Université de la Méditerranée, Aix-Marseille II
Laboratoire du Centre de Physique des Particules de Marseille

Cette thèse intitulée:

**Étude, définition et modélisation d'un
Système Distribué à Grande Échelle:
DIRAC - *D*istributed *I*nfrastructure with *R*emote *A*gent *C*ontrol**

présentée par:

Vincent Garonne
Ingénieur de Recherche au Centre National de Recherche Scientifique
Laboratoire de l'Accélérateur Linéaire

À l'École doctorale des sciences de la matière

En vue de l'obtention du grade de:

**Docteur de l'Université de la Méditerranée
(Spécialité Informatique)**

a été évalué par un jury composé des personnes suivantes:

Monsieur	Roy ALEKSAN	Président du Jury
Monsieur	Franck CAPPELLO	Examinateur
Monsieur	Eddy CARON	Co-directeur
Monsieur	Philippe CHARPENTIER	Rapporteur
Monsieur	Hervé JAOUEN	Invité
Monsieur	Stéphane LANTERI	Rapporteur
Monsieur	Renaud LE GAC	Directeur
Monsieur	Andreï TSAREGORODTSEV	Co-directeur
Monsieur	Guy WORMSER	Examinateur

Thèse acceptée le: 14/12/2005

Décembre, 2005

À mon grand-père, Jean Garonne.

RÉSUMÉ

La physique des particules traite un grand nombre de données qui nécessitent des ressources de calculs particulièrement importantes. C'est pourquoi, les applications de simulation et d'analyse d'une expérience de physique des particules se retrouvent dans un environnement de calculs distribués à grande échelle. Souvent dénommés grilles, ces environnements se différencient des machines parallèles les ayant précédés par leurs natures intrinsèquement hétérogènes, partagées et fortement dynamiques. Ils se déclinent en deux types de système : les grilles institutionnelles qui mutualisent les ressources d'organismes par accord mutuel et les systèmes communautaires de calcul global dont le pair-à-pair est un exemple. Dans cette thèse, nous étudions ces systèmes et soulignons l'intérêt d'un système hybride conjuguant les deux approches. Nous proposons une implémentation d'un système unifié *DIRAC* (Distributed Infrastructure With Remote Agent Control). Cette solution est un système léger, extensible et robuste, qui offre une plate-forme transparente et uniforme pour une seule communauté ou organisation virtuelle. Le but est d'agréger le plus grand nombre de ressources de tout type avec une simplicité de déploiement, de maintenance et d'administration. Nous détaillons les technologies et mécanismes mis en œuvre pour un tel environnement. *DIRAC* repose sur une *architecture orientée service* Agents/services régulant notamment la charge et les accès aux données dans le contexte de régime permanent et saturé (« High Throughput Computing ») générés par des simulations de Monte-carlo et des analyses de données. Ainsi, *DIRAC* a connecté plus de 6.000 processeurs répartis sur une soixantaine de sites dans le monde, a supporté plus de 5.500 tâches simultanées et a stocké, transféré et dupliqué plus de 100 téra-octets de données . Pour l'évaluation de l'ordonnancement de *DIRAC* dans un tel contexte, nous avons proposé une modélisation et développé un simulateur autorisant la comparaison de stratégies et d'architectures pour l'ordonnancement et le méta-ordonnancement. Avec cet outil, dont nous soulignons la validité, nous avons justifié l'approche de *DIRAC* « pull » face à d'autres approches centralisées et architectures de types « push ».

Mots-cléfs : Grille de calcul, calcul global, système de production, régime permanent et saturé, modélisation et simulation, (méta-)ordonnancement.

ABSTRACT

High energy physics and particularly *LHCb* will need a lot of computing resources when it comes online in 2006. A Petabyte of data each year is expected to be taken so it is not surprising that LHCb applications have been evolving with a large scale system environment in mind. Often called Grids, these environments are different from parallel machines due to their heterogeneous, shared and dynamic behaviour. Two types of large scale computing system are considered here. Firstly, Grid computing *i.e.* connected institutional resources in a trusted environment and secondly, global and peer-to-peer computing. In global and peer-to-peer computing, constraints are stronger than those in the Grid and more automatically-adaptable software is generally developed. In this thesis, we do a study and comparison of these systems and outline the benefits of having an hybrid system. We propose an implementation of such a system called *DIRAC* (Distributed Infrastructure With Remote Agent Control), which is lightweight, robust and scalable. The philosophy of *DIRAC* is to provide an uniform and virtual view of the system for one community or Virtual Organisation. The ultimate goal is to integrate the largest amount of different types of resources, with easy deployment, maintenance and administration. We detail the technologies and mechanisms put in place to faciliate such a system. *DIRAC* is organized into a Service Oriented Architecture (SOA) using agents to regulate workload and the access to data in a high throughput computing context. *DIRAC* connected more than 6.000 nodes on more than sixty sites spread all over the world and can support more than 5.500 simultaneous tasks. Also, more than 100 tera-bytes of data has been stored, transferred and replicated using *DIRAC*. To evaluate the *DIRAC* scheduling, a new model is presented and a simulator was developed for many clusters of heterogeneous nodes belonging to a local network. These clusters are assumed to be connected to each other through a global network and each cluster is managed via a local scheduler which is shared by many users. We justify our decentralized, adaptive and opportunistic approach in comparison to a centralized "Push" approach in such a context.

Keywords : Grid and global computing, production system, high throughput computing, model, simulation, (meta-)scheduling.

TABLE DES MATIÈRES

INTRODUCTION GÉNÉRALE

La mise en service de l'expérience *LHCb*, installée auprès du collisionneur proton-proton *LHC* (Large Hadron Collider), prévu pour 2007 au CERN nécessitera des ressources informatiques encore inégalées pour l'exploitation scientifique de ses données. Le domaine de la physique des particules est caractérisé par la nécessité de traiter individuellement plusieurs dizaines de milliards d'événements, véritables photographies numériques de collisions de particules élémentaires. Ces événements serviront à reconstruire les traces de particules nées de collisions proton-proton et de rechercher parmi ces photos celles où seront produites une nouvelle particule ou un phénomène encore inconnu. Cela se traduit par un volume de données gigantesque de 3 péta-octet par année, distribué sur plusieurs centres dans le monde. De plus, pour que les physiciens puissent analyser et exploiter ces données, une puissance de calcul à l'échelle d'un Système Distribué à Grande Échelle (*SDGE*) est une nécessité. L'enjeu d'un tel système est l'accès à des moyens de calculs intensifs via un réseau et un partage à grande échelle des ressources. Dans le but de concevoir et d'optimiser le détecteur *LHCb* des données simulant cette expérience ont besoin d'être produites en très grandes quantités. Il s'agit typiquement d'un contexte de « High Throughput Computing » [90] ou de régime permanent et saturé. La puissance d'un tel système s'estime selon la puissance de calcul ou la quantité de tâches exécutées pendant une période donnée asymptotiquement grande.

Diverses approches sont envisageables pour concevoir et déployer un système capable d'exploiter des ressources de calcul et de stockages réparties. Ces systèmes sont classés en considérant l'approche *institutionnelle* des grilles de calcul et celle *communautaire* du calcul global, dont le pair-à-pair est une des incarnations.

Les solutions considérées jusqu'à maintenant pour les expériences *LHC* sont principalement des solutions de grilles de calcul comme *LCG* [39], GRID3 [66] ou NorduGrid [63]. Le terme de grille de calcul ou GRID (Globalisation des Ressources Informatiques et des Données) fut popularisé par Ian Foster et Carl Kesselman en 1999 dans le livre *"The Grid : BluePrint for a New Computing Infrastructure"* [25]. Ils définissent le concept de grille de calcul comme *"Une infrastructure matérielle et logicielle qui fournit un accès sûr, consistant, omniprésent et peu coûteux à des facilités de calcul haute performance"*. Le mot grille vient de l'analogie avec le réseau électrique (*Power Grid*) qui fournit une puissance aux utilisateurs de manière simple et transparente. L'environnement des grilles est un environnement de confiance où les utilisateurs sont regroupés en organisation virtuelle [50]. Une organisation virtuelle est trans-institutionnelle et se définit par centre d'intérêts. Une grille connecte des ressources lourdes et stables de plusieurs sites de calculs. Le calcul sur grille ou *Grid Computing* les utilise pour des applications scientifiques, typiquement de très gros calculs. Une grille se focalise donc sur le calcul haute performance et fournit une qualité de service.

Le calcul global, dont le projet phare est SETI@home [47], connecte des millions d'ordinateurs personnels par internet pour un seul but, qui peut être le partage des fichiers ou un gros calcul. L'extrême volatilité des ressources, non dédiées, oblige à développer d'autres approches pour agréger les ressources souplement, dynamiquement et en toute transparence des utilisateurs.

La principale contribution de cette thèse est *DIRAC* (*Distributed Infrastructure with Remote Agent Control*) [5]. *DIRAC* a été conçu pour l'expérience *LHCb* afin de répondre aux exigences de la production simulées et pour l'analyse de ses données. L'architecture de ce système a été élaborée sur le principe d'une coopération entre différents composants, dénommée comme une *architecture orienté service* [59]. De plus, *DIRAC* s'appuie sur une organisation Agents/Service et illustre la convergence des systèmes de calcul global et des systèmes de grille de calcul. Ainsi, La philosophie de *DIRAC* est d'être un système léger, extensible et dynamique pour une seule communauté, *LHCb*. *DIRAC* est un système complet capable d'exploiter tout type de ressource comme celles de calcul, de stockage, de logiciel et réseau. Dans cette thèse, nous considérons principalement la gestion des ressources de calcul et la régulation de la charge dans un contexte de régime permanent saturé.

La méthodologie privilégiée pour l'étude d'un *SDGE* conjuguant l'aspect des grilles de calcul et du calcul global est celle de l'expérimentation. L'analyse théorique de l'ordonnancement, des protocoles et des propriétés du système ne sont pas abordés dans cette étude. Bien qu'essentielle, la non utilisation de ces outils d'analyse s'explique principalement par la difficulté d'avoir un modèle décrivant les caractéristiques d'une plate-forme de calcul distribué à grande échelle, dynamique et partagée. L'évaluation des stratégies d'ordonnancement s'effectue par un simulateur basé sur *Simgrid* et développé pour ces études. En tant que projet directement rattaché à une expérience, *DIRAC* est évalué par *LHCb* lors de son utilisation pour les « Challenges » ou défis pour préparer la mise en route du *LHC* en 2007. *DIRAC* a été approuvé pour être le système de production et d'analyse de données de *LHCb* pour les prochaines années d'exploitations de celle-ci.

Ce manuscrit se compose des chapitres suivants :

Chapitre 1. Nous présentons le domaine de la physique des particules, plus particulièrement l'expérience *LHCb*. Nous détaillons, pour celle-ci, ses besoins et caractéristiques en terme de puissance de calcul, d'espace de stockage et d'applications. Le modèle de calcul de *LHCb* privilégie la solution des Systèmes Distribués à grande Èchelle.

Chapitre 2. Ce chapitre présente brièvement l'évolution du calcul scientifique qui aboutit au Système Distribué à Grande échelle. Nous donnons ainsi un état de l'art de ces systèmes. Nous suivons une classification qui considère les grilles de calcul et le calcul global. Nous détaillons ensuite chacune de ces formes en décrivant leurs ressemblances et leurs différences. Nous observons l'avantage et l'intérêt d'avoir un système hybride conjuguant les deux aspects.

Chapitre 3. Nous introduisons dans ce chapitre le système *DIRAC* (Distributed Infrastructure With Remote Agent Control). Nous présentons sa philosophie de conception, qui est celle d'un système communautaire, léger, extensible et robuste conjuguant les aspects du calcul global et de grille de calcul. Il repose sur une *architecture orientée service* et fournit ainsi un ensemble de services pour un *SDGE* générique. Nous décrivons un état de l'art sur la gestion de la charge de calcul dans un *SDGE*, puis nous décrivons la régulation de la charge de *DIRAC* qui suit une stratégie passive « pull », typique d'un système de calcul global adapté dans un contexte de régime permanent et saturé. Par ailleurs, nous décrivons les autres entités, nécessaires à un *SDGE*, qui composent *DIRAC*.

Chapitre 4. Nous détaillons le déploiement du système *DIRAC* et l'expérience acquise avec son utilisation. L'expérience *LHCb* a utilisé *DIRAC* intensivement pendant 4 mois lors du *Data Challenge* en 2004. Nous décrivons les propriétés du modèle Agents/Service conjugué avec le paradigme « pull » et des effets seulement observables sur une plate-forme réelle. Ainsi, *DIRAC* a connecté plus de 6.000 processeurs répartis sur une soixantaine de sites dans le monde. Le nombre total de tâches exécutées est de l'ordre de 200.000 tâches. *DIRAC* a supporté plus de 5.500 tâches simultanées. Le montant des données transférées et stockées est supérieure à 100 Téra-octets.

Chapitre 5. Nous traitons dans ce chapitre de la difficulté d'évaluer rigoureusement des stratégies d'ordonnancement ou de méta-ordonnancement avec une plate-forme de *SDGE*. Nous détaillons tous les outils d'évaluation pour ensuite privilégier la simulation. Nous présentons notre modèle d'évaluation et de performances pour une plate-forme méta-ordonnancée composée de sites autonomes. Nous décrivons ensuite notre simulateur développé basé sur *Simgrid* et les travaux liés à sa validation.

Chapitre 6. Fort de l'outil de simulation, nous évaluons l'architecture décentralisée de *DIRAC* et sa stratégie passive « pull » contre une architecture centralisée appliquant une stratégie active « push ». Cette évaluation se situe sur une plate-forme hétérogène pour des tâches simples dans un contexte de régime permanent et saturé. Nous étudions ensuite la problématique d'un ordonnancement multi-critères (bi-applicatifs), en considérant le mélange de deux applications aux besoins différents sur une même plate-forme. L'une nécessite de maximiser le nombre d'unités élémentaires traitées sur une période asymptotiquement grande, tandis que l'autre favorise le temps de réponse d'une tâche. Nous définissons des stratégies passives se basant sur des priorités et des préemptions. Nous évaluons ensuite ces stratégies avec le simulateur.

Nous concluons finalement ce manuscrit en discutant nos apports et les perspectives des travaux présentés.

CHAPITRE 1

LE CONTEXTE : *LHCb*, UNE EXPÉRIENCE DE PHYSIQUE DES PARTICULES

Résumé du chapitre :
Nous présentons ici l'expérience de physique de haute énergie LHCb, *installée auprès du grand collisionneur hadronique, le* LHC. *Nous détaillons sa justification physique, ses caractéristiques et besoins informatiques. Nous mettons en évidence que la seule solution envisageable pour satisfaire les besoins informatiques associés à l'expérience* LHCb, *est d'utiliser la technologie des Systèmes Distribués à Grande Échelle.*

1.1 L'expérience *LHCb*

LHCb (*Large Hadron Collider Beauty*) est une expérience de physique des particules. Cette expérience de seconde génération étudiera la violation CP et les désintégrations rares dans le secteur de « la beauté ». La symétrie CP est le produit de deux symétries, C, la conjugaison de charge, qui transforme une particule en antiparticule et P, la parité, qui transforme un système physique en son image dans un miroir. *LHCb* sera installée avec les expériences ATLAS, CMS et Alice auprès du futur grand collisionneur hadronique, le LHC.

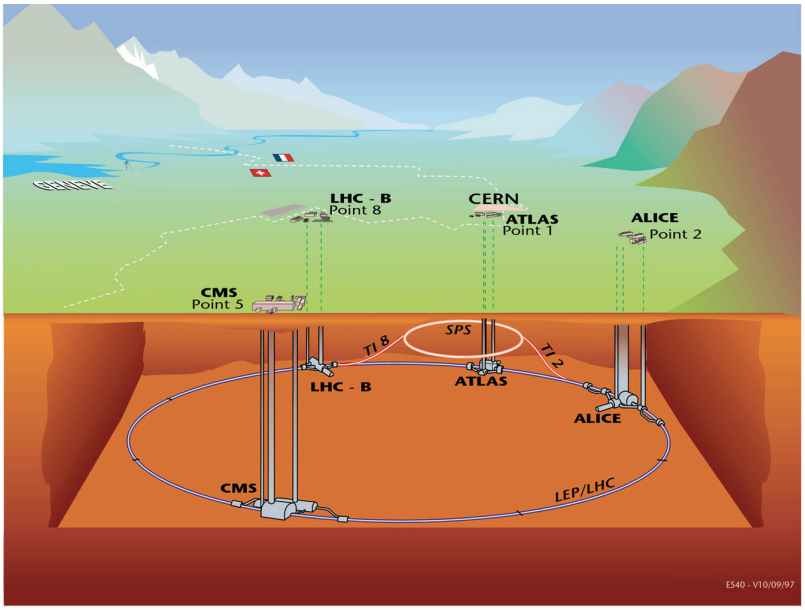


FIG. 1.1 – Illustration du LHC(Large Hadron Collider).

Le LHC, illustré sur la figure 1.1 est le plus grand accélérateur de particules qui est actuellement en construction au CERN, à la frontière franco-suisse de Genève. Le LHC est construit à 100 mètres de profondeur sur l'emplacement de l'ancien grand collisionneur électron positon, le LEP. En 2007, deux faisceaux de protons seront accélérés en sens opposés sur les 27 km de circonférence du LHC avant d'entrer en collision. De nouvelles particules émergeront de ces collisions de faisceaux. Celles-ci se produiront à une énergie encore jamais atteinte de 14 TeV [1].

1.2 La justification physique

L'expérience *LHCb* s'intéresse à la symétrie matière-antimatière en étudiant spécifiquement les mésons B, particules composées d'un quark b appelé "beauté" et d'un antiquark.

L'objectif ultime est de comprendre pourquoi l'Univers est constitué de matière plutôt que d'antimatière. Contrairement aux grands détecteurs polyvalents que sont Atlas et CMS, *LHCb* est un outil spécialisé destiné à réaliser la meilleure détection possible des mésons B. Ces particules sont émises préférentiellement dans des directions voisines du faisceau. Le détecteur est conçu pour les observer à petit angle. Le taux élevé de production de particules permettra d'approfondir l'étude de la violation de la symétrie CP dans des modes rares de désintégration du méson B.

La compréhension de la nature de la violation de CP est un enjeu majeur de la recherche en physique des particules et en cosmologie. En effet, le Modèle standard permet de rendre compte de la violation de CP à travers l'existence d'une phase qui apparaît dans la matrice de Cabibbo-Kobayashi-Maskawa (CKM) [175]. Mais l'origine de cette phase est mystérieuse. Il prédit aussi des effets violents CP, importants, dans les systèmes des mésons beaux. Par ailleurs ce module théorique de la violation de CP est peu contraint expérimentalement, laissant la porte ouverte à d'autres sources de violation de CP qui pourraient trouver leurs origines dans les modules supersymétriques. L'étude de la violation de CP et des désintégrations rares dans le secteur de la beauté permettra de tester le Modèle standard comme étant la formulation théorique de la violation de CP et, peut-être, de mettre en évidence des processus nouveaux.

1.3 Le détecteur *LHCb*

Le détecteur est composé d'un ensemble de sous-détecteurs, voir figure 1.2. L'appareil est disposé autour du tube à vide de la machine. Il permet de mesurer les trajectoires des particules émises et d'identifier ces particules.

[1] 1 TeV = 1 millier de milliards de fois l'énergie qu'un électron acquiert lorsqu'il passe du pôle positif au pôle négatif d'une pile de 1 Volt.

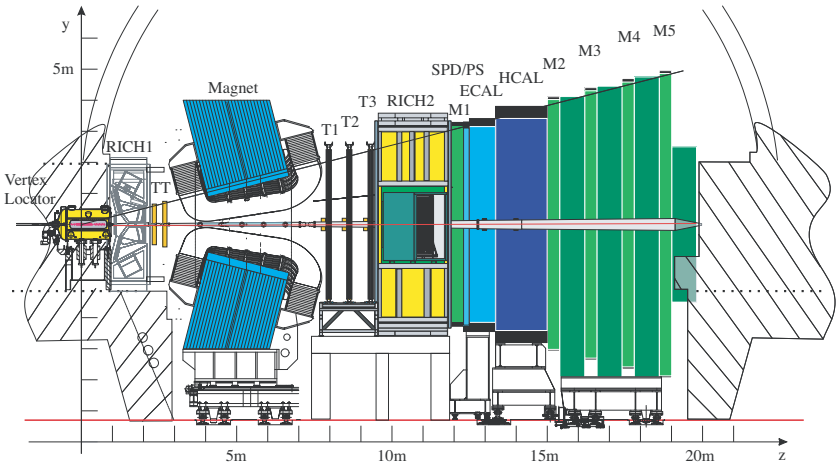


FIG. 1.2 – Le détecteur *LHCb* est composé de sous-détecteurs comme le VELO, les deux détecteurs RICH, les quatre stations « tracking » (TT, T1-T3), les quatres magnet, les détecteurs SPD et PS, les calorimètres electromagnétique (ECAL) et hadronique (HCAL), et les cinq stations muon M1-M5.

1.4 L'acquisition et le traitement des données

L'analyse et l'interprétation des données recueillies constitue le but essentiel d'une expérience de physique des particules. Mais avant cette ultime étape il est obligatoire d'acquérir les données du détecteur. Ces données décrivent les particules issues des collisions protons-protons à 14 TeV. Par ailleurs des simulations Monte-carlo sont utilisées pour concevoir, comprendre et améliorer le détecteur. Nous avons deux sources de données l'une provenant du détecteur, l'autre de simulation. La figure 1.3 illustre la chaîne d'acquisition des données réelles et simulées.

1.4.1 L'acquisition des données réelles

L'ensemble des détecteurs participe à une logique de décision complexe qui sélectionne les collisions intéressantes, en temps réel, toutes les 25 nano-secondes.

Les composants du processus de sélection s'appellent le système de déclenchement ou les « triggers ». Nous avons le *trigger* de niveau 0 (L0) processeurs câblés dont les algorithmes sont implantés dans des réseaux logiques programmables comme les *FPGA* [176]. Les *triggers* de niveau 1 sont logiciels et se composent chacun d'un routeur relié à une ferme de calcul en ligne.

Après la phase de sélection, les données brutes sont reconstruites en temps réel en utilisant des informations supplémentaires comme les données de calibration. Cette phase de reconstruction ajoute des méta-données qui caractérisent le type de physique présente, les conditions de prise de données. Elles facilitent le traitement ultérieur des données.

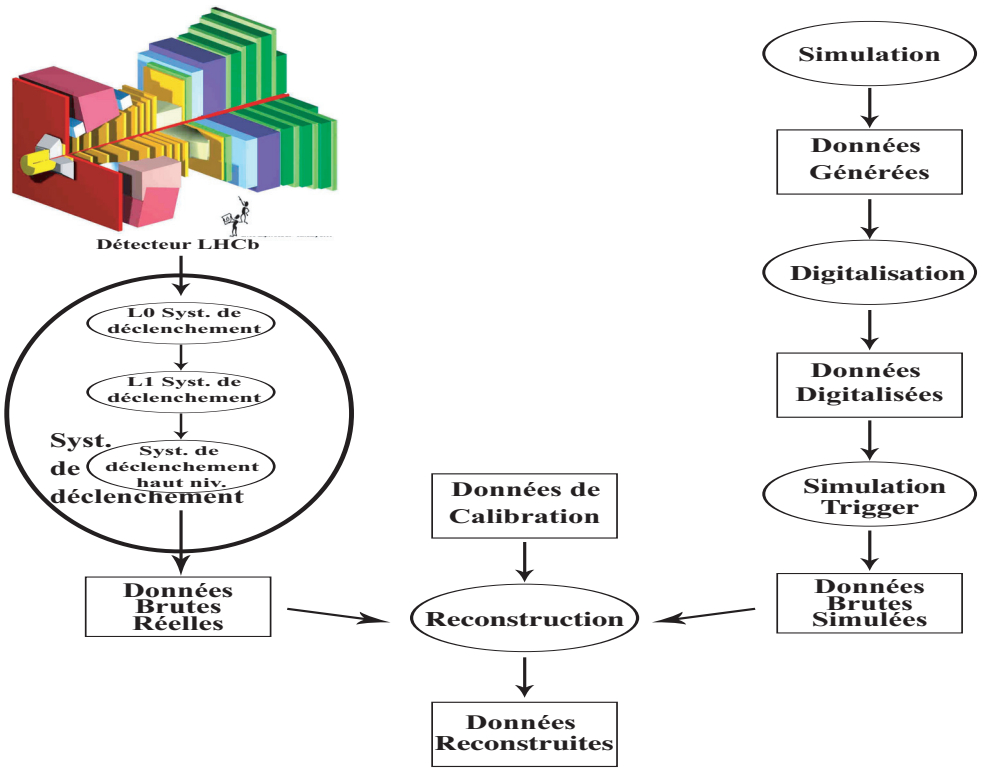


FIG. 1.3 – Le modèle logique du flot de données de *LHCb*.

La sélection diminue le nombre d'événements de 40 millions par seconde en entrée à 2.000 par seconde en sortie. La taille finale d'un événement, véritable photographie numérique en trois dimensions, est de l'ordre de 25 kilo-octets. Les événements sont ensuite stockés sur bande. Les ensembles d'événements ayant des caractéristiques similaires sont décrits et enregistrés dans une base de données. Cette base de données autorise la recherche selon des critères de physique ou des conditions de prise de données précises.

1.4.2 Les données simulées

La prise de données commencera avec la mise en fonctionnement du collisionneur *LHC* à l'été 2007. Pour préparer cette échéance, des données simulant l'expérience par Monte-carlo sont produites. La quantité des données simulées est équivalente à quelques minutes de fonctionnement de *LHCb*. Ces simulations se basent sur un modèle détaillé des propriétés du détecteur et des phénomènes physiques à étudier. C'est de plus grâce à cette modélisation que l'on peut à l'avance prévoir les performances du détecteur et étudier diverses hypothèses théoriques et leurs effets sur les résultats de physique. Les différentes étapes de la simulation sont illustrées sur la partie gauche de la figure 1.3 :

La génération d'événement.
Les particules qui émergent des collisions sont générées en utilisant des logiciels basés sur les théories de physique décrivant les interactions entre quarks.

La simulation.
Les particules sont « transportées » dans le détecteur en utilisant les lois de physique qui gouvernent le passage des particules à travers la matière.

La digitalisation.
Les interactions avec les éléments sensibles du détecteur sont simulées, reproduisant ce qui devrait être les données réelles.

1.4.3 La reconstruction et la présélection

Les données sont reconstruites en incluant les méta-données. Les formats des données simulées sont identiques aux formats des données réelles. Les données simulées sont de même enregistrées et décrites dans une base de données pour permettre leurs analyses.

La physique considérée détermine l'analyse des données. Pour faciliter celle-ci des présélections sont effectuées. Les critères de présélections se définissent au niveau d'un groupe de travail qui étudie un canal de physique précis. Chaque groupe de travail de la collaboration fournit les algorithmes de présélection. Ces algorithmes s'appliquent sur les données réelles et simulées. Après cette phase, les méta-données relatives aux sélections se rajoutent aux données sélectionnées.

1.4.4 L'analyse des données

Après la phase de présélection, les physiciens analysent les données. Une analyse est typiquement à l'échelle d'un seul physicien, ou d'un petit groupe de physiciens pour l'étude d'un canal de physique ou pour la mesure d'une quantité précise. Cette phase d'analyse génère des données quasiment privées et propres à un physicien. Ce sont par exemple des vecteurs qui seront ensuite traités par des outils statistiques.

1.5 Les caractéristiques et besoins des applications

La synthèse présentée ici s'inspire des recueils des besoins et rapports techniques des expériences du *LHC* [169–171]. Nous nous situons au niveau hors ligne qui diffère de la partie temps réels, nécessaire pour l'acquisition et la sélection des données réelles.

La partie hors ligne englobe les applications purement logicielle comme : 1) La production de données Monte-carlo, 2) la présélection et 3) l'analyse des données.

Nous avons expliqué précédemment dans la section 1.4 l'enchaînement de ces applications. Nous détaillons, maintenant, comment cette logique se traduit en terme de *workflow* ou flot de traitement d'un point de vue informatique. Un *workflow*

est composé de logiciels et de formats de données précis. Il exprime la séquence des opérations. Un *workflow* se représente comme un graphe. Il se compose de modules et de filtres de données avec des entrées/sorties bien spécifiées. La figure 1.4 montre le *workflow* des applications *LHCb* pour la production de données simulées, la reconstruction et l'analyse.

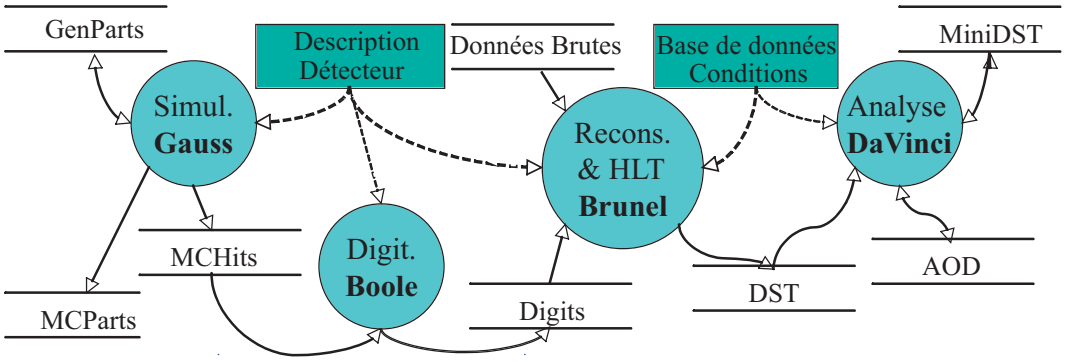


FIG. 1.4 – Le workflow pour la production et l'analyse de données dans *LHCb*.

Nous allons maintenant détailler pour chaque application ses caractéristiques qui sont résumées sur la figure 1.5.

1.5.1 La production de données

La production de données simulées se planifie au niveau de la collaboration. Un seul gestionnaire de production en est responsable. Elle consiste à générer une quantité de données déterminée, et se compose d'un grand nombre de tâches purement calculatoires non communicantes et séquentielles. Bien que la granularité de ces tâches est paramétrable par le nombre d'événements à reconstruire, il est usuel de produire peu d'événements par tâche, typiquement 500. Les calculs sont longs de l'ordre de plusieurs heures. Ces tâches sont de type *batch*. Une tâche *batch* s'exécute sans aucune interaction avec les utilisateurs. La taille d'un événement est en moyenne de 3 méga-octets et le temps associé de l'ordre de 15 minutes par événement sur un processeur rapide du marché actuel (P4 2.4 GHz).

Le temps d'exécution est linéaire par rapport au nombre d'événements demandés. En entrée, ces tâches utilisent des paramètres de configuration et génèrent en sortie les données. L'ordre de grandeur pour les données issues d'une simulation est compris entre 500 méga-octets et 1 giga-octets. Ce type d'application est qualifiée d'application multi-paramétriques ou *parameter sweep application* [51]. Une production favorise amplement le débit de calcul sur une grande période de temps déterminée. Nous sommes dans une situation de « High Troughput Computing » [90] où nous nous intéressons à maximiser le nombre de tâches ou d'opérations effectuées sur une longue période de temps. Ce contexte favorise le régime permanent et saturé d'un système de calcul.

1.5.2 La préselection

Une présélection est planifiée par la collaboration. Une tâche de présélection admet en entrée un ensemble de données. Cette tâche est séquentielle et non communicante avec d'autres. Après l'application des critères de sélection, nous obtenons en sortie les données. La taille des entrées est de l'ordre de la dizaine de giga-octets. Le facteur de réduction des données est en moyenne de 10. La granularité d'une tâche est considérée comme petit grain. Cette application est plus liée aux données que calculatoire. Le temps d'exécution de ces tâches est linéaire par rapport à la quantité de données en entrée. Cette application est aussi une application *batch* et de « High Troughput Computing ».

1.5.3 L'analyse de données

L'analyse de données est une application qui intervient de façon totalement non prédictive car elle se situe au niveau d'un physicien. Le nombre de physiciens qui analysent les données de la collaboration sera de l'ordre de 140. Une tâche d'analyse nécessite un volume de données en entrée de l'ordre du giga-octet et renvoie un résultat sous forme de fichier de l'ordre du méga-octet contenant des histogrammes. Une tâche d'analyse est de petit grain, elle est souvent qualifiée d'application *data intensive* ou grande masse de données car elle nécessite beaucoup d'opérations d'entrées/sorties sur des données. Le temps d'exécution est non linéaire par rapport aux données et non prévisible. En opposition avec les tâches de simulation, la particularité de cette tâche est de favoriser le temps de réponse de l'application. De plus, cette application peut être soit exécutée en *batch* ou de manière interactive. Une tâche interactive en opposition à une tâche *Batch* a besoin d'interactions de l'utilisateur pour continuer son exécution. Contrairement au « High Troughput Computing », nous sommes dans un contexte de « High Performance Computing » ou de haute performance. Dans celui-ci, les ressources sont disponibles dés que l'utilisateur le demande et ceci pour minimiser le temps de réponse. Cette problématique est détaillée plus longuement dans la section 3.4.7 du chapitre 3.

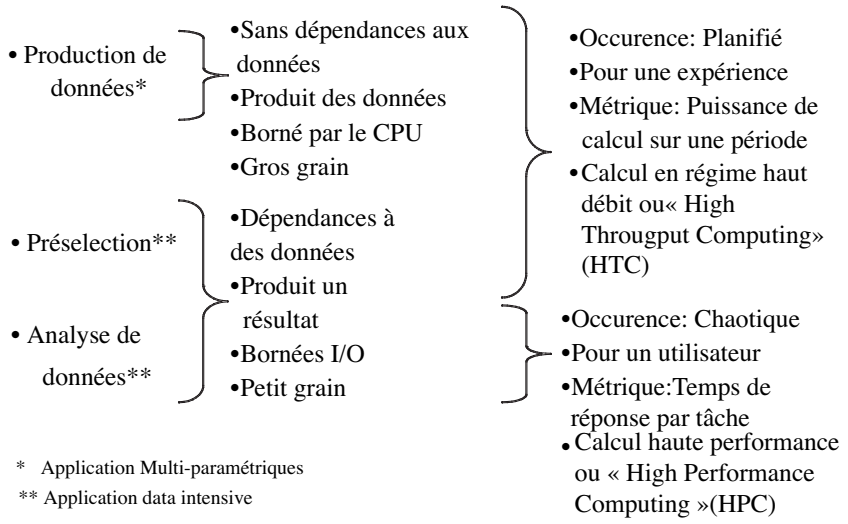


FIG. 1.5 – Résumé des caractéristiques des applications de *LHCb*.

1.5.4 Les besoins matériels

Les besoins matériels de *LHCb* s'expriment à travers des capacités de stockage et de calcul [170]. Le tableau 1.1 résume l'estimation des besoins des trois types d'application pour les premières années d'utilisation. La puissance processeur est exprimée dans l'unité de mesure CPU, le SpectInt2K[2], et la capacité de stockage en tera-octet.

Année / Besoins	2007	2008	2009	2010
CPU [M SpectInt2K]	7.78	12.97	14.45	17.88
Disque [To]	1969	3281	4015	4749
Stockage de masse [To]	2069	3433	7144	11632

TAB. 1.1 – Résumé prévisionnel des besoins en puissance de calcul et de stockages pour l'expérience *LHCb*.

Un Kilo SpectInt2K est équivalent à un processeur rapide comme le P4 2.4 GHZ (2005). En 2010, la puissance de calcul est estimée à **18.000 processeurs actuels**. Il est important de noter que nous donnons ici des estimations pour *LHCb* mais des besoins similaires voir supérieures sont aussi quantifiés pour les autres expériences

[2]Unité de mesure de puissance CPU, URL : <http ://www.spec.org>

du LHC. En effet pour l'ensemble des expériences du *LHC*, les besoins en calculs seront de l'ordre de 100.000 machines actuelles.

Au niveau du stockage nous distinguons deux niveaux : le stockage de masse pour l'ensemble des données produites ; le stockage sur disque pour le traitement de chaque application. Le stockage sur disque est de l'ordre de 4 péta-octets. Le stockage de masse sera de 3 péta-octets la première année. Les données produites sur une année sont cumulées et reportées sur les années suivantes. Aucune donnée produite par le détecteur n'est détruite.

1.6 Comparaison avec les technologies existantes

Les impressionnants besoins en calcul et de stockage de la physique des particules nous permet de faire le point sur les technologies existantes et leurs capacités.

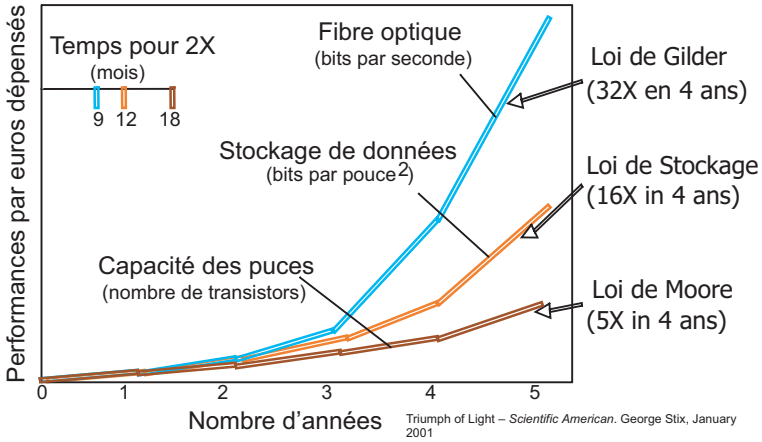


FIG. 1.6 – Évolutions des performances technologiques par euros dépensés en fonction du nombre d'années.

Aucun système n'est aujourd'hui capable de fournir la puissance de calcul et de stockage nécessaire au traitement des données de *LHCb*. En effet, le plus puissant des supercalculateurs [62] fournit maintenant une puissance équivalente à 10.000 processeurs, puissance inférieure d'un facteur 2 par rapport aux besoins *LHCb*. Outre son coût prohibitif, cette solution n'est pas envisageable pour toutes les expériences du *LHC*.

Une énorme grappe de calcul bien que moins coûteuse est bien trop complexe à mettre en œuvre, ne serait ce que pour des questions de place ou de système de refroidissement. Au niveau du stockage les capacités maximales atteintes sont de l'ordre du péta-octet. Elles sont insuffisantes par rapport au besoin de *LHCb*.

Si nous regardons de plus les prévisions pour les capacités des ressources informatiques [75, 93] comme l'illustre la figure, nous pouvons espérer de maintenant

(2005) à 2007 un facteur 100 sur la capacité des réseaux, un facteur 4 sur la puissance d'un processeur, un facteur 5 sur l'espace de stockage d'un disque dur et pour les systèmes de stockage de masse. En considérant tout ceci, le modèle des Systèmes Distribués à Grande Échelle (*SDGE*) parait le plus adapté. En effet, ce modèle réparti agrège par le réseau toutes les ressources disponibles et ceci à moindre coût. L'effort pour fournir toutes les ressources du système se partage alors avec tous les pays membres.

1.7 Le modèle de calcul

La collaboration *LHCb* est constituée d'environ 450 participants provenant de 47 laboratoires répartis dans 14 pays. Chaque laboratoire ou pays participant possède souvent des moyens de calcul et de stockage. Ce constat donna lieu au modèle de calcul suivant.

Les données de *LHCb* seront tout d'abord disponibles en un seul point : au CERN. Ensuite ces données seront redistribuées sur l'ensemble des sites participants. Ce modèle suit une structure hiérarchique et de pipeline comme le montre la figure 1.7. Ces ressources étant hétéroclites de par leurs tailles et leurs importances, l'idée de les hiérarchiser par caractéristiques est apparue. Cette hiérarchie comporte quatre niveaux ou « Tiers ». Chaque niveau répond à des fonctionnalités et spécificités précises. Pour *LHCb*, nous avons :

Tier-0.
: Le site Tier-0 est unique et se situe au CERN. Il hébergera les données brutes de *LHCb* et les re-distribuera sur les centres Tier-1. Il fournira une capacité de calcul de l'ordre de 20.000 processeurs, une capacité de stockage disque et stockage de masse de l'ordre de 6 péta-octets et une connectivité réseau 20 Gbps avec les Tier-1.

Tier-1.
: Les centres régionaux Tier-1 servent une nation ou un groupe de nations. Un centre Tier-1 devra fournir une capacité de calcul de l'ordre de 12M SpectInt2K, une capacité de stockage disque et stockage de masse de l'ordre de 6 péta-octets et une connectivité réseau avec le Tier-0 (CERN) de 10 Gbps.

Tier-2.
: Les centres Tier-2 se destinent principalement à des régions. Ils sont orientés plutôt puissance de calcul et sont faiblement couplés avec des stockages de masse. Les ressources Tiers-2 requises pour *LHCb* sont de 1000 processeurs et de 5 TB de disque.

Tier-3 / Tier-4.
: Les machines locales aux institutions membres représentent le Tier-3 du modèle. Les machines utilisateurs constituent le Tier-4.

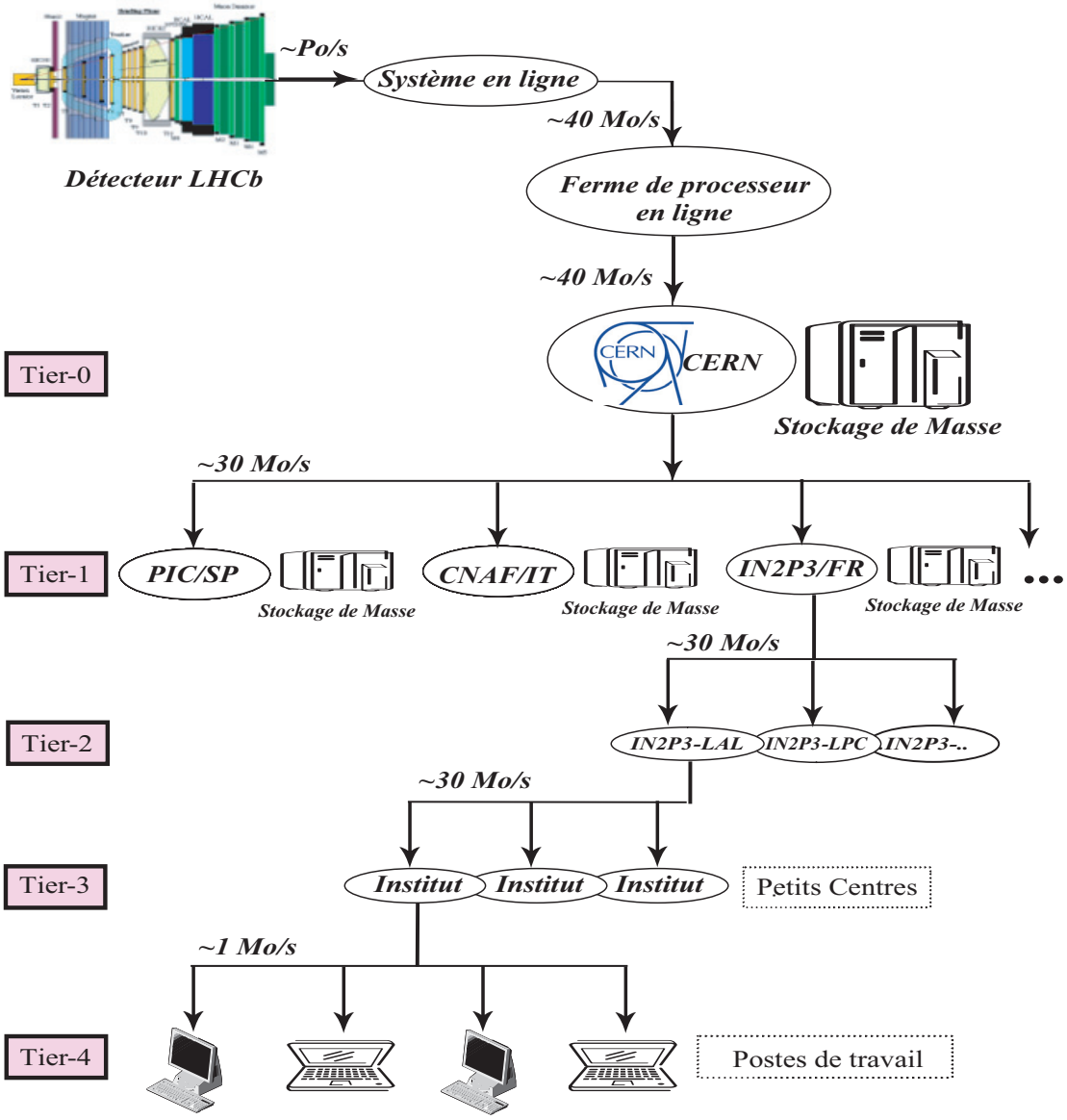


FIG. 1.7 – Distribution et partage des données de *LHCb*.

1.8 Résumé et problématique

Dans ce chapitre, nous avons présenté le domaine de la physique des particules au sein de *LHCb*. Les besoins des utilisateurs représentent des exigences jamais atteintes pour le calcul scientifique. En effet, l'expérience *LHCb* produira chaque année un volume de données de l'ordre de 1.3 péta-octets et des données simulées en quantité similaire. Ces données sont réparties sur une cinquantaine de centres de calcul en Europe. Pour que les physiciens analysent et exploitent ces données, nous devons leur fournir une puissance de calcul et de stockage de l'ordre de 18.000 processeurs actuels ce qui est très supérieure à celle disponible aujourd'hui dans un seul centre de calcul. Connecter physiquement des sites par le réseau ne suffit pas à résoudre les exigences du modèle, Il est nécessaire maintenant d'avoir un système qui puisse agréger et utiliser pleinement toutes ces ressources de stockage et de calculs. Ces nécessités ont donné naissance au concept de *SDGE*. Ce dernier sera présenté dans le chapitre 2 suivant. Nous montrerons comment cette technologie répond aux contraintes de l'expérience *LHCb*.

CHAPITRE 2

LES SYSTÈMES DISTRIBUÉS À GRANDE ÉCHELLE (*SDGE*)

Résumé du chapitre :
Dans ce chapitre nous présentons brièvement l'évolution du calcul scientifique qui aboutit au Système Distribué à Grande échelle. Nous proposons notre classification de ces systèmes en considérant les grilles de calcul et le calcul global, dont le pair-à-pair est un des exemples. Nous détaillons ensuite chacune de ces formes en décrivant leurs ressemblances et leurs différences. Nous étendons ici les concepts des SDGE *en l'appliquant aux besoins de la physique des particules.*

2.1 Introduction

À travers les évolutions technologiques liées au calcul scientifique, nous constatons que les stratégies répondant aux demandes croissantes de calcul montrent de grandes similarités. En effet, conjointement à l'amélioration constante des composants de calcul et de stockage, l'augmentation du nombre de ces composants et des capacités de communication entre eux s'est généralisée au sein des successives infrastructures de calcul. La croissance de la taille de ces systèmes aboutit aujourd'hui aux Systèmes Distribués à Grande Échelle (*SDGE*) et s'accompagne de nouvelles spécificités. Celles-ci amènent de nouvelles classes de problèmes à résoudre. Elles contraignent à re-explorer les solutions existantes et à en créer de nouvelles. Dans ce chapitre, nous décrivons les nouveaux aspects introduits par les *SDGE* et les solutions proposées. Ce chapitre débute par un bref rappel historique du calcul scientifique avec ses tendances. Nous proposons notre définition et classification de *SDGE*. Nous détaillons ensuite chacune des approches. Pour chacune d'entre elles, nous nous concentrons sur les fonctionnalités et leurs mises en œuvre pour finalement les comparer.

2.2 Historique du calcul scientifique

En réponse aux demandes croissantes de calculs, les infrastructures et les applications ont évolué de manière à se détacher de plus en plus du support matériel.

En effet, les premiers systèmes utilisés étaient des systèmes *monolithiques* comme les « mainframes » ou gros systèmes d'IBM introduis en 1960. Un système monolithique est fortement lié à la couche matérielle. Les supercalculateurs d'aujourd'hui sont les descendants des systèmes monolithiques. Ces solutions sont spécifiques à des applications et peu flexibles. Elles sont de plus propriétaires, coûteuses et difficilement extensibles.

Pour gagner de la puissance, une des stratégies a été de modifier l'architecture matérielle des machines et le nombre de composants. Bien que la loi de Moore [78], proposée en 1965, indique que les puissances des processeurs doubleront tous les 18 mois, ces solutions matérielles arrivent rapidement aux limites technologiques. Nous estimons que les limites de la miniaturisation seront atteintes dans quelques années [79,93].

Les infrastructures, principalement logicielles, sont devenues alors des systèmes *ouverts*. Devant la pluralité des solutions matérielles, un système ouvert supprime les dépendances matérielles du système. Mais les applications restent toujours dépendantes d'une infrastructure logicielle. Les systèmes ouverts les plus connus sont les systèmes d'exploitation. Dans ce cadre les applications sont disponibles par une approche client-serveur. Une application est statiquement liée à un serveur. Dans un cadre muti-applicatifs, ces configurations résultent ensuite d'une mauvaise utilisation des serveurs. Certains se trouvent trop ou pas assez utilisés.

De plus, l'amélioration constante des réseaux locaux a encouragé le passage à un modèle informatique *distribué*. Ce modèle augmente le modèle client-serveur avec, par exemple, une gestion des traitements ou des mécanismes d'exécution. Nous comprenons dans ces systèmes les grappes de machines(clusters). Ces grappes se composent des machines de bureau au sein d'une entreprise ou des machines dédiées. L'entrée de grands clusters dans le top 500 [62] des supercalculateurs est la preuve que ces systèmes rivalisent avec les supercalculateurs, et ceci, à moindre coût. Avec le développement du réseau haut débit, l'idée de généraliser ce modèle à une grande échelle fut possible. Les projets résultants s'appellent des Systèmes Distribués à Grande Échelle (*SDGE*).

Un *SDGE* est un système qui connecte et utilise des ressources hétérogènes par un réseau à grande échelle. Les *SDGE* sont des systèmes qui procurent des puissances de calcul et de stockage susceptible de répondre aux besoins les plus importants des communautés scientifiques, notamment celle de la physique des particules.

Avec ce bref historique nous constatons qu'en réponse aux demandes croissantes, les infrastructures ont évolués de monolithiques, à ouvertes et distribuées. Ces changements « darwinien » se sont souvent accompagnés de phases de transition nécessaires à la maturité d'une infrastructure, ainsi que sa standardisation. Les applications suivent alors de nouveaux modèles de programmation pour répondre au nouvelles spécificités introduites. Le passage de la programmation séquentielle à parallèle [83] est un exemple de ces évolutions. Nous commençons maintenant dans le domaine des *SDGE* à suivre cette évolution.

