisateur peut appeler dans son programme. Chaque fonction est expliquée en détail : ce qu'elle fait, ses paramètres, sa valeur de retour, etc.

Ces pages HTML sont générées avec Doxygen à partir du code source de SimGrid. Le code source contient des commentaires avant chaque fonction, et ces commentaires sont analysés par Doxygen pour produire les pages HTML automatiquement.

L'étape de documentation a duré une semaine.

### 2.4.4 Tests

L'étape la plus longue a été la phase de tests (trois semaines).

J'ai d'abord écrit mon propre petit programme de test qui vérifie sur un exemple simple de DAG que les fonctions de SimDag donnent bien le résultat attendu.

Ensuite, j'ai travaillé sur des anciens programmes qui fonctionnaient avec SimGrid 2 et utilisaient des DAGs. L'idée était de réécrire ces programmes pour qu'ils fassent appel aux fonctions de SimDag et non plus à celles de SimGrid 2, et de comparer les performances. Ces programmes permettaient d'exécuter divers algorithmes d'ordonnancement de tâches dépendantes.

Les tests se sont révélés très utiles car ils ont permis :

- de trouver et corriger des bugs ;
- d'optimiser SimDag ;
- de rajouter des nouvelles fonctions auxquelles nous n'avions pas pensées au départ.

La section 2.5.1 présente une comparaison des résultats obtenus.

### 2.4.5 Autres tâches

Les dernières semaines de mon stage ont été consacrées à diverses tâches supplémentaires :

- Les difficultés liées à l'utilisation de SURF pour implanter SimDag ont fait apparaître la nécessité de documenter SURF. J'ai donc réalisé une documentation de SURF avec Doxygen, destinée aux développeurs qui voudraient réaliser une nouvelle API au-dessus de SURF.
- J'ai également travaillé sur certaines structures de données de XBT : les dictionnaires et les ensembles. Les dictionnaires permettent d'associer un objet à un nom. Les ensembles (*sets*) peuvent associer quant à eux un objet à la fois à un nom et à un numéro.
- J'ai enfin fait beaucoup de *profiling*, c'est-à-dire que j'ai analysé l'exécution de programmes SimGrid pour trouver quels sont les morceaux de code qui prennent le plus de temps d'exécution. J'ai donc optimisé le mieux possible ces fonctions afin d'améliorer les performances. Pour diminuer le nombre d'allocations d'objets en mémoire, très coûteuses en temps d'exécution, j'ai

implanté un nouveau mécanisme appelé *mallocator*. Ce mécanisme permet de conserver les objets devenus inutiles (au lieu de les supprimer) afin de pouvoir les recycler par la suite plutôt que d'en créer de nouveaux. Cette technique a permis d'économiser énormément d'allocations mémoire.

### 2.4.6 Problèmes rencontrés

Comme expliqué plus haut, SimDag fait appel aux fonctions de SURF, le noyau du simulateur. Une des principales difficultés, lors de l'étape d'implantation, a été de comprendre comment interagir avec SURF. Le fonctionnement de SURF est assez complexe, et il n'existe pas de documentation expliquant comment on l'utilise. En effet, rédiger la documentation de SURF n'est pas une priorité puisque les utilisateurs de SimGrid ne font pas directement appel aux fonctions de SURF : ils utilisent une des API de plus haut niveau (MSG, GRAS ou SimDag). Heureusement, Arnaud Legrand, l'auteur de SURF, m'a expliqué comment procéder et a répondu à mes nombreuses questions.

La deuxième difficulté a été de réécrire les anciens programmes lors de la phase de test en utilisant SimDag. Les fonctions de SimDag sont proches de l'ancienne version de SimGrid qui gérait les DAGs, mais il y a tout de même quelques différences. Certaines fonctions de l'ancien SimGrid ne sont plus disponibles avec SimDag. Ces fonctions permettaient de prédire des valeurs à un instant donné, comme la charge d'une station de travail, la bande passante d'un lien ou encore la durée d'exécution d'une tâche. Elles ne sont plus disponibles avec SimDag SURF ne permet plus de calculer ces prédictions.