En effet, les *SDGE* élargissent le concept de la *virtualisation*. Déjà présente au niveau de la programmation, elle se généralise au niveau des ressources et des services. Elle se focalise sur les fonctionnalités logiques d'une entité et s'abstrait totalement de l'implémentation sous-jacente, laissant possible pour un même service

plusieurs implémentations. Pour coordonner plusieurs services, il faut une infrastructure de coopération, nous parlons alors d'*architecture orientée service*. Une architecture orientée service favorise la dynamicité, la flexibilité et la coopération. Mais pour assurer l'interopérabilité entre services, les interfaces et protocoles d'invocations doivent être clairement identifiés. Cette identification du rôle de chaque composant illustre les efforts de standardisation engagés dans *le Global Grid Forum* [1]. De plus de nombreux projets existent aujourd'hui, et sont référencés sur le site « Grid Computing »[2]. Nous proposons dans la section suivante une classification pragmatique de ces projets et nous montrons que nous tendons vers un système virtuel, dynamique et flexible.

2.3 Une classification des *SDGE*

En général, les technologies de *SDGE* cherchent à répondre à la problématique suivante : mieux exploiter les ressources existantes pour fournir de plus grands moyens de calcul et de stockage en toute transparence. Concrètement, il s'agit d'autoriser un ou plusieurs utilisateurs, d'utiliser à moindre coût des ressources dont ils n'ont pas l'accès individuellement. Le lien entre les ressources et les utilisateurs s'opère par une couche intergicielle, en anglais « middleware ». Elle est la brique de base regroupant l'ensemble des éléments pour la mise en œuvre d'un *SDGE*. Pour la réalisation de ces objectifs, nous distinguons deux inspirations ou courants majeurs pour les *SDGE*, l'un *institutionnel* ou professionnel et l'autre plus *communautaire* comme illustré sur la figure 2.1.

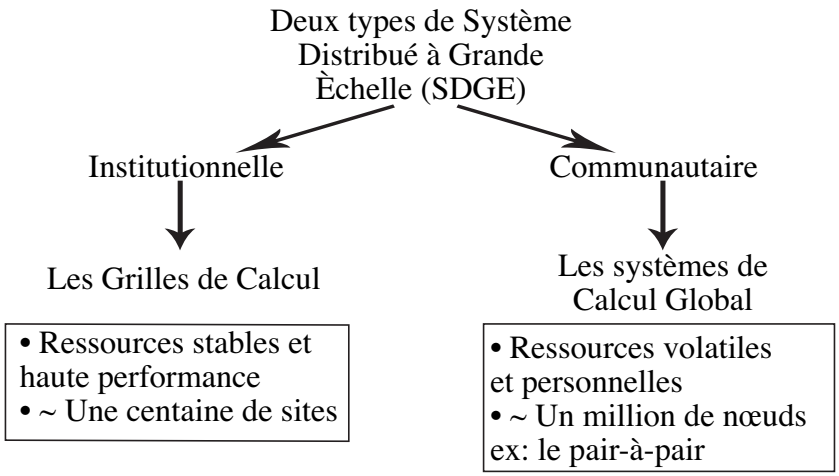


FIG. 2.1 – Taxonomie des Systèmes Distribués à Grande Échelle (*SDGE*).

[1] GGF(Global Grid Forum), URL : <http ://www.gridforum.org>
[2] Grid Computing Info Centre, URL : <http ://www.gridcomputing.com/>

Dans le premier cas, ces projets sont pilotés par des institutions possédant une manne financière suffisante pour financer complètement un système dédié au calcul scientifique. Les incarnations de ces systèmes sont les grands projets de *grille de calcul*. En parallèle à ces projets, la maturité d'Internet et la propagation des ordinateurs à moindre prix chez les particuliers favorisa une solution communautaire pour le calcul scientifique : *le calcul global* dont le pair-à-pair est une des illustrations. Contrairement au contexte institutionnel, l'approche communautaire repose sur des ressources déjà existantes, typiquement, des ordinateurs personnels et le volontariat de leurs propriétaires. Ces plate-formes possèdent des caractéristiques et contraintes distinctes d'un supercalculateur ou d'un cluster. Nous les résumons ici :

Hétérogénéité.
: Les ressources sont par nature hétérogènes. Les ressources de calcul possèdent des puissances, des architectures et des systèmes d'exploitations différents. Les ressources de calcul vont du simple CPU, au cluster ou supercalculateurs. Les ressources de stockage sont de simples disques durs, ou des systèmes de fichiers distribués, ou des systèmes de stockage de masse. Les réseaux diffèrent aussi de par leurs caractéristiques, telles que la latence ou la bande passante.

Extensibilité.
: Les *SDGE* ont souvent une propension à l'extensibilité, ou le passage à une très grande échelle, typiquement mondiale.

Selon le type de *SDGE*, ces contraintes varient. Nous allons voir dans la suite comment celles-ci différent.

2.4 Une solution institutionnelle : les grilles de calcul

Le terme de grille de calcul ou GRID (Globalisation des Ressources Informatiques et des Données) fut popularisé par Ian Foster et Carl Kesselman en 1999 dans le livre *"The Grid : BluePrint for a New Computing Infrastructure"* [25]. Ils définissent le concept de grille de calcul comme *"Une infrastructure matérielle et logicielle qui fournit un accès sûr, consistant, omniprésent et peu coûteux à des facilités de calcul haute performance"*. Le mot grille vient de l'analogie avec le réseau électrique (*Power Grid*) qui fournit une puissance aux utilisateurs de manière simple et transparente (voir figure 2.2).

FIG. 2.2 – Vision artistique d'une grille de calcul.

Le terme plus ancien de méta-ordinateur ou *méta-computer*, introduit en 1992 par Smarr et Catlett [72], est souvent utilisé pour désigner une grille de calcul.

Celui-ci véhicule l'idée d'un ordinateur virtuel offrant une puissance de calcul et de stockage inégalée.

Le terme méta-computing désignait d'ailleurs les premiers projets de grille tel que I-WAY [76] en 1995 qui visaient à connecter par un réseau haut débit des supercalculateurs d'architectures parallèles variées comme vectorielles, à mémoire distribuée ou partagée. Maintenant, une grille est aussi considérée comme l'extension de grands centres de calcul. Les ressources lourdes de plusieurs centres de calcul sont distribuées sur l'ensemble de la planète et sont connectées par des réseaux haute-performance. Le calcul sur grille ou *Grid Computing* les utilisent pour des applications scientifiques, typiquement de très gros calculs. Une grille se focalise donc sur le calcul haute performance ou « High Performance Computing » .

Le problème des procédures administratives imposées par les centres de calcul (inscription, réservation des ressources, facturation, etc.) sont souvent prépondérants dans ce contexte. Par exemple, il s'agit d'autoriser l'accès et l'utilisation des centres pour un ensemble d'utilisateurs.

Pour simplifier cette gestion les utilisateurs sont regroupés en organisation virtuelle [50]. Une organisation virtuelle (OV) est trans-institutionnelle et se définit par exemple par centre d'intérêt. Elle est assujettie à des règles et est soumise à des politiques d'accès et de partage des ressources. Des moyens sont mis en œuvre pour appliquer ces règles, notamment par l'authentification des utilisateurs, la propagation de leurs identités et les autorisations d'utilisation associées. L'utilisation des ressources repose sur une relation de confiance entre les utilisateurs et les propriétaires des ressources. Cette idée d'organisation virtuelle précise l'identification des consommateurs et des producteurs qui sont respectivement, les utilisateurs et les propriétaires des ressources. Ceci illustre encore la standardisation actuelle qui dégage une structure incrémentale commune à tous ces projets de grille de calcul.

2.4.1 Structure d'une grille de calcul

La hiérarchie structurale d'une grille est illustrée sur la figure 2.3. Bien que celle-ci diffère selon les projets, une architecture générale est importante pour expliquer certains concepts fondamentaux des grilles. Ce modèle en couche se compose de :

La couche bas-niveau (ou fabric*)* comporte la couche physique, les protocoles réseaux et les systèmes d'exploitation. Elle intègre aussi les systèmes de gestion de ressources locales comme un système de traitement par lot. Ces éléments sont indépendants de l'intergiciel de la grille mais s'intègrent dans celle-ci par des protocoles et APIs.

La couche intermédiaire est responsable de la communication entre les composants. La définition et l'implémentation de protocole « grille » se situe à ce niveau, notamment pour le passage des informations et statuts des composants. Elle intervient pour le transfert des données et la sécurité. Parmi les mesures de protection nous trouvons ceux habituellement employés dans les

réseaux, comme les pare-feux, ainsi que ceux spécifiques aux environnements répartis et aux grilles (autorisation et authentification).

La couche haut niveau fournit les intergiciels nécessaires à la gestion des ressources, la coordination de l'accès ou l'ordonnancement des tâches. De plus elle regroupe aussi tous les outils et paradigmes pouvant aider les développeurs à concevoir et écrire des applications. Nous y trouvons des librairies ou interfaces de programmation aidant les développeurs à « gridifier » leurs applications, c'est à dire à porter leurs applications sur la grille.

La couche application regroupe les applications de natures variées dont celles qui nous intéressent ici liées à l'analyse des données d'un détecteur de physique des particules.

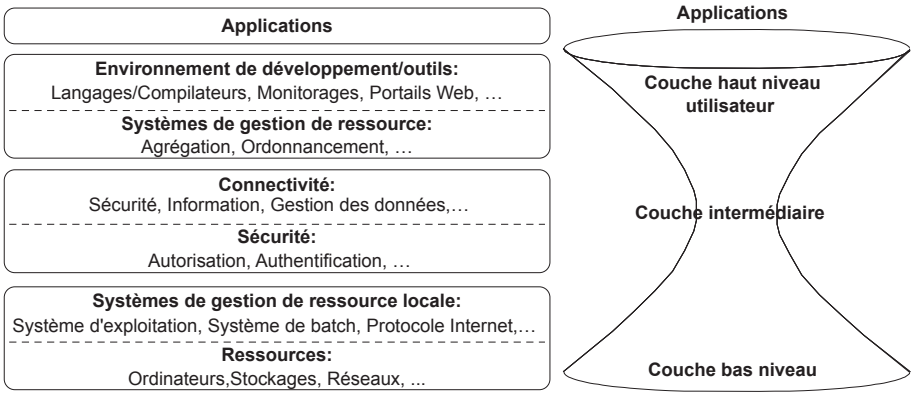


FIG. 2.3 – Le modèle en couche de l'architecture générale d'une grille de calcul.

Une grille de calcul est une plate-forme d'un usage difficile en raison de son hétérogénéité et de sa grande dynamicité. Plusieurs modèles de programmation sont possibles. La plupart d'entre eux étendent des solutions existantes pour les adapter aux contraintes de la grille. Nous présentons maintenant une sélection de projets visant à simplifier l'usage de la grille. Nous avons choisi ces intergiciels en raison de leur importance et de leur maturité aux niveaux académiques et industriels.

2.4.2 L'environnement de base : Globus

Le projet Globus [14] commencé en 1997 est un projet « open source » visant à créer les logiciels et les outils nécessaires pour la conception et la mise en œuvre de grilles de calcul. Globus est principalement développé aux Etats-Unis à l'Argonne National Laboratory par l'équipe de Ian Foster.

Le « Globus Toolkit » est formé d'un ensemble de composants et d'architectures modulaires qui permettent d'apporter des modifications et des améliorations d'une manière rapide et efficace. Globus est devenu le standard utilisé dans beaucoup de projets de grilles de calcul comme nous le verrons dans la section 2.5.

Le but de Globus est de donner une image homogène d'une grille pour ses utilisateurs. Le souhait est d'avoir un système exploitation étendu à une grille qui comprend les composants de base pour construire une grille de calcul. Globus favorise l'utilisation de composants existants, et propose aussi ces composants comme un système de traitement par lots autorisant la réservation de ressource.

L'intégration des composants de Globus avec le modèle grille générale de la figure 2.3 est illustrée sur la figure 2.4.

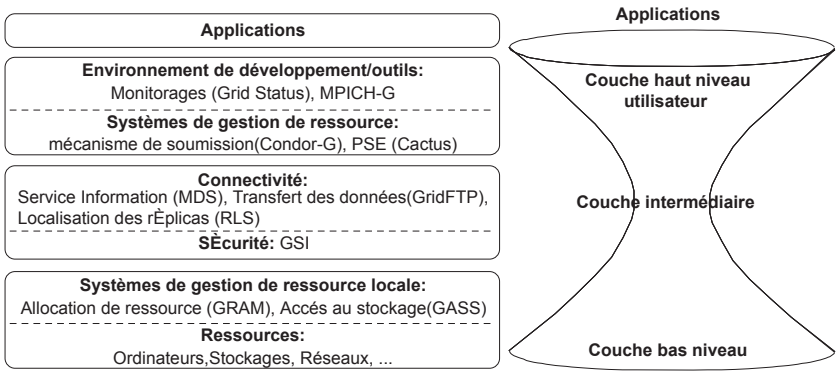


FIG. 2.4 – Le placement des composants Globus dans la hiérarchie grille.

Nous trouvons principalement des modules pour la gestion des ressources (GRAM) [21], l'accès aux données (GASS) [19,22], la gestion des communications (GridFTP) [17], la sécurité (GSI) [20], et l'information sur les ressources MDS [18]. Globus est devenu le standard « ipso facto » de beaucoup de projets actuels comme ceci sera détaillé plus loin dans le section 2.5.

2.4.3 Un environnement orienté objet : Legion

Legion [167] est un projet commencé en 1994 à l'Université de Virginie aux Etats-Unis par l'équipe de Andrew Grimshaw. Legion constitue l'un des premiers projets traitant des grilles de calcul. Il fournit une architecture spécifique afin de créer une seule machine virtuelle avec des ressources géographiquement distribuées et hétérogènes. L'approche adoptée est basée sur les principes du paradigme orienté objet. Tous les concepts sont exploités comme l'encapsulation, l'héritage et la réutilisation du code. Chaque objet est une instance de classe qui communique avec les autres objets par invocations de méthodes. Ils possèdent un ensemble de méthodes qui permettent de dialoguer avec d'autres objets de façon asynchrone. Legion fournit des interfaces pour les interactions entre les composants de base.

Parmi les classes de base, nous avons tous les gestionnaires qui contrôlent les instances des objets en les créant, les activant ou les désactivant. Les objets de ces classes ont la responsabilité de délivrer des détails et informations aux autres objets qui souhaitent communiquer. Un espace de nommage unique est défini pour

tous les objets du système. De plus, les composants physiques et logiciels sont représentés comme des objets. Les classes utilisées pour décrire les différentes entités s'obtiennent par héritage et les fonctionnalités sont surchargeables en accord avec les besoins des développeurs.

La hiérarchie basique des classes Legion est montrée dans la figure 2.5. Un hôte représente un nœud de la grille. Un dépôt ou réserve est un espace de stockage persistant par exemple un système de fichiers ou une base de données.

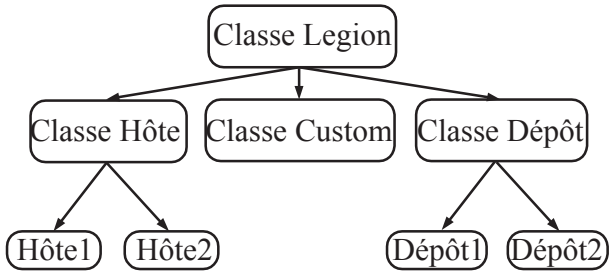


FIG. 2.5 – Hiérarchie des objets de base pour décrire un système Legion.

Des classes fournissent des fonctionnalités de plus haut niveau comme l'ordonnancement de tâches. Cette fonctionnalité utilise l'ordonnanceur, qui sert de base d'information pour les objets et les ressources du système. La localisation et l'accès aux données sont aussi proposés. Le modèle de sécurité implémenté protège les objets et les communications entre eux par la technique des certificats. Ces derniers décrivent les droits attribués par le propriétaire de l'objet. La validité et l'authenticité de ce certificat sont vérifiés. Les communications entre les objets sont cryptées et décryptées par des mécanismes reposant sur des paires de clef publique.

2.4.4 Un environnement haut niveau : NetSolve

NetSolve [23] est un exemple d'outils de grille centrés sur les applications. Il élargit le concept de PSE (« Problem Solving Environment ») [81] aux systèmes répartis. Un PSE est un environnement informatique résolvant de manière simple et transparent une classe de problème. Celui-ci inclut des méthodes et un langage masquant la complexité matérielle ou logicielle sous-jacente. Un exemple de PSE est Matlab [48] pour le calcul numérique et la visualisation. En constatant que le paradigme client-serveur est applicable aux PSE, NetSolve comme Ninf [24] ou DIET [178] l'a appliqué en utilisant l'approche d'invocation de méthode à distance RPC (« Remote Procedure Call »). Ces environnements s'appellent des NES (« Network Enabled Server »). Le nom NES signifie la mise à disposition par le réseau de serveurs pouvant résoudre des problèmes. Un NES est un environnement de calcul à distance dans lequel un utilisateur soumet un problème qu'il désire résoudre sur un serveur distant. Ce serveur solutionne le problème et renvoie un résultat.

Une standardisation des NES au sein du GGF(« Global Grid Forum ») a donné naissance au GridRPC.

Il existe trois composants principaux dans NetSolve : 1) l'agent, 2) le serveur et 3) le client, comme le montre la figure 2.6.

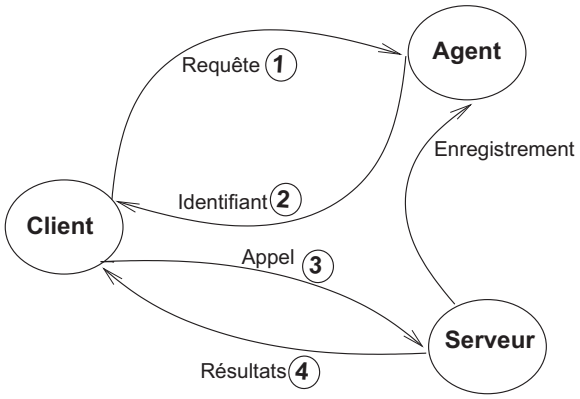


FIG. 2.6 – Relation et flot de communication entre le client, l'agent et le serveur dans un système Netsolve.

L'utilisateur accède au système via un client qui se compose d'un ensemble d'APIs. Celles-ci s'utilisent pour décrire les détails de la requête. Elle comporte notamment des dépendances à des ressources logicielles. La requête est envoyée à un agent (1). Celui-ci contient une base de données de toutes les ressources disponibles (serveurs, applications). Il détermine celui qui est le plus approprié (2). Chaque serveur transmet à l'agent des informations sur son état dynamique (occupation bande passante, charge processeur, etc.). L'agent monitore l'utilisation des serveurs et leurs taux de pannes pour améliorer l'ordonnancement. Le client contacte directement le serveur sélectionné et lui transmet son problème (3). Le serveur active les logiciels et routines locales comme un service NetSolve par l'intermédiaire d'un langage de définition de tâches et renvoie le résultat (4).

Le protocole standard de communication et l'authentification à distance se gère par jeton Kerberos.

2.5 Des exemples de grilles pour le LHC

Après avoir vu différents environnements de grilles, nous présentons ici les initiatives proposant des solutions pour le traitement des besoins du *LHC*. Nous présentons ici quelques-uns des projets majeurs. Ces projets sont sous deux formes : *une grille de recherche* et *une grille de production*.

Une grille de recherche est motivée par des travaux de recherche et d'évaluations comme le projet *Grid 5000* [182]. Les travaux sont au stade fondamental. Ils nécessitent une première phase de validation expérimentale engendrant éventuellement

des déploiements sur une plate-forme de production. Une plate-forme de recherche n'offre aucune garantie de service dans la mesure ou l'infrastructure est sans cesse modifiée pour des expérimentations.

Contrairement aux grilles de recherche, les grilles de production tel *LCG* [3] sont des plate-formes applicatives devant fournir des services de manière stable, sure, et extensible. Le mode de fonctionnement d'une grille de production est typiquement celui d'un grand centre de calcul.

Ces solutions sont principalement évaluées lors des « Data Challenges » pratiqués par les expériences du *LHC*. Ces *Data Challenges* sont de véritables démonstrations de force destinés à préparer la mise en route du LHC. Ils ont pour but de valider le modèle d'organisation et d'utilisation des données qui seront produites ainsi que la chaîne de logiciels nécessaires pour l'exploitation de ces données.

2.5.1 Les grilles de première génération

Trois grands projets innovateurs de grille ont été développés pour les besoins de la physique des particules du LHC. Ces projets sont le projet européen DataGRID [49], le projet nord-européen NorduGrid et le projet nord-américain GriPhyn [69].

DataGRID a été établi par le CERN avec la volonté d'étendre le champ d'applications à d'autres disciplines comme l'observation satellite de la terre ou la bioinformatique. L'enjeu de ce projet était de créer un réseau de recherche pour le développement de nouvelles technologies et de démontrer les capacités d'une grille à grande échelle. Le projet s'est intéressé surtout au développement de l'environnement pour l'analyse de données physiques. Un des buts de ce projet était de fournir la grille de production adaptée aux besoins du *LHC*. Il a été la base intergiciel du projet LCG [39] qui est la grille d'exploitation du *LHC*.

GriPhyn [69] est un sous projet avec PPDG [67] et IVDGL [77] du projet GRID3 [66,68]. GRID3 est un projet américain de grille multi-organisation virtuelle et multi-applicatif.

NorduGrid [63] est un projet né pour répondre au besoin du Data Challenge de l'une des expériences du *LHC*, ATLAS. Son but est de fédérer une grille des pays nordiques comprenant le Danemark, la Norvège, la Finlande et la Suède. La philosophie du projet était d'être tout d'abord un banc d'essais minimaliste fonctionnel, puis de s'enrichir de fonctionnalités nouvelles. Tout ceci se concentre sur l'absence de goulots d'étranglement et le respect des propriétaires des ressources.

Ces trois projets reposent sur le toolkit Globus avec des extensions souvent propres au projet. Le projet Grid3 et DataGRID utilisent conjointement le « Virtual Data Toolkit » (VDT) [70]. Celui-ci comprend le toolkit Globus et Condor incluant l'interface de soumission Condor-G [82] aux ressources Globus. Ces trois projets

[3] LCG, URL : <http ://lcg.web.cern.ch/LCG/>

fournissent les mêmes fonctionnalités. Ils utilisent tout le système d'information MDS de Globus pour fournir les informations statiques et dynamiques de la plateforme. L'authentification se base sur GSI et la propagation des autorisations ainsi que la gestion des *OV* par le système VOMS(Virtual Organisation Management System) [84]. Ces services sont déployés et maintenus par l'équipe support et les administrateurs de site.

Dans le cadre de DataGRID, l'utilisateur soumet une tâche au *Resource Broker* ou méta-ordonnanceur. Celui-ci est le composant qui s'occupe de choisir la ressource de calcul ou s'exécutera la tâche. Il joue donc le rôle de l'ordonnanceur du système et utilise pour faire la correspondance tâches-ressources de calcul le système d'information. Ce système d'informations permet de localiser et connaître l'état des différentes ressources disponibles. Une fois le choix de la ressource effectué, un courtier ou *Resource Broker* transmet la tâche à la ressource sélectionnée via un service de soumission. La tâche est ensuite mise en attente au niveau de la ressource pour être exécutée. Dès que la tâche est terminée, le *Resource Broker* est notifié et informe à son tour l'utilisateur. Celui-ci peut alors récupérer les résultats. Pour contourner le goulot d'étranglement causé par l'unicité de l'ordonnanceur, l'idée à été de dupliquer ce composant par *OV*.

Dans le cas de NorduGrid, cette fonctionnalité de gestion de la charge a été dupliquée au niveau de l'utilisateur. Un utilisateur est associé à un courtier qui en fonction du système d'information choisit une ressource.

Pour Grid3, chaque organisation virtuelle construit sa solution pour s'agréger et utiliser les ressources. Ainsi, les deux expériences CMS et ATLAS qui ont utilisé Grid3 lors du « Data Challenge 2004 », on définit leurs services de gestion des ressources respectivement IMPALA, CAPONE [171] en s'appuyant sur le VDT.

2.5.2 Vers des grilles de nouvelle génération

Suite à l'inspiration de virtualisation des services, les projets de grille à grande échelle ont évolué vers une approche orienté service. L'intérêt de ce mode de conception va croissant comme peut en témoigner la proposition de standardisation de ARDA [65] et *Open Grid Services Architectures* (OGSA) [55] faites par le consortium Globus. Cette dernière spécification définit le service grille qui est l'interface de chaque entité d'une grille. Cette spécification supporte la création et la destruction d'instance de service, la gestion de leurs durées de vie, la recherche, la découverte de fonctionnalité et la notification. Cet environnement respecte aussi la sécurité, notamment par l'authentification et l'autorisation.

Les premières expériences à grande échelle et les premières standardisations ont permis d'identifier les principaux services d'une grille de calcul. Nous les retrouvons dans les successeurs des projets DataGrid, NorduGrid et Grid3. Ainsi, le projet EGEE [64, 71] (« Enabling Grids for E-science in Europe ») est le projet européen en cours, successeur de DataGrid. Le projet NorduGrid évolue lui aussi dans une

approche de virtualisation des services, notamment dans le projet ARC(Advanced Ressource Connector) [83]. De même le projet Grid3 se poursuit maintenant avec le projet OSG (Open Science Grid) [38] qui applique aussi une architecture orientée service.

2.6 L'élan communautaire : le calcul global

Le calcul global désigne les systèmes connectant des ordinateurs individuels par Internet. Le paradigme qui s'applique pour ces systèmes est la récupération de cycles de processeurs inutilisés à travers la toile. En fait, cette idée était déjà appliquée au niveau d'un réseau local et connaît déjà plusieurs implémentations majeures. Parmi ces propositions nous pouvons citer Condor [46] . Le « vol de cycles » consiste à récupérer les ressources inutilisées de nœuds de calcul distribués d'un réseau pour un but commun, qui peut être le stockage, le partage d'information et le calcul. Nous décrivons ici les caractéristiques et qualités des systèmes de calcul global ainsi que notre classification des principales implémentations de ces systèmes.

2.6.1 Les caractéristiques et qualités du calcul global

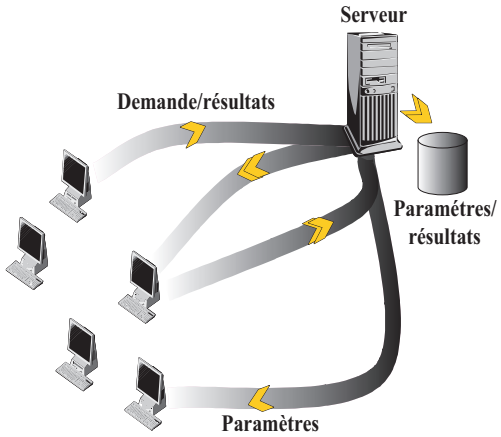
Le calcul global est la transposition du paradigme de vol de cycles sur Internet. Ce passage à une très grande échelle, oblige de revisiter cette approche pour l'adapter au contexte d'Internet. Celui-ci introduit de nouvelles problématiques. La première est l'extrême volatilité des ressources. Les ressources sont des ordinateurs individuels qui se connectent et qui quittent le système à tout instant. Une étude mesure un changement de 30% de nœuds d'un instant à l'autre dans le réseau Gnutella [191]. De plus, les latences et débits du réseau varient beaucoup. La taille du système est énorme. Potentiellement le nombre de systèmes connectés à Internet dépasse largement le million [47]. En dépit du fait que les ressources sont des ordinateurs personnels, il existe une hétérogénéité au niveau des systèmes d'exploitation, des configurations matérielles ou de la puissance des processeurs. Une autre problématique est l'absence de contrôle global des ressources.

Un système de calcul global doit être capable de s'adapter aux changements fréquents de configuration. Ensuite, l'extensibilité du système doit être garantie, ainsi qu'un fort niveau de tolérance aux pannes. Le système doit aussi pouvoir connecter toutes les machines même derrière un pare-feu. Cette contrainte de connectivité passe par un déploiement simple et silencieux du système pour les participants. Le propriétaire d'une ressource doit pouvoir contrôler l'utilisation et la contribution de sa ressource. Il est souvent nécessaire de protéger la machine source de tous codes pernicieux.

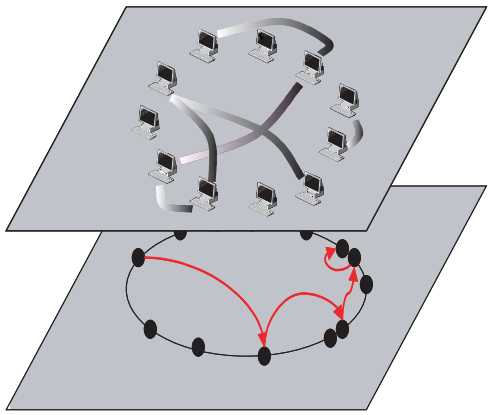
Nous verrons par la suite, comment ces systèmes de calcul global répondent à ces contraintes et exigences de qualités.

2.6.2 Classification des systèmes de calcul global

La figure 2.7(a) décrit les principes et les acteurs du calcul global. Ce modèle reste simplifié mais permet de bien comprendre les enjeux.



(a) Le principe maître-esclave. Les nœuds sont clients.



(b) Le principe pair-à-pair. Les nœuds sont clients et serveurs et déploient un réseau virtuel privé.

FIG. 2.7 – Les modèles du calcul global.

Le scénario habituel est le suivant : dès que le nœud détecte que son utilisateur est inactif, il signale à un serveur central qu'il est disponible pour effectuer des calculs sous son contrôle. Les résultats seront rapatriés en fin de calcul. Le modèle de programmation est le modèle maître-esclave [86]. Ceci s'opère le plus souvent de manière complètement transparente pour l'utilisateur habituel de ce nœud.

Les projets de calcul global se classifient comme suit [40] : Le calcul basé sur Internet ou *web computing*, les projets mono-application basés sur le modèle de calcul global, et les systèmes pair-à-pair.

Le calcul basé sur Internet est apparu dans les années 90. Les plus connues sont Jet [31], Charlotte [32], PopCorn [27], SuperWeb [30], ParaWeb [29], Javelin [28]. La plupart sont basés sur le langage Java qui se prête particulièrement bien à ce contexte. Il répond en effet aux contraintes de l'hétérogénéité car il offre un langage intermédiaire (le *bytecode*), exécutable et portable sur toutes les plate-formes. De plus la large disponibilité des machines virtuelles Java (*JVM*) sur les navigateurs facilite beaucoup le déploiement du système. En effet, un utilisateur pour participer visite une adresse Web et télécharge une applet. Le langage Java protège aussi la machine hôte en confinant l'exécution pour éviter toute intrusion malveillante (technique de confinement d'exécution ou « sandboxing » [168, 172]). Si ce modèle de sécurité est de prime abord efficace il est aussi extrêmement restrictif pour les applications. De

plus, les machines virtuelles sont souvent des facteurs de ralentissement non négligeables pour les applications. D'autre part, l'infrastructure repose sur les navigateurs dont l'arrêt et le démarrage sont encore plus fréquents que ceux d'une machine.

les projets mono-application. Les premières applications étaient la cryptographie pour casser une clef RSA [35] ou le calcul de décimale de Pi [34]. Ces premiers projets constituent les premières bases du calcul global mais le projet référence dans ce domaine pour son coté peu commun est le projet SETI@home [47] lancé en 1999 par l'Université de Californie Berkeley. Ce projet aurait utilisé selon un recensement trois millions de machines avec des pics de 200.000 machines participantes en même temps. La puissance de calcul est estimée à 30 Tera-flops. Ce chiffre n'est pas directement comparable à la puissance des supercalculateurs du top 500 [62] car celui-ci s'appuie sur la métrique du Giga-flops obtenue avec le jeu d'essai *LinPack* [177] qui est un calcul fortement parallèle. Cette expérience constitue surtout une preuve d'un concept pour atteindre une puissance inégalée de manière peu coûteuse. Elle est le premier essai à grande échelle de contribution volontaire. Toutefois ces projets ont permis de déceler des vulnérabilités. Premièrement, des problèmes interviennent sur la validité des résultats. Dans le cas de SETI@home, des utilisateurs ont modifiés le code de certaines routines engendrant des résultats corrompus. Pour lutter contre ces erreurs difficilement décelables SETI@home a introduit une redondance dans les calculs (un facteur 17). Ceci se justifie par le grand nombre de participants. D'autre part le serveur central est « un point de faille unique » (*single point of failure*). Son interruption d'activité pénalise lourdement le système. Les systèmes pair-à-pair résolvent simplement ce goulot d'étranglement.

Les systèmes pair-à-pair.

Un autre paradigme de calcul global a récemment suscité l'intérêt de la communauté scientifique : le calcul pair-à-pair, créé par des jeunes étudiants pour échanger de la musique. Cette approche généralise les idées du calcul global. Le modèle complémente le modèle classique client/serveur, aujourd'hui à la base du calcul global, en symétrisant la relation des nœuds qui interagissent, comme le montre la figure 2.7(b). La relation client-serveur n'est plus associée aux nœuds, mais aux transactions : chaque nœud peut être client dans une transaction et serveur dans une autre. Un pair est l'unité structurelle de base de tout réseau pair-à-pair. Un pair peut être toute ressource capable de se connecter au réseau et de fournir un service. Grâce à leur architecture horizontale et décentralisée, ces systèmes gèrent de façon rationnelle et dynamique les ressources disponibles en mémoire et en bande passante. Pour l'instant, cette technique a essentiellement été utilisée pour le partage de fichiers entre millions d'utilisateurs, connectés de manière intermittente et imprévi-

sible. Parmi ces projets, citons Napster [57], Freenet [61] et Kazaa [60]. Ils ont la particularité de développer leur propre service souvent sous forme de VPN(« Virtual Private Network »). Les VPN sont des réseaux en surcouche des couches standard comme TCP/IP mais qui fournissent des fonctionnalités supplémentaires comme la nomination unique d'une ressource, la découverte d'une ressource à l'aide de table de hachage distribuées (Distributed Hash Tables- DHT) ou des fonctionnalités de routage comme CHORD [89].

2.6.3 Vers des environnements génériques

Une analyse des systèmes de calcul global existants permet de constater que ces systèmes partagent un grand nombre de fonctionnalités communes. Malheureusement, ces fonctionnalités sont souvent implantées par des mécanismes propriétaires, fermés. Afin de mettre en œuvre un nouveau système, il est nécessaire de développer un grand nombre de composants logiciels qui ont déjà été implémentés précédemment.

En partant de ce constat, des environnements se proposent d'identifier plusieurs concepts et mécanismes de base propre au calcul global. Ces environnements sont développés dans une approche « Open Source ». Ces projets proposent plusieurs protocoles permettant de gérer un certain nombre de fonctionnalités communes, comme la découverte de ressources, l'invocation de services sur le réseau, la communication inter-pair, la sécurité, etc. Ces protocoles peuvent servir de briques de base pour la mise en œuvre de services. Parmi ces environnements, nous pouvons citer la solution d'entreprise Entropia [85], JXTA [16], Berkeley Open Infrastructure (BOINC) [80] et XtremWeb [52]. Ces projets sont désignés parfois sous le vocable de « Desktop Grid ».

JXTA met les fondations d'une infrastructure générique de type intergiciel pour des applications pair-à-pair. Il se présente comme une extension à l'environnement d'exécution de Java. BOINC est la continuité du projet SETI@home. Il est un support a l'exécution d'applications natives et partage une approche semblable à XtremWeb. Contrairement à SETI@home ou la participation d'utilisateur était restreinte ces plate-formes permettent : la distribution d'applications natives sans modifications ; la mise à jour des applications automatiquement ; l'authentification entre les serveurs ; et la tolérance aux pannes des serveurs par redondance.Toutefois, ces infrastructures de service standard ne possèdent pas d'interopérabilité entre elles et ne sont pas encore suffisamment répandues pour pouvoir conclure. Elles représentent néanmoins, un pas vers la virtualisation comme les systèmes de grilles de calcul.

2.7 Les grilles de calcul et le calcul global vers un système unique ?

Les grilles de calcul et les systèmes de calcul global peuvent-ils être un même système ? Selon la classification de Ian Foster [26], nous désignons par grille de calcul un système distribué dédié au calcul qui répond aux exigences suivantes :

1. Les ressources ne sont pas gérées de manière centralisée et appartiennent à divers domaines administratifs ;
2. Une grille de calcul utilise des protocoles et interfaces standards, ouverts et génériques ;
3. Une grille de calcul délivre une qualité de service non triviale.

En comparaison, nous avons les systèmes de calcul global qui :

1. Utilisent des ressources individuelles ;
2. Implémentent des protocoles « Ad Hoc » et non génériques ;
3. Ne fournissent aucune garantie pour la qualité de service.

Ces définitions excluent clairement tout rapprochement. Pourtant, nous pensons fortement que les techniques du calcul global appliquées au contexte de grilles de calcul est prometteur. Pour bien comprendre à quoi pourrait correspondre ce système nous allons faire une comparaison des deux approches.

2.7.1 Comparaison des deux approches

Des travaux récents [58, 74] constatent des similarités entre les systèmes de calcul global et les grilles de calcul. Nous comparons ici ces deux modèles selon certains points donnés dans le tableau 2.1. La première différence est dans le mode de conception de ces systèmes. Les grilles de calcul sont conçues avec *une vue administrateur* tandis que les systèmes de calcul global suivent un développement selon *une vue utilisateur et client*. Cette divergence de point de vue est fondamentale et explique les propriétés respectives. Les communautés sont très distinctes, l'une scientifique et industrielle tandis que l'autre est strictement basée sur le volontariat et le libre échange.

Les systèmes de grilles de calcul et de calcul global se différencient ensuite par le nombre de ressources participantes. Celles-ci sont est très importantes pour le calcul global. Nous l'estimons à plusieurs centaines de milliers de nœuds contre quelques dizaines de milliers pour une grille. En général, les grilles interconnectent des ressources plus puissantes, diverses et mieux connectées que des ressources de calcul global.

Les ressources grilles sont des clusters ou sites, des systèmes de stockage distribué ou de masse. Ils sont donc très hétérogènes. En opposition, les ressources du calcul global sont des ordinateurs individuels particuliers dont la différence de puissance est donnée par le projet SETI@home : les ordinateurs personnels sont en

moyenne 30% plus lents que des ordinateurs à usage professionnel. Ces ressources sont rarement parallèles et l'hétérogénéité se constate essentiellement au niveau du système d'exploitation.

La réussite d'un projet de calcul global est intimement lié à sa facilité d'utilisation et de connectivité des ressources. Le déploiement est donc volontairement simple et avec peu de dépendance externe. Il reste toujours au niveau utilisateur. Les services des utilisateurs sont logiquement limités. En opposition le déploiement des grilles est lourd et intrusif. Il peut aller jusqu'à l'installation d'un nouveau système d'exploitation et requiert les droits administrateurs. Les ressources sont sous le contrôle d'administrateurs qui les régissent de manière coordonnée par des politiques prédéfinies. Cette administration explicite augmente la stabilité des ressources et la qualité de service résultante. Elle facilite la mise à jour des logiciels de la couche de bas niveau. Mais, la contrepartie est le coût associé à ces ressources humaines. Les ressources d'un système de calcul global sont volatiles. Le départ ou l'arrivée d'un utilisateur est fréquent et nécessite une grande auto-adaptation. Les ressources sont faiblement couplées car les réseaux sont des LAN, Intranet ou Internet. Les grilles de calcul bénéficient d'une meilleure stabilité mais restent quand même sujettes à des pannes, comme des pannes de courant ou des problèmes de configuration.

Une grille de calcul doit servir plusieurs communautés et différentes applications. Un système de calcul global ne supporte souvent qu'une seule application et autorise rarement la soumission de tâche. Par conséquent, il est moins flexible pour intégrer de nouvelles applications. Dans le cas du calcul global, le système est souvent dédié pour un type d'application précise, qui peut être le partage et le stockage de données ou des calculs scientifiques. Dans ce dernier cas, la granularité des applications est petite dans un système de calcul global avec de la préemption, tandis qu'elle peut être très grosse sur une grille où le champ des applications est plus large (GridRPC, programmation par message, etc.).

L'ordonnancement des applications suit des stratégies différentes. Elle est en générale active (« push ») pour les grilles de calcul, ce qui veut dire que l'ordonnanceur collecte le maximum d'informations sur le système avant de choisir une ressource pour une tâche. Tandis que dans un modèle passif (« pull »), la ressource en fonction de son état local demande une tâche à l'ordonnanceur. Nous traitons par la suite de manière plus approfondie ces concepts en section 3.4.

L'échelle pour les données est plus importante pour les grilles. Par exemple l'expérience de physique D0 [58] déplace plus de 4 Tera octets par jour, ce qui est deux fois plus que les systèmes de partage de fichiers pair-à-pair. Les grilles sont plus orientées sur les masses de données.

Dans les grilles de calcul les contraintes de sécurités sont très fortes. Les utilisateurs sont authentifiés et l'identité propagée pour une meilleure tracabilité et comptage d'utilisation des ressources. L'utilisation des ressources se base aussi sur

une relation de confiance. Dans le domaine de calcul global, la relation de confiance est inexistante et l'anonymat des utilisateurs fréquent.

Attributs	**Grille de calcul**	**Calcul global**
Mode de conception	administrateur	*utilisateur et client*
Taille	<100 sites, 10000 nœuds	*1 million de nœuds*
Ressource de calcul	*Cluster, supercalculateur*	Ordinateur individuel
Ressource de stockage	*NFS/AFS, mass storage*	simples disques durs
Caractéristique	*stable*	volatile
déploiement	intrusif	*léger*
Applications	*multiples et génériques*	dédiées
Granularité	*Gros grain*	grain fin (préemption)
Ordonnancement	Stratégie active (« push »)	*Stratégie passive (« pull »)*
Soumission directe	*oui*	non (quelques exeptions)
Organisation	plusieurs communautés	*une communauté*
Sécurité	*importante*	faible
Authentification	*identification*	anonyme

TAB. 2.1 – Comparaison grille de calcul et calcul global.

En dépit de tout ceci, nous observons une convergence, notamment, au niveau des données. Des projets utilisés par des grilles de calcul appliquent des modèles pair-à-pair pour la gestion des données [88] dans les grilles institutionnelles. Nous pouvons aussi citer le projet JuxMem [186]. Ces projets se focalisent sur les données et les ressources de stockage. La convergence technologique des grilles de calcul et des systèmes de calcul global se voit confortée par la volonté commune de standardiser ces approches. En particulier, ceci se manifeste par l'utilisation des appels de procédure à distance, comme la propagation de protocoles SOAP [87] ou XML-RPC [37] dans le modèle client-serveur. La proposition OGSI va aussi dans ce sens. Un autre constat rapproche les grilles de calcul et le calcul global. Lors de l'utilisation d'une grille, une organisation virtuelle développe souvent sa propre solution logicielle pour agréger le système. Une organisation virtuelle utilise alors un mode de conception utilisateur. Ces systèmes implémentent de nouvelles solutions et fonctionnalités introduisant des clients et serveurs multiplexés, multiprotocoles voir des systèmes d'agents. Ces nouvelles définitions de fonctionnalité se matérialisent par des systèmes qui déploient leurs propres infrastructures en surcouche des infrastructures logicielles et matérielles existantes. L'approche *gliding in* [73] de Condor pour le déploiement d'un cluster Condor sur d'autres systèmes est très proche de la notion de réseaux virtuels privés dans le calcul global.

2.8 Conclusion

Nous considérons principalement deux Systèmes Distribués à Grande échelle, les grilles de calcul et le calcul global. En regard des projets passés et présent de grille de calcul ainsi que de calcul global, nous caractérisons que l'infrastructure d'une grille est lourde, intrusive et peu flexible. Tandis que l'infrastructure d'un système globale est flexible, légère et dynamique. Pourtant un système de calcul global se limite à peu de types d'application, de ressources et il est non sécurisé, contrairement à une grille de calcul. En constatant ces complémentarités, un système conjuguant les deux aspects serait un atout majeur pour le calcul scientifique. Nous avons mis en italique dans le tableau 2.1 les qualités qui nous semblent intéressantes. Ainsi un système extensible, générique et défini au niveau utilisateur pouvant gérer tout type de ressource avec toutes les contraintes de sécurité pour une seule organisation serait pertinent. Dans le chapitre suivant, nous présentons un système virtuel, dynamique et flexible qui répond à ces caractéristiques et constitue le principal apport de cette thèse : *DIRAC*.

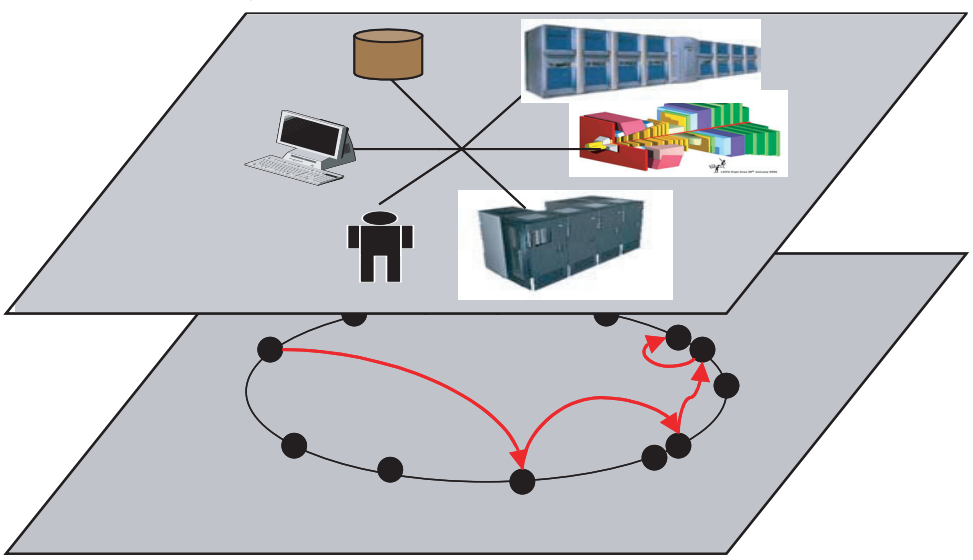


FIG. 2.8 – Grille de calcul et système de calcul global vers un même système ?

CHAPITRE 3

DIRAC - UN SYSTÈME VIRTUEL, DYNAMIQUE ET EXTENSIBLE

Résumé du chapitre :
Le système DIRAC *(Distributed Infrastructure with Remote Agent Control) est un environnement distribué à grande échelle. C'est un système léger, extensible et robuste. Il repose sur une* architecture orientée service *et fournit ainsi un ensemble de services pour un* SDGE *générique. Il s'appuie sur une organisation Service/Agent et illustre la convergence des systèmes de calcul global et des systèmes de grilles de calcul. Il applique pour la régulation de la charge un paradigme « pull » typique d'un système de calcul global avec des ressources de grille. De plus,* DIRAC *uniformise le système masquant les implémentations sous-jacentes, dans le but d'agréger le plus grand nombre de ressources avec une simplicité de déploiement, de maintenance et d'administration pour une seule communauté.*

3.1 Le contexte

En dépit de la coordination des quatre expériences du *LHC* (Large Hadron Collider) pour la mise en place de la grille de calcul de production *LCG* [39] (LHC Computing Project), il restait toujours au sein de l'expérience *LHCb* des besoins non résolus comme la non-incorporation des ressources hors du domaine *LCG*. Des lacunes fonctionnelles apparurent aussi au cours de l'évaluation du projet *EDG* [3], précurseur au projet *LCG*, qui rendaient difficile voire impossible la production de données de Monte-carlo dans un contexte de régime permanent et saturé (« high throughput computing »). De plus, il existe plusieurs solutions de grilles de calcul adaptées au *LHC* comme pour l'expérience Atlas [173]. Les évolutions et les changements de ces solutions soulèvent la problématique de fournir une interface pérenne, facilitant l'utilisation de ces moyens de calcul, pour les dix prochaines années d'exploitation de *LHCb*.

Le groupe *LHCb* du Centre de Physique des Particules de Marseille proposa alors le projet *DIRAC* (Distributed Infrastructure with Remote Agent Control) [2] dirigé par Andreï Tsaregorodtsev. En 2002, une première version [10] basée sur un modèle Agent/Service agrégea près de 500 processeurs pour la production distribuée de données. Le succès de ce système valida le concept d'agents distribués demandant des tâches à un service central sans état. Cependant, ce concept était prouvé pour une échelle et pour une classe d'applications spécifiques. Une nouvelle version de *DIRAC* fut donc écrite pour généraliser ces principes.

Nous décrivons dans la suite de ce chapitre les paradigmes appliqués au sein de *DIRAC* en section 3.2 et l'organisation générale en section 3.3. Nous détaillerons

ensuite son architecture orientée service ainsi qu'un état de l'art sur les systèmes de gestion de ressources (section 3.4). Cet état de l'art explique nos choix de conception du service de gestion de ressource de *DIRAC* que nous détaillons ensuite en section 3.5. Par la suite, nous montrerons comment ce service de gestion de ressources coopère avec les autres services comme le service de gestion de données (section 3.6). Les choix d'implémentation et les points clefs de *DIRAC* seront expliqués en section 3.8 pour ensuite détailler pourquoi *DIRAC* amène une nouvelle vision des systèmes distribués à grande échelle dans la section 3.9, puis nous exposerons nos perspectives et conclusions en section 3.10.

3.2 Structure générale de conception

L'objectif de *DIRAC* est de conjuguer les aspects de grille de calcul et du calcul global pour avoir un système léger, flexible et extensible.

La première qualité souhaitée est la généricité du système au niveau applicatif. Bien que *DIRAC* soit directement rattaché à une expérience de haute énergie *LHCb*, une volonté très forte de séparer le domaine applicatif de l'infrastructure de *DIRAC* a toujours été présente. Le déploiement des logiciels applicatifs doit être assuré de manière intelligente et non intrusive. Les principales applications considérées pour *DIRAC* sont celles de *LHCb* décrites dans le chapitre premier à savoir la production de données Monte-carlo, la présélection et l'analyse de données. Le champ de ces applications est large et représentatif d'un système distribué à grande échelle.

La communauté *LHCb* représente les utilisateurs finaux du système. Cette caractéristique et la constatation que la taille de l'organisation virtuelle *LHCb* variait peu prône une approche communautaire. En opposition à une approche généraliste qui vise à résoudre les besoins de plusieurs communautés scientifiques, de différents domaines à travers un seul système. Le partage des ressources est similaire aux politiques de partage mise en œuvre dans les grands centres de calculs. *DIRAC* s'occupe de résoudre les problèmes d'une organisation virtuelle et doit supporter seulement les applications de celle-ci, de l'ordonnancement à la distribution des logiciels. Un des autres avantages non négligeable d'avoir des utilisateurs quasi-immédiatement du système est la compréhension exacte de leurs besoins, des cas de figure d'utilisation et des retours d'utilisations contribuant à la salubrité du système.