Or, les algorithmes des programmes de test sur lesquels j'ai travaillé étaient fondés sur ces fonctions de prédiction. Il a donc fallu les remplacer par des fonctions approximatives, qui renvoient des valeurs estimées et ne tiennent pas compte des contraintes liées aux tâches en cours d'exécution. La conséquence de cela est que les algorithmes d'ordonnancement des tâches écrits dans ces programmes fournissent des ordonnancements moins bons qu'avant, puisque les fonctions de prédiction sont moins précises.

## 2.5 Résultats obtenus

### 2.5.1 Performances de SimDag

Les trois objectifs initiaux du stage ont été atteints : développer SimDag, écrire la documentation et mettre en oeuvre des tests d'assurance qualité et de performances. De plus, j'ai également contribué à d'autres parties de SimGrid à la fin de mon stage pour améliorer la qualité technique.

Les performances de SimDag sont tout à fait satisfaisantes si on les compare à l'ancienne version de SimGrid. J'ai fait le test avec un DAG de 370 tâches sur les deux versions grâce à l'un des programmes de test sur lesquels j'ai travaillé. Avec l'ancienne version, le temps de l'exécution de la simulation est de 15 secondes, contre 2.5 secondes avec SimDag. Selon les algorithmes de ces programmes de tests, la différence est plus ou moins importante, mais SimDag est toujours nettement plus rapide. Plus les DAGs sont gros, et plus la différence se fait sentir.

Ces bonnes performances sont dûes à l'utilisation de SURF, qui n'existait pas dans les versions précédentes de SimGrid et qui est très rapide. SimDag est lui aussi optimisé au mieux car il utilise des structures de données qui permettent d'accélérer les calculs en faisant le moins possible de boucles et d'allocations mémoire.

### 2.5.2 Perspectives

SimDag est terminé, mais nous avons encore de nombreuses idées de travaux qui permettraient d'améliorer SimGrid.

- A l'origine, MSG fonctionnait avec SimGrid 2. Lors du passage à SimGrid 3, ce module a été réécrit rapidement pour utiliser SURF, mais il y a de nombreuses améliorations et optimisations à lui apporter.
- Pour l'instant, GRAS utilise les fonctions de MSG. Il devrait plutôt utiliser directement celles de SURF : cela permettrait d'améliorer sensiblement ses performances.

# Conclusion

J'ai pu apporter une contribution significative au projet SimGrid. Une interface de programmation telle que SimDag était très demandée par les utilisateurs du logiciel. L'objectif initial de mon stage, qui consistait à implanter SimDag, le documenter et le tester, a été atteint et l'équipe semble satisfaite de ce travail. J'ai ensuite contribué à d'autres parties de SimGrid pour les améliorer.

Participer au développement de SimGrid a été une expérience très intéressante. Au cours de ce stage, j'ai eu l'occasion de mettre en oeuvre mes connaissances en langage C et de les approfondir en découvrant des techniques d'optimisation et des outils d'analyse de la mémoire utilisée. J'ai également acquis de nouvelles compétences informatiques en apprenant de nouveaux langages et outils : Doxygen, Perl, et LaTeX.

Enfin, ce stage a aussi été l'occasion de découvrir le fonctionnement d'un laboratoire de recherche. J'ai assisté à deux soutenances de stage de Master Recherche ainsi qu'à des conférences diverses au Loria. J'ai été confronté aux contraintes de temps et j'ai apprécié le travail en équipe. Je remercie Martin Quinson, Arnaud Legrand et Frédéric Suter pour leur grande disponibilité durant ces dix semaines. Grâce à eux, mon stage a été un succès.

Ce stage a donc représenté pour moi une expérience très enrichissante et il a confirmé mon intention de m'orienter vers le monde de la recherche.