Les ressources sont principalement celles des grilles usuelles comme les fermes de calcul avec un système de traitement par lot et les systèmes de stockage de masse mais nous prenons aussi en compte l'utilisation de simple nœud et l'interopérabilité entre systèmes de grille. L'utilisation de ces ressources s'effectue dans un contexte de confiance et de sécurité avec les administrateurs de site. Cette sécurité se base essentiellement sur l'authentification et l'autorisation.

Les ressources institutionnelles sont à priori stable et permettent de fournir ro-

bustesse et qualité de service. Néanmoins la réalité est souvent autre. Ces ressources ont une certaine dynamicité de par leur échelle mondiale. En effet, certains sites sont inutilisables pendant un certains temps suite à des mises à jour, des pannes réseaux et électriques. Des outils de diagnostics sont indispensables pour ces environnements. Il est nécessaire d'avoir un système stable, auto-adaptable et tolérant aux pannes.

Ces deux dernières qualités sont relatives au calcul global. L'extensibilité ou le passage à l'échelle du calcul global est aussi une caractéristique importante que nous désirons introduire dans *DIRAC* car le système final devra contrôler plus d'une centaine de sites représentant plus de 20.000 nœuds (voir 1.5.4).

La charte de programmation bien que restrictive est de rester au niveau utilisateur pour les agents et d'adopter une vue strictement utilisateur des ressources. La méthode de développement choisi est incrémentale. C'est-à-dire que nous avons implémenté d'abord les choses simples et fonctionnelles avant de gagner en complexité. Cette méthodologie se justifie surtout par l'aspect innovateur de ce projet et par notre expérience dans ce domaine où beaucoup de projets se montrent souvent trop complets et ambitieux au moment de leur démarrage. Nous nous appuyons de plus sur l'existant. Mais si nous manquons ou découvrons des lacunes fonctionnelles, nous l'implémentons par une surcouche des infrastructures logicielles et matérielles existantes à l'exemple du pair-à-pair et les réseaux privés virtuels. Cette surcouche logicielle offre ainsi pour chaque entité caractéristique de la grille une vue abstraite et uniforme de celle-ci, basée sur une modélisation simplifiée de ses fonctionnalités. *DIRAC* se veut donc la concrétisation de cette idée par une infrastructure déployable rapidement, facilement et nécessitant peu de maintenance.

3.2.1 Concept de l'architecture orientée service

DIRAC repose sur une architecture orientée service. La notion de service est de plus en plus répandue pour la construction d'un *SDGE*. Un service est un composant logiciel. Il est une abstraction des ressources, des informations et des services du réseau. Il comprend des interfaces (APIs), un protocole et un outil de développement (SDK) pour son implémentation. Un service est décrit par des méta-données publiées dynamiquement par un service dit registre ou une base de données. Un système distribué devient une composition de services qui dynamiquement, créent, déploient ou invoquent d'anciens ou de nouveaux services. Nous parlons alors d'architecture orientée service.

Un service est soit sans état (*stateless*) ou avec état (*stateful*). Un service *sans état* ne garde aucune information sur la connexion qu'il peut avoir avec une autre entité contrairement au service *avec état* qui garde des informations sur la connexion et offre donc des fonctionnalités personnalisées. Pour exemple, nous citons les services Web et les services grilles. Un service Web expose une interface et des opérations basées sur Internet par le protocole http. Parmi les normes, protocoles et

standard de Web service nous avons celles du consortium W3C. Elles se composent : d'un protocole RPC pour l'invocation des méthodes SOAP (Simple Object Acces Protocol), d'un format XML décrivant l'interface d'un service WSDL (Web Service Description Langage), d'une convention pour la localisation des services WSIL (Web Service Inspection Langage) et d'un langage décrivant la recherche et la composition des services UDDI (Universal Description, Discovery and Integration).

Le service grille implémenté par OGSI (Open Grid Services Infrastructure) sur la proposition d'OGSA (Open Grid Services Architecture) élargit les fonctionnalités des service Web. OGSI comprend la création, la destruction d'instance, la gestion de durée de vie, la recherche, la découverte et la notification de service. Conscients de tout les aspects positifs de cette proposition de prime abord séduisante pour l'inter-opérabilité entre logiciels de grilles, nous avons investi plusieurs mois dans l'évaluation du GT3 (Globus Toolkit 3), implémentation de OGSI. Cette évaluation a été abandonnée avant l'annonce conjointe d'IBM et de Globus de proposer WSRF (Web Services Resource Framework) [36] comme version allégée des services grilles.

Nous approuvons l'idée de service dynamique avec état voire temporaire en comparaison avec des services Web statiques et sans état. Néanmoins en 2003/2004, nous avons trouvé que OGSI n'était pas adapté à nos besoins du fait de sa lourdeur et complexité rendant impossible l'obtention de clients légers en simple utilisateur. De plus, le manque de documentation pour l'installation, la maintenance, le déverminage et les problèmes d'implémentations confirmèrent ces choix. Des problèmes similaires [185] ont étés aussi reportés. Une implémentation purement en Python, pyGridWare, du Lawrence Berkeley Laboratory a aussi été évaluée mais celle-ci était incomplète pour être pleinement utilisée.

Au sein de *DIRAC* les communications entre entités s'effectuent principalement par l'intermédiaire d'appel à distance ou RPC (« Remote Procedure Call ») et plus particulièrement XML-RPC. Le RPC a été choisi pour sa généricité et aussi pour sa diversité d'implémentation. Avec l'exacte définition des APIs, nous pouvons imaginer avoir un service disponible avec plusieurs protocoles de RPC ou nouveau système comme par exemple avec la nouvelle version du Globus Toolkit : le GT4 [184]. Cette caractéristique œuvre directement pour l'inter-opérabilité de services, facilite l'intégration et l'utilisation d'un service dans une infrastructure coopérative.

3.3 Vue d'ensemble

DIRAC se compose d'un ensemble de services légers. Le découpage des services s'inspire de la proposition ARDA [65] et OGSA [55]. Nous décomposons une grille de calcul en services indépendants. Ce découpage s'appuie surtout sur l'identification précise des fonctionnalités de chaque entité et la description des interfaces de ces composants. Cette approche vise à avoir une grande souplesse d'intégra-

tion, de remplacement ou d'évaluation des services. La réutilisation de l'existant est grandement favorisé.

DIRAC se décompose en quatre parties distinctes : les services, les agents, les ressources et les interfaces utilisateurs comme illustré dans la figure 3.1.

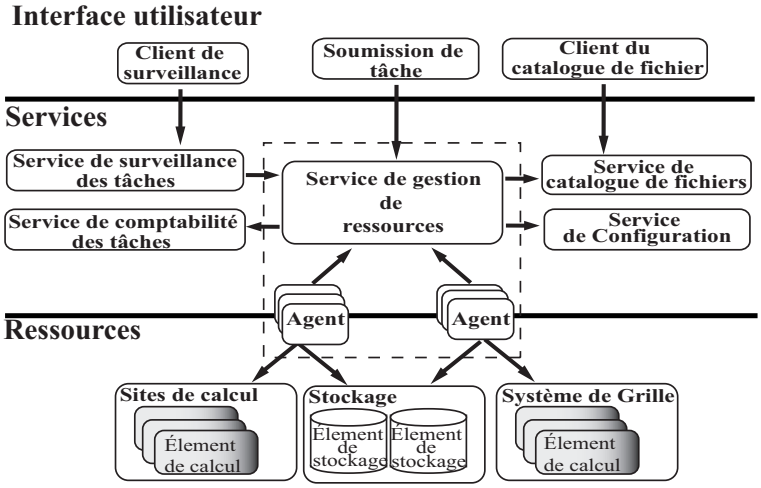


FIG. 3.1 – Vue générale de l'architecture de *DIRAC*.

L'utilisateur accède au système par une interface utilisateur. Celle-ci donne accès aux services centraux pour le contrôle ou l'extraction d'information sur le système. L'utilisateur interagit avec le système principalement en soumettant des requêtes. Celles-ci sont par exemple la soumission d'un ensemble de tâches, la demande de statut de ces tâches et la récupération des résultats. Ces interfaces sont simples et rarement modifiées. La complexité des traitements se situe surtout au niveau des services centraux.

Le noyau du système est un ensemble de services indépendants, sans états et distribués. Le lien avec les ressources s'accomplit avec les agents. Un agent est un composant logiciel actif qui interagit avec d'autres composants logiciels. La différence générale entre un agent et un service est qu'un service est passif et sans état tandis que l'agent est pro-actif et avec état. Les services peuvent être distribués sur plusieurs machines ou s'exécuter sur un seul « serveur », preuve supplémentaire de la robustesse du système.

Les services *DIRAC* sont gérés de manière centralisés et beaucoup d'efforts ont été portés pour ne pas solliciter plus d'une personne pour l'administration du système. Les services sont de plus hébergés sur des hôtes de haute disponibilité et maintenus.

Un agent est lui, associé à une ressource, tel un site de calcul. Un agent est par définition plus dynamique. Cette dynamicité favorisa l'utilisation de communications sans connexion permanente « connectless » sur TCP. Cette asynchronisme protège des ruptures de connections fréquentes dans un tel système. L'agent est

autonome et configuré en accord avec la politique d'utilisation de ce site par un administrateur. L'agent est pluri-fonctionnel. Parmi ses fonctionnalités, il participe activement à la gestion des ressources de calcul dans *DIRAC*.

Notre étude approfondie des applications que nous avons à mettre en œuvre (voir chapitre 1) ont rapidement soulevé l'impact de l'ordonnancement sur l'efficacité d'un tel système. Par exemple dans le contexte de production de données de *LHCb*, nous nous intéressons à l'utilisation maximale du système sur une longue période. Le calcul intensif favorise donc le régime permanent et saturé du système où la quantité des tâches à traiter est supérieure à la puissance de calcul disponible. Bien qu'au niveau local l'utilisation de système de traitement par lots est fréquente dans cette situation, il n'existe pas d'architecture standard au niveau global pour le calcul intensif. C'est pourquoi, nous détaillons dans la suite un état de l'art sur les systèmes de gestion de ressources dans les systèmes distribués pour comprendre les questions et enjeux dans la conception d'un tel système de gestion de ressource.

3.4 État de l'art sur les systèmes de gestion de ressources

Un Système de Gestion de Ressource (*SGR*) est l'interfaçe entre les créateurs de la charge de calcul et les ressources comme le montrent la figure 3.2 inspirée de [123].

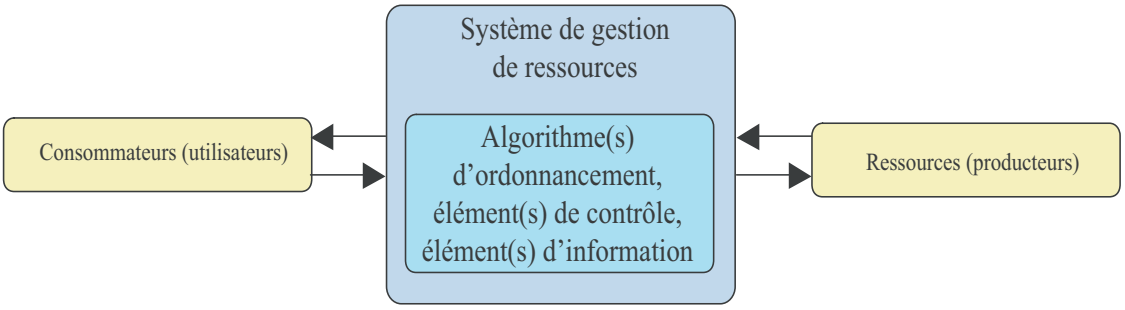


FIG. 3.2 – Fonctionnalité d'un Système de Gestion de Ressources (*SGR*).

Un *SGR* gère l'ensemble des ressources du système. Une ressource comme un processeur est une entité réutilisable employée à exécuter des tâches ou des requêtes. Une tâche est considérée comme une unité de travail. Nous utilisons indifféremment les termes de tâche et job, ainsi que processeur et nœud. Une tâche peut avoir des dépendances avec d'autres tâches. Ces dépendances sont décrites par un graphe orienté et acyclique ou *DAG* (« Diagram Acyclic Graph »). Un *DAG* est considéré comme une application composée de plusieurs tâches et se note $G = (t, d)$ où t est l'ensemble des tâches et d l'ensemble des dépendances entre les tâches. La figure 3.3 montre un exemple de *DAG*.

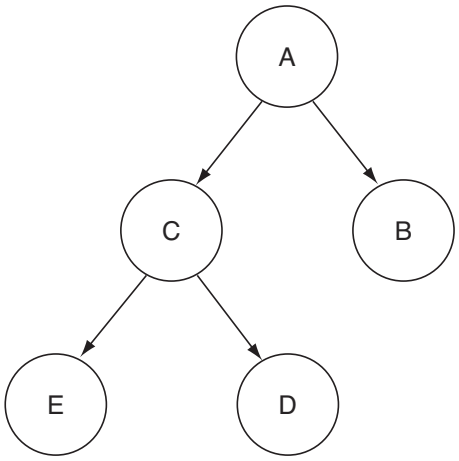


FIG. 3.3 – Exemple du *DAG* d'une application. A, B, C, D, E sont des tâches.

Les tâches dépendantes sont une classe importante à considérer et surtout pour toute la problématique qu'elles soulèvent dans un environnement où les nœuds sont faiblement couplés avec le réseau. Le degré de parallélisme d'une tâche s'exprime par le rapport du temps de calcul sur le temps de communication. Le type de tâche dominant dans les *SDGE*, sont surtout des tâches faiblement parallèles voire pas du tout. Dans la suite, nous considérons principalement les tâches simples, sans dépendances entre elles et rigides. Une tâche rigide en opposition avec une tâche malléable a un nombre de processeurs fixés à sa définition pour son exécution. Dans cette catégorie, nous considérons ensuite les tâches multi-paramétriques comme les simulations de Monte-carlo et les tâches ayant des dépendances à des données comme l'analyse de données.

Un *SGR* applique dynamiquement un algorithme d'ordonnancement prenant en entrée une charge de calcul constituée de tâches et un ensemble de ressources pour générer en sortie un ordonnancement. Un *SGR* dans un *SDGE* institutionnel intervient localement et globalement.

3.4.1 Problématique dans un *SDGE* global et institutionnel

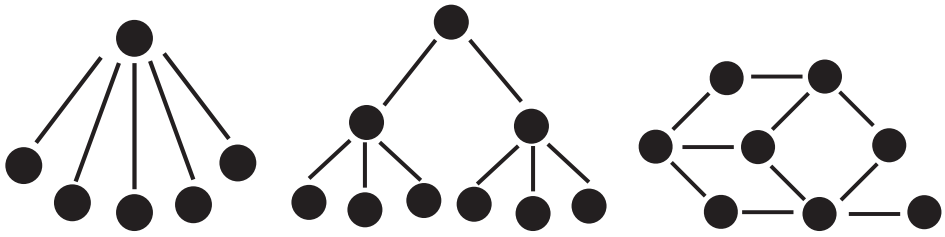
Dans les *SDGE* institutionnels et locaux, les ressources de calcul sont des grappes hétérogènes et autonomes de processeurs appartenant à un réseau local et divers domaines administratifs. Ces grappes sont non dédiées et partagées entre différents utilisateurs ou organisations virtuelles. Ainsi, une politique locale à chaque grappe définit le partage et l'accès entre les différentes organisations. Comme exemple de politiques locales, citons *First-Come-First-Serve*, *Minimal-request-Job-First*, *Shortest-Job-First* ou *BackFill* [121]. Ces politiques s'appliquent directement à travers le *SGR* local communément appelé un système de traitement par lot. Pour fédérer ces grappes et gérer la charge de calcul globale, la définition d'un *SGR* glo-

bal est nécessaire. Le contexte d'un *SGR* global est très différent d'un local. Dans un *SGR* local, nous pouvons compter sur une homogénéité des ressources et une connexion réseau haut débit dédiée. Ce qui n'est pas le cas dans un *SGR* global où l'hétérogénéité, le partage et l'autonomie des ressources, se traduisent par un fort aspect dynamique. En réponse, un *SGR* global doit être robuste, extensible et auto-adaptable.

De nombreuses taxonomies existent sur le sujet comme celle de Krauter et al. [97] qui se basent sur d'autres taxonomies antérieures [42,124,125,136] de *SGR*. Nous présentons ici une synthèse de l'ensemble de ces études en considérant d'abord comme l'un des critères de classification l'organisation des éléments participant aux décisions d'ordonnancement.

3.4.2 Modèle d'organisation des éléments de contrôle

Un élément de contrôle est hébergé sur un hôte et participe aux placements des tâches sur les ressources. Il prend des décisions d'ordonnancement. Celles-ci sont, par exemple, le transfert d'une tâche à un autre élément de contrôle ou l'allocation d'une tâche à un nœud. Ces décisions se basent sur des éléments d'information. L'organisation des éléments de contrôle intervenant dans un ordonnancement sont principalement soit [157] : centralisée (figure 3.4(a)), décentralisée (figure 3.4(b)) ou hiérarchique (figure 3.4(c)).



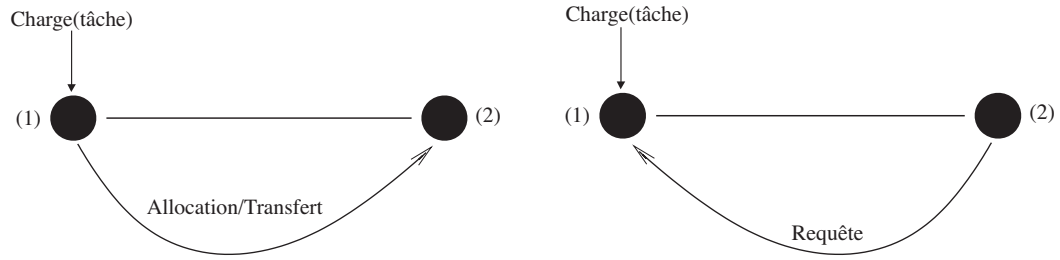
(a) Centralisée. (b) Hiérarchique. (c) Décentralisée.

FIG. 3.4 – Les principales organisations de l'ordonnancement.

Un ordonnancement centralisé, comme NetSolve [23], le « Resource Broker » de *LCG* [171] ou Condor condor-hunter, est simple à gérer mais il souffre de l'unicité du serveur central, d'un manque d'extensibilité et de tolérance aux pannes. Ces manques sont comblés par les organisations hiérarchiques et décentralisées des éléments de contrôle. Un ordonnancement décentralisé tel Javelin [28] possède une grande résistance aux pannes et aux reconfigurations du système. Un autre avantage est leur extensibilité : nous pouvons étendre leur taille sans créer de goulot d'étranglement. Par contre, leur inconvénient majeur est leur complexité. Le compromis est d'utiliser une gestion hiérarchique des ressources qui par rapport à un ordonnancement totalement décentralisé reste contrôlable. Comme exemple d'organisation hiérarchique, nous pouvons citer DIET [178] et AppLes [135]

3.4.3 Modèle de communication : « push » et « pull »

L'organisation des éléments de contrôles est importante mais leur manière de communiquer encore plus. En effet, celle-ci est directement impliquée pour les décisions, l'extensibilité et la qualité de l'ordonnancement. La communication des éléments de contrôle suit deux sortes de stratégies [98, 128] : les stratégies actives ou *Sender-initiated* qui sont initiées par les créateurs de charge, et les stratégies passives ou *receiver-initiated* qui sont initiées par des entités autres que les créateurs de charge. Nous parlons de stratégie passive comme le paradigme « pull », et de stratégie active comme le paradigme « push », toutes deux respectivement illustrées sur les figures 3.5(a) et 3.5(b).



(a) Le paradigme « push ».
(1) est actif et (2) passif.

(b) Le paradigme « pull ».
(1) est passif et (2) actif.

FIG. 3.5 – Les stratégies de contrôle.

Les stratégies actives « push » sont souvent appliquées dans les grilles de calcul et au niveau d'une grappe de processeurs, tandis que les stratégies passives « pull » sont plus fréquemment utilisées dans les systèmes de calcul global.

3.4.4 Les différentes phases d'un ordonnancement

Le choix du modèle de communication détermine les phases d'un ordonnancement dynamique. Avec une stratégie « push », nous avons trois phases. L'élément de contrôle ou ordonnanceur centralisé :

1. Collecte les éléments d'informations sur le statut des ressources ;
2. Sélectionne les ressources en appliquant un algorithme de sélection ;
3. Soumet ou transfère les tâches sur les ressources ;

Tandis qu'avec une stratégie « pull », nous avons les phases suivantes :

1. Un élément de contrôle associé à une ressource, détecte la disponibilité de celle-ci ;
2. Celui-ci demande ensuite du travail à un autre élément de contrôle ;
3. Ce dernier lui retourne une tâche en appliquant un algorithme de sélection ;

4. L'élément de contrôle soumet ou transfère la tâche sur la ressource ;

Les ordonnancements considérés sont dit paramètriquement adaptatifs, ce qui veut dire que les paramètres sont mis à jour dynamiquement en fonction de l'état courant du système. Les principales différences entres ces deux méthodes sont les éléments d'informations considérés et les algorithmes d'ordonnancement appliqués.

3.4.5 Les éléments d'informations

Un élément d'information a pour rôle de maintenir l'état courant du système. Cet état est utilisé par les éléments de contrôle pour effectuer le placement des tâches sur les ressources. *L'état global* du système est une collection de sources d'information, localisées dans des ressources distinctes. Toutes ces ressources doivent coopérer. Dans le cas d'un système local, le maintien de l'état global est relativement simple. Dans le contexte d'un *SDGE*, ce maintien est coûteux. Une solution comme le paradigme « pull » utilise *un état partiel* basé sur une vue locale de chaque ressource. Par contre, le paradigme « push » nécessite une vue sur toutes les ressources et donc une collection d'éléments d'informations, aussi appelé un système d'information.

Les systèmes d'informations

Différents systèmes d'informations existent pour fournir des informations statiques et dynamiques d'une plate-forme. Nous distinguons des modèles hiérarchiques comme MDS [18], R-GMA [92] et des modèles totalement décentralisés tel NWS [126]. L'organisation de l'information pour faciliter les requêtes est soit relationnelle comme R-GMA, hiérarchique MDS [18] ou sous forme de graphe tel le projet 2K [127] basé sur le modèle objet CORBA. La propagation de l'état d'une ressource et les requêtes de découvertes de ressources engendrent souvent des échanges de messages. Ceux-ci s'effectuent principalement selon trois approches : les échanges explicites ; les échanges périodiques ; les échanges relatifs effectués seulement lors du changement de l'état d'une ressource. Les requêtes sont effectuées par les entités chargées de l'ordonnancement.

De manière générale, les systèmes d'informations sont vitaux pour l'approche « push ». Le schéma des informations défini au préalable est souvent trop complet et oblige à maintenir plus d'informations que nécessaires. La délimitation entre informations statiques et dynamique est de plus souvent mal définie et le traitement de ces informations identiques. Un système d'information est directement impliqué dans les performances de l'ordonnancement « push » et représente un point de faille pour contenir l'ensemble des informations statiques et dynamiques d'une plate-forme mondialement répartie, dynamique et partagée. L'obtention de toutes ces informations est coûteuse et nécessite une infrastructure assez lourde à déployer qui supporte mal le passage à l'échelle. Ceci fut le cas pour avec MDS, basé sur la

technologie LDAP [45], dans le cadre du projet européen EDG. Le projet Nordugrid [63] utilise MDS mais à une échelle assez réduite d'une vingtaine de sites. Un système d'informations doit être stable, répondre sans grande latence aux requêtes et fournir des informations cohérentes pour les décisions d'ordonnancement.

3.4.6 Les algorithmes d'ordonnancement pour la mise en relation

Les algorithmes d'ordonnancement sont conditionnés par la politique d'utilisation des ressources, les critères d'évaluations de leurs performances [111] et aussi les éléments d'informations sur lequels ils se basent. Les critères d'évaluations ou fonctions objectives évaluent la pertinence d'un ordonnancement. Une fonction objective représente la quantité que la statégie de mise en relation tente d'optimiser. Celle-ci varie beaucoup entre les stratégies choisies et se définit soit par rapport au système comme Condor ou aux applications tel AppLes, soit pour des aspects économiques comme Nimrod-G [187].

Un critère système est par exemple de maximiser l'utilisation des ressources. Un des critères d'application est de minimiser le temps de réponse de l'application. Des systèmes comme DIET ou AppLes proposent des ordonnancements applicatifs avec plusieurs critères. Ils utilisent au maximum les connaissances sur les applications pour faire des prédictions et les incluent dans l'algorithme de l'ordonnancement. Les techniques employées [134] se basent sur des heuristiques, des méthodes d'apprentissage ou des méthodes stochastiques basées sur l'historique accumulé du système. Des jeux d'essais et étalonnages sont pratiqués pour déterminer les performances des machines d'exécutions et les caractéristiques des applications.

Un algorithme d'ordonnancement est centralisé ou distribué. Un algorithme centralisé signifie que les décisions d'allocation s'effectuent en un seul endroit. C'est le cas de l'approche « push », où l'algorithme de sélection de ressource décide de l'allocation d'une tâche dès sa soumission. Celui-ci s'effectue en deux phases. Il collecte d'abord les ressources candidates puis choisit parmi cet ensemble un candidat. Obtenir la solution optimale, qui par exemple minimise le temps d'exécution de la tâche, est connu pour être non déterministe polynomiaux (NP-complet) [143]. Des solutions s'en approchant existent dans le cas statique mais dans un environnement dynamique et hétérogène les stratégies se basent plutôt sur des heuristiques aidées éventuellement par des prédictions. Par exemple, pour le problème de l'allocation de n tâches indépendantes, les algorithmes suivants ont été proposés : *Fast Greedy*, *Min-min*, *Max-min*, *Greedy* ou glouton ou des algorithmes génétiques [136]. Par exemple, pour n ressources et p tâches, nous effectuons avec une approche naïve $n \times p$ correspondances que nous comparons ensuite entre elles pour trouver un couple qui parait le meilleur. Si nous cherchons une ressource pour une tâche la complexité est alors de l'ordre du nombre de ressources (*O(card(Ressources))*) qui est dans notre cas de l'ordre d'une centaine de sites et de quelques milliers de nœuds.

Dans le cas de l'approche « pull », l'algorithme de l'ordonnancement est décentralisé et se découpe en deux phases. Une tâche est mise en attente et est examinée pour l'allocation en réponse à des événements d'ordonnancement. Un événement d'ordonnancement peut être déclenché à l'arrivée d'une tâche ou à la fin d'une tâche. Ce dernier événement est synonyme de la disponibilité d'une ressource. Le problème est alors de déterminer cette disponibilité en estimant la charge d'une ressource. Cette tâche est complexe et dépend totalement du type de la ressource. La deuxième phase est de trouver une tâche qui correspond à cette ressource. La complexité algorithmique de cette recherche devient indépendante du nombre de ressources, et est de l'ordre de nombre de tâches en attente (*O(card(tâches en attente))*). Cette complexité est réduite si nous regroupons les tâches en classes par caractéristiques.

3.4.7 La problématique du calcul intensif et saturé

Dans le domaine des nouvelles expériences de la physique des particules, l'ordre de grandeur envisagé se situe autour d'une centaine de grappes réparties dans le monde et représentant 20.000 nœuds. L'autre caractéristique de ce domaine est le contexte de calcul intensif « High Troughput Computing » [90]. Nous nous intéressons alors à l'utilisation maximale du système sur une longue période. Le calcul intensif favorise le régime permanent et saturé du système. Bien qu'au niveau local l'utilisation de système de traitement par lots est fréquente, il n'existe pas d'architecture standard au niveau global pour le calcul intensif. La question est de savoir quelle organisation des éléments de contrôle choisir ainsi que leurs modes de communications et stratégies de mise en relation.

À travers un même modèle d'organisation des éléments de contrôle, le modèle de communication influe directement sur l'extensibilité. Si nous considérons une même organisation centralisée puis appliquons successivement un mode de communication « pull » et « push ». Nous pouvons caractériser les points suivants.

Dans une approche « push », les solutions existantes mettent en place un *super-scheduler* ou méta-ordonnanceur centralisé avec des statégies de contrôle « push ». La complexité de la mise en relation augmente avec le nombre de ressources à gérer. chaque ressource perd son autonomie. Cette solution est adaptée à une situation de charge plus faible où l'ordonnanceur peut passer un temps non négligeable à déterminer la ressource cible. L'événement déclencheur de l'ordonnancement est l'arrivée d'une tâche ce qui traduit que la charge de soumission est la principale source de goulots d'étranglements. Ces décisions se basent sur des informations plus complètes du système et obligent donc d'avoir un système d'informations performant. Cette situation autorise de trouver des solutions plus proches d'un optimal. Cet optimal peut être le temps de réponse minimal pour une tâche. Le système est *a priori* moins sensible à tout changement de la plate-forme.

Avec le paradigme « pull » centralisé, nous avons une approche plus réactive

aux variations de la plate-forme car les décisions se basent sur une vue locale. Il n'est donc pas nécessaire de définir un système d'information ce qui facilite sa mise en œuvre. L'événement déclencheur de l'ordonnancement est la disponibilité d'une ressource ce qui traduit que la charge liée aux ressources est la principale source de goulots d'étranglements. De plus, la complexité de la mise en relation est indépendante du nombre de ressource ce qui peut encore faciliter le temps de réponse. Rappelons que le serveur central de SETI@home [47] supporte des pics de 200.000 machines participantes en même temps. L'inconvénient est que la qualité de cet ordonnancement est moins performante dans la minimisation du temps de réponse car il se base sur des vues partielles du système. Par contre, il favorise l'utilisation de toutes les ressources tout en préservant leurs autonomies. Ce dernier semble mieux indiqué dans notre cas pour une charge lourde et donc dans un contexte de calcul intensif et saturé. Mais le calcul global n'applique ce paradigme qu'à un type limité de ressources comme des ordinateurs personnels. La problématique était de savoir comment l'appliquer dans un contexte plus généraliste de *SDGE* institutionnel avec tout type de ressource. C'est à ce dessein que le système *DIRAC* propose justement un exemple de solution à travers le système de gestion de ressource [7].

3.5 La gestion des ressources au sein de *DIRAC*

DIRAC met en place des files d'attente centrales globales et des éléments de contrôles, appelés agents, déployés sur des ressources. *DIRAC* utilise la stratégie *« pull »* où le rôle des agents est de demander des tâches quand ils détectent la disponibilité des ressources locales. *DIRAC* emprunte cette idée aux systèmes de calcul global ou de vols de cycles [40], dans lesquels un serveur central distribue des tâches à des nœuds en fonction de leurs disponibilités. *DIRAC* transpose ce concept au niveau de ressources de calcul distribuées en définissant un critère de disponibilité pour détecter la sous-utilisation de chaque ressource. Ces ressources peuvent être de simples ordinateurs, des systèmes de traitement par lot comme le montre le modèle hiérarchique sur la figure 3.6, ou même un système de grille.

L'agent vérifie périodiquement la disponibilité de la ressource. Le problème est de déterminer la période appropriée. Une petite période augmente le coût des communications dans le réseau et la charge sur la ressource. Un compromis est de définir cette période par rapport à la fréquence de changement d'état de la ressource comme par exemple, la moyenne des durées de séjour d'une tâche sur une ressource. La détection de la disponibilité de la ressource se base sur *un seuil de charge*. La difficulté est de définir ce seuil de charge de manière adéquate avec le type de ressource. Par exemple pour un simple ordinateur dédié, ce critère de disponibilité serait que l'utilisation de processeur ne dépasse pas un pourcentage donné. Nous illustrons dans l'équation (3.1) un critère de disponibilité possible pour un système

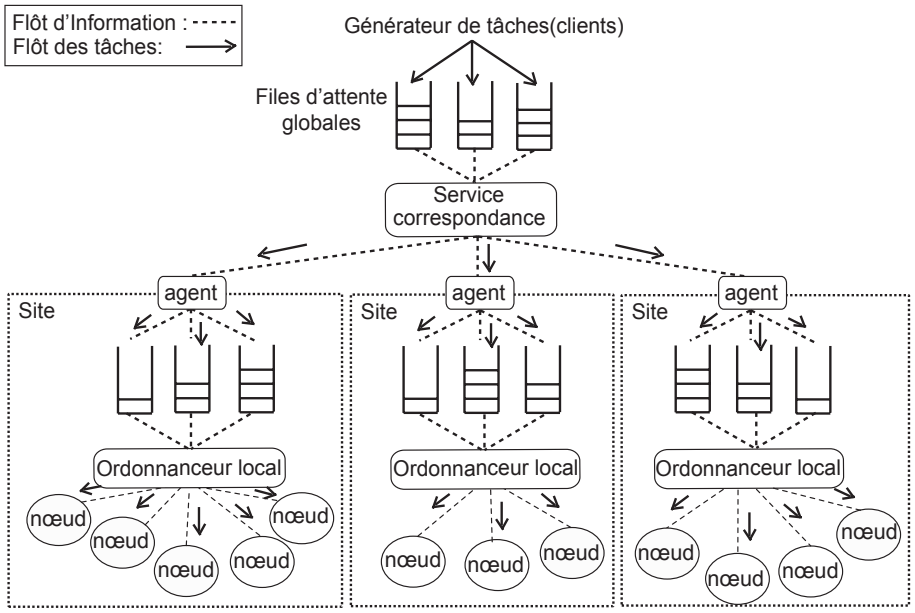


FIG. 3.6 – Le modèle d'ordonnancement hiérarchique de *DIRAC* avec des systèmes de traitement par lots.

de traitement par lots dédié qui se situe au niveau d'une file d'attente.

ε est une valeur choisie de manière arbitraire ou empirique entre 0 et 1. En cas de site partagé, ce critère ne s'appliquerait que sur les tâches dont l'agent est propriétaire.

$$\frac{Nombre\ de\ tâches\ en\ exécution}{Nombre\ de\ tâches\ en\ attente} < \varepsilon,\ tel\ que\ \varepsilon \in R_*^+\ et\ \varepsilon \in]0,1[. \tag{3.1}$$

Dès qu'une ressource est détectée comme disponible (c'est-à-dire respectant 3.1), l'agent dédié à cette ressource demande une tâche à un service *correspondance*. Cette requête s'effectue avec la description de la ressource, qui indique son état dynamique et statique. Le service *correspondance* résout la correspondance entre la description de la ressource et les tâches disponibles en file d'attente. Il applique un protocole « soft-state » en opposition à une protocole « hard-state ». Ceci signifie que le service *correspondance* n'a aucune connaissance *à priori* sur les ressources et ne maintient aucune information sur elles. Le modèle utilisé pour décrire ces ressources est de plus très important pour cette opération de mise en relation tâche/ressource.

3.5.1 Le langage de description des ressources et des tâches

Dans un environnement de calcul distribué, il est nécessaire de décrire les caractéristiques et les besoins d'une tâche de calcul. Ceci inclut par exemple, les entrées/sorties, l'environnement, les logiciels, les contraintes temporelles. De façon similaire les ressources de calculs doivent être décrites de manière assez satisfaisante pour déterminer l'adéquation d'une tâche avec une ressource. Cette description comprend parmi d'autres éléments, l'environnement, les performances et la mémoire.

Beaucoup de langages de description prennent leurs racines dans le langage développé par IBM pour les mainframes le JCL (Job Control Language). Parmi les exemples de langages, Globus a développé le langage RSL (Resource Specification Language) [137] pour le composant GRAM. Le Global Grid Forum (GGF) est proche de finaliser un langage de description basé sur XML, le JSDL (Job Submission Description Language) [138].

Le langage utilisé dans le projet *DIRAC* pour décrire les ressources, les tâches et les opérations de correspondance est *ClassAds* du projet *CONDOR* [90].

L'un des grands avantages de *ClassAds* est sa facilité pour l'inclusion de nouveaux attributs pour les ressources et les tâches. En effet, il est actuellement utilisé par beaucoup de projets de grille. Le JDL (Job Description Language) [140] développé par le projet européen *EDG* et maintenant étendu par le projet EGEE, se base sur *ClassAds* ; Il applique notamment, le schéma GLUE [139] qui définit un ensemble d'attributs par défaut pour un environnement de grille. Utiliser *ClassAds* avec le schéma GLUE, permet de garantir une inter-opérabilité avec d'autres projets.

3.5.2 Le langage ClassAds

Le langage *ClassAds* offre la possibilité d'exprimer pour une entité ses caractéristiques et ses besoins. Une entité est éventuellement consommatrice ou productrice. Cette dualité ne traduit aucune différence entre les ressources et les tâches. Il permet à une tâche de spécifier les restrictions relatives aux ressources qu'elle accepte, et à une ressource de spécifier les restrictions relatives aux tâches qu'elle demande. L'opération de mise en relation ou *match-making* s'applique sur deux descriptions *ClassAds*. Une description *ClassAds* contient une liste d'attributs. Chaque attribut s'exprime sous la forme suivante :

$$Attribut ='' Valeur''$$

où Valeur peut être de type différents comme une liste, une valeur numérique ou une chaîne de caractères. Suivi par des règles exprimées comme suit :

$$Requirements = (other.Attribut1 > 5.0)\&\&(other.Attribut2 ==' 2')$$

Chaque ligne est évaluée a une valeur précise. Dans le cas ci-dessus c'est une valeur booléenne. Une propriété préfixée par « other » désigne une propriété de la description à laquelle nous comparons la description courante. Si l'évaluation est vraie après la confrontation des deux descriptions, alors les conditions de correspondance sont validées. Dans le cas d'une correspondance ressource/tâche, l'utilisateur spécifie la liste des besoins de la tâche tels que l'architecture de la machine et le système d'exploitation. De manière similaire, la ressource exprime une liste de contraintes pour la plate-forme d'exécution comme par exemple, la mémoire maximale disponible. Cette opération de mise en relation peut s'effectuer au niveau d'une ressource, une tâche ou par un tiers. Dans le cadre de *Condor* c'est un tiers composant qui s'appelle le *Match-maker*.

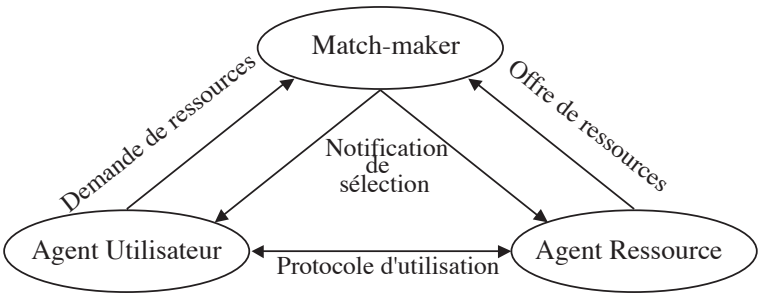


FIG. 3.7 – L'opération de mise en relation dans Condor.

La figure 3.7 montre comment le *Match-maker* apparie les demandes de travail issues des agents utilisateurs ou consommateurs avec les offres d'utilisation de l'agent propriétaire de la ressource. *Condor* se positionne surtout sur un domaine administratif tandis que *DIRAC* propose un élargissement à plusieurs domaines administratifs.

Nous illustrons dans le tableau 3.1 un exemple de deux descriptions *ClassAds* qui se correspondent par l'opération de mise en relation. Nous pouvons aussi voir en annexes (annexes D et E) des exemples de description de tâche *DIRAC* pour l'analyse de données et la production de données.

3.5.3 Le service de soumission en détail

Dans le contexte de *LHCb*, il n'est pas rare de soumettre plus de 30.000 tâches en une seule fois. L'architecture du système est spécialisée pour fournir la meilleure robustesse à un taux élevé de soumissions. Pour ceci, le service de gestion de charge de *DIRAC* se compose lui même de plusieurs services. Cette pluralité des services est destinée à fournir une grande robustesse au système et surtout à supporter des soumissions intensives. Chaque service peut s'exécuter sur une machine séparément des autres.

Ces services, représentés sur la figure 3.8, sont :

MonType = "Machine" ; TypeCible = "Tache" ; Name = "mon.ip.fr" ; Cpus = 1 ; Arch = "MonProcesseur" ; OpSys = "MonOS" ; Requirements = Other.MonType == "Tache" ;	MonType = "Tache" ; TypeCible = "Machine" ; Programme = "MonProgramme" ; Arguments = "MesArguments" ; Requirements = Other.MonType == "Machine" && Other.Cpus == 1 ;
***ClassAds* Agent**	***ClassAds* Ressource**

TAB. 3.1 – Exemple de descriptions *ClassAds* qui se correspondent de manière commutative par l'opération de mise en relation.

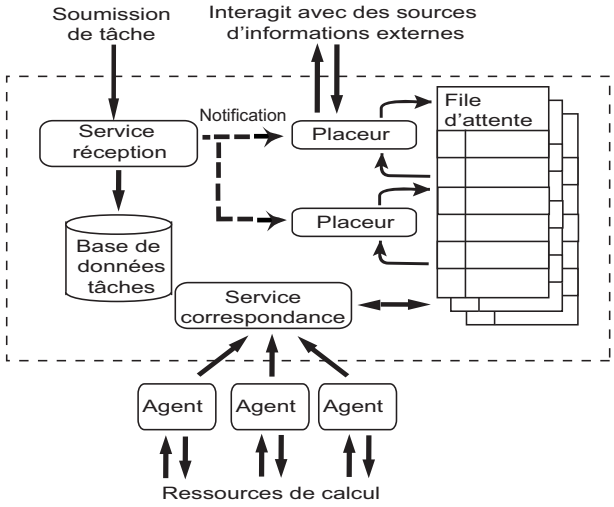


FIG. 3.8 – Architecture du service de gestion de charge dans *DIRAC*.

1. *Le service réception.* Accepte la soumission de tâches des clients, les enregistre dans la base de données des tâches et notifie les *placeurs* de l'arrivée d'une tâche.
2. *La base de données des tâches.* Contient toutes les informations à propos des paramètres des tâches et l'état dynamique des tâches. Nous donnons en annexes les états défini pour les tâches *DIRAC* (annexe H) ainsi que le schéma relationnel utilisé pour la description de cette base de donnée (annexe B).
3. *Les placeurs.* appliquent une discipline de placements des tâches dans les files d'attentes. Ils trient les tâches reçues par le service réception et les distribuent dans les files d'attente. Ces placements peuvent éventuellement se baser sur des informations extérieures comme la charge de travail et la disponibilité de données.

4. *Le service de correspondance* Alloue les tâches aux ressources en les sélectionnant à partir des files d'attentes et utilise le langage de mise en relation *ClassAds*.

Le découplement des fonctionnalités de réception de tâches et d'allocation garantissent des temps de service meilleurs par rapport à un système monolithique qui effectue tout en une passe, de la soumission à l'allocation. Ceci garantit une plus grande robustesse de service.

Pour diminuer la complexité des opérations de correspondance qui nous le rappelons dans un cas « pull » est de l'ordre du nombre de tâches dans le système, les tâches sont regroupées par caractéristiques et besoins. Ces catégories définissent des classes de tâches. Un exemple de classe de tâche peut être les tâches de production de données nécessitant plus d'un certain temps de calcul. Nous appliquons d'abord les disciplines de service entre les classes de tâches pour la mise en relation, puis dans chaque classe de tâche. Cette opération a une complexité de $O(n)$ où n est le nombre de classes de tâches. Cette opération est indépendante du nombre de ressources dans le système et du nombre de tâches. Ce regroupement de tâches autorise à les différencier en attribuant des priorités différentes par classe de tâches, c'est-à-dire par file d'attente.

Les files d'attente sont de deux types : exécution ou routage. Une tâche est en file d'attente d'exécution si toutes les conditions nécessaires sauf celle d'une ressource disponible pour son exécution sont satisfaites. Une tâche est en file d'attente de routage tant que les conditions nécessaires à son exécution ne sont pas satisfaites. L'exemple le plus fréquent est la dépendance à des données qui ne sont pas encore disponibles. Une fois ces conditions satisfaites, la tâche est transférée dans une file d'attente d'exécution.

Un *placeur* est un composant générique qui s'ajoute dynamiquement. Après la définition de son algorithme, sa mise en œuvre est rapide. Un *placeur* peut se spécialiser pour traiter un type de tâche précis. Les *placeurs* placent les tâches en file d'attente d'exécution en scrutant aussi les files d'attente de routage. Ils interfèrent donc avec des services extérieurs pour déterminer si les conditions d'éligibilité à l'exécution d'une tâche sont satisfaites.

Nous avons essayé de garder le maximum de souplesse dans le système car nous pouvons imaginer une organisation hiérarchique des *placeurs* ou même avoir un *placeur* qui implémente une approche « push » ou hybride. Ceci a été fait dans le souci de ne pas nous enfermer dans un seul choix condamnant toute évolution du système. Mais pour l'instant parmi les *placeurs*, un seul interfère avec un service extérieur. Ainsi, les tâches nécessitant des données sont traitées par *placeur* spécial qui interagit avec le service de gestion de données pour déterminer la disponibilité ou non des données pour cette tâche. Si oui, il place la tâche en exécution. L'événement déclencheur pour l'allocation de cette tâche est la disponibilité d'une ressource possédant les données. L'agent associé à cette ressource détecte la disponibilité et

effectue les transactions.

3.5.4 L'agent, un élément de contrôle proactif

Un agent est déployé très proche d'une ressource de calcul et interagit directement avec elle. Typiquement une ressource est un site de calcul régi par un système de traitement par lots. Cet agent est entièrement sous le contrôle d'un administrateur du site. Il s'exécute et se configure de différentes façons dépendant de la politique locale. Un agent est facilement déployable et nécessite une connectivité sortante pour contacter les services *DIRAC*.

L'agent est pluri-fonctionnel. Ceci se traduit par le fait qu'il ne sert pas seulement à la gestion de la charge, il peut servir par exemple à la gestion des données, la tolérance et aux recouvrements des pannes comme nous le verrons dans la suite. La conception de l'agent inclut un conteneur de modules et un ensemble de modules de base. Le diagramme de classes de la figure 3.9 illustre le mode de conception utilisé pour l'instanciation de ces modules.

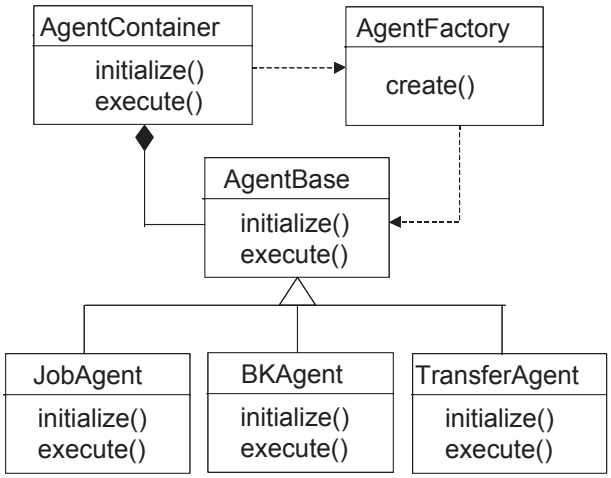


FIG. 3.9 – Diagramme de classe UML de la conception modulaire d'un agent.

Les modules sont exécutés de manière séquentielle ou parallèle. Un site a la possibilité d'exécuter plusieurs agents possédant leur propre ensemble de modules. Par exemple, un agent contient les modules pour la gestion des tâches et un autre contient les modules pour la gestion des données. Ce concept rend les agents *DIRAC* très flexibles car des nouvelles fonctionnalités sont ajoutées facilement et dynamiquement. Chaque administrateur de site décide quels ensembles de modules il exécute, ou peut personnaliser ces modules.

Le plus important de ces modules est le module de requête de tâches. Il surveille l'état de la ressource de calcul et demande des tâches au service *correspondance* quand il détecte une période d'activité creuse. À la soumission de la tâche globale

au système de traitement par lots local, il stocke les paramètres dans une base de données locale. Ceci permet de vérifier le statut des tâches et de déceler les défaillances. Cette information peut aussi être vérifiée par une interface légère de service fournie par un module de l'agent (voir section 3.8.4).

3.5.5 Le modèle des ressources de calcul

Une ressource de calcul ou *computing element* est une vue abstraite d'une ressource de calcul. Il expose une API standard pour l'exécution et le monitorage d'une tâche. En utilisant cette API un agent traite facilement avec des ressources hétérogènes. À présent, *DIRAC* s'interface avec LSF, PBS, BQS, GLOBUS/GRAM, NQS, Sun Grid Engine, Condor, Globus, de simples PCs et des systèmes de grilles complets comme *LCG*. Chaque implémentation s'occupe de traduire les besoins des tâches globales en paramètres locaux compréhensibles par le système de gestion de ressources. Il est à noter que l'interface de la gestion de ressources de *DIRAC* expose une interface identique à notre modélisation d'une ressource de calcul. Il est aisé dans ce cas là d'avoir un système hiérarchique et imaginer une ressource de calcul *DIRAC* comprenant des ressources reliées à un autre système *DIRAC*. Cette hiérarchie rajouterait de la robustesse à l'ensemble du service.

Nous avons vu comment la charge de calcul est gérée dans *DIRAC* à travers le service de gestion de ressources. Ce service est une des pièces centrales du système qui collaborent avec d'autres services. Nous allons détailler maintenant quels sont ces services et comment ils fonctionnent ensemble.

3.6 Le service de gestion des données

Une des choses très complexes à considérer au sein de *DIRAC* est la gestion des données. Nous donnons ici une description des composants du système de gestion de données de *DIRAC*. Celui-ci comprend la réplication, les transferts fiables, l'enregistrement des données et l'accès aux méta-données puis le stockage persistant sur les sites de stockages. La première entité à décrire pour un service de gestion de données est le modèle utilisé pour les ressources de stockage.

3.6.1 Le modèle des ressources de stockage

Une ressource de stockage ou Storage Element (SE) est aussi une vue abstraite. Elle se définit par une machine hôte, un protocole et un chemin. Cette définition est enregistrée dans le service de configuration (voir section 3.7.2) et est utilisée par n'importe quel agent ou tâche pour télécharger ou charger des données. *DIRAC* supporte les protocoles usuels, à savoir gridftp, bbftp, sftp, ftp, rfio, dcache, rootd ou un accès à un disque local. L'interface d'accès au SE est identique à celle du gestionnaire de réplicatas du projet *LCG* pour favoriser l'interopérabilité.

3.6.2 Les catalogues de fichiers

DIRAC offre une interface simple pour localiser les fichiers physiques à partir d'alias et d'identifiants universels de fichiers.

DIRAC s'interface avec trois catalogues de fichiers, l'un provenant de *LHCb*, l'autre du projet Alien [56] et le dernier issu de *LCG*. Ces catalogues s'utilisent de manière inter-changeable. Dans le cas du « data challenge » *LHCb*, les deux premiers ont été utilisés avec des informations de réplicatas dans le but de fournir une redondance pour ces composants vitaux du système de gestion de données.

Ceci illustre l'avantage de l'approche orientée service qui autorise d'utiliser des composants d'implémentations différentes mais fournissant les mêmes fonctionnalités avec une même interface.

3.6.3 Un transfert de données fiable

Un transfert de fichier est une opération fragile. Il peut être interrompu par une panne réseau, une panne matérielle de stockage ou des erreurs dans les logiciels impliqués. Il n'est pas inaccoutumé de perdre le résultat d'une longue tâche à cause d'un transfert de données qui a échoué. Pour cette raison *DIRAC* intègre un mécanisme de transfert de fichiers fiable.

Chaque site maintient une base de données de requêtes (voir figure 3.10). Une requête est soit un transfert de données, une réplication ou un enregistrement dans les catalogues de fichiers. Une requête peut contenir plusieurs opérations. Un module spécial de l'agent appelé « Transfert Agent » vérifie continuellement le contenu de la base de données de requêtes. Il les exécute. En cas d'échec, la requête est maintenue dans la base. Les requêtes partiellement accomplies sont modifiées pour réessayer seulement les opérations non faites.

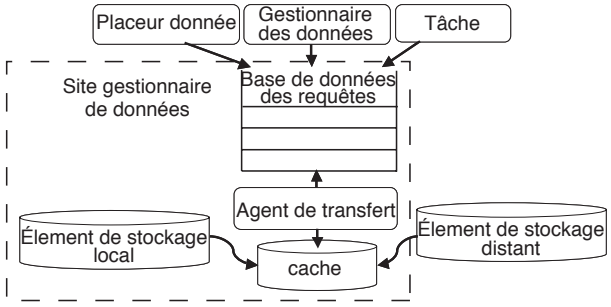


FIG. 3.10 – Le transfert de données fiable dans *DIRAC* sur un site.

La base de données des requêtes est accessible par une tâche régulière ou par des tâches spéciales dont le seul but est de positionner un transfert de données. Dans les deux cas, le progrès de l'exécution des requêtes se surveille par les outils standards de surveillance de tâches fournis par le système de gestion de charge.

3.7 Les autres services

Nous avons investi beaucoup d'efforts dans la gestion des ressources et des données. Ces services centraux sont vitaux pour le bon fonctionnement d'un *SDGE*, mais il s'inscrit néanmoins dans une infrastructure de collaboration entre services. En effet, nous trouvons d'autres services dans la proposition [65] utiles au bon fonctionnement d'un *SDGE*. Nous présentons une sélection de ces services implémentés dans *DIRAC* qui coopèrent avec les services présentés précédemment et ont nécessité notre intervention ou conception complète. Ces contributions sont issues d'un travail collaboratif dont la liste suivante présente les contributions respectives :

- Service de surveillance : V. Garonne et M. Sanchez,
- Service de comptabilité : V. Garonne, R.Graciani-Diaz , J. Saborido et R. Vizcaya-Carrillo,
- Service de configuration : I.Stokes-Rees.

3.7.1 Le service de surveillance et de comptabilité

Le fait d'avoir des outils de visualisation et de diagnostic de pannes est vital pour comprendre le comportement, pour analyser les performances d'un système de calcul distribué et le coordination de l'exécution des tâches. Le service de surveillance de *DIRAC* [9] met en place un modèle client/serveur comme dans les infrastructures de surveillances de *SDGE*. En plus de ce système, il utilise aussi Ganglia [94] ou MonaLISA [95]. Ces types de services sont des services point à point « end-to-end», et restent dans un modèle « pull ». Les informations sont émises de la ressource au service de surveillance qui les recueille en les estampillant de son horloge. Ceci fournit une horloge universelle pour donner l'état global du système.

Le service de surveillance fournit une interface aux agents et tâches pour mettre à jour les états de tâches, les machines hôtes ou des mesures réseaux. Ces composants ont donc aussi le rôle de sondes logicielles ou senseurs.

Ce service permet aussi aux utilisateurs d'interroger l'état d'une tâche. Il retient seulement les informations des tâches qui sont actives dans le système.

Cependant l'implémentation d'un système de surveillance dans un *SDGE* a une propriété intéressante, la mesure modifie souvent le système lui-même. Ces modifications sont dûes au trafic réseaux et consommation systèmes liés à cette infrastructure de surveillance.

Les tâches finies ou échouées sont enregistrées dans le service de comptabilité et effacées du système de gestion de charge. Le service de comptabilité détermine les consommations du système, des utilisateurs et les performances du système. Nous montrons dans les annexes (annexe G) les captures d'écrans des interfaces Web du service de surveillance et de celui de comptabilité.

La migration des tâches du système de gestion de charge au service de comp-

tabilité est effectuée par des agents de maintenance. Ceux-ci sont autonomes et chargés de l'entretien du système. Le nettoyage des tâches finies du système de gestion de charge en fait partie mais ça peut être aussi la détection automatique d'anomalies comme le marquage d'une tâche dans un état particulier dû à une absence de message de sa part aprés un laps de temps calculé.

Nous avons clairement distingué les rôles de chaque composant à l'instar de XtremWeb [40] où le rôle de coordination de l'exécution des tâches est assuré par le même composant qui effectue l'ordonnancement : le coordinateur. Dans le projet BOINC [80], cette coordination s'effectue au niveau du serveur central. Dans *DIRAC*, la coordination est effectuée par des agents autonomes et l'ordonnancement par les *placeurs* et le service correspondance.

3.7.2 Le service de configuration

Une problématique commune des systèmes distribués et des architectures orientées services est de partager des informations à travers le système. Dans ces systèmes, nous avons un réseau de services. Chacun d'entre d'eux nécessite de se configurer et de trouver les informations de configuration des autres services dynamiquement. Les utilisateurs doivent ensuite savoir comment accéder à ces services et les utiliser. En considérant un service de configuration pour *DIRAC* [8], il est ressorti que les mécanismes existants, tels UDDI [91], MDS [18], et R-GMA [92] étaient trop puissants, complexes et exigeaient une infrastructure lourde pour les utiliser.

Un système d'appariement de nom/valeur accessible par le réseau a été mis en place. L'implémentation applique les principes d'une implémentation légère et simple propre à la philosophie de *DIRAC*. Les composants utilisent le service configuration à travers un service de configuration locale. Celui-ci obtient toutes les informations à partir d'un fichier local, d'un service distant, ou une combinaison des deux. Nous montrons un exemple de fichier de configuration pour un agent en annexe F. Les informations distantes sont mises en cache. Plusieurs services de configuration sont utilisés dans le cas ou l'un d'entre eux serait défaillant. Les valeurs locales sont toujours privilégiées par rapport aux valeurs distantes. Si une valeur n'est pas retrouvée localement, le composant client interrogera alternativement tous les services Configuration disponibles.

3.8 Les choix d'implémentations

DIRAC repose entièrement sur des produits appartenant à la communauté du logiciel libre. L'implémentation de *DIRAC* est écrite entièrement en Python [179, 180], un langage objet interprété de haut niveau. Le langage Python est placé sous une licence libre. L'interpréteur Python est aujourd'hui disponible sur la majorité

des plates-formes existantes (BSD, GNU-Linux, Mac OS, Windows). L'interprétation induit une certaine lenteur d'exécution. Mais la simplicité du langage permet au programmeur d'écrire un code propre et naturellement optimisé. Comme dans un contexte de *SDGE*, le langage d'implémentation influe peu sur les performances du service, la vitesse d'exécution reste tout à fait acceptable. La portabilité des programmes, la rapidité de développement et la facilité de maintenance sont des atouts majeurs de ce langage. De plus, l'outil SWIG [181] permet d'étendre les fonctionnalités de Python en le liant à des bibliothèques C ou C++ comme la bibliothèque *ClassAds* utilisée dans *DIRAC*. Bien que Java, utilisé par exemple par Javelin [28], Jxta [16] et XtermWeb [52], expose beaucoup de ces qualités, nous ne l'avons pas choisi car ce langage n'était pas présent par défaut sur les systèmes d'exploitation ce qui limiter le déploiement du système. De plus, Java en comparaison avec Python semble plus lourd car il nécessite la machine virtuelle java JVM (*Java Virtual Machine*). Néanmoins Java est plus complet que Python notamment sur le confinement d'exécution ou « sandboxing » que Python tolère seulement sur l'utilisation du processeur. Il est à noter qu'un interpréteur Jython permet d'appeler du python et du java dans le même programme.

Une autre des grandes forces du langage Python réside dans l'existence d'une bibliothèque standard très complète et d'un nombre important de bibliothèques externes. La bibliothèque python *pyOpenSSL* fournit la bibliothèque *OpenSSL* [189] qui comprend les certificats x509 [190]. Par conséquent, l'infrastructure de sécurité de *DIRAC* se base sur ces certificats pour l'authentification et l'autorisation. Nous avons évité l'utilisation lourde de GSI [20] de globus. La couche de communication sécurisée passe par la couche sécurisée ssl et http. La bibliothèque standard Python supporte notamment de nombreux protocoles de communication par http dont le XML-RPC utilisé dans les communications entre les Client/Services et les Agents/services.

XML-RPC [37] est un protocole de type RPC qui utilise le format XML comme support pour l'encodage des messages et http pour le transport. A ce titre il est très proche de SOAP, les principales différences avec ce dernier étant la simplicité radicale de sa spécification. XML-RPC a été adopté dans de nombreux projets de logiciels libres et il existe à l'heure actuelle 67 implémentations pour les langages les plus répandus. Faire une instance et exposer une interface multi-threadé en Python est très rapide et robuste. Cette grande diversité d'implémentations promeut XML-RPC comme un composant de choix pour l'inter-opérabilité et l'extensibilité. Par contre XML-RPC comme SOAP souffre de performances médiocres. Les données sont transmises dans un format ASCII ce qui entraîne un surcoût très important de bande passante. En revanche, contrairement à SOAP, XML-RPC offre un support pour les données binaires via l'encodage base64. Cet encodage réduit sensiblement la taille des données transmises et donc atténue la dégradation de performances.

La persistance des tâches et de leurs états sont stockées dans une base de don-

nées MySQL.MySQL est un serveur de bases de données de type SQL (Structured Query Language). Les bases de données locales des agents utilisent SQLite [54] qui est une bibliothèque qui propose un moteur de base de données SQL sans reproduire le schéma habituel client-serveur. Elle est intégrée directement aux programmes et donc au niveau utilisateur.

Dans la suite, nous détaillons les points clefs d'implémentations qui ont contribué au succès de *DIRAC* tels que la tolérance aux pannes, un robuste et simple service de configuration et l'utilisation de la messagerie instantanée.

3.8.1 Le déploiement des agents et logiciels

En gardant l'implémentation en Python, les clients et agents nécessitent seulement un interpréteur Python pour leurs installations et une connectivité internet sortante pour connecter les services *DIRAC*. Ceci permet d'exécuter les agents sur n'importe quelle ressource et réseaux incluant celles derrières des parefeux et des réseaux privés utilisant le système de translation d'adresse NAT (Network Address Translation) pour accéder internet.

L'installation est entièrement dans l'espace utilisateur limitant les risques d'insécurité. Ceci convient aussi aux membres *LHCb* qui administrent un agent *DIRAC* sur le centre de calcul local mais n'ont pas les accès administrateurs.

Ces composants distribués aux utilisateurs et centres de calcul sont d'une taille inférieure à un méga-octets. Ce qui facilite des installations rapides et des mises à jour des logiciels.

Les logiciels requis par les applications sont installés dans une approche « paratrooper ». C'est-à-dire que chaque tâche installe les logiciels s'ils ne sont pas déjà présents. Les logiciels sont mis dans des caches et disponibles pour les futures tâches utilisant le même agent. La distibution se base sur *CMT* [131] et *PACMAN* [130, 174].

3.8.2 La tolérance aux pannes

Dans un environnement de calcul distribué, il est quasiment impossible d'assurer que le réseau, les stockages et services distants seront constamment disponibles. Le résultat est qu'une opération distante échoue souvent dans les trois cas : panne pour connecter une ressource distante, appel bloquant durant une opération distante ou une terminaison anormale.

Ces pannes sont souvent temporaires. Un nouvel essai après un certain laps de temps ou le choix d'une ressource alternative est souvent fructueuse. Ceci permet de compléter l'opération primaire avec un délai dû aux nouveaux essais. Dans le but de traiter avec ces modes de pannes les mécanismes suivants sont utilisés dans *DIRAC*.

Le service de configuration est l'ossature centrale pour un accès coordonné aux informations concernant tous les services *DIRAC*. Chaque composant utilise le service de configuration local. La pluralité des sources d'informations et les appels distants introduisent une redondance qui rend le système robuste aux éventuelles pannes. Pour se prémunir de ces pannes trois stratégies sont définies.

1. La duplication : plusieurs services possèdent des services de sauvegarde disponibles tout le temps.
2. Un système de basculement ou *fail-over*. pendant la connexion à un service critique, après un nombre d'essais suffisant, une requête sur un service alternatif est tenté.
3. La mise en cache : la mise en cache de requêtes et de leurs résultats lors de la première demande pour accélérer les opérations suivantes utilisant ces mêmes valeurs.

3.8.3 Mécanismes de recouvrement des pannes

En réponse aux pannes fréquentes, le système incorpore des mécanismes automatiques de recouvrement des pannes. Un mécanisme de recouvrement restitue dans un état cohérent l'entité touchée par une panne. Si nous restaurons l'état le plus récent, nous parlons alors de « forward recovery » ou encore de restauration vers l'avant. Si cela n'est pas possible comme avec une panne trop complexe, il faut alors assurer la consistance de l'entité pour l'ensemble du système. Ces mécanismes garantissent un système plus autonome avec un minimum d'interactions humaines. La plupart de ces recouvrements sont mis en place après avoir rencontré et identifié ces pannes. Celles-ci se situent au niveau des services et agents, et au niveau des tâches.

Au niveau des services et agents. Tous les services et agents s'exécutent sous le contrôle d'un démon *runit* [33]. *runit* assure qu'un composant redémarrera après une panne ou le redémarrage de la machine hôte. Un de ses attributs est la gestion de processus qui limite la consommation de mémoire et le nombre d'ouvertures de fichiers dans le système. De cette manière, un service ne peut bloquer le système entier. *runit* génère aussi des estampilles automatiques et des traces rotatives facilitant le déverminage. Un composant peut être arrêté, suspendu, redémarré ou temporairement désactivé à travers *runit*. De plus, il ne requiert pas de droit utilisateur pour son utilisation.

Au niveau des tâches.

Pour chaque tâche, une enveloppe de script prépare l'environnement d'exécution, télécharge les données nécéssaires et reporte l'état de la tâche et les caractéristiques du nœud d'exécution. Il lance un processus de surveillance ou « autopilot ». Ce processus vérifie périodiquement l'état de la tâche et envoie

un signal qualifié de battement de cœur ou « heart beat signal » au service de monitorage. Ce mécanisme s'inspire du projet *Heartbeat Monitor* [129]. Une interface par messagerie instantanée fournit un canal de contrôle sur la tâche. À la fin d'une tâche, le processus de surveillance reporte toutes les informations d'exécution comme par exemple la mémoire et le temps CPU consommé au service de monitorage. Finalement, il traite aussi les erreurs d'exécution et les transmet conjointement. Nous distinguons pour les tâches deux types d'erreurs. Celles directement causées par le système comme un service de soumission local défaillant, et celles directement liées aux applications comme une boucle infinie. *DIRAC* ne traite de manière automatique que le premier type d'erreurs. Une classification de ces erreurs et leurs traitements a été établie. Des erreurs déclenchent parfois la re-soumission de la tâche locale ou globale.

3.8.4 La messagerie instantanée dans le domaine des grilles

Dans le but de fournir une messagerie asynchrone fiable, *DIRAC* incorpore un protocole de messagerie instantanée dans tous ses composants. L'engouement public pour la messagerie instantanée ont conduit à des projets optimisés utilisant des standards bien définis. Les versions les plus robustes supportent aujourd'hui plusieurs milliers à des dizaines de milliers d'utilisateurs en même temps. Tandis qu'ils utilisent surtout des communications de personne à personne, il est clair que des communications machine à machine ou de machine à personne sont possibles. C'est dans ce domaine que *DIRAC* démontre une nouvelle application de cette technologie.

Tandis que les services *DIRAC* exposent leurs interfaces via le protocole XML-RPC, il y avait toujours un besoin de fournir un canal de contrôle et de surveillance pour les agents et les tâches. Aucune information n'est connue à priori sur le lieu et la date de l'exécution d'un agent ou d'une tâche. De plus un réseau local n'autorise pas un agent ou une tâche à exécuter un serveur XML-RPC accessible de l'extérieur. Ceci suggère l'utilisation d'une infrastructure de communication dynamique et asynchrone initiable par un client.

Le protocole de messagerie XMPP (eXtensible Messaging and Presence Protocol), maintenant une norme Internet IETF [53], est utilisé dans quatre composants de *DIRAC* : au niveau de l'interface utilisateur, les tâches, les agents et les services. Le serveur XMPP utilisé est Jabber. XMPP fournit les fonctionnalités standard de messagerie instantanée comme le chat, le groupe de discussion, le diffusion de message et le message de point à point. De plus, un mécanisme RPC appelé IQ(Information/Query) peut être utilisé pour exposer une interface à n'importe quelle entité XMPP. Finalement, le mécanisme de favoris ou roster facilite la surveillance automatique et quasi en temps réels des entités XMPP par leurs présences.

Les services *DIRAC* utilisent XMPP aux endroits où la tolérance aux pannes et les messages asynchrones sont importants. Par exemple, le service de réception utilise XMPP pour notifier le *placeur* quand il reçoit une nouvelle tâche. Quand l'optimiseur reçoit cette nouvelle tâche il la trie dans la file d'attente appropriée. La fonctionnalité IQ permet aux utilisateurs d'obtenir des informations instantanées à propos des tâches en exécution chose souvent critique pour les tâches interactives, ou pour le contrôle de tâche. Il favorise aussi grandement le debbugage et le recouvrement sur panne d'une tâche bloquée.

XMPP est conçu spécialement pour avoir des clients légers. Il offre la fonctionnalité de boite de messages en stockant les messages jusqu'à ce qu'une entité soit disponible pour les lire. En mélangeant les fonctionnalités IQ et celles des messages standards XMPP, il devient possible pour les utilisateurs de localiser et de communiquer avec les agents, les tâches et services de n'importe où.

La principale problématique qu'entraîne l'utilisation de la messagerie instantanée est la sécurité. Ce tunnel pose évidement des problèmes de sécurité pour l'authentification des utilisateurs. Il est à noter, qu'un groupe de recherche du Lawrence Berkley Laboratory a développé un serveur XMPP qui accepte les certificats Globus x509 pour l'authentification. Ceci est une avancée positive pour l'utilisation de cette technologie dans le domaine des systèmes distribués.

3.9 *DIRAC* : Un système hybride entre grille de calcul et calcul global

DIRAC incorpore les deux courants majeurs des *SDGE*. Il utilise des ressources de grille en appliquant les stratégies du calcul global pour réguler la charge de calcul. Mais l'aspect le plus intéressant et innovateur pour l'illustration de cette convergence se symbolise par la virtualisation offerte par *DIRAC*. Celle-ci se manifeste quand des systèmes déploient leurs propres infrastructures en surcouche des infrastructures logicielles et matérielles existantes. Ainsi Condor utilise XtremWeb pour créer une grappe Condor associant les fonctionnalités de Condor et Condor-G [73, 132]. *DIRAC* approfondit ce principe en le généralisant avec toutes les ressources comme nous l'illustrons avec la grille de calcul *LCG*.

LCG englobe aujourd'hui un grand nombre de ressources de calcul. Cette plateforme comprend au dernier recensement plus d'une centaine de sites représentant plus de 10.000 nœuds. Une des ambitions de *DIRAC* est d'utiliser toutes les ressources disponibles même celles à travers un autre système comme *LCG* pour fournir un même système virtuellement uniforme. La problématique était alors de définir l'agrégation de *LCG* dans l'infrastructure *DIRAC*.

Initialement, une des premières idées a été d'utiliser *LCG* comme une ressource classique, c'est-à-dire un centre de calcul. En effet, *LCG* peut exposer une interface identique à notre abstraction d'une ressource de calcul et définir un critère de

disponibilité. Mais le taux de pannes de *LCG* mesuré à 61% (voir chapitre 4.1), nous a incité à définir une approche plus intéressante et prometteuse, basée sur des agents légers et rapidement déployables qui font parti du système de gestion de charge de *DIRAC*.

Nous définissons deux déploiements pour les agents. Un déploiement *direct* où l'agent déployé sur la ressource soumet des tâches. Ce cas est identique à celui utilisé avec les sites de calcul et expliqué en section 3.5. L'autre déploiement est qualifié de *indirect* ou *dynamique*. L'agent maître déployé sur la ressource détecte si celle-ci est disponible. Dans le cas positif, il interroge le service *correspondance* pour savoir si des tâches sont disponibles. Si tel est le cas, il soumet des agents pilotes à la ressource et réitère sa demande. Techniquement, l'agent pilote soumis est enveloppé dans un script « enveloppe » ou « wrapper ». Bien que l'implémentation de l'agent déployé sur le site et soumis soit le même, nous distinguons leurs rôles en les qualifiant respectivement d'agent *maître* et d'agent *pilote*. Ce dernier est considéré par le système sous-jacent comme une tâche normale et bénéficie du système d'ordonnancement comme un mécanisme de déploiement. Ensuite comme illustré sur la figure 3.11, l'agent une fois sur le nœud demande du travail au service *correspondance* avec la description du nœud générée au préalable. Le nœud doit répondre alors aux mêmes contraintes qu'un agent classique. Ce qui se traduit surtout par une connectivité sortante.

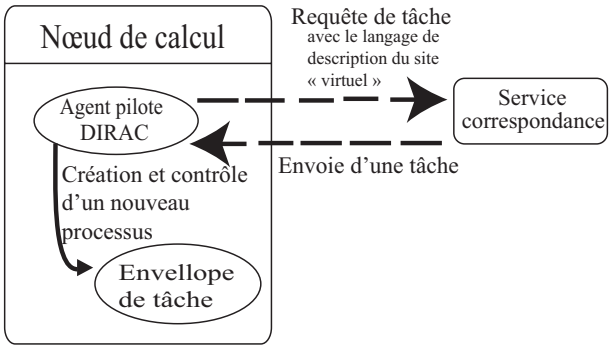


FIG. 3.11 – Illustration du déploiement dynamique de l'agent *pilote*.

Si une tâche est disponible, l'exécution de cette tâche est effectuée. Dans le mode de réservation simple l'agent meurt après l'exécution d'une tâche. En cas de non disponibilité d'une tâche, l'agent meurt immédiatement après avoir reçu une réponse négative du service correspondance. De plus, nous nous agrégeons aussi les ressources de stockage du site en incluant des transferts à la fin d'exécution des tâches. Par cette stratégie nous illustrons bien toute la souplesse de *DIRAC*. Nous agrégeons toutes les ressources disponibles pour une organisation virtuelle. Nous nous approprions entièrement la plate-forme par un système virtuel et identique. Nous définissons notre propre grappe *DIRAC* de nœuds virtuels et dynamiques. La

définition de leur agrégation et organisation reste libre pour l'allocation des tâches.

3.10 Conclusions, discussions et questions ouvertes

Nous discutons ici les points clefs de *DIRAC* qui présentent tous les aspects positifs d'un système conjuguant les grilles de calcul et le calcul global en un système virtuel, léger et extensible. L'approche orientée service de *DIRAC* prouve toute la flexibilité d'une telle approche qui permet une rapidité de développement et d'intégration d'un système distribué à grande échelle. Le paradigme « pull », appliqué au travers du modèle Agent/services est novateur dans un contexte de grilles de calcul.

Une de nos premières constatations fut de voir une forme de récursivité dans les fonctionnalités. Pour exemple, nous avons la fonction d'ordonnancement qui est à plusieurs niveaux : d'abord à celui d'une grille, puis celui d'un système de traitement par lots et enfin au niveau d'un nœud. À chaque niveau nous retrouvons donc une même abstraction de ressources de calcul. Nous suivons alors un modèle d'organisation hiérarchique, où les machines d'un même niveau communiquent entre elles ou avec celles de niveaux immédiatement supérieurs ou inférieurs. La citation de B.Mandelbrot, mathématicien et inventeur des images fractales, s'applique alors bien à l'organisation de notre système : *« Le tout peut être divisé en partie plus petites, Chacune répétant le tout comme en écho. »*.
Mais si nous considérons que l'entité terminale de cette récursivité est un nœud, nous pouvons au contraire déplier cette hiérarchie et avoir d'autres modèles d'organisation, comme nous l'avons expérimenté avec *LCG*. Cette approche permet de voir *DIRAC* comme un émulateur virtuel du simple ordinateur au système complet de grilles de calcul, comme le montre la figure 3.12.

En effet, le déploiement dynamique des agents *pilotes* sur les nœuds donne une vue uniforme et abstraite de toutes les ressources disponibles pour une organisation virtuelle. Il permet de s'approprier les nœuds de manière homogène et fournit pour chaque nœud toutes les fonctionnalités de *DIRAC*. L'organisation de ces nœuds peut suivre l'approche « pull » de *DIRAC*. Néanmoins les fonctionnalités offertes par la messagerie instantanée suggèrent d'autres organisations logiques plus prometteuses, comme une organisation à plat ou par cellule plus typique des systèmes pair-à-pair. Dans une organisation à plat, tous les nœuds peuvent communiquer entre eux. Dans une organisation par cellule, les nœuds au sein d'une cellule communiquent entre eux. Une autre cellule ne voit pas l'organisation interne d'une autre cellule. La définition des cellules est une autre problématique. Une cellule peut être définie géographiquement ou regroupée par caractéristique. Dans une telle organisation, nous pouvons imaginer de la migration de tâche d'un site à un autre ou une organisation pair-à-pair pour la répartition de charge.

Par contre, d'un point de vue pratique, ces nouvelles approches posent des pro-

blèmes en matière de sécurité comme la délégation de l'authentification et l'autorisation d'un utilisateur à partir d'un nœud au lieu de l'entrée d'un site. La tracabilité locale est alors perdue et l'organisation virtuelle se charge de fournir cette information. La nécessité de soumettre des agents sous l'identification d'une organisation au lieu d'un individu est problématique et envisage de revoir les infrastructures de sécurité. Sur ce point là, nous connaissons encore mal tous les avantages ou inconvénients que peuvent nous apporter les mécanismes de machine virtuelle comme Xen [133] qui permettent de multiplexer virtuellement une machine en plusieurs, choses utiles dans un environnement trans-institutionnel. Nous pouvons aussi noter que nous utilisons des systèmes et des ressources de types grilles. Une évaluation intéressante serait d'essayer d'agréger des ressources plus volatiles de calcul global dans un seul et même système.

DIRAC est un système en production depuis mai 2004 nous allons décrire dans le chapitre suivant ses performances et les expériences acquises.

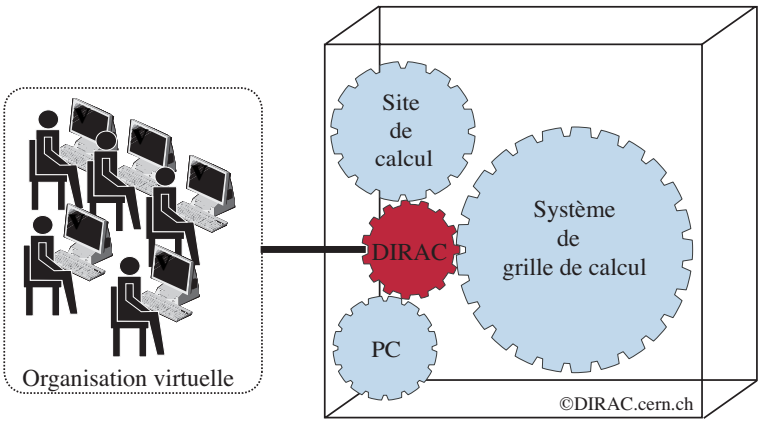


FIG. 3.12 – *DIRAC* comme un émulateur d'autres systèmes pour une organisation virtuelle.

CHAPITRE 4

LE DÉPLOIEMENT DE *DIRAC*

Résumé du chapitre :
Dans ce chapitre, nous présentons le déploiement du système DIRAC *et l'expérience acquise avec son utilisation du système distribué à grande échelle* LCG. *L'expérience* LHCb *utilise* DIRAC *en production depuis mai 2004. Pendant le* Data Challenge*, qui a duré quatre mois, le modèle Agent/Services conjugué avec le paradigme « pull » s'est révélé approprié et efficace dans un environnement dynamique. En effet,* DIRAC *a connecté plus de 6.000 processeurs répartis sur une soixantaine de sites dans le monde. Le nombre de tâches exécutées est de l'ordre de 200.000.* DIRAC *a supporté plus de 5.500 tâches simultanées. Le montant des données transférées et stockées est supérieure à 100 Téra-octets.*

4.1 Le contexte

L'expérience de physique des particules *LHCb* (*Large Hadron Collider Beauty*) utilise *DIRAC* (Distributed Infrastructure with Remote Agent Control) en production depuis mai 2004. *DIRAC* est utilisé intensivement lors des « Challenges » de *LHCb*. Le but de ces *challenges* est de tester, valider et agréer le modèle de calcul de *LHCb* jusqu'à l'utilisation constante du système pour l'exploitation de *LHCb* en 2007. Ce passage à la production révèle des comportements relatifs à un système réel, souvent non prédictibles. Nous présentons ici principalement les résultats obtenus lors du « Data Challenge 2004 ». Celui-ci s'est déroulé de mai à octobre 2004. Il avait pour but : de valider l'infrastructure distribuée de *DIRAC* en utilisant des ressources *LCG* (LHC Computing Project) et hors *LCG*; de vérifier les logiciels de *LHCb*; et de générer des données simulées pour l'analyse. La quantité des données à générer était fixée à 60 téra-octets ce qui représentera l'équivalent de quinze minutes de fonctionnement de *LHCb*. Le déploiement de *DIRAC* s'est effectué avec l'aide logistique des administrateurs de sites de calcul. Des procédures entre l'équipe *DIRAC*, l'équipe *Data Management* de *LHCb* et les administrateurs de sites ont été mises en place pour les retours sur le système. Mon rôle dans ce projet était d'être le responsable du service de gestion de ressources de sa conception à son évaluation puis son déploiement et support.

Dans ce chapitre, nous discutons de l'expérience pratique acquise lors du déploiement de *DIRAC*, ainsi que des caractéristiques d'un système et d'une plate-forme réels. Nous analysons les résultats obtenus lors du déploiement à l'échelle de *DIRAC*. Ces résultats se basent sur une analyse des données issues du service de surveillance, du service de comptabilité et des journaux ou *logs* du système. Nous décrivons d'abord les caractéristiques de la plate-forme mondiale de production en

section 4.2. Par la suite, nous soulignons les performances de *DIRAC* dans la section 4.3 ainsi que les les informations recueillies sur les tâches et leurs exécutions dans la section suivante 4.4. Nous détaillons dans la section 4.5 la gestion des données puis dans le section suivante 4.6 l'inter-opérabilité de *DIRAC* avec le système de grille *LCG*. Nous détaillons ensuite en section 4.7 les problèmes rencontrés, puis nous conclurons.

4.2 La plate-forme de production

DIRAC se compose de deux plates-formes. Une première minimaliste qualifiée de « banc d'essai » ou *testbed* utilisée pour tester et valider de nouvelles stratégies et implémentations. Après une phase de tests, ces modifications sont reportées sur la plate-forme de production, plus stable par définition. La plate-forme de banc d'essai est de plus utilisée pour valider un site avant de le mettre en production. Cette procédure se révéla très utile pour se prémunir des erreurs de configuration d'un site, fréquentes sur une plate-forme de cette ampleur. La plate-forme de production lors du DC04 se composait de 58 sites de calcul répartis sur 16 pays comme le montre la figure 4.1.

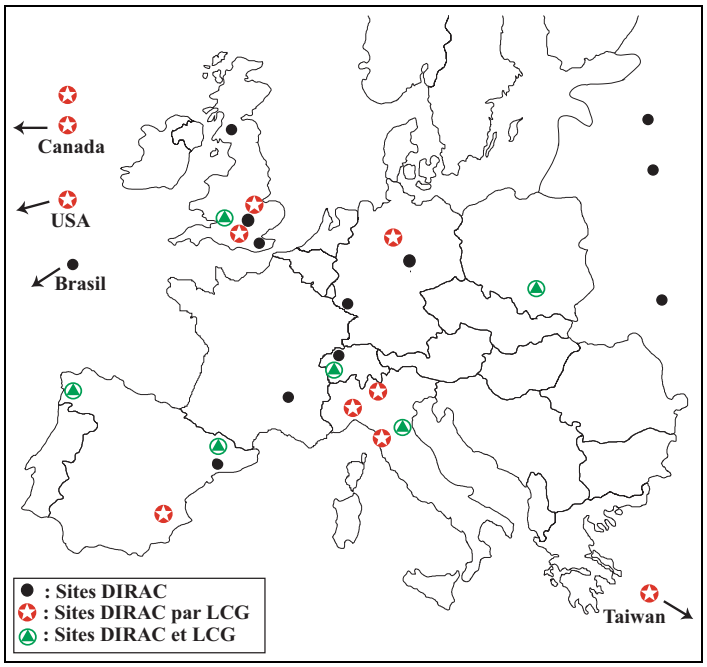


FIG. 4.1 – Carte de la plate-forme de production *DIRAC* lors du DC04. La plate-forme réunit des sites de *LCG* et hors *LCG*.

La particularité de la plate-forme est l'inclusion des ressources *LCG*. Celles-ci représentaient alors une quarantaine de sites. Les sites *LHCb* représentaient une

vingtaine de sites. La taille des sites participants varie beaucoup. Cela va de 20 processeurs partagés avec d'autres utilisateurs à plus de 500 processeurs dédiés. La figure 4.2 donne un instantané de la distribution du nombre de tâches en exécution par site. Cette figure illustre la disparité de puissance des sites qui est aussi fortement conditionnée par la puissance des nœuds.

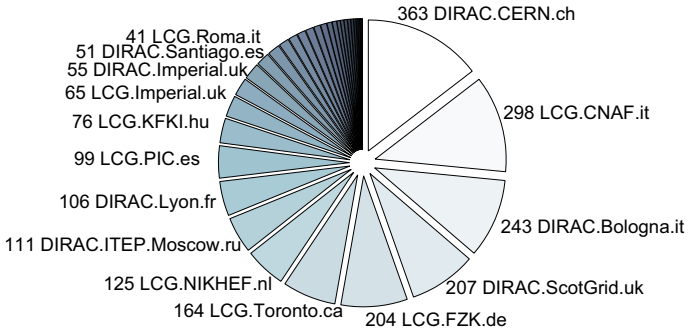
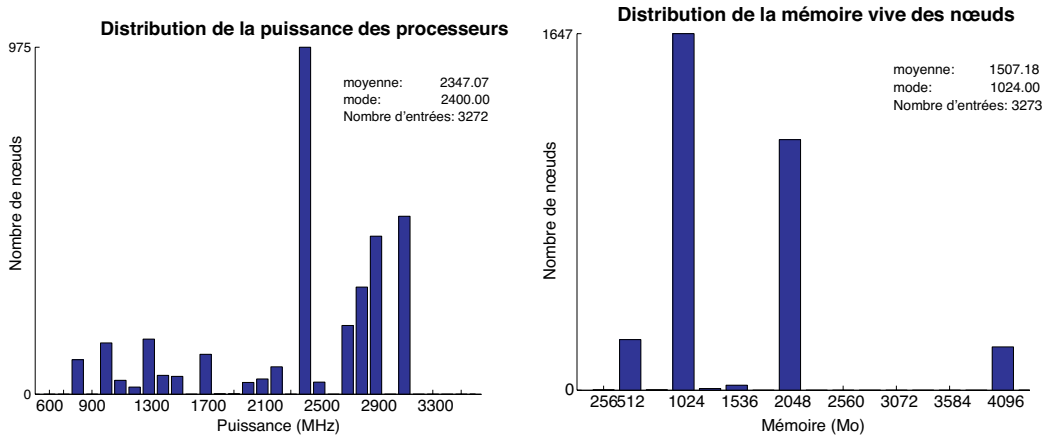


FIG. 4.2 – Répartition des tâches en exécution par site (Image générée par le service de surveillance).

Le recensement des ressources utilisées indique que *DIRAC* a connecté 5.743 nœuds différents lors du DC04. Les figures 4.3 illustrent les caractéristiques de ces nœuds sur la plate-forme. Ces caractéristiques sont la puissance des nœuds (voir figure 4.3(a)), exprimée en GHz, et la mémoire vive disponible en méga-octets (voir figure 4.3(b)).



(a) Distribution de la puissance des processeurs.

(b) Distribution de la mémoire vive des machines.

FIG. 4.3 – Caractéristiques des nœuds de la plate-forme.

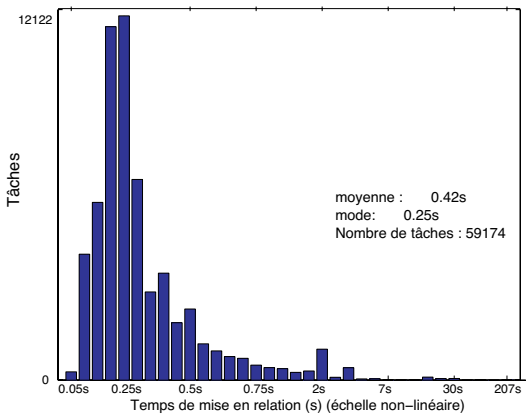
Pour un échantillon de 3270 machines, la répartition du nombre de processeurs par machine est la suivante : 2% de mono-processeurs ; 80 % de bi-processeurs ;

18% de quadri-processeurs. Une majorité des machines sont des bi-processeurs et la puissance moyenne se situe autour des 2 GHz, avec en moyenne une mémoire vive de 1024 méga-octets. L'architecture de processeur la plus répandue est le Pentium IV.

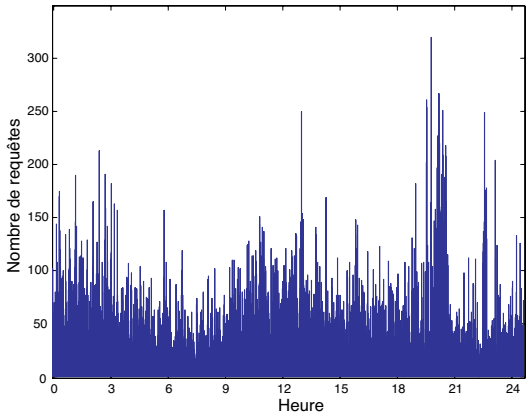
4.3 Performances de *DIRAC*

Contrairement à un système de type « push », où la charge sur les services est exclusivement initiée par les créateurs de charge, les services d'un système « pull » sont surtout sollicités par les ressources et dans beaucoup de cas par les nœuds directement. Cette caractéristique oblige de prêter une grande attention à la robustesse des services, ainsi qu'à leurs temps de vie.

Le système se révèle très robuste car les services multi-threadés XML-RPC, conjugués avec des caches de connexion de base de données, sont capable de soutenir un grand volume de requêtes concurrentes, comme illustré sur la figure 4.4(b) pour le service configuration. Sur cette figure, nous observons une charge constante sur ce service avec une moyenne de 50 demandes et des pics atteignants les 200.



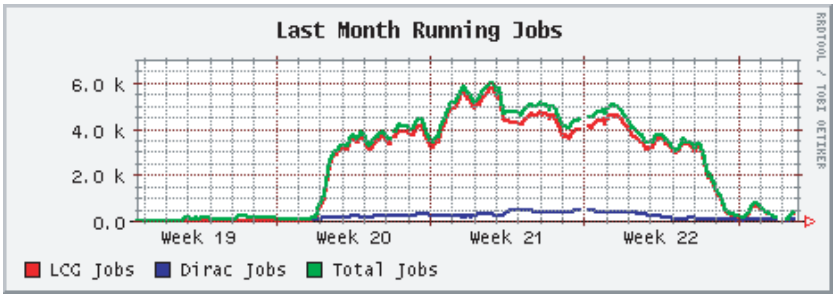
(a) Distribution du temps de mise en relation tâche-ressource pour 223.000 tâches lors du DC04.



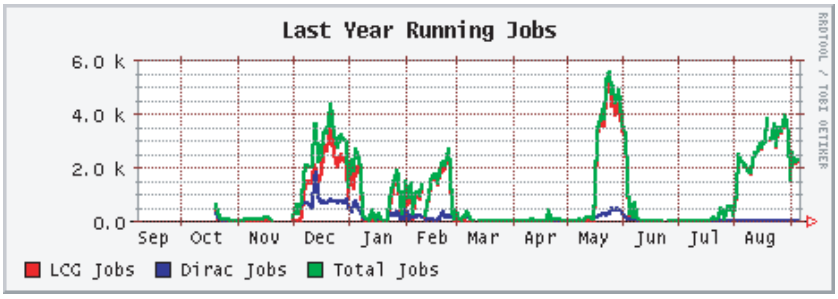
(b) Requêtes par minute sur le service configuration sur une période de 24 heures.

FIG. 4.4 – Caractéristiques du service correspondance et de configuration

La figure 4.4(a) montre le temps de service pour la mise en relation tâche-ressource lors du DC04. Le temps moyen de mise en relation est de $0.42s$ et 85% des opérations de mise en relation prennent moins de deux secondes avec plus d'une dizaine de milliers de tâches en attente. Le nombre de tâches pouvant être exécutées simultanément avoisine les 6.000 tâches avec une moyenne située autour de 4.000 comme illustré sur la figure 4.5. Le système supporte plus de 5.500 tâches qui s'exécutent de façon concurrente et reportent leurs états.



(a) sur un mois.

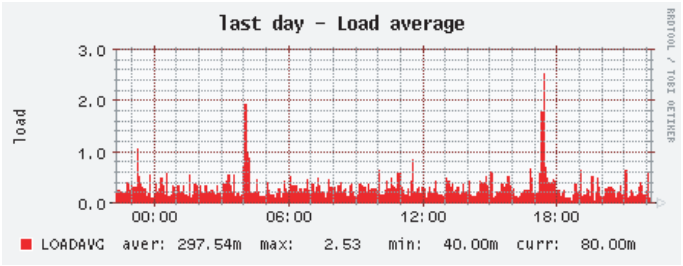


(b) sur une année.

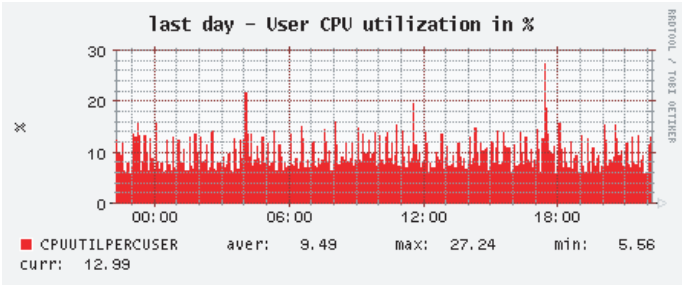
FIG. 4.5 – Évolution du nombre de tâche en exécution.

L'ensemble des services centraux sont hébergés sur un seul serveur au CERN. Les traces d'utilisation de la machine hôte montrent comme illustré sur la figure 4.6 que le serveur n'est absolument pas saturé avec une charge inférieure à 0.5 et une utilisation du processeur inférieure à 10%. Il supporte bien la charge au niveau processeur. Les caractéristiques modestes du serveur, un Intel Xeon 2.40 MHz avec deux giga-octets de mémoire vive, indique que nous sommes encore loin des limitations du système, car la mise à l'échelle n'a pas encore révélé les limitations du système. D'autant que le déploiement des services est possible sur plusieurs machines pour augmenter la robustesse.

La figure 4.7 montre le diagramme de Gantt de l'utilisation des ressources de juillet à octobre 2004. Les nœuds sont regroupés par site et l'échelle croissante de noirceur est corrélée à la densité d'utilisation du site. Nous voyons que certains sites sont utilisés de manière intensive. Ceux-ci sont habituellement dédiés. D'autres sites ont des participations plus sporadiques. Certaines zones blanches sont liées à l'arrêt d'utilisation des ressources *LCG* causés par les problèmes rencontrés (voir section 4.7). La constatation générale est que l'environnement est fortement dynamique, justifiant l'approche réactive et opportuniste de *DIRAC* pour la gestion des tâches.



(a) Charge du processeur.



(b) Utilisation du processeur.

FIG. 4.6 – Statistique du processeur du serveur hébergeant les services centraux.

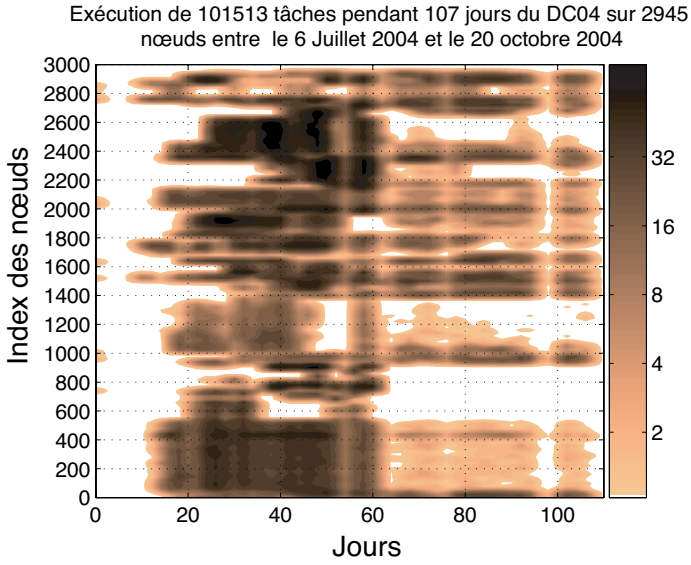


FIG. 4.7 – Diagramme de Gantt de l'occupation des nœuds par les tâches de production. Chaque ligne horizontale représente un nœud unique, groupé par site. La densité des tâches est représentée par l'échelle de noirceur.

4.4 Caractéristiques des tâches

Les tâches du DC04 sont des simulations de Monte-carlo ou qualifiées de tâches multi-paramétriques [51]. Plus de 40.000 tâches se sont exécutées lors du mois de mai 2004. La durée moyenne d'exécution est de 20 heures. La figure 4.8 exprime le temps de consommation CPU en fonction du nombre d'heures où la tâche est restée en exécution sur le nœud. La noirceur croissante reflète la densité des tâches. Cette droite se rapprochant fortement d'une droite affine, nous concluons que la consommation CPU est proche de 100% et qu'une petite portion de tâches souffre de délais occasionnés par des transferts ou des requêtes bloquées ou très lentes. En moyenne, l'exécution d'une tâche se compose de 93% de temps purement calculatoire et de 7% d'opérations d'entrées/sorties ou d'attente de la disponibilité d'autres ressources. Chaque tâche produit en moyenne 400 méga-octets de données, qui sont ensuite répliquées.

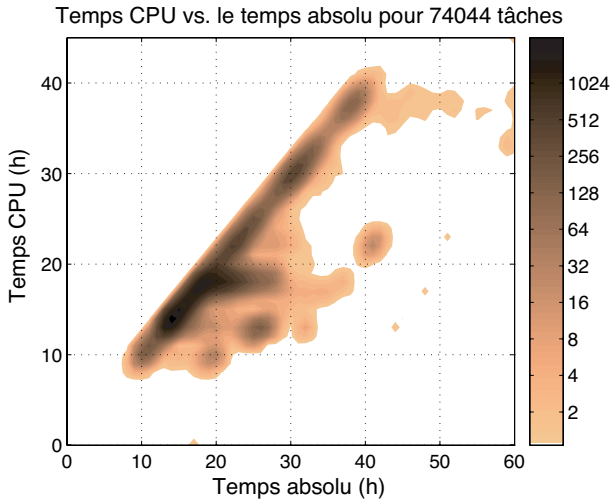


FIG. 4.8 – Consommation du temps CPU en fonction du temps total d'exécution pour 74.044 tâches.

4.5 Les caractéristiques relatifs aux données

La gestion et la régulation de la charge de calcul pour la production, à travers le système de gestion de ressources, sont traitées efficacement. Par contre, la gestion des données représente une grande problématique au sein de *DIRAC*. La politique appliquée est que toutes les données produites sont rapatriées et stockées sur le site *Tier-0* qui est le CERN. Les données sont aussi partiellement répliquées sur les sites classifiés comme *Tier-1* pour faciliter leur analyse. Cette classification des sites comporte quatre niveaux ou « Tiers ». Chaque niveau répond à des fonction-

nalités et spécificités précises relatives à l'importance du site et à leurs rôles pour l'exploitation de *LHCb* (voir section 1.7).

La liste des sites classés comme *Tier-1* pour *LHCb* est donnée dans le tableau 4.1. Celui-ci donne le récapitulatif des données produites et stockées après le DC04. Nous indiquons la répartition des données par site. Ces données représentent seulement les données contenant les objets physiques reconstruits pour *LHCb*. Les fichiers intermédiaires ne sont pas pris en compte. Si nous comptabilisons toutes les données produites en incluant les données intermédiaires ce montant s'élève à *72 téra-octets* de données pour le DC04.

	Centre de calcul	**Quantité des données(TO)**
Tier 0	CERN (Suisse)	12.6
Tier 1	CNAF (Italie)	12.6
	RAL (Angleterre)	6.5
	PIC (Espagne)	5.4
	FZK (Allemagne)	4
	CC/IN2P3 (France)	1.5

TAB. 4.1 – Récapitulatif du montant des données produites et stockées sur les sites participants au DC04.

La figure 4.9 montrent la distribution du taux de transfert des données mesuré entre les sites *Tier-1* et *Tier-0* (CERN) pour toutes les tâches du DC04. Pour chaque tâche, nous déterminons le rapport entre le montant transféré et le temps requis pour le transfert entre le site *Tier-1* et le CERN. Le taux de transfert moyen par tâche est de 5 méga-octets par seconde.

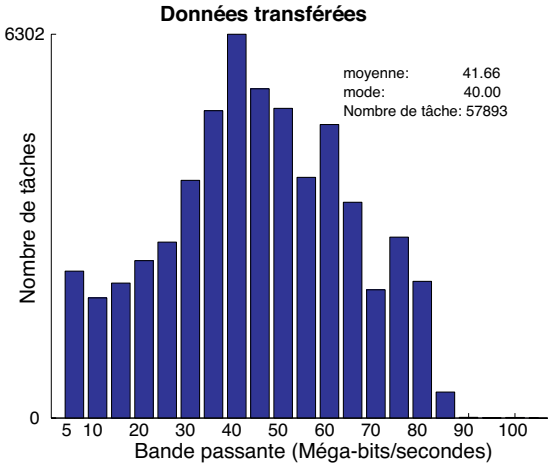


FIG. 4.9 – Distribution du taux de transfert moyen des données par tâches lors du DC04 entre les sites *Tiers-1* et le CERN(*Tiers-0*).