# Glosssaire

**API :** Application Programming Interface. Ensemble de fonctions proposées aux programmeurs qui utilisent un module.

**ARC OTaPHE :** Action de Recherche Coopérative : Ordonnancement de Tâches Parallèles en milieu Hétérogène

**DAG :** Directd Acyclic Graph (Graphe Acyclique Orienté)

**ESIAL :** Ecole Supérieure d'Informatique et Applications de Lorraine

**GRAS :** API de SimGrid 3 permettant de développer réellement des applications distribuées

**Loria :** Laboratoire Lorrain de Recherche en Informatique et ses Applications

**INRIA :** Institut National de Recherche en Informatique et en Automatique

**LGPL :** Licence Publique Générale Limitée

**MSG :** API de SimGrid 3 permettant de simuler des applications distribuées

**SD (SimDag) :** nouvelle API de SimGrid 3 développée au cours de mon stage

**SG (SimGrid) :** simulateur d'applications distribuées sur des plates-formes réalistes

**SURF :** noyau de SimGrid 3, sur lequel reposent MSG, GRAS et SimDag

**UHP :** Université Henri Poincaré, Nancy 1

**XBT :** boîte à outils de SimGrid 3, qui fournit des fonctions de bas niveau aux autres modules

# Références

[1] LORIA : http://www.loria.fr

[2] AlGorille : http://www.loria.fr/equipes/algorille

[3] SimGrid : http://simgrid.gforge.inria.fr

# Annexes

# A Exemple de programme utilisant SimDag

Le code qui suit est un exemple de programme très simple qui fait appel aux fonctions de SimDag. Il crée un environnement à partir d'un fichier XML passé en paramètre de la ligne de commande, puis il crée quatre tâches avec des dépendances entre elles. Il attribue ensuite des temps de calcul et de communication à chacune des tâches, avant de lancer la simulation. A la fin de la simulation, il affiche les temps d'exécution des quatre tâches.

```
int main(int argc, char **argv) {
  int i;

  /* initialisation de SimDag */
  SD_init(&argc, argv);

  /* creation de l'environnement */
  const char * platform_file = argv[1];
  SD_create_environment(platform_file);

  /* creation des tâches et de leurs dépendances */
  SD_task_t taskA = SD_task_create("Tâche A", NULL, 10.0);
  SD_task_t taskB = SD_task_create("Tâche B", NULL, 40.0);
  SD_task_t taskC = SD_task_create("Tâche C", NULL, 30.0);
  SD_task_t taskD = SD_task_create("Tâche D", NULL, 60.0);

  SD_task_dependency_add(NULL, NULL, taskB, taskA);
  SD_task_dependency_add(NULL, NULL, taskC, taskA);
  SD_task_dependency_add(NULL, NULL, taskD, taskB);
  SD_task_dependency_add(NULL, NULL, taskD, taskC);

  /* paramètres d'exécution des tâches */
```

```
int workstation_number = 2;
SD_workstation_t w1 = SD_workstation_get_list()[0];
SD_workstation_t w1 = SD_workstation_get_list()[1];
SD_workstation_t workstation_list[] = {w1, w2};
double computation_amount[] = {2000000, 1000000};
double communication_amount[] =
  {
    0, 2000000,
    3000000, 0
  };
double rate = -1.0;

/* planification des tâches */
SD_task_schedule(taskA, workstation_number, workstation_list,
 computation_amount, communication_amount, rate);
SD_task_schedule(taskB, workstation_number, workstation_list,
 computation_amount, communication_amount, rate);
SD_task_schedule(taskC, workstation_number, workstation_list,
 computation_amount, communication_amount, rate);
SD_task_schedule(taskD, workstation_number, workstation_list,
 computation_amount, communication_amount, rate);

/* simulation */
SD_task_t *changed_tasks;
changed_tasks = SD_simulate(-1.0);

/* affichage des résultats */
for (i = 0; changed_tasks[i] != NULL; i++) {
  printf("La tâche '%s' a commencé à la date %f "
         "et s'est terminée à la date %f.\n",
SD_task_get_name(changed_tasks[i]),
SD_task_get_start_time(changed_tasks[i]),
SD_task_get_finish_time(changed_tasks[i]));
}
```

```
  /* on libère la mémoire */
  free(changed_tasks);
  SD_task_destroy(taskA);
  SD_task_destroy(taskB);
  SD_task_destroy(taskC);
  SD_task_destroy(taskD);

  /* on quitte SimDag */
  SD_exit();
  return 0;
}
```