Parmi les grands problèmes rencontrés, les délais de transfert furent important car un nombre conséquent de sites ont subi des pannes réseaux. Le résultat de ces défaillances est l'accroissement de la taille des files d'attente locales de transfert. Les sites se retrouvent rapidement avec des files d'attente de transfert supérieures à 40 giga-octets à transférer. Le recouvrement de ces transferts génère une saturation de la bande passante sortante du site. Ces lacunes impliquent une gestion plus fine du réseau et l'utilisation de sondes logicielles ou senseurs et d'outils comme NWS [126], VISPERF [142], NetLogger [141] et GANGLIA [94]. Ces outils de mesure ou de prédiction sont inévitables pour un système de gestion de données complexe et perfectionné.

4.6 L'inter-opérabilité avec *LCG*

Nous présentons les résultats détaillés de l'utilisation de *LCG* comme expliqué en section 3.9. *LCG* met en place un ordonnancement centralisé et une organisation de type « push », diamétralement opposée à celle de *DIRAC*.

L'exécution de 123.000 tâches avec *LCG* fournit un riche ensemble de mesures de performances. La relative homogénéité des ressources de *LHCb* facilite l'analyse des performances. Les statistiques, liées à l'utilisation de *LCG* du 16 juillet au 27 octobre 2004, sont montrées dans la figure 4.10.

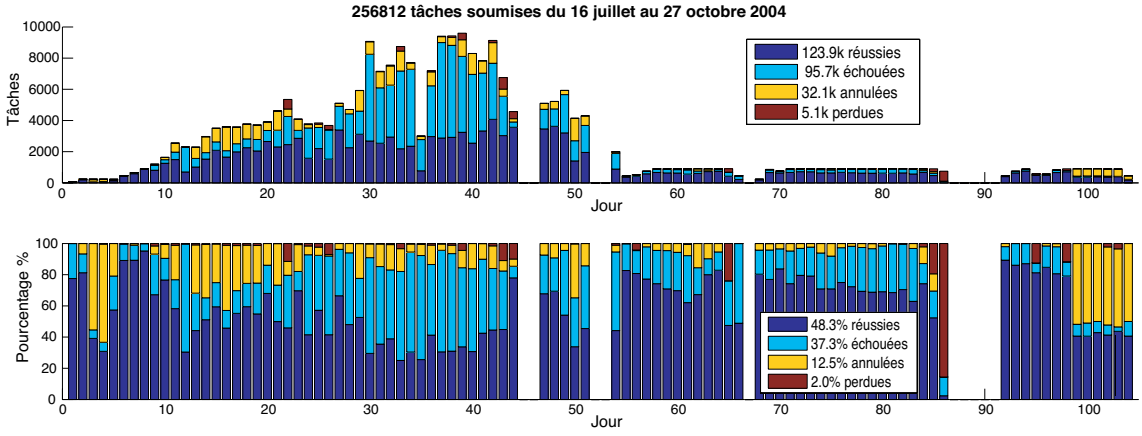


FIG. 4.10 – Résultats des tâches soumises à *LCG* durant le DC04.

Sur l'ensemble du DC04, le taux de réussite mesuré de *LCG* est de 61%. Celui-ci reflète le rapport du nombre de tâches soumises sur celui des tâches considérées par *LCG* comme accomplies avec succès. Ce taux de réussite s'applique sur les agents *pilotes* soumis au système. Dans l'approche indirecte *DIRAC*, les agents *pilotes* sont soumis par l'agent *maître* déployé sur l'interface utilisateur du système et une

fois sur le nœud ces agents *pilotes* demandent des tâches au service *correspondance* (3.9).

La figure 4.11 montre les performances du méta-ordonnanceur ou le « Resource Broker » de *LCG* pour la soumission et la mise en relation tâche-ressource. Le temps moyen d'une tâche de la soumission à l'allocation sur une ressource est de 50 secondes, soit presque une minute par tâche. Si nous comparons ce chiffre avec celui de *DIRAC* (figure 4.4(a)), nous trouvons un facteur cent ! Il faut noter que les temps de soumission sont négligeables dans *DIRAC* et qu'il est directement exposé à une haute charge et à des soumissions à haute fréquence. Tandis que dans l'approche indirecte prônée dans *LCG*, le taux de soumission est contrôlable et stable car il correspond à des agents soumis de manière adaptée et contrôlée par *DIRAC*.

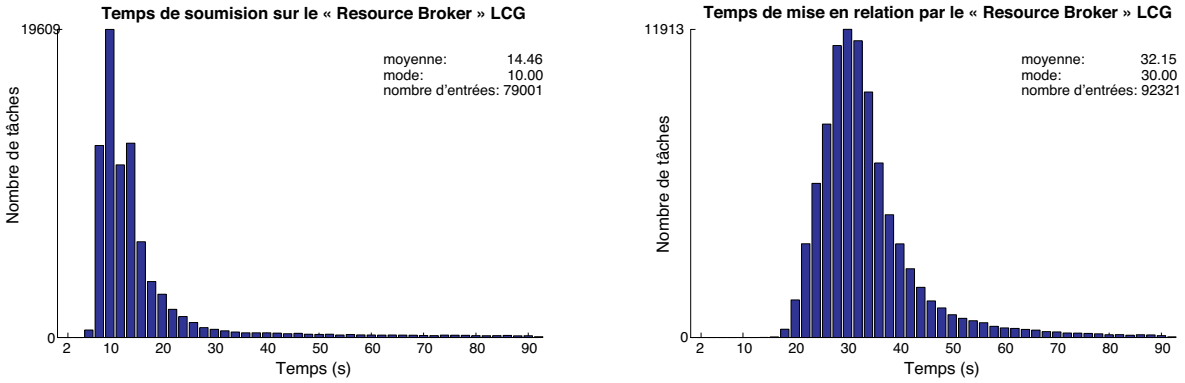


FIG. 4.11 – Distribution des temps de soumission et d'ordonnancement du méta-ordonnanceur de *LCG* le « Resource Broker » pour 100.000 tâches.

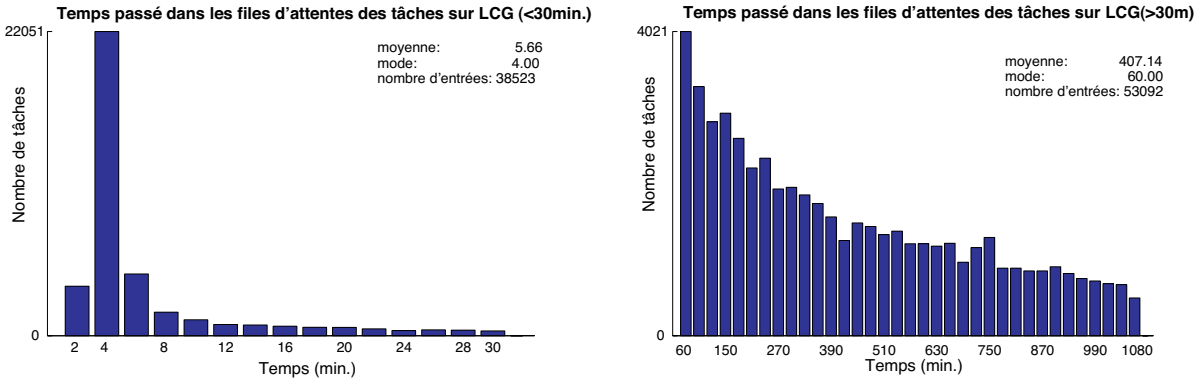


FIG. 4.12 – Distribution du temps d'attente sur les files d'attente des sites.

La figure 4.12 montre la distribution du temps d'attente des tâches sur les sites. Nous voyons clairement que l'attente est bien moindre pour les tâches courtes

inférieures à 30 minutes (environ 5 minutes), que pour les tâches longues (près de 6 heures d'attente). Si nous considérons que ces deux tailles correspondent à des files d'attente distinctes, nous pouvons supposer que le système est plus occupé par de longues tâches, ce qui se traduit par un grand temps d'attente dans ces files. Les petites tâches occupent moins le système. Les temps d'attente résultants sont donc moindres. En outre, nous pouvons supposer que des sites ont des politiques qui favorisent les tâches courtes comme *Shortest-Job-First* [121].

La figure 4.13 présente l'évolution du nombre de tâches en exécution avec *DIRAC* en fonction du temps. Nous présentons ces résultats avec les ressources *LCG*, hors *LCG* et cumulées.

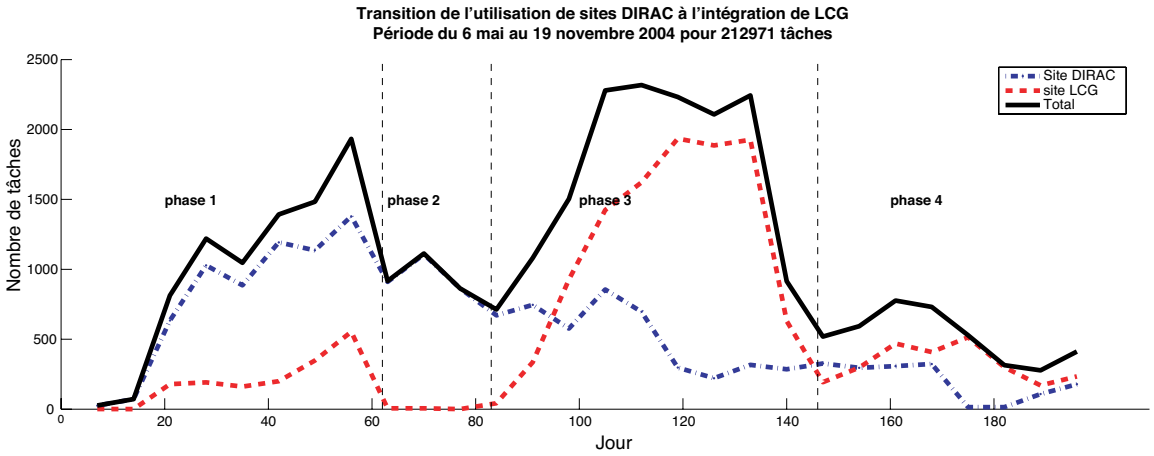


FIG. 4.13 – Transition des tâches sous systèmes de traitement par lots aux tâches *LCG* (seulement les tâches exécutées avec succès sont représentées).

Nous observons quatre phases distinctes du système lors de l'interface avec *LCG* comme représentées sur la figure 4.13.

Phase 1. Ici *DIRAC* a été interfacé avec *LCG* pour la première fois ce qui explique l'augmentation de la pente de la courbe. Les sites *DIRAC* et *LCG* sont rajoutés au fur et à mesure dans le système ce qui explique la courbe en augmentation. Le fort dégré de la pente sur certaines périodes témoigne de la rapidité du système à intégrer et utiliser les ressources. Après un mois d'utilisation, l'arrêt d'utilisation de *LCG* a été adopté. Le but de cette pause était de résoudre un nombre conséquent de problèmes sur *LCG* comme détaillé plus loin en section 4.7.

Phase 2. Durant la deuxième phase, l'équipe *DIRAC* a travaillé de concert avec l'équipe *LCG* pour résoudre les problèmes identifiés lors de la phase une. Pour cet exercice, nous utilisions une plate-forme d'expérimentation sans utiliser les ressources *DIRAC*.

Phase 3. Après la correction des problèmes de jeunesse, il a été possible d'utiliser *LCG* rapidement. Nous observons une pente encore plus prononcée que dans la phase une jusqu'à l'obtention d'un plateau de 40 jours qui correspond à un régime stable et de saturation de toutes les ressources du système *LCG* et les ressources *DIRAC*. La plateau de saturation s'illustre encore mieux pour les ressources seules de *DIRAC* du jour 130 à 170. Ceci est une illustration de l'idéal d'un système de régime permanent et saturé où toutes les ressources sont utilisés. La pente descendante des tâches *LCG* s'explique par l'arrêt progressive du système.

Phase 4. Le but de cette phase était de produire le montant de données exigé par *LHCb* plutôt que de tester les infrastructures des différents systèmes. Cette phase fut suivi d'une pause pour évaluer le système et continuer à un taux bas mais constant d'utilisation du système jusqu'à l'aboutissement des objectifs.

4.7 Les défaillances et pannes observées

Peu de pannes furent observées dans le logiciel de *DIRAC*. Celles-ci apparurent au début du DC04 et furent rapidement corrigées. Ces modifications se focalisèrent sur l'extensibilité des services centraux. Parmi ces réglages, nous avons dû augmenter le nombre de connexions disponibles pour un serveur MySQL. Un autre problème est intervenu pour le stockage des fichiers de paramètres des tâches et de leurs sorties ou « input/output sandbox ». Le système de fichiers *Posix* est limité à $2^{15} = 32.000$ entrées. Ce qui empêche d'avoir plus de 32.000 tâches dans le système. Pour résoudre ce problème, nous avons mis à plat un système de fichiers avec l'aide d'une base de données.

Un autre problème rencontré fut la non-disponibilité des services centraux et pour quelques uns leur unicité. Par exemple en cas de panne réseau comme nous l'avons observé beaucoup de tâches échouaient et les ressources souffraient de famine. Ces incidents incitèrent l'augmentation de la disponibilité des services centraux et de réduire les dépendances des tâches et agents au services centraux pour rendre possible l'attente du retour à la normale des services centraux sans trop de préjudice sur l'intégrité du système.

En dehors de ce problèmes mineurs, l'utilisation de *LCG* révéla d'autres problématiques lors de son intégration avec *DIRAC*.

4.7.1 La normalisation des files d'attente et des puissances de calcul

La soumission à *LCG* inclut une estimation de la durée de la tâche. Cette estimation s'utilise ensuite pour l'allocation de la ressource. Les ressources sont des systèmes de traitements par lots mettant en place des files d'attente. Une file d'attente possède des attributs caractéristiques comme la durée maximale d'une tâche sur un nœud de calcul et l'ensemble des nœuds associés de la grappe. Cette

durée maximale est intrinsèquement liée à la puissance de l'ensemble des nœuds liés à une file d'attente. L'expression de la durée d'une tâche pour l'allocation n'a une signification que si une normalisation du temps d'exécution des tâches existe. Cependant aucun mécanisme convenable n'existe pour résoudre ce point précis. Certains sites normalisent de manière interne le temps maximal d'exécution de leurs files d'attente. La répercussion de ces définitions locales est que certaines tâches *LHCb* s'exécutent avec une réservation insuffisante de temps de calcul. De plus des sites surchargent des processeurs en autorisant l'exécution de plusieurs tâches ou utilisent l'*hyper-Threading*. Ces techniques rendent encore plus difficile l'estimation du temps de calcul disponible. La conséquence directe de ce manque d'homogénéité est la fin prématurée de milliers de tâches.

La mise en relation de la tâche-ressource directement à partir du nœud, permie d'apporter une solution à ce problème. Avant d'utiliser un nœud, l'agent effectue une rapide vérification entre le temps défini pour la file d'attente et celui effectivement mesuré sur le nœud. Pour ceci, nous utilisons un jeu d'essai ou « benchmark ». Celui-ci, nous donne une mesure de normalisation de puissance du nœud, utile pour faire ensuite une demande au service correspondance.

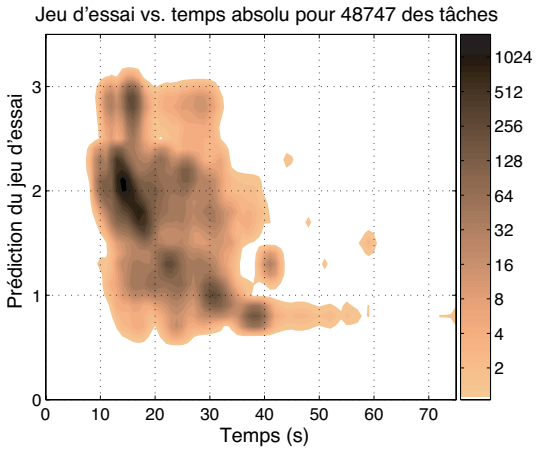


FIG. 4.14 – Puissance mesurée par un jeu d'essai *LHCb* en fonction du temps d'exécution de tâche.

La figure 4.14 montre la corrélation entre le jeu d'essai et le temps d'exécution des tâches. La prédiction est quasi-linéaire pour une majorité des tâches ce qui valide la pertinence du jeu d'essai.

4.7.2 Les espaces disque de travail

Les tâches *LHCb* requièrent un espace disque de 500 méga-octets à 1.5 giga-octets. Les espaces de travail qui utilisent un système de fichiers distribués tel NFS sont souvent surchargés quand plusieurs couples nœuds-tâches les sollicitent. Des

sites ne fournissaient pas assez d'espace pour la terminaison des tâches. Le cas d'une machine hébergeant deux processeurs est aussi souvent problématique car deux tâches peuvent de manière concurrente partager le même espace disque et ainsi échouer par manque d'espace. Ces pannes entraînent alors un effet qualifié de « trou noir ». Si un nœud d'un site possède un espace disque de travail insuffisant, la tâche allouée à ce nœud échouera. Elle sera remplacée par une tâche qui à son tour échouera et ainsi de suite. Un nœud « aspire » donc rapidement une quantité monumentale de tâches. La difficulté est de traiter ces états inconsistants, de resoumettre ces tâches et surtout de bannir du système le site défectueux. Nous avons implémentés chacun de ces mécanismes dans *DIRAC*.

4.7.3 Disponibilité des fichiers de sorties après une défaillance

Après l'exécution d'une tâche, les fichiers de sorties ou « output sandbox » sont retournés à l'utilisateur. Malheureusement, si une erreur intervient pendant l'exécution, les fichiers de sorties sont effacés ou non enregistrés par le système. Ces fichiers sont souvent nécessaires pour diagnostiquer les causes d'un échec, mais ceux-ci ne sont pas disponibles dans le cas d'une terminaison anormale. De plus, les messages d'erreurs de *LCG* dans ces situations sont trop limités pour déterminer la cause exacte d'un échec. Pour ceci, ces erreurs obligent souvent des interactions longues et directes avec l'administrateur de site incriminé si l'échec intervient au niveau d'un site ou une analyse des journaux des serveurs *LCG*.

4.7.4 La sécurité

LCG repose sur l'infrastructure de sécurité GSI (Globus Security Infrastructure) [20] qui se base sur les certificats x509. L'utilisateur avant de soumettre des tâches doit s'identifier auprès de l'infrastructure de sécurité. Cette phase authentification et de vérification de ces droits génère un « proxy » ou mandataire de sécurité pour une durée fixée. Cette procuration permet de transmettre l'identité et d'autres méta-informations sur l'utilisateur. Le mandataire est transmis avec la tâche pour toute la chaîne de soumission, de l'interface utilisateur, à l'ordonnanceur pour finalement l'authentification sur un site et l'exécution de la tâche sur un nœud. Le premier inconvénient de ces mécanismes est qu'ils ne permettent pas de partager des détails sur les tâches au sein d'une même organisation virtuelle. Ceci rendait difficile le travail collaboratif. Des défaillances ont aussi été trouvées avec la gestion des mandataires de tâches. Cette gestion générait des tâches avec des mandataires expirés et avait pour conséquence l'échec de ces tâches. Pour éviter cette situation des tâches furent annulées manuellement avant l'expiration de la durée des mandataires. La mise en place de serveur de renouvellement automatique de mandataire comme des serveurs *Myproxy* [161] est aussi une solution à cette problématique.

4.7.5 Un ordonnancement non adapté

L'ordonnancement mis en place dans *LCG* répondait faiblement à une problématique de régime permanent et saturé. Les stratégies d'ordonnancement de *LCG* s'appliquent dans un mode « push » avec l'utilisation d'un ordonnanceur centralisé, le « Resource broker », se basant sur un système d'information. Les algorithmes de mise en relation tâche-ressource appliquent des critères qui ne favorisent pas l'utilisation maximale de la plate-forme. Le *Resource broker* sélectionne d'abord les ressources candidates puis ensuite utilise un critère pour déterminer la pertinence de la combinaison « ressource/tâche » pour ensuite attribuer des valeurs avec une relation d'ordre. Il choisit ensuite la combinaison jugée comme meilleure pour lui. Un des critères le plus souvent utilisé est de choisir l'estimation du temps de réponse minimal d'un site. Cette stratégie est aussi qualifié de ETT (*Estimated Traversal Time*) [110]. Chaque site calcule cette valeur, puis la publie ensuite dans le système d'information. Les méthodes pour calculer cette valeur se basent sur l'historique et des méthodes de prédiction [160]. Bien que souvent utilisé et efficace au niveau local [162,163], la prédiction dans un contexte global a un effet moins bénéfique. En effet, une mauvaise prédiction d'un site cumulée avec des erreurs de configurations amènent souvent à la surcharge de site et la famine d'autres sites. Cette situation est inappropriée dans un contexte de régime permanent où nous souhaitons une utilisation maximale de chaque ressource disponible.

Les outils *LCG* se sont de plus révélés insuffisants pour traiter des requêtes pour un large volume de tâches. Pendant le DC04, des milliers de tâches ont été soumises chaque jour. Il était difficile de surveiller et de gérer ces tâches.

4.7.6 Les erreurs lors de l'exécution

Pourcentage	Intitulé
70%	Succès
14%	Pas de tâche disponible au niveau du service correspondance
4%	Exception
4%	Erreur d'installation du logiciel applicatif
2%	Erreur/*bug* du logiciel applicatif
1%	Erreur d'initialisation
<1%	Plus d'espace disque disponible
<1%	autres

TAB. 4.2 – Proportions des erreurs et succès survenus lors de l'exécution des agents *pilotes* sur *LCG*.

Avec l'approche de *DIRAC* où nous soumettons des agents *pilotes* et effectuons l'opération de mise en relation sur le nœud, nous contournons les erreurs précédemment énoncées car ces erreurs affectent directement les agents *pilotes* et non

les tâches *LHCb*. Néanmoins une fois sur le nœud, nous rencontrons aussi des problèmes. Le tableau 4.2 donne la proportion des erreurs et leurs causes.

Par l'approche indirecte, le taux de réussite de *LCG* mesuré de 61% pour les agents *pilotes*, passe à 70% pour les tâches *LHCb*. Nous observons une défaillance mesurée à 14% d'agents *pilotes* qui meurent suite à une réponse négative à leurs demandes de travail. C'est le coût payé par l'approche indirecte. En dépit que ces agents occupent la ressource pour quelques secondes et donc ne chargent pas le système.

4.8 Conclusions

L'architecture orientée service prouve que la flexibilité offerte par cette approche permet un rapide développement et intégration dans un système distribué. Le paradigme « pull » et le modèle Agent/Service est bien passé à l'échelle car nous n'avons pas vu de problèmes majeurs avec plus de 5500 tâches en exécution concurrente. Par contre, ce modèle nécessite de prêter une grande attention à la robustesse des services sollicités par tous les nœuds, au mécanisme de détection et traitement d'erreurs et de collection de résultats. Une plate-forme réelle a un comportement dynamique auquel *DIRAC* s'adapte bien. L'utilisation de *LCG* avec l'approche indirecte où nous utilisons le service de gestion de ressource de *LCG* comme système de déploiement pour nos agents *pilotes* prouve les intérêts de cette stratégie. Nous obtenons alors un gain de près de 10% sur le taux d'erreur de *LCG* avec un faible surcoût. En plus de diminuer, le taux d'erreurs pour les tâches, nous nous agrégeons et approprions un système de type « push » pas adapté à un contexte de régime permanent et saturé en constituant notre ensemble de nœuds *DIRAC* et appliquant de façon parallèle notre gestion des ressources.

Bien que ce paradigme marche bien avec des applications multi-paramétriques, il reste néanmoins des questions sur des applications qui nécessitent des bons temps de réponse au niveau individuel, d'équité entre utilisateurs et son influence sur le partage des ressources entre organisations virtuelles. Une problématique intéressante est de déterminer si le système *DIRAC* peut utiliser plusieurs critères d'ordonnancement. Les limites du système réel pour l'évaluation des stratégies et la nécessité d'outils d'analyse plus performants est évidente.

CHAPITRE 5

MODÉLISATION ET SIMULATION

Résumé du chapitre :
Dans ce chapitre, nous traitons du problème de la modélisation d'un SDGE*, de son architecture et de l'évaluation du (méta-)ordonnancement. Nous présentons une modélisation et un simulateur de grands systèmes de calcul distribué. Une telle plate-forme se compose de grappes de processeurs hétérogènes appartenant à un réseau local, c'est-à-dire des centres de calcul ou sites, interconnectées entre elles par un réseau global. Ces grappes sont accessibles via un ordonnanceur local et sont partagées entre les utilisateurs. Nous traitons aussi ici la validité de notre outil de validation pour nos expérimentations.*

5.1 Introduction

Une plate-forme de *SDGE* hétérogène et non dédiée a un comportement dynamique, instable et non prévisible, comme nous l'avons montré précédemment avec la plate-forme de *DIRAC*. Dans ces conditions, il est très difficile, voire impossible d'avoir recours à des expériences grandeur nature pour évaluer des architectures, des stratégies et des heuristiques d'ordonnancement. De par la non-reproductibilité des expériences, aucune comparaison rigoureuse ne peut être entreprise. De plus, l'aspect hiérarchique de l'ordonnancement, où des sites autonomes gèrent eux-mêmes l'allocation des tâches sur les nœuds, complique cette évaluation, ainsi que celle d'une architecture globale et d'un méta-ordonnanceur.

Ce chapitre présente les enjeux et les problèmes liés à l'évaluation d'un méta-ordonnancement avec une plate-forme de *SDGE*. Nous présentons premièrement les outils disponibles pour la comparaison d'ordonnancement en section 5.2 et puis particulièrement les technologies de simulation (section 5.3) dont plus en détail *Simgrid* (section 5.4). Nous présentons ensuite notre modèle de performances en section 5.5. Enfin, nous exposons le simulateur basé sur *Simgrid* implémentant cette modélisation dans la section 5.6 et les travaux relatifs à sa validation (section 5.7) puis nous concluons.

5.2 Les outils d'évaluation

Les outils disponibles pour l'évaluation et la compréhension d'un *SDGE* par ordre croissant de réalisme sont les outils mathématiques, les techniques de simulation puis l'émulation et enfin les systèmes réels [104].

Les études mathématiques, basées sur des programmes linéaires ou des modèles stochastiques, permettent de trouver les bornes supérieures ou inférieures d'un

algorithme d'ordonnancement. Mais ces études sont réalisées sur des modèles simplifiés, restrictifs et souvent non réalistes, alors qu'un *SDGE* est une combinaison d'éléments statiques et dynamiques tel que le réseau.

L'émulation est un autre outil qui pallie à ce problème. Il est utilisé pour expérimenter les systèmes et les réseaux avec un plus fort degré de réalisme. En effet, l'émulation consiste à exécuter des expérimentations sur des plates-formes réelles où les paramètres statiques et dynamiques sont contrôlés et gérés avec précision. Comme projet actuel, nous pouvons citer le banc d'essai pour l'émulation du réseau *Netbed* basé sur *Emulab* [112] et *DummyNet* [106] puis le projet *Wan in Lab* [104]. Ces projets se focalisent sur le réseau contrairement au projet français *Grid eXplorer* [105] plus généraliste. Ce projet est associé au projet *Grid 5000* [182].

En analogie avec les grands projets de physique des particules tel *LHCb*, ce projet souhaite être un outil de mesure grandeur nature, stockant les paramètres d'expérimentations comme les caractéristiques réseaux dans une base de données.

La perte des conditions expérimentales ainsi que la non reproductibilité des expériences sont les différences principales entre l'émulation et les projets réels. Ceux-ci sont des systèmes en production sur des plates-formes non dédiées comme *DIRAC*.

Aucun outil d'émulation n'était disponible au moment des travaux présentés dans ce manuscrit. D'autre part, la durée nécessaire à un test grandeur nature est souvent longue et présente un coût élevé. L'instabilité d'une plate-forme non dédiée interdit l'expérimentation comme moyen de comparaison. Pour cette étude nous avons décidé d'utiliser de la simulation de système distribué à grande échelle.

5.3 La simulation

La simulation d'un système de calcul distribué se base sur un modèle empirique ou mathématique d'un système réel qui prédit le comportement du système à partir de paramètres et de conditions initiales. La simulation est utile pour beaucoup de raisons. Par exemple, elle permet de faire facilement varier les caractéristiques des plates-formes et des applications, en apportant des résultats quasi immédiats.

Il existe aujourd'hui un nombre croissant de simulateurs disponibles. Nous présentons une sélection des outils de simulation, souvent couplés avec des projets réels.

MicroGrid [99] simule spécifiquement les environnements basés sur Globus. Il permet d'exécuter des applications réelles sur un banc d'essai virtuel. Le simulateur Bricks [100] associé avec le projet de *NES Ninf* [24] simule des systèmes qui utilisent un modèle client-serveurs. Il se focalise plus particulièrement sur les performances du système. OptorSim [101] se base sur une modélisation de l'environnement du projet DataGrid. Il a été construit initialement pour l'étude de la gestion des données comme la réplication. Le modèle a été étendu pour l'ordonnancement de tâches.

Ce projet était rattaché au projet de EDG [49].

Le projet MONARC [183] est un outil de simulation basé sur des événements discret en java. Ce projet se destine directement à l'étude d'un système distribué à grande échelle pour les besoins du LHC et de ses physiciens, notamment pour l'analyse des données. Ce simulateur permet d'évaluer les différentes architectures de traitement des données en compte et prend aussi le comportement et les schémas d'accès aux données par les physiciens.

La librairie *Ptolemy II* [115] a été aussi utilisée pour la simulation de l'environnement distribué de DataGrid [110]. ChicSim [103] est aussi un système pour la simulation de grilles de données. Il repose sur Parsec [114] et permet l'autorisation d'ordonnancement couplés avec l'aspect données et réseau.

GridSim [108] a été créé pour étudier les stratégies d'allocation dans un modèle économique [42]. Il s'appuie sur JavaSim [113].

PANDA [109] permet la simulation d'applications parallèles au niveau d'un cluster. HuskySim [107], et BeoSim [102] traitent aussi des clusters mais dans un contexte élargi aux grilles de calculs et de méta-ordonnancement.

Les projets existants sont trop spécialisés pour un modèle particulier ou pas assez aboutis pour être utilisés. Pour le travail de simulation décrit dans cette thèse, nous avons utilisé *Simgrid* [43, 44]. Nous avons choisi *Simgrid* du fait de sa grande utilisation pour la simulation d'ordonnancement des grilles de calculs et surtout sa facilité d'implémentation de nouveau modèle et scénarios d'ordonnancement, utile pour évaluer tous les aspects d'ordonnancement, de méta-ordonnancement, de déploiement de *DIRAC*. De plus, *Simgrid* fournit un rapport temps de la simulation sur le temps simulé supérieure de plusieurs ordres de grandeur inférieur à de l'émulation. Par exemple, ce rapport est de l'ordre de 10^{-6} pour un ordonnancement maître-esclave avec des tâches simples et gros grain sur une plate-forme avec une cinquantaine de machines et un Pentium 3 avec un processeur 1 GHz comme machine de simulation.

5.4 Mise en œuvre en *Simgrid*

Simgrid propose donc un simulateur à base d'agents communicants pour pouvoir simuler des ordonnanceurs décentralisés, centralisés, hiérarchiques et étudier des politiques locales d'ordonnancement. *Simgrid* est un outil à événement discret fournissant des fonctions de base qui permettent de modéliser, de décrire et de créer un simulateur. Nous présentons ici les concepts principaux :

Un agent est défini par un code possédant des données privées et un lieu d'exécution (ou hôte). La notion d'agent ou processus est importante pour simuler des décisions d'ordonnancement indépendantes les unes des autres.

Un hôte est l'endroit où peut s'exécuter un agent. Il est représenté par une capacité de calcul, des boîtes aux lettres dans lesquelles des agents cherchent

des messages, et des données privées pouvant être accèdées par les agents s'exécutant sur ce site.

Une tâche est traitée localement par un agent ou transférée vers un autre site. La plupart des algorithmes d'ordonnancement repose sur la notion de tâche. Elle est définie par une quantité de calcul nécessaire à son traitement, par la taille des fichiers nécessaires à sa migration et par des données privées. Notons qu'avec ce type d'expression d'un ordonnancement, une tâche n'est pas exécutée ou transférée par exemple dès qu'une ressource est libre mais dès qu'un agent décide de l'exécuter ou de la transférer.

Une route est une agglomération de ressources de communications représentant des liens physiques. Il n'y a pas de routage dans *Simgrid*. Chaque paire d'hôtes est reliée par une route et les communications d'un agent à l'autre se font donc en n'ayant pas d'accès direct aux ressources de communication.

Un canal est un numéro de boîte aux lettres. Cette notion ressemble à celle de port TCP et permet à certains agents de communiquer entre eux sans interférer avec les autres,

Selon ce formalisme, l'ordonnancement se décrit alors en terme d'agents s'exécutant sur des hôtes et interagissant en émettant, recevant ou traitant des tâches. L'envoi et la réception se fait à l'aide des canaux. Un agent peut envoyer une tâche sur le canal d'un hôte et en recevoir une en consultant la boîte aux lettres correspondant à un canal donné.

Les concepts précédents permettent d'encoder tout les types de programmation et surtout une plate-forme multi-sites ainsi que des architectures globales et leurs stratégies respectives mais avant toute implémentation une phase de modélisation des entités concernées est primordiale.

5.5 Modélisation

Pour la modélisation d'un *SDGE*, nous considérons tout d'abord la plate-forme avec l'interconnexion des nœuds et leurs puissances respectives. Ensuite, la charge de calcul qui décrit les caractéristiques des tâches, puis finalement les architectures et les stratégies des ordonnanceurs et méta-ordonnanceurs.

Considérons une plate-forme de méta-ordonnancement constituée de $\mathcal{C}$ grappes de processeurs. Chaque grappe $\mathcal{C}_i$ possède un ensemble de nœuds de calcul $\mathcal{N}_i$.

Chaque grappe $\mathcal{C}_i$ appartient à un domaine local, i.d. un LAN (Local Area Network). Ce réseau local décrit un graphe d'interconnexion des nœuds. Chaque lien de ce graphe est pourvu d'*une bande passante locale* notée $bwt_{\mathcal{C}_i}$ d'*une latence locale* notée $latence_{\mathcal{C}_i}$. La connexion d'une grappe $\mathcal{C}_i$ au réseau global ou WAN (World Area Network) se fait par un routeur. Un graphe décrit aussi cette interconnexion avec des liens ayant les mêmes propriétés que précédemment soit *la bande passante globale* $bwt_{\mathcal{C}}$ et *la latence globale* $latence_{\mathcal{C}}$.

Nous observons ainsi une topologie hiérarchique comme l'illustre la figure 5.1.

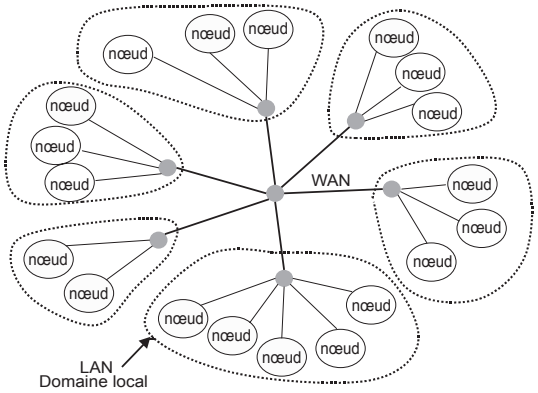


FIG. 5.1 – Exemple d'une topologie simple d'un système de méta-ordonnancement.

5.5.1 Les générateurs de topologies

Pour générer le graphe conforme à la modélisation énoncée, différentes approches existent. Une méthode consiste à décrire une topologie réelle. Pour ceci des outils de découvertes de caractéristiques comme ENV [144] existent. Le passage à l'échelle de ces outils n'étant pas garanti nous avons décidé de privilégier des générateurs aléatoires de topologie. Les générateurs de graphes sont principalement de trois types : aléatoire, basé sur le degré ou hiérarchique.

Les premiers sont les plus simplistes. Ils consistent par exemple à placer des points aléatoirement et à les connecter avec une probabilité fixe. Des modèles plus élaborés furent mis en place dont celui de Waxman [146] qui prend en compte plus de paramètres comme la distance entre deux nœuds, le nombre d'arêtes et l'hétérogénéité de la longueur d'arête. En dépit de l'utilisation de ces outils dans des études, le réalisme de ces approches sont limitées et d'autres études montrent que les lois uniformes ne sont pas adaptées.

Par exemple, Michalis, Petros et Christos Faloutsos [15] ont établi, par des instantanés de la topologie d'Internet à différentes périodes, des lois exponentielles simples décrivants le graphe et son évolution. Ces lois relient le degré des nœuds à leurs fréquences ou portent sur la fréquence des degrés, la taille du voisinage et les valeurs propres de la matrice d'adjacence. Cet article initia des modèles basés sur le degré comme Inet [148, 152], Brite [149] et CMU [150] et PLRG [151].

Cependant, des travaux [147, 155] prouvent que les lois sont corrélées entre elles. Si une de ces lois est vérifiée, les autres l'étaient également. De plus, les graphes générés par ces outils ne montre qu'un aspect hiérarchique qu'a une grande échelle de quelques milliers de nœuds [154]. Des générateurs hiérarchiques comme Tiers [153, 156] et Transit-Stub [153] incluent l'aspect d'organisation des réseaux

(WAN, MAN, LAN) comme illustré sur la figure 5.1 et sont les mieux adaptés selon GridG [145]. Nous avons donc choisi pour nos simulations de générer les topologies hiérarchiques avec Tiers.

5.5.2 Caractéristiques des nœuds de calculs

Soit le couple (i, j) définissant le $j^{ième}$ nœud d'une grappe $\mathcal{C}_i$. Chaque nœud (i, j) est caractérisé par une puissance $puissance_{i,j}$ du processeur. Pour exprimer cette puissance, définissons une unité le NCU (Normalyzed Computing Unit) qui équivaut à une unité de calcul. Cette unité se détermine notamment par des benchmarks d'une application précise sur différents types de machines en fonction du temps d'exécution absolu sur une machine référence. Ainsi la puissance d'un nœud est le nombre d'unités de calcul traitées par unité de temps. Nous pouvons ensuite exprimer l'hétérogénéité de la plate-forme et désigner *la puissance moyenne* de la plate-forme par :

$$puissance_m = \frac{1}{\sum_{i \in \mathcal{C}} card(\mathcal{N}_i)} \sum_{i \in \mathcal{C}, j \in \mathcal{C}_i} puissance_{i,j} \tag{5.1}$$

5.5.3 Le modèle de charge et d'exécution

Nous distinguons deux niveaux de charge, une locale et l'autre globale. La charge globale correspond aux tâches soumises par le système de *SDGE*. Ces tâches sont appelées méta-tâches. La charge locale composée de tâches est la charge propre à une grappe. Une méta-tâche m_k est traduite au niveau local en une simple tâche k.

Une tâche k est caractérisée par quatre attributs : $attributs_k$=$\{tl_k$, $taille_k$, $proc_k$, $group_k\}$ où tl_k est *la date de soumission locale* sur un site, $taille_k$ **la taille** exprimée en NCU, $proc_k$ *le nombre de processeurs* requis pour son exécution et $group_k$ *l'organisation* qui soumet la tâche. Une méta-tâche mk est aussi caractérisée par $meta\text{-}attributs_k$=$\{t_k$, $attributs_k\}$ où t_k est *la date de soumission* de la tâche mk au système de méta-computing global.

Modéliser la charge d'un système de méta-computing revient à déterminer pour chaque tâche k de l'ensemble des tâches $\mathcal{T}$ soumises à une grappe C_i les caractéristiques $attributs_k$. Il en va de même pour chaque méta-tâche mk de l'ensemble de tâches $\mathcal{MT}$ soumises au système de méta-computing et leurs méta-attributs. Les méthodes pour générer une charge sont de trois types : une charge aléatoire, une charge issue des traces d'un système réel ou une charge stochastique. Une charge aléatoire est peu réaliste de même que celles issues de systèmes réels sont souvent relatives à la capacité et au temps de réponse du système réel et donc trop spécifiques. Une charge stochastique est souvent la plus réaliste car elle prend en compte les capacités du système observées et incluent ces caractéristiques comme

des paramètres du modèle. Celle-ci sera notre choix. Des travaux étudiant les traces de centres de calcul [158, 159] proposent des modèles probabilistes complets. Nous notons $S(\mathcal{T})$ *la fonction de distribution qui génère l'ensemble des tailles* d'un ensemble de tâches $\mathcal{T}$. Soit CA *la coupure appliquée* à cet ensemble de tailles fixant la taille minimale et maximale. Nous désignons par $A(\mathcal{T})$ *la fonction de distribution qui génère l'ensemble des temps de soumission* d'un ensemble de tâches et $\lambda_{\mathcal{T}}$ *le taux de soumission moyen.*

Nous considérons un modèle d'exécution simplifié pour les tâches. Nous distinguons pour une tâche simple k le temps d'exécution suivant sur le nœud j :

$$ExecTime_{j,k} = TempsCPU_{j,k} + TempsI/O_{j,k} + TempsControle_{j,k} \tag{5.2}$$

$TempsCPU$ est le temps CPU consommé. $TempsI/O$ est le temps passé en entré sortie par exemple pour la lecture ou l'écriture de données. $TempsControle$ est le surcoût occasionné par la gestion de l'exécution de la tâche comme le changement de contexte.

5.5.4 Modélisation des architectures locales

Au niveau local, les nœuds d'un même site sont gérés par un système de gestion des ressources comme un système de traitement par lots. Sur le marché la plupart des solutions existantes mettent toutes en place des ordonnanceurs utilisant des files d'attente (ou queues). Ces files d'attente sont souvent définies en fonction des caractéristiques des tâches, par exemple leurs tailles. Ces ordonnanceurs locaux régissent le partage des ressources entre utilisateurs par des politiques locales reposant sur des stratégies de quotas et de priorités. Ainsi nous avons pour une grappe $\mathcal{C}_i$, *un ensemble de files d'attente* $\mathcal{F}_i$. Chaque file $f_{i,l}$ de $\mathcal{F}_i$ est composée d'un ensemble de nœuds $\mathcal{N}_{f_{i,l}}$ de la grappe. Un même nœud i, l, j peut appartenir à une ou plusieurs files. Les tâches soumises au site sont ensuite affectées à ces files en attendant leur exécution en fonction de la stratégie d'ordonnancement locale basée sur leurs priorités et contraintes. De ce fait, une file $f_{i,l}$ contient un ensemble de tâches $t_{i,l}$ en attente d'exécution. Nous définissons aussi la profondeur de la file d'attente comme $profondeur_{i,l} = card(t_{i,l})$ qui est le nombre de tâches en attente dans la file à un instant. Nous notons $t_{max_{i,l}}$ le temps maximal d'une tâche en exécution sur un nœud i, l, j de la file d'attente $f_{i,l}$.

5.5.5 Les indicateurs de performance

Un *SDGE* est un système complexe de par toutes les entités impliquées et sa nature distribuée. Mesurer et évaluer les performances du système est difficile. La définition même de l'efficacité varie selon les approches. Les métriques de performances se découpent en deux catégories. Selon le point de vue, nous avons les métriques utilisateurs et les métriques propriétaires ou administrateurs du système.

Les métriques utilisateurs se focalisent sur les performances des tâches tandis que les administrateurs se concentrent sur l'utilisation de leurs ressources.

5.5.6 Les mesures et métriques

Nous définissons pour une tâche k, les trois états suivants : *queued*, *running* et *done*. L'état *queued* signifie que la tâche est en file d'attente. Quand la tâche est en exécution elle est en état *running*. L'état *done* indique que la tâche a fini son exécution. *Les dates correspondantes aux changements d'états running, queued et done pour une tâche k sont respectivement* r_k, q_k *et* d_k. Ainsi *le temps d'attente local d'une tâche k* est la date de début d'exécution moins la date de soumission soit $r_k - tl_k$, *le temps d'exécution* est $d_k - r_k$ et *le temps de réponse local* est $d_k - tl_k$.

Pour une méta-tâche mk, nous avons *le temps d'attente global* qui correspond à la date de début d'exécution moins la date de soumission au système soit $r_k - t_k$ et *le temps de réponse global* est $d_k - t_k$.
Pour l'ensemble de méta-tâches $\mathcal{MT}$, nous définissons *le temps d'attente moyen* :

$$attente_m = \frac{1}{card(\mathcal{MT})} \sum_{k \in \mathcal{MT}} (r_k - t_k) \tag{5.3}$$

le temps d'exécution moyen :

$$execution_m = \frac{1}{card(\mathcal{MT})} \sum_{k \in \mathcal{MT}} (d_k - r_k) \tag{5.4}$$

et *le temps de réponse moyen* :

$$moyen_m = \frac{1}{card(\mathcal{MT})} \sum_{k \in \mathcal{MT}} (d_k - t_k) \tag{5.5}$$

Nous définissons aussi *le makespan* qui est le temps total pour terminer l'exécution de toutes les tâches dans l'ensemble $\mathcal{MT}$:

$$makespan = \max_{k \in \mathcal{MT}}(d_k) - \min_{k \in \mathcal{MT}}(t_k) \tag{5.6}$$

Le débit de calcul ou *throughput* est défini comme le nombre d'instructions élémentaires ou de tâches exécutées sur une période de temps :

$$throughput = \frac{\sum_{k \in \mathcal{MT}}(Taille_k)}{\max_{k \in \mathcal{MT}}(d_k) - \min_{k \in \mathcal{MT}}(t_k)} \tag{5.7}$$

Le *slowdown* d'une tâche mk est défini comme le temps de réponse sur le temps d'exécution soit $\frac{d_k - t_k}{d_k - r_k}$. Celui-ci souligne l'importance de la durée de la tâche. Soit

le slowdown moyen :

$$slowdown_m = \frac{\sum_{k \in \mathcal{MT}} (d_k - t_k)/(d_k - r_k)}{card(\mathcal{MT})} \tag{5.8}$$

Le stretch factor exprime le facteur de ralentissement de l'exécution d'une tâche mk par rapport à une machine de référence soit $\frac{d_k - r_k}{Taille_k}$. Soit *le stretch factor moyen* :

$$stretch\ factor = \frac{\sum_{jk \in \mathcal{MT}} \frac{(d_k - r_k)}{Taille_k}}{card(\mathcal{MT})} \tag{5.9}$$

L'écart type du temps moyen d'attente nous donne une métrique pour estimer l'équité et la variabilité des performances :

$$ecart_m = \frac{1}{card(\mathcal{MT})} \sqrt{\sum_{k \in \mathcal{MT}} (r_k - t_k)^2 - \left(\sum_{k \in card(\mathcal{MT})} \frac{(r_k - t_k)}{card(\mathcal{MT})} \right)^2} \tag{5.10}$$

Le flow-time est la somme de tous les temps de réponse des tâches. Soit :

$$FlowTime = \sum_{k \in \mathcal{MT}} (d_k - t_k) \tag{5.11}$$

L'utilisation des ressources se définit par le taux d'utilisation des nœuds soit :

$$taux\ utilisation = \frac{\sum_{k \in \mathcal{MT}} (d_k - r_k)}{Makespan} \times 100\% \tag{5.12}$$

5.6 Description du simulateur développé

Après vous avoir exposé notre modélisation d'un système multi-sites, nous montrons ici l'implémentation de celle-ci dans notre simulateur. Ensuite nous jugerons de la pertinence de ce modèle en étudiant la problématique de la validation d'un tel outil.

5.6.1 Module de description de la plate-forme

Notre simulateur est interfacé avec le générateur de graphes *Tiers* [156]. Il faut spécifier le nombre de WAN, le nombre de LAN, le nombre de nœuds par LAN et le nombre de liens redondants. Le graphe généré est une topologie réseau où chaque routeur, pont ou répéteur et nœuds sont représentés par les sommets, et les arcs représentent les liens réseaux comme illustré sur l'exemple de la figure 5.2. Celui-ci est utilisé pour nos expérimentations. Nous attribuons la latence et la bande

passante des liens en fonction du type de réseau. Un réseau local de type LAN a une plus grande capacité qu'un réseau global de type WAN.

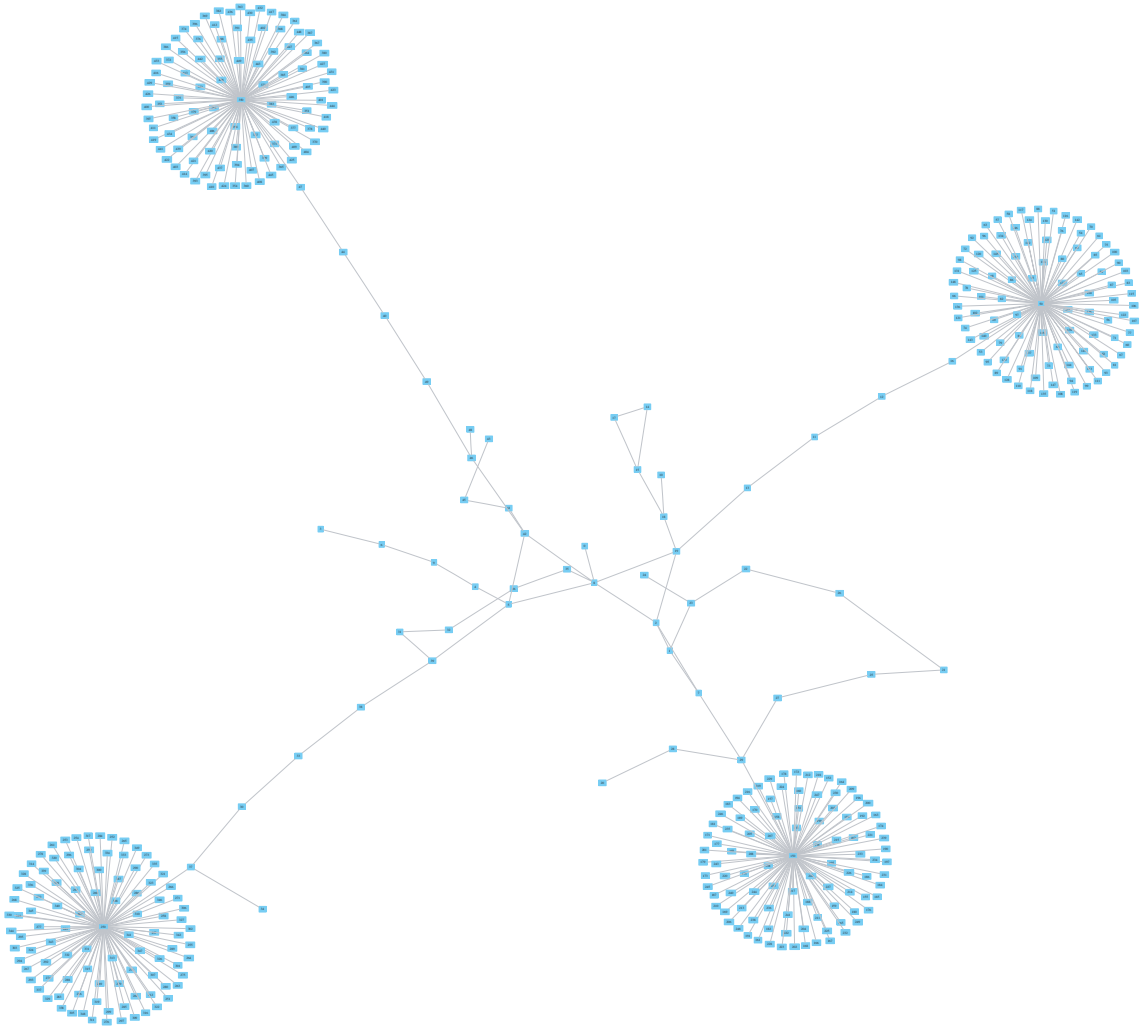


FIG. 5.2 – Topologie générée par *Tiers* et utilisée pour les expérimentations.

Pour l'attribution de la puissance, nous définissons un échantillon de nœuds de référence. Chaque nœud est pondéré par un pourcentage. Ce pourcentage exprime la proportion de ce type de nœud présent sur la plate-forme. La puissance NCU des nœuds et leurs pondérations s'inspirent des performances obtenues avec *DIRAC* pour une application de physique sur la plate-forme de production (voir chapitre 4). Cette plate-forme se composait de 4.000 nœuds et d'une vingtaine de configurations de nœuds différents. En fonction du nombre total de nœuds et des proportions attribuées à chaque nœud de référence, nous générons l'ensemble P de toutes les puissances disponibles. Ensuite, pour chaque nœud nous procédons à un tirage aléatoire dans cet ensemble. Nous ôtons ensuite une occurrence de cette valeur de

l'ensemble P et ce jusqu'à l'attribution de la puissance pour tous les nœuds. Nous disposons ainsi d'une plate-forme hétérogène.

5.6.2 Générateur de charge

Simgrid implémente déjà le concept de tâche. Nous avons rajouté les méta-données supplémentaires de notre modèle comme l'organisation soumettant la tâche. Nous avons aussi mis en œuvre un générateur de charges. Celui-ci génère des charges en fonction de différentes lois de distribution comme des lois gamma, de Poisson, de Weibull, normale, hyper-exponentielle (voir Annexe I). Ces lois ont été identifiées par les études statistiques [158, 159]. De plus, pour avoir un système partagé nous avons implémenté un agent qui représente soit un client du système soit une organisation et qui soumet une charge spécifique. Ce simulateur permet donc de simuler le comportement individuel d'un utilisateur ou d'une organisation virtuelle. Ainsi, nous pouvons avoir un système soumis à plusieurs charges différentes en même temps et évaluer les interactions.

5.6.3 Les systèmes de gestion de ressource local

Au niveau local, l'entité de base d'un système de méta-ordonnancement est le système de traitement par lot. Les deux principales raisons d'utilisation d'un système de traitement par lot sont : de maximiser l'utilisation entre les ressources partagées ; d'assurer effectivement l'équité d'utilisation de ces ressources.

Simgrid ne fournissant pas de système de traitement par lot, nous avons implémenté un système générique. Le modèle de conception utilisé est illustré sur la figure 5.3. Celui-ci se base sur une étude des implémentations existantes comme LSF, PBS et SGE [118–120]. Les algorithmes usuellement utilisés sont « First-Come-First-Serve », « Minimal-Request-Job-Fist », « Shortest, Backfill » [121, 122].

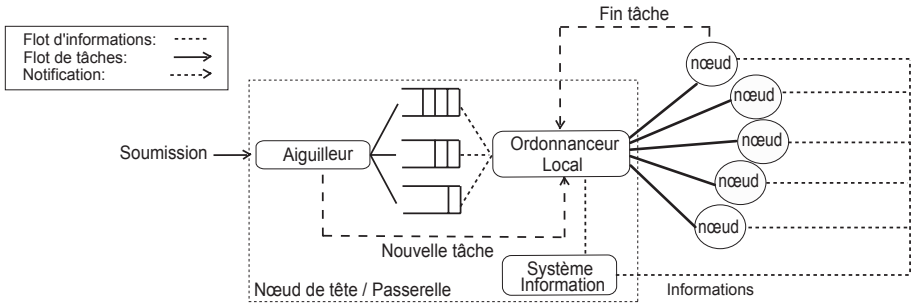


FIG. 5.3 – Structure d'un système de traitement par lot générique.

Un système de traitement par lot a une architecture centralisée maître-esclave. Un nœud de tête ou passerelle héberge les composants principaux de l'ordonnanceur local : l'aiguilleur, les files d'attente, le système d'information puis l'ordonnanceur

local. Chaque nœud contient un composant qui communique avec la passerelle. La soumission d'une tâche est traitée par un aiguilleur qui en fonction des besoins exprimés dans les méta-données de la tâche affecte celle-ci à une file d'attente. Il notifie ensuite l'ordonnanceur. Celui-ci interrogera le service de monitorage et de comptabilité local pour ensuite élire un nœud candidat. Si aucune ressource n'est disponible, la tâche reste en file d'attente. Après l'envoi de la tâche au nœud, la tâche est exécutée. À la fin de l'exécution de la tâche, l'ordonnanceur est notifié. Ceci enclenche un cycle d'ordonnancement, l'ordonnanceur examine les files d'attente et détermine si une tâche est apte à être exécutée. Les classes sont triées par classe de tâches comme il est usuel de le faire sur un système de traitement par lots. La configuration locale de chaque ordonnanceur est paramétrable par fichier. Cette configuration comprend le nombre de files d'attente, leurs caractéristiques, l'appartenance d'un nœud à une ou plusieurs files ou le nombre de tâches pouvant s'exécuter sur un nœud. L'attribution des nœuds aux tâches se fait par ordre décroissant de puissance de nœuds, c'est-à-dire que nous commençons par le nœud le plus puissant. Pour déterminer la puissance des nœuds et leur états, l'ordonnanceur utilise le système de monitorage implanté. Il peut aussi utiliser le système de comptabilité pour appliquer un ordonnancement équitable entre utilisateurs et organisations virtuelles.

5.6.4 Systèmes de monitorage et de comptabilité

Un système de monitorage et de comptabilité local possède une base de données qui contient l'état courant des nœuds. Cette base est mise à jour par les nœuds du système, l'aiguilleur et l'ordonnanceur. Elle contient aussi tous les changements d'état des tâches soumises au site et les informations relatives à leurs exécutions. Cette base comporte aussi l'énumération des tâches exécutées sur le système et les méta-données correspondantes. Pour pouvoir analyser les résultats, nous enregistrons pour chaque expérimentation les informations de chaque tâche. Ces informations permettent le calcul des mesures et métriques définies en section 5.5.6.

5.6.5 Les choix technologiques

Le simulateur est programmé en C++ et utilise les librairies C de *Simgrid*. Un module de configuration XML a été implémenté pour communiquer les paramètres de simulation. Le service de comptabilité utilise la base de données sqlite pour faciliter l'analyse des résultats avec l'aide de requêtes SQL. La langage *ClassAds* (voir section 3.5.2) est utilisé pour décrire les tâches et pour les fonctions de mise en relation. La génération des nombres aléatoires a été implémentée en C++ et Python.

5.7 Travaux relatifs à la validation du simulateur

Pour déterminer la pertinence de notre outil une phase de validation est primordiale. La vérification permet de valider le modèle d'abstraction et détermine sa consistance par rapport à la représentation du système, notamment pour les structures logiques du modèle. La validation donne une estimation du taux d'erreur de l'outil d'analyse pouvant décrire et prédire le comportement du système. Elle est primordial pour valider tous les paramètres du modèle ou indiquer qu'il faut compléter le modèle avec de nouveaux paramètres.

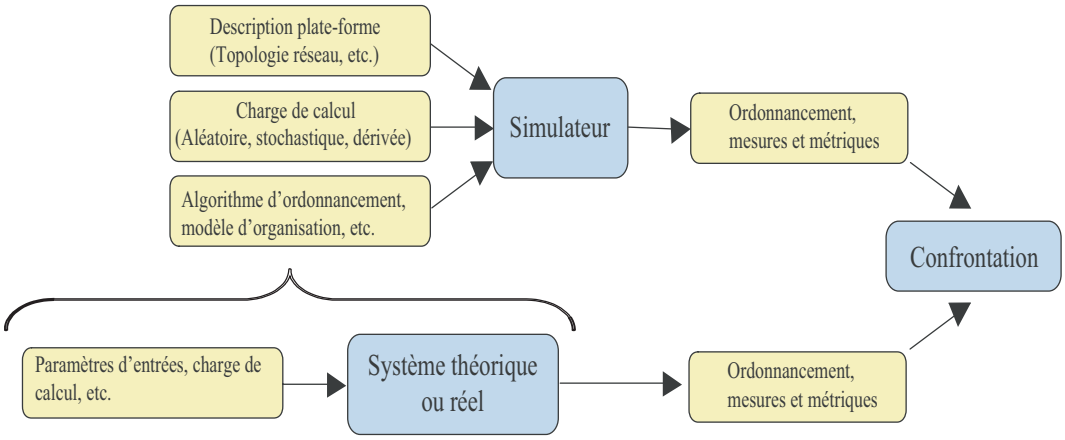


FIG. 5.4 – Protocole d'utilisation du simulateur.

La figure 5.4 montre le protocole d'utilisation du simulateur. La première étape est de fournir les paramètres d'entrées au simulateur. Ceux-ci comprennent la description de la plate-forme, la charge de calcul et les stratégies d'ordonnancement ainsi que le modèle d'organisation de cet ordonnancement. La deuxième étape est de les injecter dans le simulateur et de les comparer avec d'autres systèmes théoriques ou réels. Ainsi, le simulateur a été validé de manière théorique puis expérimentale.

5.7.1 Validation théorique

Pour la validation théorique de notre simulateur, un système à file d'attente [116, 117] est utilisé. Dans ce modèle, comme illustré sur la figure 5.5, une file d'attente simple est caractérisée par : Un processus d'arrivée de clients décrivant les instants auxquels les clients entrent dans le système ; Un processus de service décrivant le temps requis pour servir un client ; Un nombre de serveurs, tous identiques, spécifiant le nombre maximal de clients pouvant être servis simultanément. A ces caractéristiques de base s'ajoutent parfois la capacité de la file correspondant au nombre maximal de clients pouvant être présents à un instant donné aussi bien en attente qu'en service, la taille de la population des clients susceptibles d'utiliser

le serveur, et la discipline de service décrivant l'ordre dans lequel les clients sont servis.

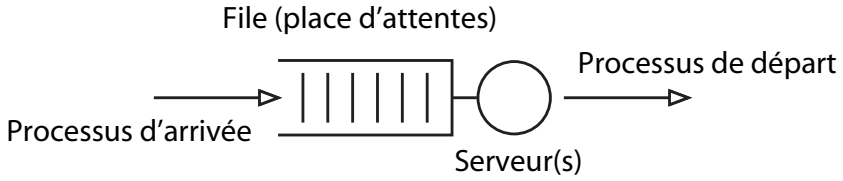


FIG. 5.5 – Structure générale d'un système de files d'attente.

Afin de simplifier la description des éléments décrivant un système à file d'attente, Kendall a introduit la notation suivante (forme simplifiée) : $X/Y/Z$ où chaque symbole correspond à une caractéristique de la file d'attente. Ainsi, X désigne le processus d'arrivée, Y le processus de service et Z dénote le nombre de serveurs soit dans notre cas le nombre de processeurs.

Nous considérons pour notre validation un système $M/M/m$. Dans ce cas, les arrivées suivent un processus de Poisson de taux λ et la durée de chaque service est une variable exponentielle de paramètre μ. Un processus d'arrivée est un processus de Poisson de taux λ si et seulement si les durées entre arrivées successives sont indépendantes et suivent toutes une loi exponentielle de paramètre $\frac{1}{\lambda}$. La distribution des inter-arrivées et celle des services sont donc toutes deux exponentielles. Ce système comporte de plus, un ensemble de m processeurs parallèles et une file d'attente appliquant la discipline de service PAPS (Premier Arrivé Premier Servi).

Nous appelons *état* d'un système à l'instant t le nombre n(t) de clients présents dans le système à un instant t. Un client est « présent dans le système » s'il est en file d'attente ou en cours de service. L'évolution du nombre de clients, ou ici de tâches, dans un tel système est une chaîne de Markov à temps continu. L' évolution cette variable d'état suit un processus dit de « naissance et de mort » dont le graphe représentatif est donné dans la figure 5.6. Dans ce graphe, les sommets correspondent aux états possibles du système, et chaque arc indique une transition possible d'un état vers un autre.

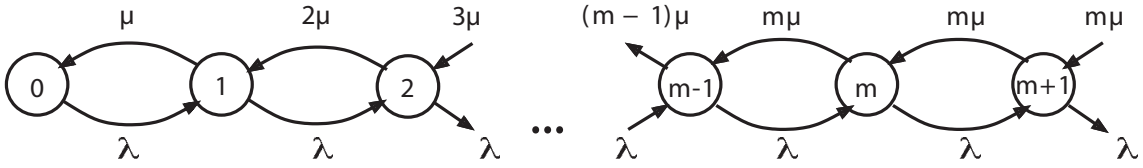


FIG. 5.6 – Représentation du graphe de transition d'un processus de « naissance et de mort » d'un système $M/M/m$.

Le processus d'état stochastique est un processus de naissance et de mort si, pour chaque $k = 0, 1, 2, ...$, il existe des paramètres λ_k et μ_k (avec $\mu_0 = 0$) tels

que, lorsque le système est dans l'état n, le processus d'arrivée est poissonnien de taux λ_k et le processus de sortie est poissonnien de taux μ_k. Pour conserver les propriétés de conservation, nous fixons le nombre de processeurs n requis par les tâches comme identique ($n = 1$). Le taux de service est proportionnel au nombre de nœud occupés par des tâches, il s'ensuit que :

$$\begin{cases} \lambda_k = \lambda, & k \geq 0 \\ \mu_k = \mu k, & 1 \leq k < m \\ \mu_k = m\mu, & k \geq m \end{cases} \tag{5.13}$$

Les quantités fondamentales dans le cadre des modèles de files d'attente sont les probabilités d'état, que nous définissons par $p_k(t)$ qui est la probabilité de l'état k à l'instant t, c'est-à-dire la probabilité que k clients soient présents dans le système à l'instant t. Nous définissons la probabilité à long terme ou probabilité stationnaire p_k comme $\lim_{t\to+\infty} p_k(t)$ qui est la probabilité que, à un instant quelconque dans le long terme, exactement n clients soient présents dans le système. La probabilité p_k est défini par :

$$p_k = p_0(\lambda/\mu)^k, avec\ p_0 = \left[1 + \sum_{k=1}^{\infty} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}\right]^{-1}. \tag{5.14}$$

En substituant les valeurs λ et μ dans les équations de 5.13, nous avons alors :

$$\begin{cases} p_k = p_0 \frac{(\lambda/\mu)^k}{k!}, & \text{k} < \text{m} \\ p_k = p_0 \frac{(\lambda/\mu)^k}{m! m^{k-m}}, & \text{k} \geq \text{m} \end{cases} \tag{5.15}$$

où en exprimant que $p_k = \sum_{k=0}^{\infty} p_n = 1$, nous pouvons déduire la valeur de p_0.

$$p_0 = \left[1 + \sum_{k=1}^{m-1} \frac{(\frac{\lambda}{\mu})^k}{k!} + \frac{(\frac{\lambda}{\mu})^m}{m!(1 - \frac{\lambda}{m\mu})}\right]^{-1} \tag{5.16}$$

en se basant sur la fonction des moments, nous avons le nombre moyen de clients en attente :

$$\overline{N}_q = \frac{p_0(\frac{\lambda}{\mu})^m}{m!} \frac{\frac{\lambda}{m\mu}}{(1 - \frac{\lambda}{m\mu})^2} \tag{5.17}$$

Par le théorème de Little, nous trouvons le temps moyen d'attente dans la file d'attente. Celui-ci est $\overline{W} = \frac{\overline{N}_q}{\lambda}$. Le temps de service moyen étant $\frac{1}{\mu}$, le temps moyen d'exécution est $\overline{T} = \overline{W} + \frac{1}{\mu}$. De plus par le théorème de Little, nous connaissons le nombre moyen de clients présents dans le système (en attente et en service) soit $\overline{N} = \lambda\overline{T}$. Une autre métrique intéressante est le taux d'utilisation du système $\overline{U}$

définit comme :

$$\overline{U} = \sum_{k=1}^{m-1} kp_k + m \times P(Attente) = \sum_{k=1}^{m-1} kp_k + mp_0 \frac{1}{m!} \left(\frac{\lambda}{\mu}\right)^m \left(\frac{1}{1 - \frac{\lambda}{m\mu}}\right) \quad (5.18)$$

où la probabilité d'être en attente $P(Attente)$, qui est la probabilité que tous les serveurs soient occupés soit la présence de plus de m clients dans le système ($P(\geq m\ clients\ présents)$), se base sur la formule d'Erlang. Nous utilisons ce modèle simple pour vérifier les statistiques fournies par notre simulateur.

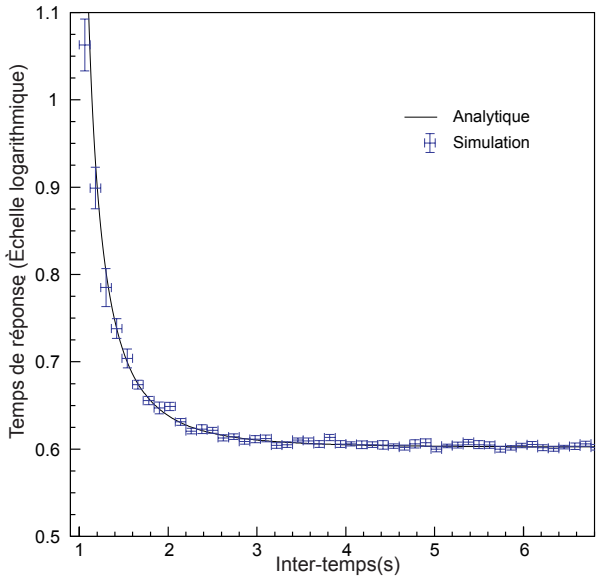


FIG. 5.7 – Comparaison des temps de réponse entre la simulation et la théorie dans un système de file d'attente M/M/4.

La figure 5.7 montre la confrontation des temps de réponses analytiques $\overline{T}$ et simulés dans un système M/M/4 en fonction des inter-temps. Les temps d'exécution sont distribués exponentiellement avec une moyenne de 4 unités de temps. Les inter-temps sont aussi distribués exponentiellement. Les temps de réponses simulés sont obtenus par la méthode des moindres carrés avec 16 exécutions indépendantes de 1000 tâches pour une moyenne d'inter-temps donnée. Les résultats entre les résultats observés et les résultats attendus coïncident avec un taux erreur moyen de 0.6%. De plus, nous appliquons le test du Khi-deux d'homogénéité et obtenons une valeur de $0.002 << 1$ qui nous fait conclure que notre simulateur d'un système de traitement par lots est validé du point de vue théorique.

5.7.2 Validation expérimentale

Nous avons utilisé une grappe d'ordinateurs dédiée et hétérogène dont la configuration est résumée dans le tableau 5.1.

Attributs	**Valeurs**		
Nombre de nœuds($card(\mathcal{N}_i)$)	3		
Type	PII	PII	PIII
Processeur(MHz)	350	400	600
Mémoire vive(mo)	128	128	128
Puissance($NCU.s^{-1}$)	32.12	52.12	100.00
Ordonnanceur	openPBSv2.3		
Politique	Premier Arrivé Premier Servi (PAPS)		
Réseau local	Mégabit Ethernet		

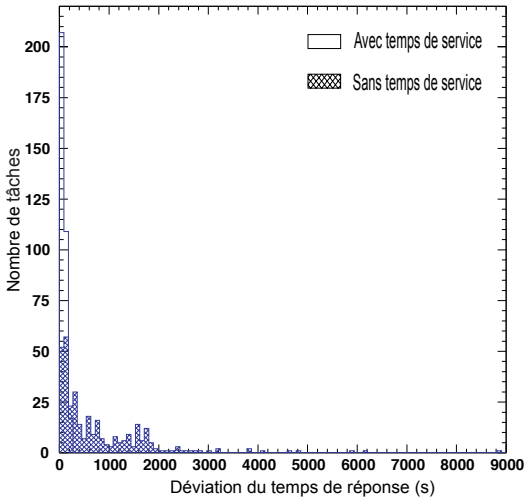
TAB. 5.1 – Résumé des caractéristiques de la plate-forme utilisée pour la validation du simulateur.

Nous avons utilisé le système *DIRAC* pour valider notre simulateur. Nous avons déployé un agent *DIRAC* sur le site et utilisé un générateur de tâches. Ce générateur génère et soumet des tâches de calcul séquentielles indépendantes et donc sans communication. Les temps de soumission suivent une loi de distribution de Poisson.

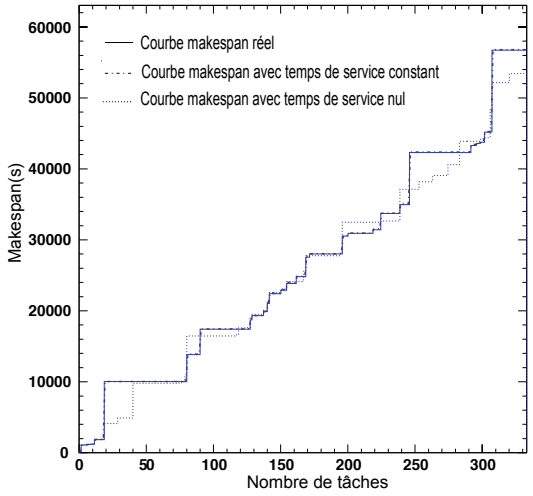
Le jeu d'essai utilisé est un programme qui implémente un compteur de consommation CPU. Il prend un paramètre qui est le temps CPU à consommer avant de se terminer. Cette taille est générée pour chaque tâche et suit une loi de Weibull. Cette loi généralise la loi exponentielle et modélise des durées de vie qui sont dans notre cas des durées d'exécutions. Les temps de réponse et d'attente du système sont recueillis par le service de monitorage de *DIRAC*. Nous dérivons ensuite cette charge pour l'injecter dans notre simulateur. Pour décorreler le temps d'exécution du nœud sur lequel une tâche s'exécute, nous normalisons ce temps avec la puissance NCU du nœud. La puissance NCU des nœuds est obtenue par des jeux d'essais, et est reportée dans le tableau 5.1. La topologie utilisée est une topologie simple où chaque nœud est relié au nœud de tête comme illustré sur la figure 5.3 par un simple lien d'une bande passante de 100 méga-octets par seconde et de latence nulle.

La figure 5.8(a) montre deux distributions de la valeur absolue de la déviation observée entre la réalité et la simulation pour les temps de réponse de chaque tâche. Le nombre de tâches est de 330, i.d. $card(\mathcal{MT}) = 330$. Dans la première distribution en hachuré, nous observons un fort taux d'erreurs de l'ordre de 80%. Après une étude des traces, nous avons caractérisé deux temps de service μ_{rec} et μ_{env}. Le temps de service μ_{rec} est le temps entre l'arrivée d'une tâche et l'envoi de

cette tâche sur un nœud ou en file d'attente. Le temps de service μ_{env} est le temps de prise en compte de la fin d'une tâche par l'ordonnanceur et de la disponibilité d'un nœud. Cette différence entre la réalité et la simulation s'explique par le fait que l'ordonnanceur effectue ces choix avec des données différentes de la réalité. Les nœuds étant hétérogènes, cela entraîne des conséquences immédiates sur le temps de réponse d'une tâche. En prenant en compte les temps de services mesurés sur le système réel pour les paramètres μ_{rec} et μ_{env}, nous corrigeons ce taux d'erreur en injectant ces paramètres sous forme de fichiers de traces. Dans ces conditions, nous obtenons exactement le comportement de la réalité. Ceci valide le code de notre simulateur. Nous avons réitéré l'expérience en positionnant les temps de service à des constantes. Ces constantes sont les temps de services moyens mesurés sur le système test ($\mu_{rec} = 3.75s$, $\mu_{env} = 2s$). Ces paramètres diminuent le taux d'erreur car nous observons un taux d'erreur moyen de 10.5% pour le temps de réponse ce qui explique la disparition de la queue dans la distribution de la déviation en blanc sur la figure 5.8(a).



(a) Affichage de la distribution de la déviation pour les temps de réponse (valeur absolue).



(b) évolution du makespan en fonction du nombre de tâches dans la réalité.

FIG. 5.8 – Confrontation entre la simulation et la réalité, avec $\mu_{rec} = \mu_{env} = 0$ et $\mu_{rec} = 3.75s$, $\mu_{env} = 2s$.

La figure 5.8(b) montre l'évolution du makespan en fonction du nombre de tâches en abscisse. Pour les temps de services constants, nous observons un taux d'erreur moyen de 12%, ce qui est acceptable. Notre simulateur au niveau local est donc réaliste. Nous pouvons ainsi procéder à l'évaluation de stratégies et d'architectures de *SDGE*. Notons, qu'une possible amélioration serait de faire suivre pour les paramètres de temps de services μ_{rec} et μ_{env} une loi de distribution qui approche le comportement réel.

5.8 Conclusion

Dans ce chapitre, nous avons soulevé la problématique de l'évaluation d'ordonnancement dans un contexte de *SDGE*. Une telle plate-forme se compose de grappes de nœuds hétérogènes appartenant à un réseau local, e.g. centres de calcul ou sites, interconnectées entre elles par un réseau global. Ces grappes sont accessibles via un ordonnanceur local et sont partagées entre les utilisateurs. Nous avons prouvé la validité du simulateur en le confrontant avec des résultats issus de la théorie des files d'attente et avec un système de traitement par lots réel. Cette confrontation, nous donne une différence moyenne de 10.5% par rapport à la réalité pour les temps de réponse et de 12% pour le makespan, ce qui est acceptable. Cette phase a permis d'inclure des temps de service constant dans notre modèle. De plus, après prise en compte des traces du système réel dans notre simulation, nous obtenons des temps identiques avec la réalité. Notre simulateur est donc réaliste et décrit bien le comportement d'un système de traitement par lots réel. L'outil de simulation autorise l'évaluation de stratégies et d'architectures pour l'ordonnancement dans un environnement réaliste.

CHAPITRE 6

SIMULATIONS D'ORDONNANCEMENT SUR UNE PLATE-FORME HÉTÉROGÈNE

Résumé du chapitre :
Nous avons analysé l'ordonnancement de DIRAC *dans un contexte de calcul intensif. Nous justifions l'approche distribuée, adaptative et opportuniste utilisée dans* DIRAC *par rapport à une approche centralisée. Cette étude s'est notamment basée sur la difficulté d'avoir un état global cohérent en prenant en compte l'état de dégradation du système d'information centralisée.* DIRAC *contourne cette difficulté en se basant sur une vue locale de chaque ressource. D'autres stratégies relatives à l'influence du déploiement sur l'ordonnancement dans* DIRAC *sont aussi évaluées ici. Nous montrerons comment un déploiement dynamique augmente les performances de* DIRAC *et provoque une charge plus homogène sur le service central. Nous évaluons aussi les capacités d'évolutions de* DIRAC *en proposant des stratégies et une étude préliminaire pour un ordonnancement multi-critères basé sur des priorités et des préemptions.*

6.1 Introduction

Afin d'évaluer les performances de *DIRAC*, nous avons mis en œuvre et validé le simulateur décrit dans le chapitre 5. Nous pouvons à présent simuler les caractéristiques de *DIRAC* dans un contexte de régime permanent et saturé [90]. Nous comparons ses performances avec une architecture centralisée et des stratégies actives « push » dans différentes situations de méta-ordonnancements. En s'inspirant d'une problématique de l'expérience *LHCb*, qui est l'exécution des deux applications aux besoins antagonistes au sein d'une même communauté ou organisation virtuelle (*OV*) [50], nous étudions les possibilités d'évolution de *DIRAC* pour un ordonnancement bi-applicatif et les évolutions apportées à notre simulateur pour prendre en compte l'aspect des données.

Nous rappelons d'abord les notions et concepts de la gestion des ressources dans la section 6.2 puis nous détaillons les architectures et stratégies de méta-ordonnancement considérées (section 6.3), et ensuite les différents paramètres de la simulation tel que la modélisation de la plate-forme en section 6.5. Les résultats obtenus sont ensuite analysés dans la section 6.6. Nous traitons ensuite de la problématique d'un ordonnancement bi-applicatifs avec la description de nouvelles stratégies et une étude préliminaire expérimentale de ces stratégies en section 6.6.1. Nous conclurons sur les perspectives induites par cette études.

6.2 La gestion des ressources

Nous présentons une synthèse des notions traitées dans la section 3.4 (chapitre 3). Un système de gestion de ressources comporte des éléments de contrôle qui prennent des décisions de transfert ou d'allocation de tâche en se basant sur des éléments d'informations. Nous nous plaçons dans le cas des méta-ordonnancements dynamiques avec des tâches simples. La qualité d'un ordonnancement dynamique est intrinsèquement lié à l'architecture du système et à la vue résultante de cette architecture, comme le résume la taxonomie des systèmes de gestion de ressources que nous proposons dans la figure 6.1.

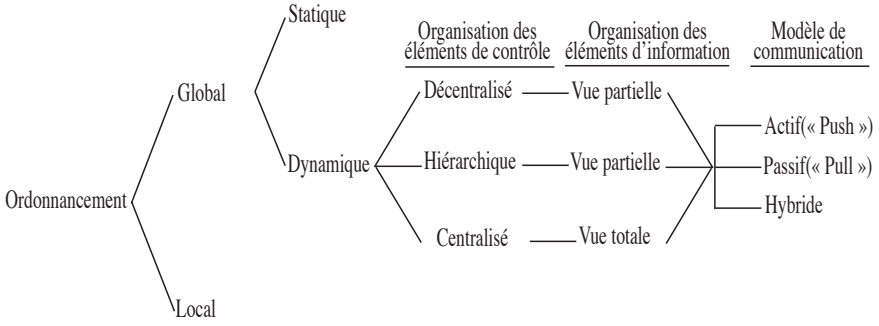


FIG. 6.1 – Taxonomie des systèmes de gestion de ressources.

Ainsi pour un ordonnancement centralisé, un seul élément de contrôle prend la décision du placement des tâches en fonction d'un état global cohérent. Tandis que pour un ordonnancement décentralisé, plusieurs éléments de contrôle interviennent dans ces décisions. Ils se basent plus fréquemment sur une vue partielle et locale. Le mode de communication entre les éléments de contrôle est soit « push » ou « pull ». Dans le mode « push », le créateur de la charge initie l'algorithme de mise en relation ressource/tâche alors que dans le mode « pull » l'algorithme de mise en relation est déclenché par le consommateur de la charge : la ressource. Nous comparons par la suite les deux architectures avec leurs modèles de communication et algorithme de mise en relation dans un contexte de régime permanent saturé.

6.3 Les architectures globales considérées

Dans les projets de grilles actuels tels que [49, 107, 157, 164, 188] sur l'ordonnancement multi-sites, les décisions se basent sur une connaissance totale de l'état du système à l'instant t. Les architectures sous-jacentes consistent en un méta-ordonnanceur centralisé qui se base sur un système d'informations centralisé comme l'illustre la figure 6.2. Le système d'information global contient les informations dynamiques et statiques sur l'état de la plate-forme. Les informations dynamiques sont mises à jour par des senseurs déployés sur les sites [18, 126]. Ils interrogent le

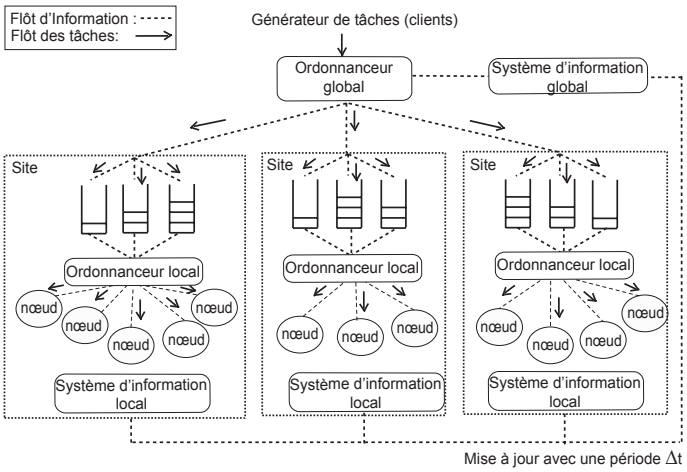


FIG. 6.2 – Exemple d'architecture avec un ordonnanceur centralisé.

système d'information local puis positionnent les informations relatives au site dans le système d'information global. Une des difficultés d'un système réel est de maintenir une vue de l'ensemble de la plate-forme la plus réaliste possible. Idéalement, un agent effectue une mise à jour dès que l'état du site subit un changement. Ce changement correspond à la fin d'une tâche ou à l'arrivée d'une tâche. Mais concrètement cette solution entraîne un surcoût important de messages et nécessite des mécanismes de notification. L'utilisation d'une période $\triangle t$ de rafraîchissement de l'information est un compromis entre cohérence des informations et nombre de messages.

Dans le cadre de *DIRAC*, c'est l'approche décentralisée qui a été retenue comme illustré sur la figure 6.3, et décrite précédemment dans le chapitre 3.

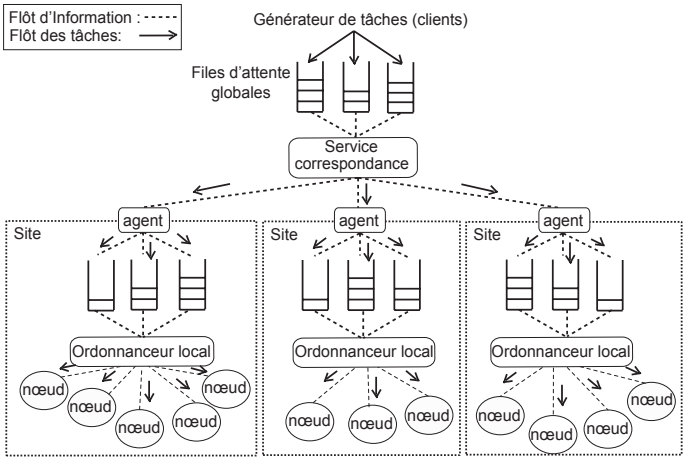


FIG. 6.3 – L'architecture décentralisée de *DIRAC*.

Le modèle de communication des éléments sont de type « push » dans le cas de l'architecture centralisée et de type « pull » dans celui de *DIRAC*. Après avoir fixé l'architecture et les modèles de communication, la définition des stratégies d'ordonnancement respectives à chaque approche est primordiale.

6.4 Les stratégies de méta-ordonnancement considérées

Nous présentons deux architectures de méta-ordonnancement différentes. Nous pouvons appliquer différentes stratégies d'ordonnancement avec ces architectures. De plus, les algorithmes d'ordonnancement sont conditionnés par les critères d'évaluations choisis [111]. Le contexte de régime permanent et saturé vise l'utilisation de toutes les ressources. Nous proposons pour chaque architecture respective une stratégie qui maximise l'utilisation des ressources, c'est-à-dire une stratégie distribuant la charge sur toutes les ressources disponibles en ne laissant aucune ressource sous-utilisée ou inutilisée.

6.4.1 La stratégie centralisée

L'algorithme d'ordonnancement « push » choisi pour l'architecture est qualifiée de « Join Shortest Queue » (JSQ) dans la littérature [165]. L'ordonnanceur à chaque réception de tâche sélectionne l'ensemble des ressources qui correspondent à cette tâche puis choisit dans cette ensemble la ressource la moins chargée, c'est-à-dire la file d'attente $f_{i,l}$ du site C_i possédant la profondeur minimale $profondeur_{i,l}$. La profondeur d'une file correspond au nombre de tâches en attente dans cette file. L'algorithme de cette stratégie est disponible dans les annexes (annexe K, algorithme 2). Cette stratégie centralisée favorise l'utilisation des ressources car elle répartit la charge sur l'ensemble des ressources. Nous allons détailler quelle stratégie nous mettons en œuvre dans *DIRAC* pour maximiser l'utilisation des ressources.

6.4.2 L'ordonnancement de *DIRAC* en détail

Contrairement à la stratégie précédente où un seul élément, l'ordonnanceur global, prend les décisions de choisir un site, nous avons deux éléments de contrôle, l'agent et le service *correspondance* qui prennent les décisions de méta-ordonnancement dans *DIRAC*. Comme nous l'avons souligné dans le chapitre 3, cette distribution des rôles s'inspire des projets de vol de cycles par internet [47]. Mais dans le contexte d'internet, le critère de disponibilité est souvent la détection de l'absence de l'utilisateur ou une consommation processeur inférieure à un seuil fixé. La difficulté est de traduire cette logique à des files d'attentes de site. Nous proposons alors le comportement suivant pour les agents déployés sur les passerelles des sites. L'agent

dans le mode *direct* en fonction de l'état d'un site demande une méta-tâche au service *correspondance*. Il vérifie, avec une période σt, la disponibilité de la ressource. le critère de disponibilité utilisé est donné en (6.1). Il s'applique sur une file d'attente $f_{i,l}$ du site et exprime que le rapport des tâches en attente ($profondeur_{i,l}$) sur celui du nombre de nœuds associés a cette file ($\mathcal{N}_{f_{i,l}}$) reste inférieur à une valeur εt, fixée de manière empirique ou arbitraire.

$$\frac{profondeur_{i,l}}{card(\mathcal{N}_{f_{i,l}})} < \varepsilon,\ tel\ que\ \varepsilon \in R_*^+\ et\ \varepsilon \in]0,1[. \tag{6.1}$$

Cette égalité garantit que la charge de chaque site est proportionnelle à la capacité de celui-ci et assure l'égalité des temps moyens d'attente dans les files d'attentes de tous les sites. Un nombre constant de tâches est dans chaque file d'attente des sites. Le comportement de l'agent est détaillé dans l'algorithme 3 en annexe K. Cet algorithme s'applique au niveau des files d'attente $f_{i,l}$ d'un site C_i de manière cyclique ou « round robin ».

L'autre élément qui intervient dans l'ordonnancement est le service *correspondance*. Nous qualifions celui-ci de passif car il n'active son traitement qu'en réponse à des événements extérieurs, en l'occurrence des requêtes d'agents. À ces requêtes, le service *correspondance* cherche dans une file d'attente globale une tâche qui correspond à la requête de l'agent. La politique appliquée au niveau du service *correspondance* est qualifiée de FRFS (« Fit Resource First Served »). C'est à dire que la première ressource qui convient à une tâche est choisie. L'algorithme 4 dans l'annexe K décrit ce comportement.

Le comportement de l'agent exposé ici est dans le mode *direct* où l'agent sur le site dit *maître* soumet directement des tâches simples au site. Mais nous avons vu notamment dans le cas de l'utilisation de *LCG* (voir chapitre 5), l'intérêt d'avoir une autre approche qualifiée d'*indirect* qui change le déploiement des agents et la localisation. Nous étudions dans la suite les conséquences, les apports ou les pénalités de ce mode.

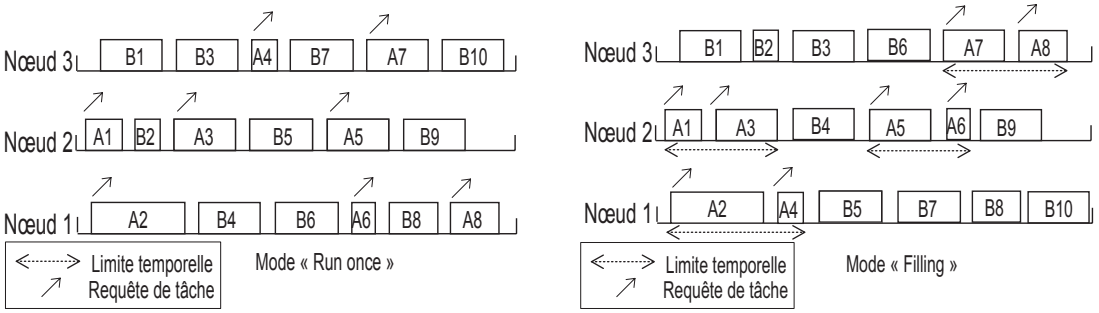
6.4.3 Les stratégies liées aux déploiements des agents *DIRAC*

Nous voulons évaluer l'importance du déploiement au sein de *DIRAC*. Pour ceci, nous considérons deux déploiements pour les agents. Un déploiement statique ou *direct* où l'agent *maître* déployé sur la passerelle d'un site soumet des tâches dans les files d'attente. L'autre déploiement est qualifié de dynamique ou d'*indirect* et a été utilisé avec le projet de grille *LCG*. Celui-ci est une notion proche de la réservation de ressource. Dans celui-ci, l'agent déployé sur le site détecte si la ressource est disponible selon le critère 6.1. Dans le cas positif, il interroge le service *correspondance* pour savoir si des tâches sont disponibles. Si tel est le cas, il soumet des agents *pilotes* à la ressource et réitère sa demande. L'algorithme 5 dans l'annexe K exhibe ce comportement. Les agents *pilotes* soumis suivent ensuite le cheminement

normale d'une tâche de la file d'attente locale à l'allocation d'un nœud pour finalement se terminer par l'exécution de l'agent *pilote* sur le nœud. L'agent *pilote* sur le nœud demande du travail au service *correspondance* utilisant la description du nœud générée au préalable. Si une tâche est disponible, l'exécution de cette tâche est effectuée. Nous distinguons ensuite plusieurs scénarios pour le comportement de l'agent *pilote* sur le nœud comme le montre la figure 6.4.

Dans le mode de réservation simple ou *run once*, l'agent *pilote* meurt après l'exécution d'une tâche comme le montre la figure 6.4(a). L'algorithme de ce mode est disponible dans l'annexe K (algorithme 6).

En mode de réservation *filling* (figure 6.4(b)), il récupère encore une tâche en accord avec les contraintes locales imposées par l'ordonnanceur local. Ces contraintes sont principalement des contraintes de temps absolu et de consommation CPU comme le temps maximal $t_{max_{i,l,j}}$ d'une tâche en exécution sur un nœud i, l, j de la file d'attente $f_{i,l}$ du site j. L'algorithme implémente donc un compteur de temps comme le montre l'algorithme 7 en annexe K. En cas de non disponibilité d'une tâche, l'agent meurt immédiatement après avoir reçu une réponse négative du service *correspondance*.



(a) Agent *pilote* en mode « run once ».

(b) Agent *pilote* en mode « filling ».

FIG. 6.4 – Exemples des deux modes de déploiement indirect sur un site de trois nœuds, partagé entre deux organisations virtuelles (OV) A et B. Les tâches sont préfixées dans leurs numérotations par la lettre de leurs OV respectives. Les tâches A (agents *pilotes*) sont ordonnancés par *DIRAC* et B correspondent à des tâches *batch* classiques.

Ce simple exemple nous aide à pressentir l'intérêt et le gain possible au passage d'un tel déploiement dans un contexte partagé par rapport au mode « run once ». Par exemple, les tâches $A3$, $A4$ et $A6$ sont exécutées plut tôt et le *makespan* de l'ensemble des tâches A, donné par la distance du début de $A1$ et la fin de $A8$ diminue.

6.5 Paramètres des expérimentations

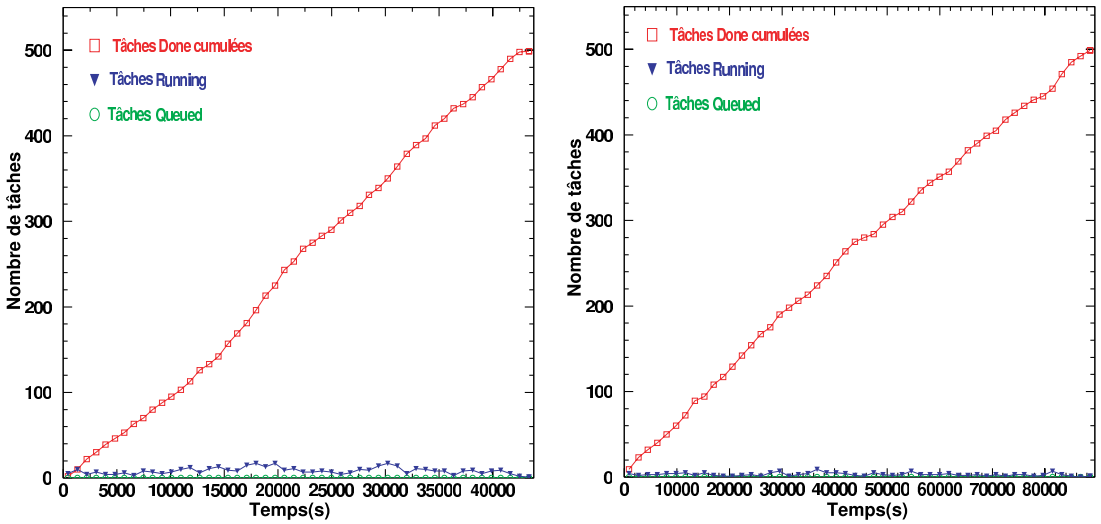
Le tableau 6.1 précise les paramètres de la plate-forme et des charges. La taille des messages de contrôle est de 30 kilo-octets pour les deux architectures. Les caractéristiques des charges sont inspirées des études empiriques [158, 159].

		Paramètres	*Notations*	*Valeurs*
Plate-forme		Nombre de sites	$card(C)$	3
		Nombre de nœuds par site	$card(N_i)$	20
		Nombre de files d'attente par site	$card(F_i)$	1
		Puissance moyenne des nœuds	$puissance_m$	96 $NCU.s^{-1}$
		Politiques Locales	M/M/$card(N_i)$/FCFS	FCFS/PAPS
		Temps maximal d'exécution	$t_{max_{i,j}}$	24000s
		Bande passante locale/-globale	bwt_C/bwt_{C_i}	1000 $Mbit$/100 $Mbit$
		Latence locale/globale	$latence_C$, $latence_{C_i}$	0s
Charge		Type de tâches	$card(proc_k)$	1
		Distribution des tailles	$S(\mathcal{M}t) \rightarrow \{taille_k\}$	$Weibull(\alpha = 142.2, \beta = 0.45)$
		Coupure sur les tailles	$\mathcal{C}(taille_k)$	$37300 < taille_k < 242800$
	Globale	Nombre de tâches	$card(\mathcal{M}t)$	500
		Distribution des temps d'arrivées	$A(\mathcal{M}t) \rightarrow \{tl_k\}$	$Poisson(m = 0.05, s = 4)$
		Inter-temps moyen	$1/\lambda_{\mathcal{M}t}$	19s
	Locale	Nombre de tâches par site	$card(\mathcal{T}_i)$	500
		Distribution des temps d'arrivées	$A(\mathcal{T}_i) \rightarrow \{t_k\}$	$Poisson(m = 0.011, s = 4)$
		Inter-temps moyen	$1/\lambda_{\mathcal{T}_i}$	87s

TAB. 6.1 – Synthèse des paramètres pour les expériences.

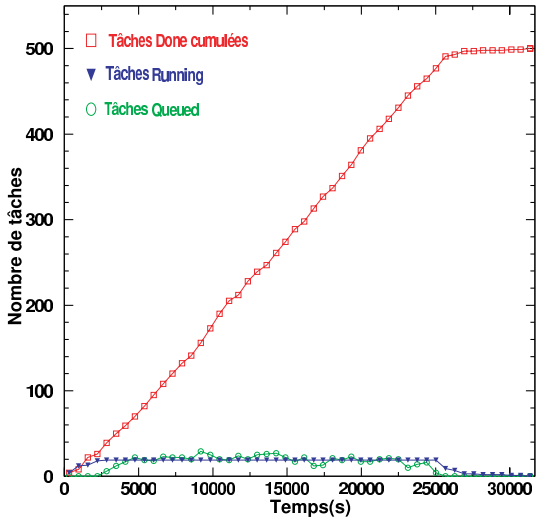
De plus, nous définissons par rapport à une plate-forme donnée, le niveau de charge par un taux d'occupation de la plate-forme. La formule 5.12 du taux d'occupation est définie en section 5.5.6. Nous exposons dans les figures 6.5 différents cas d'utilisation de la plate-forme avec des taux de soumission croissant.

Les figures 6.5 illustrent ces trois types de charge sur un seul site de 20 nœuds. Ainsi, nous avons une charge lourde de taux d'occupation 99% (figure 6.5(c)), une charge moyenne générant un taux d'occupation de 58% (figure 6.5(a)) et une charge légère avec un taux d'occupation de 31% (figure 6.5(b)). Une charge lourde permet d'évaluer les ordonnancements en régime permanent et saturé. Nous utilisons une charge lourde pour l'évaluation des architectures et stratégies.



(a) Charge à 58%.

(b) Charge à 31%.



(c) Charge à 99%.

FIG. 6.5 – Illustration des différents types de charge pour une plate-forme donnée. Les taux d'occupation moyen des nœuds sont indiqués en légende.

6.6 Comparaison des architectures et stratégies

La figure 6.6 montre l'évolution du cardinal des tâches dans les différents états *queued* et *running* pendant la durée de l'expérience pour une plate-forme dédiée. La troisième courbe est la courbe cumulée de toutes les tâches en états *done*. La figure 6.6(b) est pour l'ordonnancement centralisé avec la période de rafraîchissement du système d'information nulle ($\triangle t = 0$) et la figure 6.6(a) pour *DIRAC*.

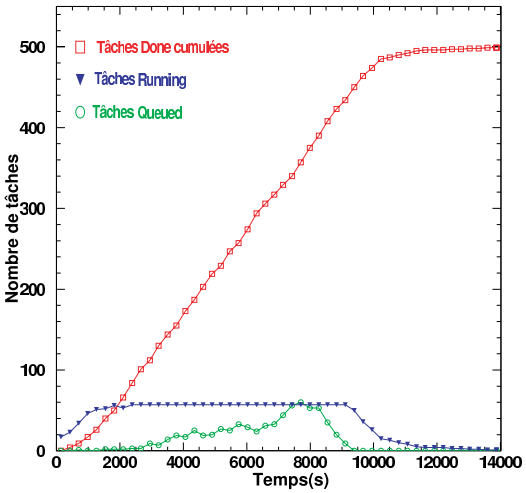
Pour les deux approches, nous observons une phase d'initialisation du système puis la saturation de toutes les ressources et finalement l'arrêt du système. La phase de saturation nous donne la capacité maximale de la plate-forme. Il correspond à la somme de tous les nœuds disponibles soit $\sum_{i \in \mathcal{C}} \mathcal{N}_i$, ici 60. Les deux approches saturent donc les ressources. Une des caractéristiques de l'approche *DIRAC* par rapport à l'approche centralisée est le fait que le nombre de tâches en état *queued* est constant. L'approche *DIRAC* pose donc moins de charge sur les systèmes locaux parce qu'il y a moins de tâches dans les systèmes locaux à chaque instant. Ceci a été bien apprécié par les administrateurs des sites.

La figure 6.7(a) compare le makespan, le temps de réponse moyen local et global ainsi que le temps d'exécution pour les quatre stratégies évaluées. Le meilleur makespan est obtenu par l'approche centralisée mais l'ensemble des *makespan* sont comparables car il reste dans une bande de 10%. Le temps de réponse le plus petit est issu de approche *DIRAC* avec la réservation en mode *filling*. Il diminue d'un facteur 2, celui observé pour l'approche centralisée. Le meilleur *makespan* ne correspond pas au meilleur temps de réponse, ce qui est une conséquence de l'hétérogénéité de la plate-forme.

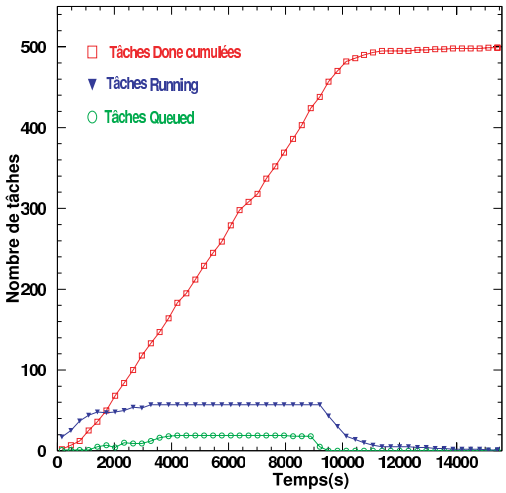
Les temps d'exécution sont du même ordre $1400s$ pour toutes les stratégies, ceci s'explique par le fait que les sites possèdent en moyenne la même capacité ($11.000 NCU.s^{-1}$). Ceci reflète notre plate-forme réelle où la puissance des nœuds est en moyenne autour de 2 GHz et notre méthode d'attribution de la puissance des nœuds utilisée par site. La différence des temps de réponse provient donc des temps d'attente local et global. Le temps d'attente global le plus important de $3750s$ est celui lié à l'ordonnancement *DIRAC* statique. Par contre pour cette approche, le temps d'attente local comparé à l'approche centralisée est moindre d'un facteur 2. Dans le cas d'une approche *DIRAC* dynamique, le temps moyen d'attente local est nul car la correspondance se fait directement à partir du nœud et il s'exprime alors pour les agents soumis aux ressources. Le passage du déploiement statique à la réservation donne une amélioration de 10% sur le temps de réponse moyen. Le mode de réservation *filling* donne près de 50% de gain sur le temps de réponse moyen obtenu par l'approche centralisée.

La figure 6.7(b) illustre le nombre de demandes par unité de temps sur le service *correspondance*, ceci dans le cas du déploiement statique (en haut), avec réservation mode simple (au milieu) et avec réservation mode *filling* (en bas). Lors de la phase d'initialisation de la plate-forme, la charge est importante sur le service

correspondance pour un déploiement statique et est moindre pour un déploiement dynamique. Le nombre total de demandes avec l'approche statique est de 872 et de 699 dans le cas dynamique avec une charge plus homogène sur le service *correspondance*.

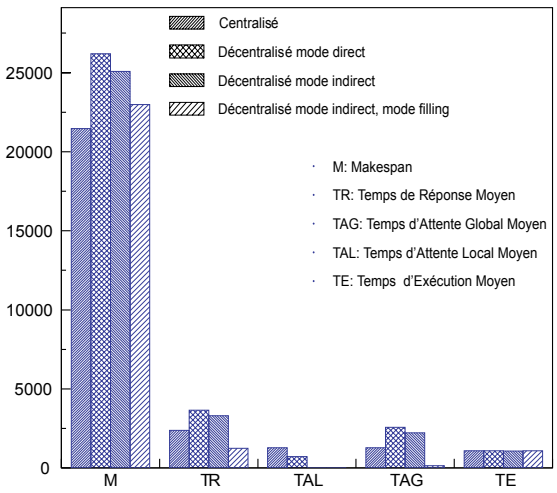


(a) Approche centralisée, $\triangle t = 0$.

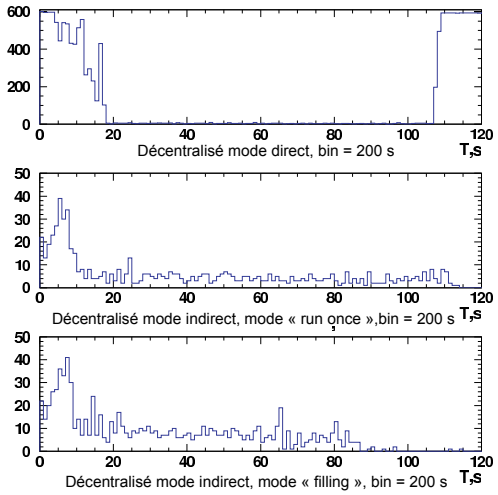


(b) Approche *DIRAC*.

FIG. 6.6 – Évolution du cardinal des états des tâches avec une plate-forme dédiée.



(a) Métriques et mesures vs. temps(s). Comparaison des stratégies : centralisée, *DIRAC* avec un approche statique et les deux autres modes de *DIRAC*.



(b) Taux de requête de tâche sur le service *correspondance* pour les trois stratégies de DIRAC en fonction du temps (binage de 200s).

FIG. 6.7 – Résultats et caractéristiques des stratégies évaluées.

Après avoir étudié les performances de ces deux approches dans un cas d'utilisation, nous étudions maintenant la confrontation de *DIRAC* dans un contexte de méta-ordonnancements concurrents.

6.6.1 Méta-ordonnancements concurrents

Une plate-forme distribuée est partagée entre différentes *OV*. Il arrive de plus que chaque organisation virtuelle soumette des tâches par l'intermédiaire d'un système de gestion de charge particulier. Nous parlons d'ordonnanceur communautaire et sommes dans le cas d'une plate-forme dites co-méta-ordonnancée ou de méta-ordonnancement concurrent. Cette situation arrive dans le projet *DIRAC* car des sites sont partagés entre deux systèmes de *SDGE* : *DIRAC* et *LCG*. La problématique est de savoir quelles influences ont ces méta-ordonnancements l'un sur l'autre ou de manière plus générale si une plate-forme peut supporter plusieurs méta-ordonnancements concurrents.

Pour cette évaluation, nous considérons la même plate-forme décrite précédemment en section 6.5. Nous choisissons de prendre deux *OV* A et B. Ces deux *OV* soumettent deux charges globales distinctes. Ces charges sont générées avec les mêmes paramètres utilisés en section 6.5. Nous examinons ensuite les trois cas de méta-ordonnancement concurrents suivants :

- Les *OV* A et B soumettent respectivement par deux méta-ordonnanceurs décentralisés.
- Les *OV* A et B soumettent respectivement par deux méta-ordonnanceur centralisés.
- L' *OV* A soumet par un méta-ordonnanceur décentralisé et l' *OV* B soumet par un méta-ordonnanceur centralisé.

Quand les deux méta-ordonnancements sont identiques, nous parlons de méta-ordonnancements homogènes centralisés ou décentralisés sinon nous sommes dans le cas de méta-ordonnancements hybrides. La figure 6.8 donne les métriques et mesures pour les deux *OV* conjointement et séparément dans les cas de méta-ordonnancement précités. Le *makespan* conjoint pour les deux *OV* est meilleur pour les approches décentralisées homogènes mais reste comparable dans une bande de 15% avec consécutivement le makespan conjoint centralisé homogène puis le pire cas observés du méta-ordonnancement hybride où nous mesurons une augmentation supérieure à 30%. Les *makespans* respectifs des deux *OV* sont similaires dans les deux cas homogènes et cette différence est plus importante d'un tiers avec le méta-ordonnancement hybride. Dans ce cas, l'approche centralisée à un meilleur makespan de 30% par rapport à celui de l'approche décentralisée. Si nous regardons séparément les deux makespan respectifs des deux *OV* dans la situation hybride, ils subissent une détérioration supérieure à un facteur 2 par rapport aux *makespans* homogènes. Les plus petits temps de réponse sont obtenus avec l'approche centralisée homogène et restent proches de ceux du cas décentralisé homogène.

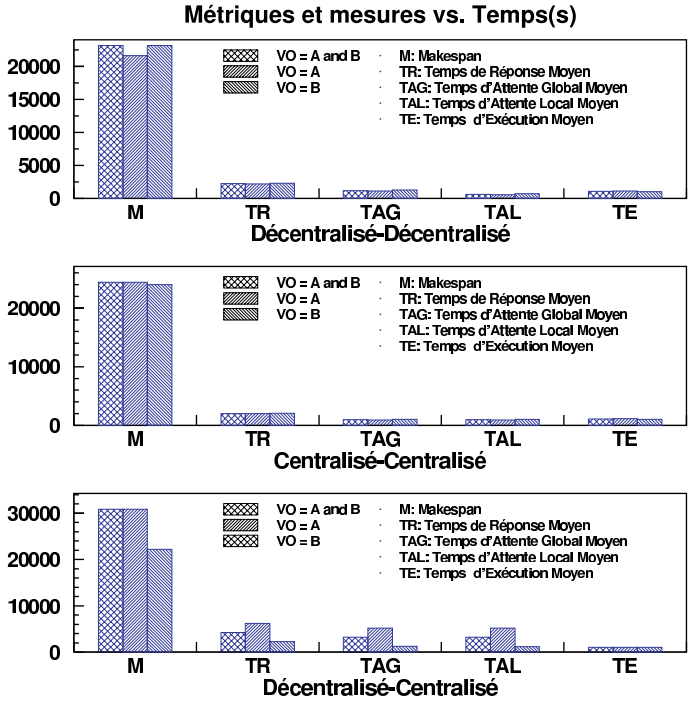


FIG. 6.8 – Comparaison de méta-ordonnancements concurrents et leurs influences sur deux organisations virtuelles (A et B) dans différents cas : A et B soumettent par deux méta-ordonnanceurs décentralisés (en haut). A et B soumettent par deux méta-ordonnanceurs centralisés (au milieu). A et B soumettent respectivement par un méta-ordonnanceur décentralisé et un méta-ordonnanceur centralisé (en bas).

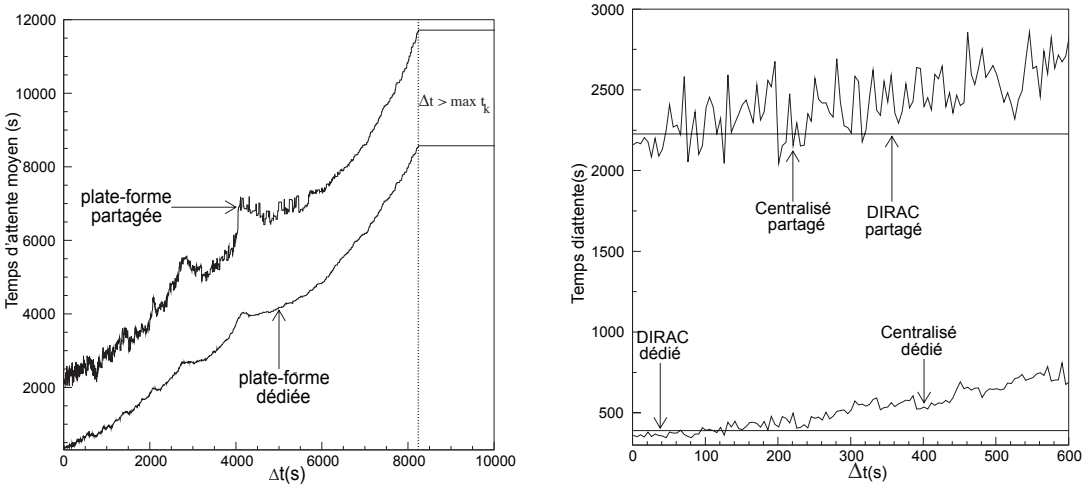
Ces temps sont détériorés d'un facteur supérieure à 2 dans le cas hybride. L'approche décentralisée subit toujours la plus grande dégradation avec un tiers de différence avec l'approche centralisée. L'approche centralisée est plus réactive et tolère mieux un méta-ordonnancement concurrent. L'approche décentralisée par contre est moins réactive. Les temps d'attentes globaux sont aussi moindres pour l'approche centralisée. L'approche centralisée semble donc mieux indiquée, mais ceci reste toujours dans un cas parfait où la vue du système est cohérente. Nous allons étudier dans la suite l'influence de cette vue du système dans nos stratégies et architectures.

6.6.2 Comportements aux limites

Une des problématiques avec des simulations est de transcrire ou de répercuter avec réalisme des effets issus d'une plate-forme réelle. Une autre problématique d'un système est aussi de connaître son comportement aux limites. Cette connaissance est utile pour comprendre son comportement. Nous essayons ici de répondre à ces deux problématiques.

Parmi nos observations, nous avons la dégradation de la plate-forme qui intervient pour l'approche « push » centralisée de *LCG* (voir section 4.7). Pour retranscrire cet effet dans le simulateur, nous prenons comme mesure de la dégradation de la plate-forme la détérioration du système d'information. *DIRAC* n'utilise pas de système d'information centralisé, ne dépend donc pas de la fréquence de rafraîchissement de l'information centralisé et se base sur une vue locale pour ces décisions d'ordonnancement. Si une détérioration de l'information intervient celle-ci se situe donc individuellement pour chaque site et ne nuit pas à l'ensemble du système. Néanmoins nous montrons ici le comportement de *DIRAC* aux limites en mesurant la dégradation que subit si tous les agents subissait des latences.

Nous prenons toujours la même plate-forme et les caractéristiques de charges globales et locales décrites précédemment en section 6.5. Nous considérons successivement une plate-forme dédiée, c'est à dire avec une seule charge globale soumisse par une *OV* puis nous considérons le cas partagé où les sites ont une charge locale et une seule charge globale. Nous montrons les causes de cette dégradation sur les temps d'attente des tâches.



(a) Balayage de 0 à 10.000s.

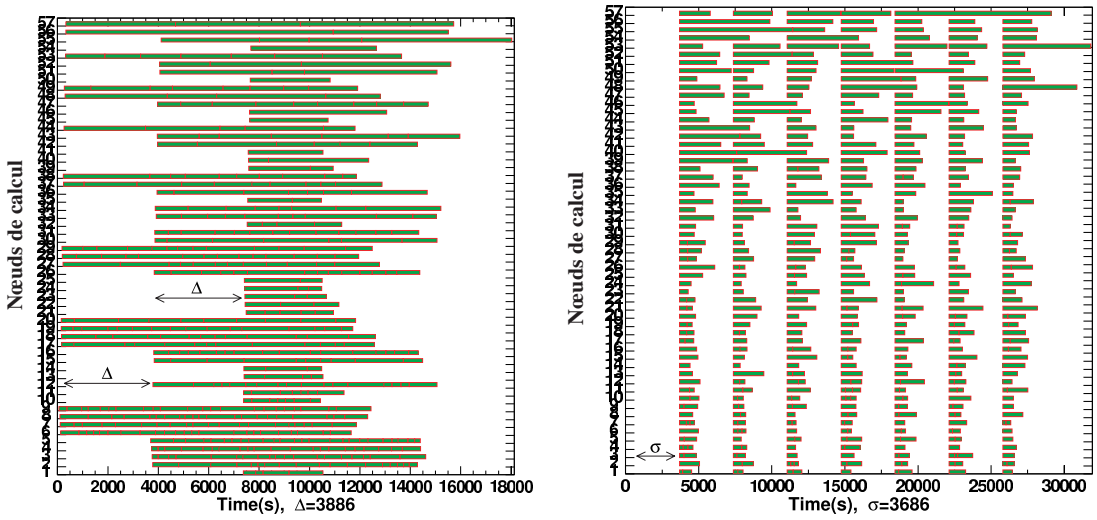
(b) Zoom de 0 à 600s et comparaison avec le temps moyen d'attente de *DIRAC*.

FIG. 6.9 – Temps moyen d'attente pour les tâches en fonction de la période de rafraîchissement $\triangle t$ du système d'information pour l'approche centralisée avec une plate-forme dédiée et partagée.

La figure 6.9(a) montre l'impact de la variation de la période $\triangle t$ pour l'ordonnancement centralisé sur le temps d'attente moyen $attente_m$ dans un contexte dédié puis partagé. Nous avons effectué pour ceci une variation du paramètre $\triangle t$ avec un pas de 5s. Les temps d'attente moyens obtenus pour *DIRAC* sont donnés à titre indicatif car ceux-ci ne dépendent pas de $\triangle t$ (figure 6.9(b)).

Pour un $\triangle t$ inférieur à $95s$, le temps d'attente moyen reste meilleur pour l'ordonnancement centralisé dans un contexte dédié et à peu près inférieur à $60s$ dans un contexte partagé, par contre il se dégrade rapidement ensuite. L'effet est plus chaotique dans un contexte partagé. Le plateau observé donne la borne supérieure qui correspond à la situation où toutes les tâches sont affectées au même site soit dans la situation où $\triangle t > \max_{k \in \mathcal{M}t} t_k$.

Les figures 6.10 illustrent l'occupation des nœuds pour l'ensemble de la plate-forme avec la période de rafraîchissement du système d'informations centralisé σt très grande pour l'approche centralisée ((3686s) en figure 6.10(a) et avec la période de vérification de l'état des ressource par l'agent *DIRAC* σt très grand (3686s) en figure 6.10(b). Les nœuds sont représentés par ordre croissant de puissance de haut en bas. Le temps d'exécution moyen varie entre $1025s$ et $1250s$. Le temps moyen entre deux arrivées successives de tâches est de $17s$. Nous observons avec un grand σt supérieur au temps moyen d'exécution, que les tâches sont séquentialisées en fonction de cette période.



(a) Approche centralisée, $\Delta t = 3686$s. (b) Approche décentralisée, $\sigma t = 3686$s.

FIG. 6.10 – Diagramme de *Gantt* pour l'ensemble des nœuds de la plate-forme. Les nœuds sont représentés par ordre croissant de puissance de haut en bas.

Pour un Δt très grand, nous observons que les tâches sont allouées par site. Au début de la période de soumission, toutes les tâches sont affectées au site qui parait le moins chargé dans le système d'information. Après la mise à jour de celui-ci, un autre site est toujours choisi car il apparaît comme le moins chargé pendant une période Δt.

Dans le cas de *DIRAC*, nous observons une autre forme de fragmentation. Les tâches sont ordonnancées par tranche de σt. Ici, les agents sont synchronisés sur

la même période, ce qui explique cet ordonnancement par lot. Contrairement à l'approche précédente, la dégradation est linéaire.

Dans la suite, nous détaillerons nos observations issus des expérimentations pour comparer les avantages et inconvénients de chaque solution.

6.7 Discussion sur les stratégies et architectures

Dans un cas idéal où l'information sur la plate-forme est cohérente, l'approche centralisée offre de bons résultats mais chacun sait qu'il est impossible de garantir qu'une plate-forme distribuée d'une telle envergure reste stable et de fournir une vue parfaite et instantanée. Les principales et fréquentes défaillances sont les pannes réseaux, le dépassement de l'utilisation de disques, l'indisponibilité des services, les mauvaises configurations des systèmes locaux et les pannes de courant comme nous le décrivons dans le chapitre 4. Dans un tel contexte, il est très difficile d'avoir une image globale fiable de l'ensemble de la plate-forme. Cet ordonnancement est totalement dépendant de la performance du système d'information. Ce système souffre souvent aussi de problème d'extensibilité.

L'approche *DIRAC* contourne ce problème car une des caractéristiques de cet ordonnancement est l'absence totale de vue globale du système. Il se base sur une vue locale et partielle des ressources. Chaque ressource en fonction de son état demande et reçoit une charge adaptée à sa capacité. Les tâches sont séquentialisées et l'événement déclencheur de l'ordonnancement est la disponibilité d'une ressource, par opposition à l'ordonnancement centralisé où l'événement déclencheur est la soumission d'une tâche.

L'approche centralisée est très sensible en terme de performance à la moindre détérioration du système d'information de la plate-forme. Si nous prenons comme mesure de dégradation l'état de la vue du système, avec une mise à jour tout les $60s$ dans notre contexte, nous avons une pénalité de 20% sur le temps de réponse par rapport à une mise à jour parfaite (0%). Un changement d'état impromptu d'une ressource n'est pas pris en compte immédiatement par le système. Pendant ce laps de temps dans un contexte de calcul intensif, les choix sont désastreux. Ceux-ci peuvent être la surcharge d'un site comme nous l'avons observé avec *LCG* (voir chapitre 4, section 4.7.5). Ces approches souffrent souvent aussi de problème de famine pour certaines ressources en opposition à *DIRAC* où toutes ressources disponibles sont utilisées immédiatement. Avec une défaillance du système d'information, l'effet de dégradation serait augmenté. La détérioration de la collection des éléments d'information utilisée pour les choix d'ordonnancement intervient pour les deux solutions. Pour obtenir respectivement une vue locale de la ressource avec une période σt et une vue totale du système avec une période de rafraîchissement Δt, *DIRAC* et l'approche centralisée doivent définir ces périodes par rapport au temps moyen d'exécution et l'inter-temps de soumission, sinon nous observons des phé-

nomènes de fragmentations corrélées à ces périodes.

L'aspect dynamique de la plate-forme oblige un ordonnancement opportuniste, réactif et non prédictif. De plus dans *DIRAC*, les résultats obtenus sont équivalents en terme de performance à une approche centralisée avec une période de rafraîchissement Δt supérieure à $60s$. Cette approche est plus stable, plus souple et évite de mettre en place une infrastructure de système d'informations. De plus, elle permet la réservation de ressource qui augmente considérablement d'un facteur trois ses résultats en mode *filling* car la tâche est directement délivrée au nœud et diminue les pertes de temps. Cette souplesse améliore les temps de réponse et en échange de son gain, nous payons le coût d'une charge importante sur le composant *correspondance* avec près de 50% de requêtes en plus. Cette amélioration a un coût qui correspond au déploiement inutile d'agents qui meurent tout de suite après la réponse négative du service *correspondance* (299 dans notre cas). Le nombre de déploiements inutiles peut être diminué, si nous opérons une soumission d'agents *pilotes* en fonction de la disponibilité des tâches dans la file d'attente globale.

DIRAC est approprié dans un contexte de régime permanent et saturé. Il a été appliqué pour des applications de productions de données créant un tel contexte. Nous examinons maintenant les possibilités d'évolution de *DIRAC* notamment dans le cas concret de mélanger des tâches d'analyses et de productions pour une même *OV* par un même système de *SDGE*, *DIRAC*.

6.8 Ordonnancement bi-applicatifs

Au sein d'une même communauté, plusieurs applications coexistent et s'exécutent de manière concurrentielles. Le critère d'ordonnancement d'une application est fortement conditionné par les besoins et demandes de l'application elle-même. Par exemple, pour l'expérience de physique des particules *LHCb*, deux types d'applications existent et possèdent des besoins et des caractéristiques antagonistes. L'une, la production Monte-carlo de données, nécessite de maximiser l'utilisation des ressources, tandis que l'autre, l'analyse des données, tend à minimiser le temps de réponse pour l'utilisateur.

Le cas d'un méta-ordonnancement multi-critères est rarement considéré au sein d'une plate-forme hétérogène. Dutot et al. considère dans [96], un ordonnancement bi-applicatif pour l'utilisation du système et la minimisation du temps de réponse, mais ces travaux se situent au niveau d'une grappe de nœuds. L'ordonnancement bi-applicatif se traduit souvent par des priorités différentes entre les tâches. Cette notion de priorité entre tâches induit souvent vla préemption de tâche au bénéfice d'une autre de priorité supérieure. La préemption de tâche signifie que nous suspendons une tâche provisoirement pour permettre l'exécution d'une tâche de plus haute priorité. Ce procédé est courant au niveau d'un système d'exploitation notamment pour un modèle temps partagé, mais il reste peu appliqué pour les *SD-*

GE. Nous avons étudié cet aspect de méta-ordonnancement multi-critères dans un environnement partagé. Pour ceci, nous définissons de nouvelles stratégies passives pour *DIRAC* et nous procédons ensuite à leurs évaluations. Nous nous inspirons pour nos hypothèses des caractéristiques des applications réelles de *LHCb* comme décrit dans le chapitre 1. Nous formulons notre problème comme suit.

Nous considérons deux types d'applications une application de type A, une autre de type B et leurs charges respectives. Dans la réalité, l'application A correspond à la production de données et l'application B à l'analyse de données. La charge de type A est lourde et génère à elle seule une situation de régime permanent et saturé. Nous nous attacherons avec cette application à privilégier le débit de calcul. Nous considérons que l'application B intervient moins souvent et est prioritaire par rapport à l'application A. La charge de B prioritaire est légère d'un facteur 3 par rapport à la charge A. Nous désirons favoriser les temps de réponse des tâches de l'application B prioritaire et mélanger les charges A et B sur les mêmes ressources. Nous nous posons donc la question de savoir comment ces deux applications peuvent coexister et comment ces deux charges interagissent. Le gain obtenu pour une application a des conséquences directes pour l'autre application. Pour ceci, nous proposons d'utiliser la même architecture d'ordonnancement *DIRAC*. Nous définissons pour ceci de nouvelles stratégies basées sur la priorité statique d'une application par rapport à une autre.

6.9 Stratégies basées sur des priorités statiques

La figure 6.11 montre notre classification des différentes stratégies passives décentralisées avec ou sans priorité dans le cadre de *DIRAC*.

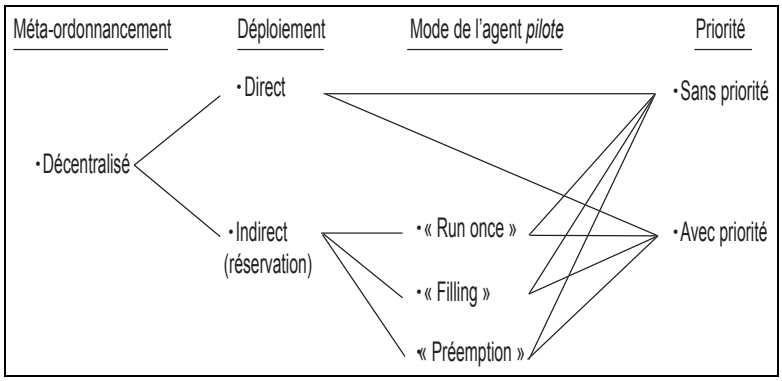


FIG. 6.11 – Taxonomie des stratégies dans l'approche « pull » *DIRAC*.

Pour prendre en compte la différence de priorité entre les applications, nous rajoutons des méta-données (type de l'application et sa priorité) aux méta-taches. Le service *correspondance* gère les priorités en attribuant les méta-tâches de plus hautes priorités comme décrit dans l'algorithme 8 en annexe K. Ces priorités sont

attribués de manière statique entre l'application A et B prioritaire. Nous venons de voir comment nous gérons l'attribution de priorité sur la partie du service correspondance. La distribution des décisions d'ordonnancement dans *DIRAC* comprend le service *correspondance* et l'agent. Avec l'introduction de la priorité dans les tâches, le comportement de l'agent reste strictement le même que décrit précédemment en section 6.4 et repris dans les algorithmes 3, 5 et 6 et 7 dans l'annexe K. Nous définissons par surcroît une nouvelle stratégie pour l'agent *pilote*.

6.9.1 Stratégie de préemption dans le mode « pull »

Nous définissons un nouvel algorithme « pull » de préemption au niveau de l'agent *pilote* pour le mode indirect (algorithme 9, annexe K).

L'agent *pilote*, une fois déployé sur le nœud, initialise un compteur de temps (ou de CPU). Il crée la description du nœud et demande une méta-tâche mk au service *correspondance* en respect avec les contraintes temporelles. Il lance ensuite l'exécution de mk. En parallèle, l'agent demande des tâches de priorité supérieure à mk au service *correspondance*. Cette demande s'effectue avec le temps restant sur le nœud calculé à partir du compteur et des tailles de tâches planifiées sur le nœud. En cas de non succès, il réitère sa demande tous les τ unité de temps. En cas de succès, l'agent vérifie si la charge dépasse un seuil arbitrairement fixé. Ceci est le nombre de tâches en exécution sur le nœud soit la charge du nœud. Si ce seuil est dépassé l'agent préempte une tâche s'exécutant de priorité minimale et l'empile sur une pile. À la fin d'une tâche, l'agent reprend une tâche préemptée en dépilant la pile. La condition d'arrêt de cet algorithme est le fait qu'il n'existe plus de tâche s'exécutant ou en attente d'exécution sur le nœud ou le dépassement du temps disponible sur le nœud. Cet algorithme prend en charge la gestion du compteur de temps dans tous les cas de figure.

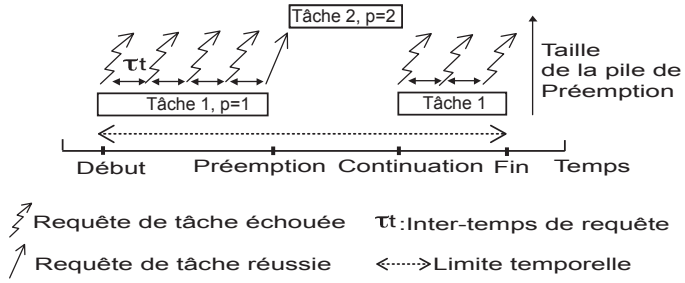


FIG. 6.12 – Le principe de la préemption illustré sur un nœud.

6.9.2 Étude expérimentale préliminaire de la préemption

Nous fixons les mêmes caractéristiques pour la plate-forme que dans la section 6.5. Nous considérons une plate-forme dédiée soumise à deux charges. La

charge globale pour l'application A a les mêmes paramètres que décrites dans le tableau 6.1. La charge B prioritaire a les mêmes caractéristiques pour les temps d'exécution mais a un taux de soumission inférieur d'un facteur 3 à celle de l'application A.

Le tableau 6.2 nous donne les résultats des expérimentations obtenues pour l'application B, avec τt fixé petit ($\sim$ 1 s). Dans l'approche indirecte, la première constatation est que le temps de réponse local ne s'exprime pas pour les tâches car il s'applique en fait aux agents *pilotes*. Les temps d'exécutions restentdsq<w toujours très proches ($\sim$ 1000 s). Les performances sont donc directement influencées par les temps d'attente.

Nous exprimons nos gains par rapport aux temps de référence qui sont les temps de B dans le mode *direct* sans priorité soit la première ligne du tableau. Pour le temps d'attente, le passage du mode *direct* à *indirect* fait gagner 8%. Le mode *filling* fait gagner 90%. L'ajout de la priorité permet de gagner successivement par rapport aux temps d'attente référence 79%, 97%, 98% à 99% avec respectivement le mode *direct*, le mode *indirect* « run once », le mode *indirect* « filing », et finalement le mode *indirect* préemption. L'influence du changement de comportement de l'agent *pilote* « run once » à « filling » n'apporte pas non plus de grande amélioration avec les priorités.

Stratégies décentralisées « pull »	**M**	**TR**	**TAL**	**TA**	**TE**
mode *direct*, sans priorité	34245.0	4906.4	703.1	3879.7	1026.7
mode *indirect*, sans priorité	31770.3	4554.4	0.0	3563.9	990.5
mode *direct*, filling, sans priorité	24028.6	1334.8	0.0	302.6	1032.2
mode *direct* , priorité	23885.6	1732.1	718.4	789.7	942.5
mode *indirect*, run once, priorité	23979.6	1114.0	0.0	84.5	1029.5
mode *indirect*, filling, priorité	26105.0	1127.7	0.0	42.0	1085.7
mode *indirect*, préemption	22681.2	954.8	0.0	1.9	952.9

TAB. 6.2 – Mesures et métriques pour les différentes stratégies de l'application B prioritaire qui nécessite de favoriser le temps de réponse (M : Makespan, TR : Temps de Réponse moyen, TAL : Temps moyen d'Attente Locale, TA : Temps d'Attente global, TE : Temps d'Exécution moyen).

Pour estimer le coût à payer, nous regardons la charge sur le service *correspondance* pour les différentes stratégies. La figure 6.13 nous donne le taux de requête sur le service *correspondance* pour les différentes stratégies.

Dans le cas *direct*, nous observons toujours pendant la phase d'initialisation de la plate-forme un fort taux de demande ($\sim$ 50). Le passage au mode *indirect* baisse cette charge de moitié pendant la phase d'initialisation mais par contre reste toujours homogène sur le durée de l'expérimentation. Le mode « filling » augmente le niveau de charge de 10%. L'ajout de la priorité dans le mode « filling » et « run

once » ne modifie pas la charge de manière significative. Par contre, l'ajout de notre mode préemptif augmente beaucoup la charge (facteur 100) sur le service correspondance ce qui est directement lié à la période τt.

Cette première évaluation permet de présager de très bons apports par rapport aux stratégies liés au déploiement. Celle qui nous permet d'avoir un meilleur apport est la préemption dans le mode *indirect*. Néanmoins, il peut être intéressant de mieux comprendre et borner les paramètres de cette stratégie. Nous comprenons que ce gain est lié par exemple au taux de soumission des tâches, leurs durées ou la capacité de la plate-forme. C'est ce dont nous allons discuter dans la section suivante.

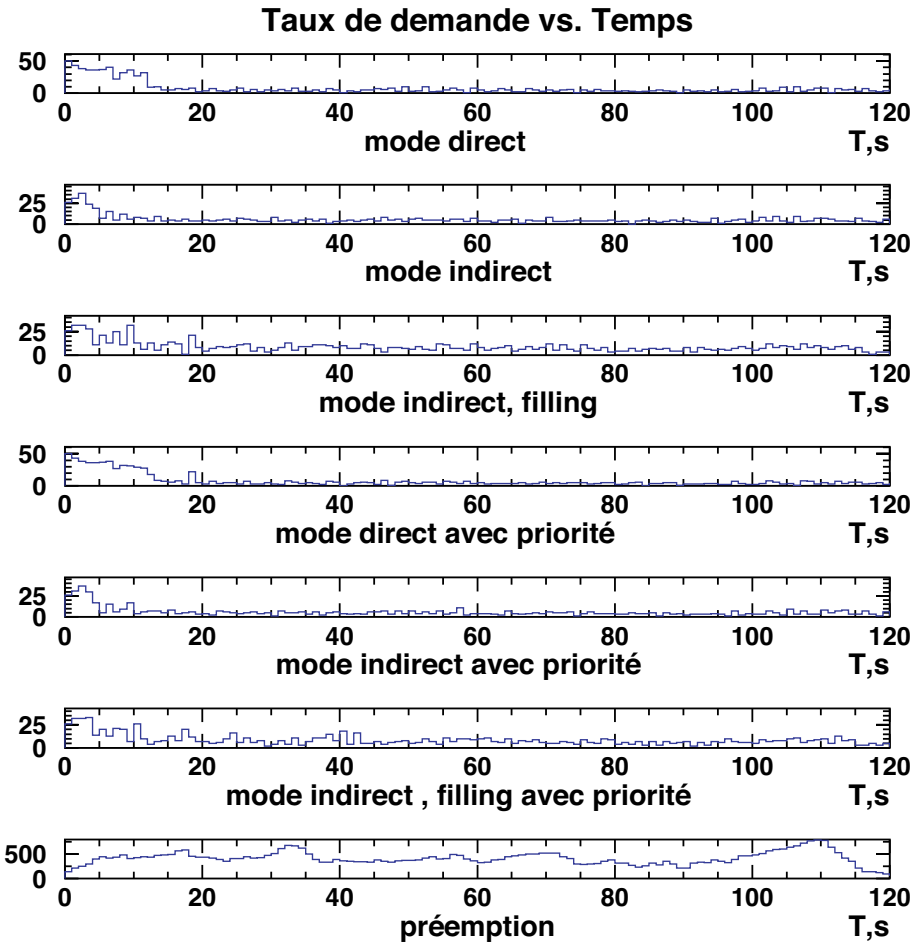
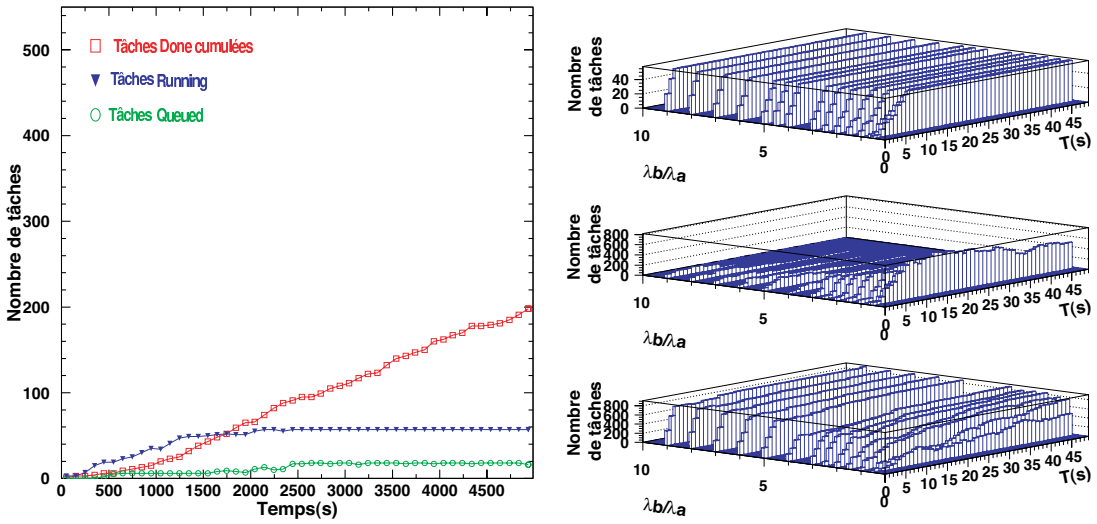


FIG. 6.13 – Taux de requête sur le service *correspondance* avec une plate-forme partagée pour les différentes stratégies.

6.9.3 Mise au point des conditions d'expérimentations

Nous désirons étudier plus particulièrement le gain important issu de la préemption mesuré à 99% pour le temps d'attente. Pour ceci nous nous plaçons maintenant dans un contexte dédié, avec la même plate-forme qu'en section 6.9.2, et considérons, la charge de type A qui génère à elle seule une situation de régime permanent et saturé comme l'illustre la figure 6.14(a) et la charge B prioritaire. La charge des applications A est caractérisée par un taux de soumission λ_A et un temps d'exécution moyen μ_A. En ajoutant la charge de l'application B, de taux de soumission λ_B et de temps d'exécution moyen μ_B, nous pouvons alors évaluer l'impact du mélange des deux charges pendant le régime permanent et saturé.

Un taux de soumission est donné pour une capacité précise de la plate-forme. Nous décorellons la capacité de la plate-forme en exprimant le rapport des deux taux de soumission ($\frac{\lambda_B}{\lambda_A}$). De plus, pour étudier l'impact du taux de soumission de l'application B λ_B, nous faisons varier λ_B en fixant les autres paramètres. L'effet de cette variation sur le cardinal de tâche s'exécutant en fonction du rapport des taux de soumission et du temps, est illustré sur la figure 6.14(b).



(a) Évolution du cardinal des tâches pour la charge liée à l'application A seule ($\lambda_A = 0.08$, $\mu_A = 1000\ s$, $T = 5000\ s$).

(b) Évolution des tâches en exécution vs. le temps et le rapport $\frac{\lambda_B}{\lambda_A}$ pour la charge liée à l'application A et B (en haut), pour la charge A (au milieu) et ensuite pour la charge B (en bas).

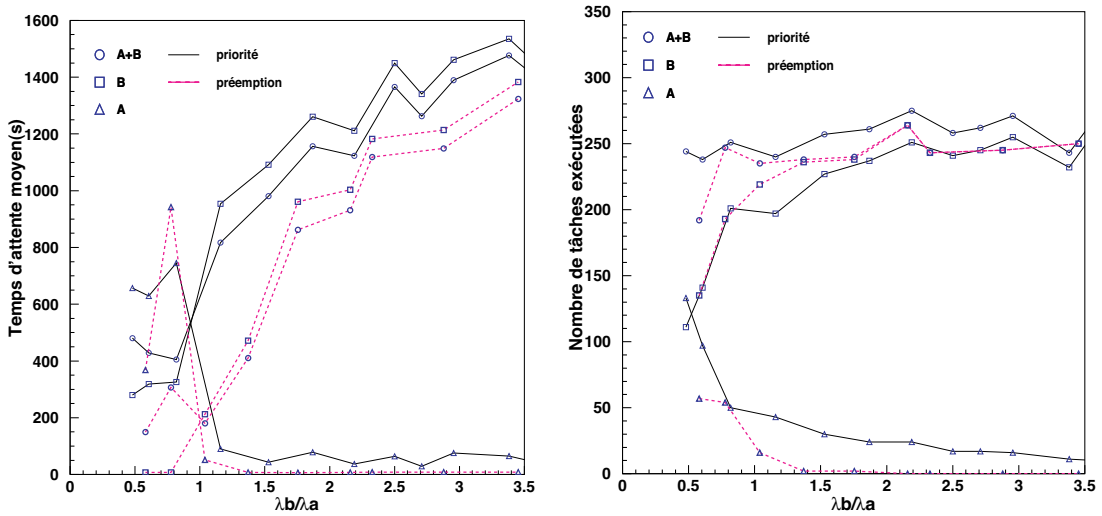
FIG. 6.14 – Illustration des conditions d'expérimentations.

Nous observons sur la figure 6.14(b) du haut que la phase de saturation des ressources est atteinte plus rapidement. Quand le taux de soumission de B augmente, l'application A s'exécute de moins en moins jusqu'à ne plus s'exécuter, au contraire de l'application B qui commence à saturer toutes les ressources (voir figure 6.14(b)

du bas). Dans la suite, nous nous intéressons seulement à une période représentative T de l'expérimentation, c'est-à-dire, après avoir obtenu la saturation de la plate-forme comme l'illustre la figure 6.14(a).

La figure 6.15(a) donne le temps de réponse moyen pour les deux applications puis séparément en fonction de $\frac{\lambda_B}{\lambda_A}$. La figure 6.15(b) représente le nombre de tâches exécutées pour les deux applications puis séparément en fonction de $\frac{\lambda_B}{\lambda_A}$.

Avec la préemption, le temps d'attente de B est nul avec un rapport de taux inférieur à 0.8. L'augmentation ensuite du temps d'attente signifie qu'il commence à y avoir plus de tâches B que de nœuds disponibles donc de l'attente, soit le début de la saturation des ressources par l'application B seule. Nous pouvons mesurer le gain de la préemption par rapport à la simple priorité pour l'application B en regardant la surface entre les deux courbes priorité et préemption pour B pour les temps d'attentes. Si le temps de réponse diminue, c'est dû au fait que de moins en moins d'applications A s'exécutent et que le système souffre de famine pour ces applications. Ce phénomène intervient vers le rapport 0.7. Cette borne exprime le changement des performances pour les deux applications et quand l'application A commence à souffrir de famine et que B est exécutée exclusivement.



(a) Temps de réponse moyen pour les deux applications puis séparément en fonction de $\frac{\lambda_B}{\lambda_A}$.

(b) Nombres de tâches exécutées pour les deux applications puis séparément en fonction de $\frac{\lambda_B}{\lambda_A}$.

FIG. 6.15 – Comparaison entre la préemption et la priorité.

La contrainte forte de cette étude est la négligence de l'aspect des données. Nous montrons maintenant comment nous allons incorporer cet aspect dans notre modèle et simulateur pour l'évaluation de la préemption.

6.10 Extension du modèle pour la prise en compte des données

Nous envisageons maintenant de compléter notre modèle et simulateur pour prendre en compte l'aspect des communications et des données. Nous allons brièvement décrire l'extension de notre modèle.

Chaque tâche est caractérisée par sa dépendance à un ensemble de fichiers qui sont hébergés par les différents éléments de stockage ou *SE* (« Storage Element ») de la plate-forme. Un même fichier peut d'ailleurs être stocké sur différents *SE*. Nous parlons alors de réplication de fichiers et les différents exemplaires d'un même fichier sont appelés des réplicatas. Une tâche qui s'exécute dans l'environnement de méta-calcul est supposée importer à travers le réseau les fichiers qui ne sont pas locaux au nœud. De ce fait, nous devons ajouter au temps d'exécution d'une tâche le temps de transfert de tous les fichiers nécessaires à ce calcul. Ceci conduit pour une tâche quelconque un temps d'exécution qui correspond à la formule (6.2).

$$t_{total} = t_{transfert} + t_{exécution} \tag{6.2}$$

Nous définissons de plus le temps moyen de transfert de donnée TT pour l'exécution d'une tâche. Nous considérons les données, tâches et propriétés suivantes.

- Nous avons un ensemble $\mathcal{F} = \{file_1, file_2, \ldots\}$ de fichiers répartis sur les *SE* de la plate-forme. Dans la réalité, cet ensemble n'est pas fixe, car de nouveaux fichiers y sont ajoutés en permanence. Il est important de noter que les fichiers sont en lecture seule et peuvent de ce fait, exister en autant de copies que l'on souhaite sans créer un risque d'informations incohérentes dans la grille.
- Chaque tâche utilise un ensemble de fichiers distincts et un même fichier peut être utilisé par plusieurs tâches.
- Nous avons un ensemble $\mathcal{SES} = \{SE_1, SE_2, \ldots\}$ de *SE* disponibles. Chaque *SE* possède un ensemble de fichiers et un même fichier peut avoir des réplicatas sur différents *SE*. Chaque *SE* possède une capacité C qui correspond au nombre maximal de fichiers qu'il peut héberger. Le *SE* est dit saturé lorsque cette capacité est atteinte. La capacité totale du système est la somme des capacités de tous les *SE*.

Après la définition de cette extension de modèle, nous devons l'inclure dans notre simulateur.

6.10.1 Répercussion sur le simulateur

Pour inclure ce modèle, deux composants sont rajoutés au simulateur pour prendre en compte l'aspect des données : 1) le Storage Element (*SE*), qui est un élément de stockage de données et 2) le catalogue des fichiers (*File C atalog*).

L'élément de stockage est le composant qui stocke les données, soit les fichiers. Le *SE* implémenté est multi-ports, c'est-à-dire que plusieurs transferts sont possibles en même temps. Le *lfn* (*logical file name*) est le nom absolu ou logique d'un

fichier. Le nommage des fichiers dans un *SDGE* se compose d'une paire *lfn-pfn*. Il désigne un seul fichier et est unique. Le *pfn* (*physical file name*) associé à un *lfn* est le chemin physique de la localisation d'un fichier. Celui-ci se compose par exemple dans la réalité de l'élément de stockage contenant le fichier, le protocole d'accès et le chemin absolu. Un *lfn* peut posséder plusieurs *pfn*. Le catalogue des fichiers résout cette correspondance.

6.10.2 Expérimentations

Nous considérons toujours la même plate-forme décrite précédemment en section 6.5. La stratégie utilisée pour nos expérimentations est une approche « data in place ». Nous envoyons les tâches sur les sites où se situent les données. Les transferts de données sont donc locaux du *SE* au nœud. Nous rappelons que la bande passante locale est de 1000 méga-octets et la globale de 100 méga-octets. Un *SE* est associé à un site. Nous générons tout d'abord l'ensemble des fichiers et leurs tailles puis les distribuons aléatoirement sur les *SE*. Nous ne répliquons pas les fichiers. Pour chaque tâche, nous gênerons la dépendance aux données se trouvant sur le même *SE*. Les charges A et B ont les mêmes caractéristiques que dans la section 6.9.2.

Le tableau 6.3 présente les résultats obtenus. Nous constatons que l'ajout de la dépendance aux données ne modifie pas les apports de la préemption car celui-ci est toujours de 99%. Les résultats sont similaires à ceux exposés dans le tableau 6.2 et section 6.9.2.

Stratégies décentralisées	**M**	**TR**	**TAL**	**TA**	**TE**	**TT**
mode *direct*, sans priorité	32481.7	4505.6	687.8	3511.5	994.1	3.6
mode *indirect*, run once, sans priorité	30366.4	4115.3	0.0	3206.2	909.1	3.3
mode *indirect* filling , sans priorité	23311.6	1206.4	0.0	261.3	945.1	3.0
mode *direct*, priorité	23661.5	1993.5	699.8	950.8	1042.7	3.1
mode *indirect*, run once, priorité	24021.8	1252.0	0.0	279.3	972.7	3.6
mode *indirect* filling, priorité	26193.0	1152.5	0.0	138.4	1014.1	3.2
mode *indirect*, préemption	23112.7	987.8	0.0	5.1	982.6	3.2

TAB. 6.3 – Mesures et métriques pour l'aspect données et l'application B prioritaire. (M : Makespan, TR : Temps de Réponse moyen, TAL : Temps moyen d'Attente Local, TA : Temps d'Attente global, TE : Temps d'Exécution moyen).

6.10.3 Discussion

Dans le cas d'un ordonnancement bi-applicatifs, si nous considérons la minimisation du temps de réponse d'une application B par rapport au débit d'une

autre application A, la stratégie *indirecte* en mode *filling* et la préemption apporte le plus grand gain (99%). La stratégie *filling* garde une charge moins lourde sur le service *correspondance* tandis que la préemption sollicite ce service à très haut débit. Cette importante charge constitue une pénalité pour l'amélioration des performances mais l'évaluation de mécanismes à la demande, de notification ou d'enchère (*bidding*) [192] de type « push », pour améliorer l'ordonnancement de l'application B et diminuer la charge sur le service *correspondance* nous semble pertinent pour la suite de ces travaux.

Si nous cherchons à minimiser le temps d'attente des applications prioritaires B dans le régime saturé des ressources, la charge des applications prioritaires doit rester légère et ne pas dépasser un certain seuil. Ce seuil correspond à $\frac{\lambda_B}{\lambda_A} \sim 0.7$ dans notre étude. Éventuellement, l'attribution des priorités aux tâches doit être ajustée pour prendre en compte ce seuil. Dans le régime où $\frac{\lambda_B}{\lambda_A} < 0.7$, il n'y a pas de détérioration significative du *makespan* pour les tâches de type A. Pour ces tâches, c'est la minimisation du *makespan* qui est le critère d'optimisation.

Quand la charge de B devient lourde et dépasse un seuil du rapport des taux de soumission (caractérisé dans notre cas à 0.7), A souffre de famine. Pour régler cette problématique, il faudrait mettre en place des priorités dynamiques avec prise en compte du vieillissement des tâches ou de l'énumération de tâches A et B déjà exécutées, des critères d'équité entre utilisateurs [165] à travers une politique des administrateurs du *SDGE*

L'introduction de la dépendance de données et avec l'approche « data in place » donne toujours un gain important avec la préemption et ne modifie pas les performances obtenues sans données. Bien que le modèle est simplifié, nous présageons mieux les avantages, souplesse et évolution que peut encore offrir *DIRAC* dans ce contexte et l'apport de favoriser des transfert locaux plus que globaux sur la plate-forme. L'aspect de gestion des données doit être plus fine que celle proposée. Les aspects réseaux doivent se renforcer de réalisme et des véritables stratégies de réplications avec des heuristiques et utilisation des statistiques d'utilisation des fichiers.

Cette étude sera surtout validée expérimentalement puis re-analysée avec le simulateur. Les manques pour compléter cette étude sont des véritables comportements d'utilisateur sur la grille déterminant mieux les descriptions réelles de charge. Beaucoup de scénarios sont imaginables avec les approches *indirect*. Les technologies de *checkpointing* comme dans le projet Condor [166] conjugué avec de la migration de tâche d'un nœud à un autre mérite aussi d'être explorées. La complémentarité des tâches considérées, une consommant du temps de calcul tandis que l'autre passe essentiellement du temps en entrées-sorties laisser à penser qu'une évaluation d'ordonnancement à temps partagé comme présenté en [41] serait intéressante.

La principale limitation à ces approches se situe surtout au niveau de la sécurité.

Dans un environnement *SDGE*, faire l'opération de mise en relation au niveau du nœud d'un site suppose de changer l'identité de l'utilisateur au niveau du nœud. Pour des raisons de traçabilité, l'administrateur du site doit savoir qui utilise ses machines donc ce mécanisme doit exister. Nous observons aujourd'hui de plus en plus d'utilisateurs ou de concepteurs de système surcouche initié par *DIRAC* avoir le même discours et demander la mise en place de telles mécanismes rendant bientôt possible le déploiement réel de toutes ces stratégies.

6.11 Conclusions et perspectives

Avec l'outil de simulation, nous avons démontré qu'une approche centralisée est meilleure en terme de performance pour le calcul intensif qu'une approche décentralisée avec une différence de 15%. Mais ceci est dans le cas idéal où la période de rafraîchissement est nulle. Au delà de $95s$ dans un contexte dédié, l'approche « pull » a des résultats équivalents et surtout plus stables. Ceci se confirme dans un contexte partagé. L'approche passive « pull » autorise des scénarios plus riches qui permettent d'améliorer les performances de 50% par rapport à l'approche centralisée, notamment par la réservation de ressources.

L'approche passive de *DIRAC* autorise aussi beaucoup de scénarios moins faciles, à implémenter avec une approche centralisée. *DIRAC* permet un méta-ordonnancement multi-critères dans un environnement partagé. Nous proposons différentes stratégies passives pour l'ordonnancement de deux applications aux besoins antagonistes. Nous faisons une étude expérimentale de ces stratégies basées sur des priorités. Notre stratégie de préemption passive semble la plus intéressante et fournit un gain énorme de 99% par rapport à une approche *DIRAC* simple. La préemption garde ses atouts avec des dépendances à des données et des transfert locaux. Il reste toutefois des problématiques comme la gestion des données qui semble un sujet d'étude à part entière. Une des perspectives immédiates de ces travaux est d'appliquer ces stratégies dans le système réel pour les applications de *LHCb* et de les valider expérimentalement. Il serait intéressant de mener des travaux sur la migration de tâches d'un site fortement chargé à un site moins chargé.

CONCLUSION

Cette thèse a présenté les problématiques des systèmes de calcul distribués à grande échelle, les techniques à utiliser et les mécanismes à mettre en œuvre pour la définition d'un système distribué à grande échelle, appliqué à l'expérience de physique des particules *LHCb*. *DIRAC* conjugue les aspects des grilles institutionnelles et de calcul global pour être un système léger, robuste, sécurisé, communautaire et extensible.

Le choix d'une architecture logicielle orientée service, reposant sur des services sans états et des agents, est une des clefs du succès de *DIRAC*. Les protocoles de procédure à distance reposant sur des connexions « stateless » sont adaptés à l'hétérogénéité et à la dynamicité, d'une plate-forme de calcul à grande échelle. Ces techniques et une régulation de charge basée sur une stratégie passive « pull », où l'ordonnanceur ne maintient aucune information sur les ressources du système, ont permis des déploiements à grande échelle dans le cadre de la collaboration *LHCb*. Ainsi, *DIRAC* a connecté plus de 6.000 processeurs répartis sur une soixantaine de sites dans le monde, pouvant exécuter plus de 5.500 tâches simultanées et aussi gérer la gestion des données résultantes avec un montant supérieur à 100 téra-octets stockées et transférées en deux mois et demi.

Les classes d'applications les mieux adaptées pour ce type de système sont les applications séquentielles multi-paramétriques faiblement ou nullement couplées. Notre étude approfondie des applications à mettre en œuvre ont rapidement montré l'impact de l'ordonnancement sur l'efficacité d'un tel système, notamment dans un contexte de régime permanent saturé. Nous nous sommes donc attachés à l'étude, définition, modélisation du service de gestion de ressources et des services annexes comme le système de surveillance.

Pour l'évaluation de l'ordonnancement de *DIRAC*, nous avons proposé une modélisation et un simulateur autorisant la comparaison de stratégies et d'architectures pour l'ordonnancement et le méta-ordonnancement. La validation théorique et pratique de ce simulateur certifie le réalisme et traduit bien le comportement d'un système de traitement par lots réel. En effet, nous avons obtenu une différence moyenne de 10.5% par rapport à la réalité pour les temps de réponse et de 12% pour le makespan. Avec cet outil, nous avons justifié l'approche de *DIRAC* face à d'autres approches centralisées de types « push ». Nous avons démontré qu'une approche centralisée est meilleure en terme de performance pour le calcul intensif qu'une approche décentralisée, mais ceci est reste comparable au performance de *DIRAC* dans une bande de 15% et est vrai seulement si nous avons un état global cohérent. Une approche centralisée est fragile dans un contexte de système partagé à grande échelle, contrairement à *DIRAC* qui préserve l'autonomie de chaque site et une auto-adaptabilité du système.

L'approche utilisée pour l'inter-opérabilité avec d'autres systèmes tel *LCG* ouvre la voie à de nouvelles façons d'appréhender les systèmes de calcul distribués. Ceci se symbolise par un système qui déploie sa propre infrastructure en surcouche des infrastructures logicielles et matérielles existantes, notamment par des agents mobiles accentuant l'importance d'un déploiement non intrusif, léger et rapide. Nous avons évalué avec le simulateur le potentiel de ces approches notamment pour conjuguer deux types d'applications avec des besoins antagonistes, qui se traduisent par des priorités différentes. Nous mesurons un gain de 99% avec des stratégies basées sur de la préemption par rapport à la première stratégie de *DIRAC*. L'introduction de l'aspect des données et une approche « data in place » donne des résultats similaires.

Le résumé de nos contributions est le suivant :

- Étude, définition et implémentation d'un système distribué à grande échelle avec le service de gestion de ressources et d'autres services annexes. Ce système répond à la problématique du calcul scientifique dans un contexte de régime permanent saturé et est en production depuis mai 2004 (chapitres 3 et 4) ;
- Modélisation, formalisation de la méthodologie et réalisation/validation d'un simulateur pour l'étude de méta-ordonnancement dans le contexte d'une plate-forme de système distribué à grande échelle (chapitres 5 et 6) ;
- Diffusion des résultats dans plusieurs revues, conférences et colloques et à l'occasion de divers séminaires.

Les résultats de ces travaux ouvrent de nombreuses perspectives, et nous allons maintenant présenter quelques unes d'entre elles.

Dans un avenir proche, l'un des objectifs principaux est compléter l'étude préliminaire des stratégies liées au méta-ordonnancement multi-critères sur la plate-forme réel de *DIRAC*. Ce passage à la réalité nécessitera les outils de sécurité adaptés comme la délégation de l'autorisation et de l'authentification au niveau des nœuds de calcul. Ce point est en discussion dans la communauté des grilles et est en bonne voie de validation car d'autres groupes de travail, inspirés par nos travaux, commencent à exprimer les mêmes besoins.

Les principales évolutions du système se baseront sur l'aspect très complexe de la gestion des données dans un système distribué à grande échelle, comme la prise en compte des contraintes de latence sur l'accès, le transfert et la réplication des données. Ces aspects sont totalement nouveaux et surtout atteignent des échelles jamais atteintes auparavant. Nous pouvons faire une analogie entre un système de gestion de charge et un système de gestion de données, les tâches correspondent alors à des transferts que nous pouvons ordonnancés, préemptés et migrés. De plus, la gestion de la charge est intrinsèquement liée aux données, ce qui laisse à penser que la suite des travaux présentés dans ce manuscrit se situera sur l'aspect des données.

Le système générique de base étant en place, *DIRAC* peut s'enrichir de nouvelles fonctionnalités, être appliqué dans d'autres cadres et élargir son champ d'applications. D'autres domaines comme la biologie et d'autres expériences de physique sont intéressés par *DIRAC* et souhaitent faire son évaluation. Ces collaborations permettront d'approfondir les modèles d'organisation de *DIRAC* comme un modèle hiérarchique et s'enrichir de nouveaux cas utilisateurs.

Au niveau de la modélisation et simulation d'une plate-forme méta-ordonnancée, ces travaux sont en intégration au sein de *Simgrid* comme contribution. L'utilisation prochaine du simulateur dans d'autres projets augmentera sa capacité, sa maturité. Nous comptons élargir ses capacités comme la prise en compte de tâches parallèles et de plus de stratégies d'ordonnancement au niveau local.

LISTE DES ABRÉVIATIONS

API Application Programming Interface

ASP Application Service Provider

DAG Directed Acyclic Graph

DIRAC Distributed Infrastructure with Remote Agent Control

GSI Globus Security Infrastructure

HTTP Hypertext Transfer Protocol

IETF Internet Engineering Task Force

ISP Internet Application Provider

LAN Local Area Network

LDAP Lightweight Directory Access Protocol

MAN Metropolitan Area Network

MDS Meta Directory Service

MPI Message Programming Interface

MPP Massively Parallel Architecture

NES Network Enabled Server

NWS Network Weather Service

OGSA Open Grid Services Architecture

OV Organisation Virtuelle

PC Personal Computer

PSA Parameter Sweep Application

PVM Parallel Virtual Machine

RMI Remote Method Invocation

RPC Remote Procedure Call

RSA Rivest Shamir Adelman

RSL Resource specifiation Language

PSE Problem Solving Environment

SETI Search for Extraterrestrial Intelligence

SDGE Système Distribué à Grande Échelle

SGR Système de Gestion de Ressources

SDK Software Development Kit

SOAP Simple Object Acces Protocol

SQL Structured Query Language

SSL/TLS Secure Sockets Layer/Transport Layer Secutity

UDDI Universal Description Discovery and Integration

URI Uniform Resource Identifiers

WAN World Area Network

W3C World Wide Web Consortium

WSDL Web services Description Language

WSRF Web services Resource Framework

XML eXtensible Markup Language

LISTE DES PUBLICATIONS

Revues internationales avec comité de lecture

[1] V. Garonne, E. Caron, and A. Tsaregorodtsev. Definition, modeling and simulation of a grid computing system for high throughput computing. *FGCS Future Generation Computer Science*, To appear, 2006.

[2] I. Stokes-Rees, A. Tsaregorodtsev, V. Garonne, R. Graciani, M. Sanchez, M. Frank, and J. Closier. Developing LHCb Grid Software : Experiences and Advances. *Concurrency and Computation :Practice and Experience*, 18(1–18), 2005.

[3] S. Burke, F. Harris, Ian Stokes-Rees, I. Augustin, F. Carminati, J. Closier, E. van Herwijnen, A. Sciaba, D. Boutigny, J. J. Blaising, Vincent Garonne, Andrei Tsaregorodtsev, P. Capiluppi, A. Fanfani, C. Grandi, Roberto Barbera, E. Luppi, G. Negri, L. Perini, S. Resconi, M. Reale, A. De Salvo, S. Bagnasco, P. Cerello, K. Bos, David L. Groep, W. van Leeuwen, J. Templon, O. Smirnova, O. Maroney, F. Brochu, and D. Colling. Hep applications and their experience with the use of datagrid middleware. *Journal of Grid Computing*, 2(4) :369–386, 2004.

Conférences internationales avec comité de lecture

[4] V.Garonne, A.Tsaregorodtsev, and E.Caron. A study of meta-scheduling architectures for high throughput computing : Pull vs. push. In *The 4th International Symposium on Parallel and Distributed Computing, University of Lille1 , France, July 4th - 6th 2005*. IEEE Computer Society, 2005.

[5] V.Garonne, A.Tsaregorodtsev, and I.Stokes-Rees. Dirac : A scalable lightweight architecture for high throughput computing. In *5th International Workshop on Grid Computing (GRID 2004), 8 November 2004, Pittsburgh, PA, USA, Proceedings*, pages 19–25. IEEE Computer Society, 2004.

[6] I. Stokes-Rees, A. Soroko, C. Cio, A. Tsaregorodtsev, V. Garonne, R. Graciani, M. Sanchez, M. Frank, J. Closier, and G. Kuznetsov. Developing lhcb grid software : Experiences and advances. In *UK e-Science All Hands Meeting*, Nottingham, UK, 9April 2004.

[7] V.Garonne I.Stokes-Rees, A.Tsaregorodtsev. Dirac workload management system. In *CHEP'04*, Interlaken, 10 2004.

[8] V.Garonne I.Stokes-Rees, A.Tsaregorodtsev. Grid information and monitoring system using xml-rpc andinstant messaging for dirac. In *CHEP'04*, Interlaken, 10 2004.

[9] V. Garonne, R.Graciani-Diaz, J.Saborido, M.Sanchez, and R.Vizcaya-Carrillo. A Lightweight Monitoring and Accounting System for LHCb DC04 Production. In *CHEP'04*, Interlaken, 10 2004.

[10] A. Tsaregorodtsev, V. Garonne, et al. Dirac - distributed infrastructure with remote agent control. *2003 Conference for Computing in High-Energy and Nuclear Physics (CHEP 03), La Jolla, California.*, C0303241 :TUAT006, 2003.

Conférences nationales avec comité de lecture

[11] E. Caron, V. Garonne, and A. Tsaregorodtsev. Étude d'architectures de méta-ordonnancement pour le calcul intensif en régime permanent et saturé. In *RenPar 2005. 16eme Rencontres Francophones du Parallélisme*, Croisic, 5-8April 2005.

Conférences nationales sans comité de lecture

[12] V. Garonne. Grille de calcul et physique des particules. In *Journées Jeunes Chercheurs*, La roche en Ardennes, November 2004.

Rapports de Recherche

[13] E. Caron, V. Garonne, and A. Tsaregorodtsev. An evaluation of meta-scheduling architecture for high trougthput computing. In *Internal INRIA research report*, 5-8April 2005.

ANNEXES

LISTE DES ANNEXES

Annexe A

Liens utiles

Le site internet de *DIRAC* :
`http://dirac.cern.ch/`

Le site Web de la distribution *DIRAC* :
`http://lhcb-wdqa.web.cern.ch/lhcb-wdqa/distribution/`

Accès internet du répertoire CVS du code source dirac par *viewcvs* :
`http://isscvs.cern.ch/cgi-bin/viewcvs-all.cgi/DIRAC/?cvsroot=dirac`

Le site *Savannah* [1] *DIRAC* :
`http://savannah.cern.ch/projects/dirac2`

Le site *LCG* de *LHCb* :
`https://uimon.cern.ch/twiki/bin/view/LCG/LhcbPage`

Le site Web du service de comptabilité *DIRAC* :
`http://lhcb01.pic.es/DIRAC/Accounting/`

Le site Web du service de configuration *DIRAC* :
`http://lhcb01.pic.es/DIRAC/Configuration/`

Le site Web du service de surveillance *DIRAC* (production) :
`http://lhcb01.pic.es/DIRAC/Monitoring/Production/`

Le site Web du service de surveillance *DIRAC* (test) :
`http://lhcb01.pic.es/DIRAC/Monitoring/Test/`

Le site Web de Simgrid :
`http://simgrid.gforge.inria.fr/`

[1] *Savannah* est une branche du célèbre projet *SourceForge*.

Annexe B

Cycle d'une tâche dans *DIRAC* et les états correspondants

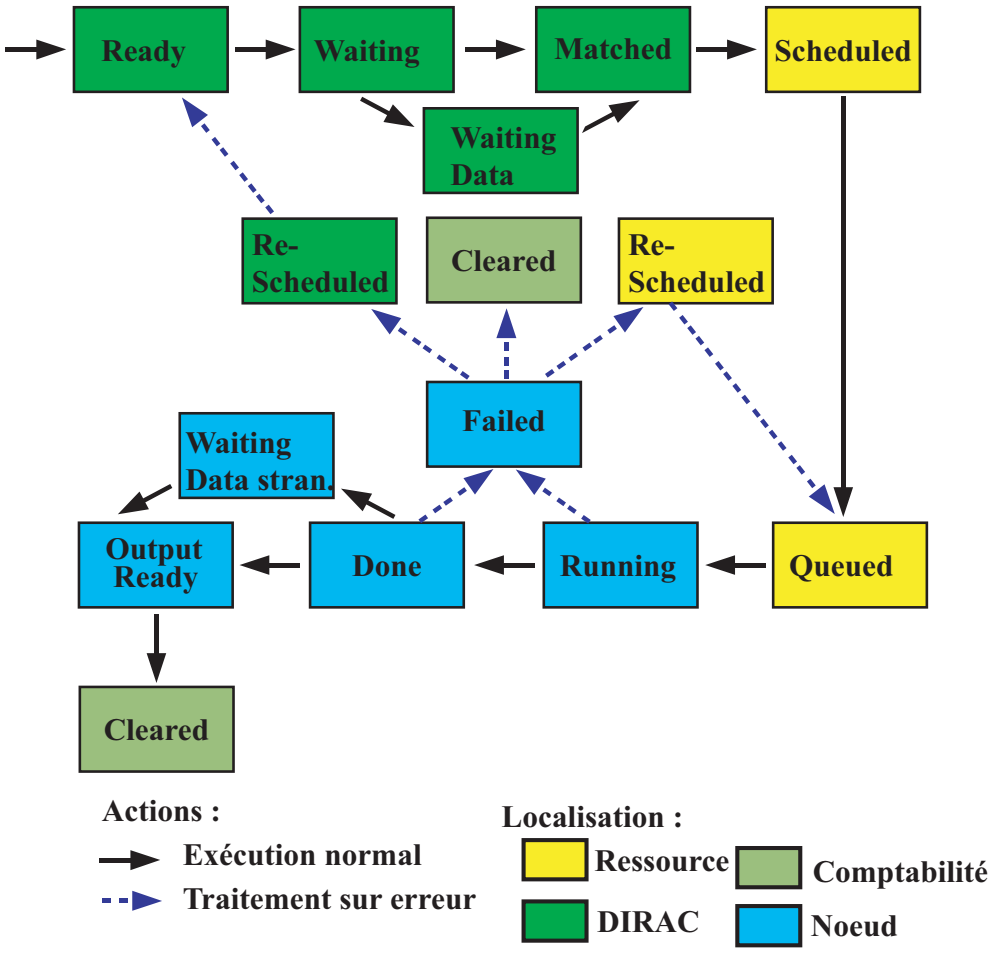


FIG. B.1 – Cycle d'une tâche dans *DIRAC*.

Pour une tâche soumise à *DIRAC*, nous avons défini, en accord avec la figure B.1, les états suivants :

Ready. La tâche vient d'être soumise à *DIRAC* et est en cours de placement dans les files d'attente globales.

Waiting. La tâche est disponible dans les files d'attente globales et attends la demande d'un agent.

Waiting Data. Cet état est utilisé quand la tâche soumise nécessite des données en entrées qui ne sont pas encore disponibles dans le système.

Matched. La tâche a été mise en relation avec une ressource.

Scheduled. La tâche est soumise à la ressource.

Queued. La tâche est planifiée sur la ressource.

Running. La tâche est en exécution sur le nœud.

Done. La tâche a fini son exécution et prépare l'envoi des données (« outputsandbox » et données produites).

Waiting Data Transfert. Cet état signifie que les données produites par la tâche ont fini d'être transférées.

OutputReady. L'envoi des données est terminé. L'utilisateur peut alors retirer l'« outputsandbox »

Cleared. La tâche est effacée du système de gestion de charge pour être transmise vers le service de comptage.

Failed. Cet état indique une erreur lors de l'exécution de la tâche. l'état *Stalled* est aussi utilisé pour indiquer qu'une tâche ne signale plus son état.

Re-scheduled. La tâche est ré-ordonnancée localement ou globalement selon le type de l'erreur rencontrée.

Annexe C

Les commandes *DIRAC*

Les commandes d'utilisation du système *DIRAC* sont présentées ici. Il faut noter que toutes ces commandes sont aussi disponibles directement par les librairies Python. Nous distinguons les commandes utilisateurs pour la gestion des ressources et des tâches puis les commandes administrateurs du système.

C.1 Les commandes utilisateurs pour la gestion des tâches

> dirac-job-submit [nom de fichier]

Cette commande soumet une tâche à *DIRAC*. La description de la tâche est en *ClassAds*. Cette commande retournera l'identifiant de la tâche (*tâcheId*).

> dirac-job-status [*tâcheId*]

Cette commande donne le statut (voir annexe B) courant de la tâche.

> dirac-get-job-parameter [*tâcheId*]

Cette commande retourne les méta-informations associées à un tâche *DIRAC*. Certaines de ces informations sont définis par le système comme la consommation CPU, et d'autres sont ajoutables par l'utilisateur.

> dirac-job-logging-info [*tâcheId*]

Cette commande donne toutes les dates associées aux événements d'une tâche comme le passage de l'état *running* à l'état *done*.

> dirac-job-alive [*tâcheId*]

Cette commande est une primitive qui en fonction du signal de battement de cœur vérifie si la tâche est oui ou non toujours en exécution.

> dirac-job-delete [*tâcheId*]

Cette commande annule une tâche et la supprime du service de gestion de ressources.

> dirac-job-get-output [*tâcheId*]

Cette commande sert à récupérer les fichiers de sorties ou « output sandbox » d'une tâche.

> dirac-job-get-input [*tâcheId*]

Cette commande sert à récupérer les fichiers d'entrées ou « input sandbox » d'une tâche.

> dirac-job-reschedule [*tâcheId*]

Cette commande replace une tâche pour l'exécution.

> dirac-proxy-init

Cette commande s'utilise pour créer le *proxy* ou mandataire de sécurité. L'utilisateur doit fournir son certificat x509. Celui-ci est utilisé pour l'authentification et les autorisations de l'utilisateur.

C.2 Les commandes utilisateurs pour la gestion des données

Les commandes de gestion de données sont principalement utilisé pour interroger et interférer avec les fichiers de catalogue qui donnent la correspondance entre le nom logique d'un fichier, le *lfn*, et son nom physique, le *pfn*. Un *pfn* se compose d'un élément de stockage (SE), d'un protocole d'accès et d'un chemin d'accès. Nous incluons dans le nom du SE le protocole dans les commandes suivantes, e.g. *gridftp ://se1*.

> dirac-rm-copy [lfn] [SE] [chemin]

Cette commande copie le fichier lfn dans l'élément de stockage SE au chemin indiqué.

> dirac-rm-copyAndRegister [lfn] [SE] [chemin]

Cette commande copie le fichier lfn dans l'élément de stockage SE au chemin indiqué et enregistre le nouveau replica dans les catalogues de fichiers.

> dirac-rm-register [lfn] [pfn] [taille] [SE]

Cette commande enregistre un fichier dans les catalogues de fichier avec sa taille et son SE hôte.

> dirac-rm-copyDir [répertoire] [SE] [chemin]

Cette commande copie le répertoire dans l'élément de stockage SE au chemin indiqué.

> dirac-rm-get [lfn]

Cette commande donne les pfn associés au fichier lfn.

> dirac-rm-remove [lfn]

Cette commande supprime un fichier et tous ses réplicas.

> dirac-rm-replicate [lfn] [SEdestination]

Cette commande copie un fichier lfn dans un SE.

> dirac-admin-clear-mask

Cette commande sert à ré-inititialer le masque ou la liste des sites autorisé à prendre des tâches avec *DIRAC*.

> dirac-admin-allow-site [Site]

Cette commande autorise un site à prendre des tâches de *DIRAC*.

> dirac-admin-ban-site [Site]

Cette commande exclut un site de la liste des sites autorisé à prendre des tâches.

> dirac-admin-get-mask

Cette commande est un accesseur pour obtenir la liste des sites autorisés à prendre des tâche.

> dirac-admin-set-mask [mask]

Cette commande est un modifieur pour positionner une nouvelle liste de sites autorisés à prendre des tâche.

> dirac-admin-agent [AgentId]

Cette commande lance un environnement permettant d'interagir avec un agent par la messagerie instantanée Jabber. Cet environnement autorise en outre l'interrogation de la base de donnée locale pour vérifier l'état des tâches localement.

Annexe D

Exemple d'une description d'une tâche d'analyse

Nous présentons ici un exemple de description JDL utilisé dans *DIRAC* pour décrire une tâche d'analyse, utilisant des données en entrées.

```
[
        Requirements = ( member("CNAF_Castor",other.LocalSE));
        Arguments = "jobDescription.xml";
        JobName = "DaVinci_1";
        OutputData =
           {
              "/lhcb/test/DaVinci/v1r0/LOG/DaVinci_v12r11.alog"
           };
        parameters =
            [
                STEPS = "1";
                STEP_1_NAME = "0_0_1"
            ];
        SoftwarePackages =
           {
              "DaVinci.v12r11"
           };
        JobType = "user";
        Executable = "$LHCBPRODROOT/DIRAC/scripts/jobexec";
        StdOutput = "std.out";
        Owner = "paterson";
        OutputSandbox =
           {
              "std.out",
              "std.err",
              "DaVinci_v12r11.alog",
              "DVNtuples.root",
              "DVHistos.root"
           };
        StdError = "std.err";
        ProductionId = "00000000";
        InputSandbox =
           {
              "PrimVtxAnalysis.opts",
              "jobDescription.xml"
           };
        InputData =
           {
              "LFN:/lhcb/production/DC04/v2/DST/00000742_00003493_10.dst"
           };
    ]
```

Annexe E

Exemple d'une description d'une tâche de production

Nous présentons ici un exemple de description JDL utilisé dans *DIRAC* pour décrire une tâche de production de données de 500 événements.

```
[
    Requirements = other.MaxCPUTime >= 200000 ;
    Arguments = "00001018_00000001.xml";
    JobName = "00001018_00000001";
    OutputData =
       {
          "/lhcb/production/DC04/v2/SIM/00001018_00000001_1.sim",
          ...
          "/lhcb/production/DC04/v2/LOG/Gauss_00001018_00000001_1.txt"
       };
    parameters =
        [
            STEP_3_NAME = "00001018_00000001_3";
            STEP_2_NUMBER_OF_EVENTS = "500";
            STEP_4_NAME = "00001018_00000001_4";
            NUMBER_OF_EVENTS = "500";
            STEP_3_NUMBER_OF_EVENTS = "500";
            STEP_5_NAME = "00001018_00000001_5";
            STEPS = "5";
            STEP_1_NAME = "00001018_00000001_1";
            JOB_IDENTIFIER = "00000001";
            STEP_2_NAME = "00001018_00000001_2";
            STEP_1_NUMBER_OF_EVENTS = "500"
        ];
    SoftwarePackages =
       {
          "Boole.v6r5",
          "XmlDDDB.v22r4",
          "Brunel.v24r5",
          "DecFiles.v8r0",
          "Gauss.v15r19",
          "ParamFiles.v3r6"
       };
    JobType = "production";
    Executable = "$LHCBPRODROOT/DIRAC/scripts/jobexec";
    StdOutput = "std.out";
    Owner = "lhcbprod";
    OutputSandbox = { "std.out", "std.err"};
    StdError = "std.err";
    ProductionId = "00001018";
    InputSandbox =  {  "00001018_00000001.xml"};
]
```

Annexe F

Exemple du fichier de configuration d'un agent

Ceci est un exemple de fichier de configuration pour la configuration d'un agent.

```
[Agent]
Root         = /data/DIRAC
Site         = marlhcb.in2p3.fr
Mode         = test
AgentName    = marlhcb.in2p3.fr
LogLevel     = DEBUG
LogFile      = /data/DIRAC/log/agent.log
LogOutput    = stdout,file
PollTimeout = 60

# This lists the queue definitions
[JobAgent]
CEUniqueIds = marlhcb.in2p3.fr/inprocess
#CEUniqueIds = marlhcb.in2p3.fr/fork,marlhcb.in2p3.fr/pbs-example

# These must match the names provided in [JobAgent]/CEUniqueIDs
# Normally all the Queues will be the same Batch System (LRMSType),
# however, in theory, you can have one DIRAC agent managing multiple
# queues on different batch systems.

[marlhcb.in2p3.fr/inprocess]
Name               = inprocess
LRMSType           = InProcess
MaxRunningJobs     = 1
MaxTotalJobs       = 1
TotalCPUs          = 1
MaxCPUTime         = 100000
CPUScalingFactor   = 1
LocalAccountString = garonne
SEUniqueID         = marlhcb.in2p3.fr-File
WorkingDirectory   = /data/DIRAC/work

[marlhcb.in2p3.fr/fork]
Name               = fork-queue
LRMSType           = Fork
MaxRunningJobs     = 1
MaxTotalJobs       = 1
TotalCPUs          = 1
MaxCPUTime         = 100000
CPUScalingFactor   = 1
LocalAccountString = garonne
SEUniqueID         = marlhcb.in2p3.fr-File
WorkingDirectory   = /data/DIRAC/work
```

```
# A typical PBS queue, with commenting
[marlhcb.in2p3.fr/pbs-example]
# For Batch Queues, "Name" must match the submit queue name
Name               = example
# Must be one of: PBS LSF BQS Globus LCG Fork Condor
LRMSType           = PBS
# Will stop submitting once max running jobs is reached
MaxRunningJobs     = 5
# Altnatively will stop submitting when max total jobs is reached
MaxTotalJobs       = 10
# Total CPUs available in batch queue is used to determine how "deep"
# the queue is
TotalCPUs          = 5
MinCPUTime         = 0
MaxCPUTime         = 100000
CPUScalingFactor   = 1
LocalAccountString = garonne
SEUniqueID         = marlhcb.in2p3.fr-File
WorkingDirectory   = /data/DIRAC/work

[marlhcb.in2p3.fr-File]
SEHost             = marlhcb.in2p3.fr
SEProtocol         = File
SEPath             = /data/DIRAC/files

# These need to be configured properly to point to
# local, tier1, and tier0 storage

[LocalSE]
SEName     = marlhcb.in2p3.fr-File
# NOTE: You should set this to your closest Tier1SE site
[Tier1SE]
SEName     = marlhcb.in2p3.fr-File

[TransferAgent]
LocalCache = /data/DIRAC/cache


# Store SIM, DIGI, DST to SE list of your choice
# Usually DIGI goes to Tier1, DST to Tier1 and Tier0
[ProductionJobConfig]
DST  = Tier0SE

# Usually you will want to set the following two options to "yes"
# so that data transfers take place after data generation and that
# data generated to the local SE are registered in the File Catalog
DelayedDataTransfer = no
RegisterLocalData   = no
```

```
AllowInJobTransfer = yes
LogTmpDirectory     = /data/DIRAC/log

[InformationService]
List = http://lxgate03.cern.ch:9130, http://lbnts2.cern.ch:9130

[AutomaticUpdate]
DoAutomaticUpdate = 1
LocalAgentRelease = 1.0.2
```

Annexe G

Le service de monitorage et de comptage

Nous présentons les interfaces Web du service de monitorage et de comptage.

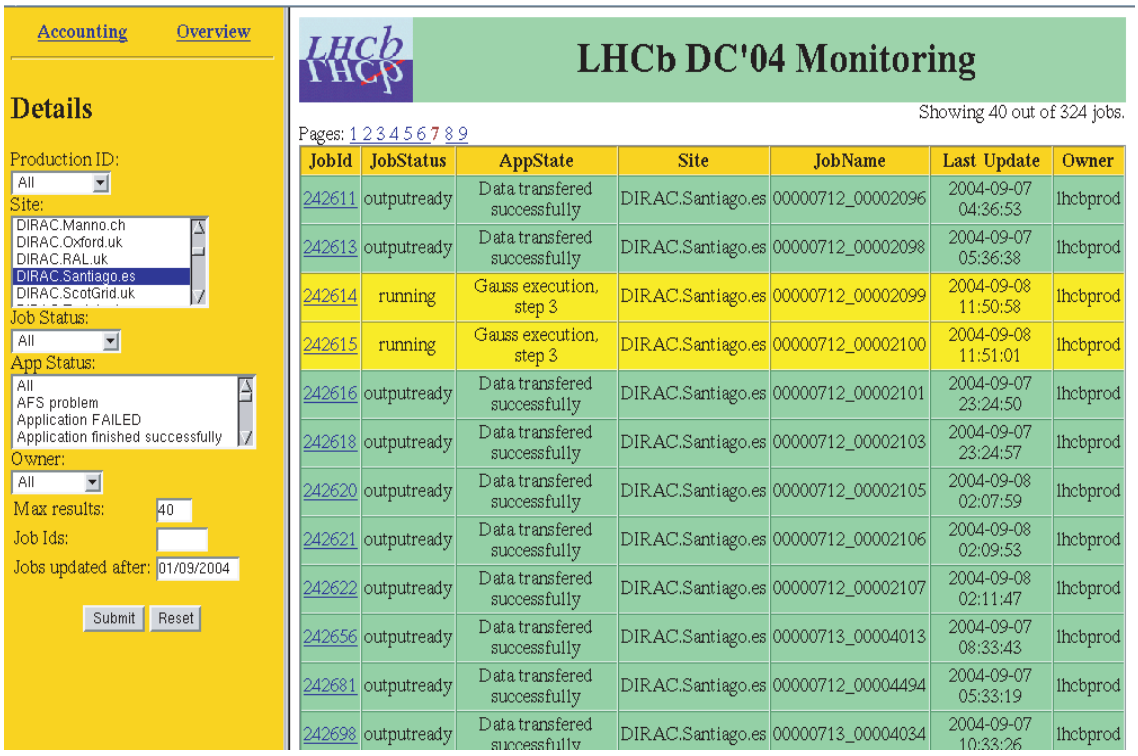


FIG. G.1 – Interface Web du service de monitorage.

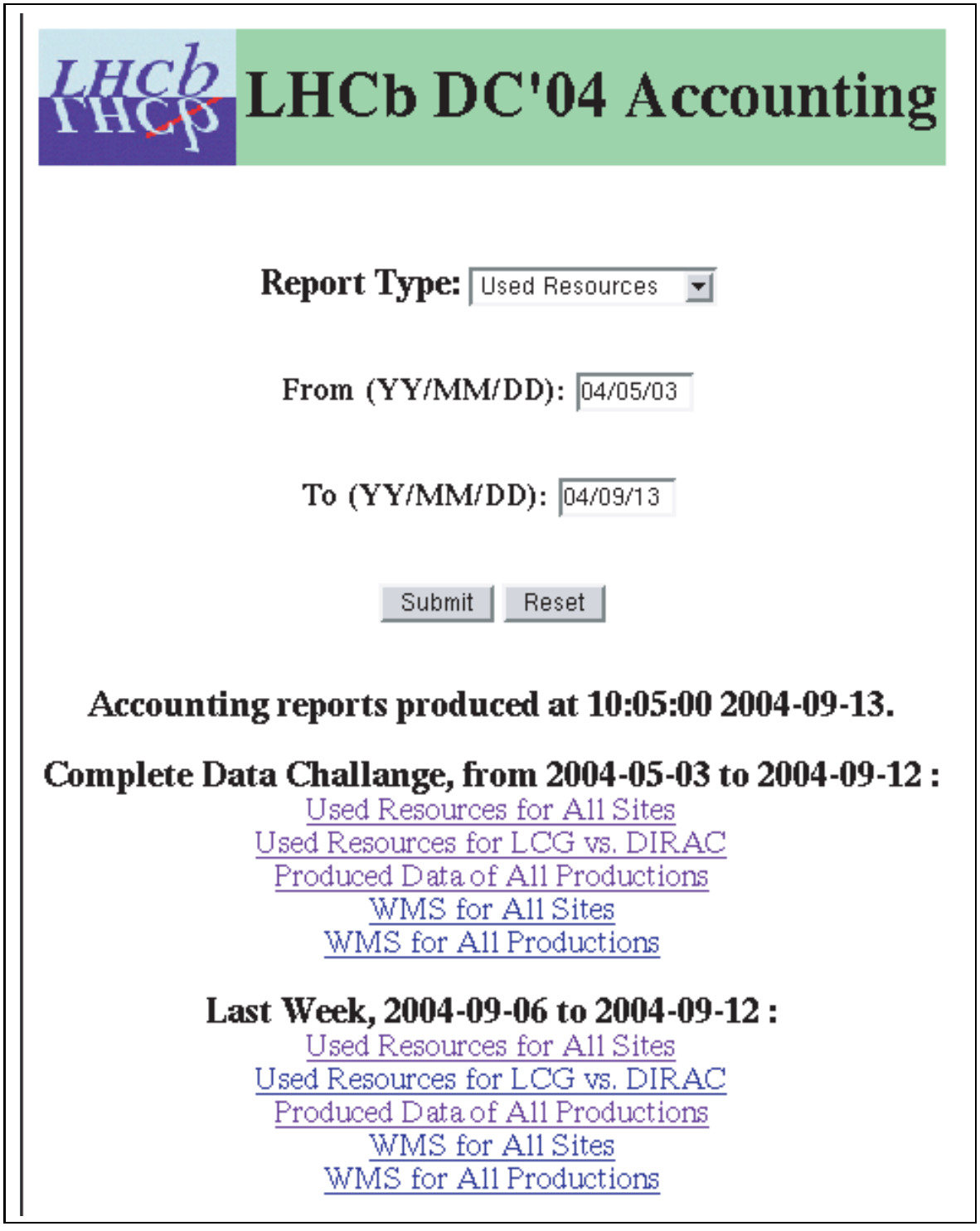


FIG. G.2 – Interface du service de comptage.

Annexe H

Description de la base de données du service de gestion de charge

Nous montrons le schéma MySQL utilisé pour définir la base de données du service de gestion ressource.

```
-----------------------------
-- file : Create JobDB Tables
-- date : 26/09/03
-- Auteur: Vincent Garonne
-----------------------------

CREATE DATABASE JobDB;
-------------------------------------------------------------------------------
use mysql;
USE JobDB;
-- USE test;
-- USE oxdevel;

-------------------------------------------------------------------------------
drop table if exists Job;
CREATE TABLE Job (
    JobID integer not null auto_increment primary key,
    JDL BLOB,
    JobType varchar(100),
    Type  enum ("Normal",
                "Interactive",
                "Checkpointable",
                "MPICH",
                "Partitionable",
                "Checkpointable,
                Interactive",
                "Checpointable, MPICH") not null,
    Site varchar(100),
    Owner varchar(100),
    SubmissionDate Date,
    SubmissionTime time,
    ProductionId     char(8),
    JobName          varchar(32),
    LastUpdate       TIMESTAMP,
    ApplicationStatus varchar(128),
    InputSandBox  enum ('True', 'False') not null,
    OutputSandbox enum ('True', 'False') not null,
    Status enum        ('submitted',
                        'running',
                        'waiting',
                        'waitingdata',
                        'ready',
```

```
                        'done',
                        'rescheduled',
                        'outputready',
                        'matched',
                        'scheduled',
                        'queued',
                        'failed',
                        'stalled',
                        'deleted' ) not null
);
-------------------------------------------------------------------------

drop table if exists InputData;
CREATE TABLE InputData (
    JobID integer not null,
    file varchar(100),
    PRIMARY KEY (JobID, file),
    FOREIGN KEY(JobID) REFERENCES Job(JobID)
);

-------------------------------------------------------------------------

drop table if exists OutputData;
CREATE TABLE OutputData (
    JobID integer not null,
    file varchar(100),
    PRIMARY KEY (JobID, file),
    FOREIGN KEY(JobID) REFERENCES Job(JobID)
);

-------------------------------------------------------------------------

drop table if exists TaskQueues;
CREATE TABLE TaskQueues (
    TaskQueueId integer not null auto_increment primary key,
    name        varchar(100),
    Private     enum ('True', 'False') not null,
    Priority    integer,
    Requirements BLOB
);

-------------------------------------------------------------------------

drop table if exists Optimizer;
-- TaskQueueID integer not null auto_increment,
CREATE TABLE  Optimizer(
    OptimId  integer not null auto_increment primary key,
    name     varchar(100),
    Priority integer
);
```

```
-------------------------------------------------------------------------------
drop table if exists OptToQueue;
CREATE TABLE OptToQueue (
    OptimId     integer not null,
    TaskQueueId integer not null,
    PRIMARY KEY (OptimId, TaskQueueId),
    FOREIGN KEY(OptimId) REFERENCES Optimizer(OptimId),
    FOREIGN KEY(TaskQueueId) REFERENCES TaskQueues(TaskQueueId)
);

-------------------------------------------------------------------------------
drop table if exists TaskQueue;
CREATE TABLE TaskQueue (
    TaskQueueId integer not null,
    JobID       integer not null,
    Rank        integer,
    PRIMARY KEY (JobID, TaskQueueId),
    FOREIGN KEY(JobID) REFERENCES Job(JobID),
    FOREIGN KEY(TaskQueueId) REFERENCES TaskQueues(TaskQueueId)
);

-------------------------------------------------------------------------------
drop table if exists JobParameters;
CREATE TABLE JobParameters (
    JobID integer not null,
    Name varchar(100) not null,
    Value BLOB  not null,
    FOREIGN KEY(JobID) REFERENCES Job(JobID)
);
ALTER  TABLE JobParameters ADD PRIMARY KEY(JobID, Name);

-------------------------------------------------------------------------------
drop table if exists LogInfos;
CREATE TABLE LogInfos (
    JobID integer not null,
    Event varchar(100) not null,
    Date Date,
    Time time,
    FOREIGN KEY(JobID) REFERENCES Job(JobID)
);

-------------------------------------------------------------------------------
DROP TABLE IF EXISTS InformationService;

CREATE TABLE InformationService (
    Profile varchar(100),
    Item    varchar(100) NOT NULL,
    Value   varchar(100) NOT NULL,
    PRIMARY KEY (Profile, Item)
```

```
);

-------------------------------------------------------------------------

DROP TABLE IF EXISTS ISandbox;
CREATE TABLE ISandbox (
    JobID  integer not null,
    File   varchar(100) NOT NULL,
    Value  LONGBLOB NOT NULL,
    Type  enum  ('ascii', 'binary') not null,
    PRIMARY KEY (JobID, File)
);
-------------------------------------------------------------------------

DROP TABLE IF EXISTS OSandbox;
CREATE TABLE OSandbox (
    JobID  integer not null,
    File   varchar(100) NOT NULL,
    Value  LONGBLOB NOT NULL,
    Type  enum  ('ascii', 'binary') not null,
    PRIMARY KEY (JobID, File)
);
-------------------------------------------------------------------------

DROP TABLE IF EXISTS SandboxReady;
CREATE TABLE SandboxReady (
    JobID  integer not null,
    RetrievedOutput enum ('True', 'False') not null,
    OutputDate Date,
    OutputTime time,
    PRIMARY KEY (JobID)
);

-------------------------------------------------------------------------
DROP TABLE IF EXISTS CommonMask;
CREATE TABLE CommonMask (
    MaskId integer not null auto_increment primary key,
    Mask BLOB,
    Date Date,
    Time time
);
INSERT INTO  CommonMask (Mask, Date, Time) VALUES
('[Requirements = true;]',CURDATE(),CURTIME());
-------------------------------------------------------------------------
```

Annexe I

Génération de nombres pseudo-aléatoires

Nous décrivons succinctement les principes pour la génération de nombres pseudo-aléatoires selon des lois de probabilité. Nous l'illustrons pour la loi normale, Weibull et exponentielle.

I.1 Principes de la simulation

Une suite de nombres pseudo-aléatoires est une suite constructible dont les propriétés statistiques sont proches de la suite $(U_1, U_2, ...)$ où les U_i sont des variables aléatoires (v.a) indépendantes et de loi uniforme sur $[0, 1]$. Pour calculer des quantités qui peuvent s'exprimer sous forme d'espérance, c'est-à-dire de la forme $E[X]$, nous générons un grand nombre $(X_1, ..., X_n)$ de tirages de X indépendants. Nous utilisons pour ceci la loi forte des grands nombres :

$$E[X] = \lim_{n \to +\infty} \frac{1}{n} \sum_{i=1}^{n} X_i$$

Pour simuler des v.a. de loi donnée, nous disposons principalement de deux moyens :

L'inversion de la fonction de répartition. Soit la fonction de répartition de la loi F et U une v.a. uniforme sur [0,1]. La loi de $X = F^{-1}(U)$ a comme fonction de répartition F. F^{-1} est ici l'inverse droite à droite de F soit :

$$F^{-1}(\alpha) = inf\{x, F(x) \leq \alpha\}.$$

Nous utilisons cette méthode pour les v.a à valeurs discrètes comme la loi exponentielle, la loi de Weibull, etc.

La méthode de rejet. Si la fonction de répartition ne s'inverse pas mais que nous connaissons la densité f, nous utilisons alors la méthode du rejet : soit g une densité pour laquelle nous savons simuler facilement des v.a. et telle que $f(x) \leq ag(x)$ pour un coefficient a donné. Le principe est défini dans l'algorithme 1.

I.2 La loi exponentielle

X suit une loi exponentielle de paramètre $\lambda > 0$ lorsque la densité de sa loi est donnée par :

$$\lambda \; exp(-\lambda x) \; pour x \geq 0.$$

Algorithme 1 Méthode du rejet.

Soit g une densité telle que $f(x) \leq ag(x)$ pour un coefficient donné
u initialisé, a donné
Tant que $u \leq f(x)/ag(x)$ **faire**
  $x \leftarrow$ Générer un tirage de loi de densité g
  $u \leftarrow$ Générer un tirage d'une loi uniforme sur [0,1]
Fin Tant que
x est le résultat

Sa moyenne $E[x]$ et sa variance $var(X)$ sont respectivement $1/\lambda$ et $1/\lambda^2$. Nous la générons de la manière suivante :

$$X = -\frac{1}{\lambda}\ logU_1$$

I.3 La loi normale

Soient U_1, U_2 des v.a. indépendantes de loi uniforme sur $[0, 1]$. X suit une loi normale de paramètres μ et σ^2 si la densité de sa loi est donnée par :

$$\frac{1}{\sqrt{\sigma 2\pi}} exp(-\frac{-(x-\mu)^2}{2\sigma^2})$$

Nous avons $E[x] = \mu$ et $var(X) = \sigma^2$. Pour générer des tirages selon une loi normale 0 et 1, nous considérons :

$$X = cos(2\pi U_1)\sqrt{-2\ logU_2}$$

I.4 La loi de Weibull

La loi de Weibull modélise des durées de vie. X suit une loi de Weibull de paramètre $\lambda > 0$ et $c > 0$ si la densité est donnée par :

$$\lambda c(\lambda x)^{c-1} exp(-(\lambda x)^c)\ pour\ x \geq 0.$$

Nous avons :

$$E[X] = \frac{\Gamma(1+\frac{1}{c})}{\lambda}$$

$$var(X) = \frac{\Gamma(1+2/c) - \Gamma(1+1/c)^2}{\lambda^2}$$

Pour générer une telle v.a., nous utilisons :

$$X = \frac{(-log(1-U_1))^{\frac{1}{c}}}{\lambda}$$

Annexe J

Liste des notations et des symboles du chapitre 4 et 5

Nous rappelons les notations et symboles utilisés dans le chapitre 5 et 6.

Notations	**Valeurs**
$card(\mathcal{E})$	Fonction donnant le cardinal d'un ensemble $\mathcal{E}$
$\mathcal{C}$	Ensemble des grappes des nœuds ou sites
$\mathcal{C}_i$	Une grappe de nœuds ou site
$bwt_{\mathcal{C}_i}$	Bande passante locale d'un site $\mathcal{C}_i$
$latence_{\mathcal{C}_i}$	Latence locale d'un site $\mathcal{C}_i$
$bwt_{\mathcal{C}}$	Bande passante globale entre les sites $\mathcal{C}$
$latence_{\mathcal{C}}$	Latence globale entre les sites $\mathcal{C}$
$\mathcal{N}_i$	Ensemble des nœuds de calcul de la plate-forme
(i,j)	Le $j^{ième}$ nœud d'une grappe $\mathcal{C}_i$
$puissance_{i,j}$	Puissance de calcul du processeur du nœud (i,j), exprimée en NCU par unité de temps
NCU	Normalized Computing Unit ou une unité de calcul élémentaire
$puissance_m$	Puissance moyenne de la plate-forme
k	Tâche locale attribuée à un site
$attributs_k$	Ensemble des attributs d'une tâche k
tl_k	Date de soumission locale k
$taille_k$	Taille de k exprimée en NCU
$proc_k$	le nombre de processeurs requis par k(tâche rigide simple, i.d. $Card(proc_k) = 1$)
$group_k$	Organisation propriétaire de la tâche k
mk	Méta-tâche attribuée à un système de méta-computing global
$meta\text{-}attributs_k$	Méta-attributs de mk associé avec une tâche locale k et ses attributs $attributs_k$
t_k	Date de soumission de la tâche mk au système de méta-computing global
r_k, q_k, d_k	La date de début d'exécution, la date d'état en attente et la date de terminaison d'une tâche k sur une site
$\mathcal{T}$	Ensemble des tâches soumises à une grappe C_i
$\mathcal{MT}$	Ensemble des méta-tâches soumises au système de méta-computing
$S(\mathcal{T})$	Fonction de distribution qui génère l'ensemble des tailles d'un ensemble de tâches $\mathcal{T}$
Suite page suivante...	

Notations	**Valeurs**
CA	Coupure appliquée fixant la taille minimale et maximale des tâches.
$A(\mathcal{T})$	Fonction de distribution qui génère l'ensemble des temps de soumission d'un ensemble de tâches $\mathcal{T}$
$\lambda_{\mathcal{T}}$	Taux de soumission moyen de $\mathcal{T}$
$\mathcal{F}_i$	Ensemble de files d'attente d'un site $\mathcal{C}_i$
$f_{i,l}$	La $l^{ième}$ file d'un site $\mathcal{C}_i$
$\mathbf{N_{f_{i,l}}}$	Ensemble de nœuds associé à $f_{i,l}$
$\mathcal{T}_{i,l}$	Ensemble de tâches $\mathcal{T}_{i,l}$ en attente d'exécution
$profondeur_{i,l}$	Profondeur ou nombre de tâches en attente dans la file à un instant t
$t_{max_{i,l}}$	Temps maximal d'une tâche en exécution sur un nœud de la file d'attente $f_{i,l}$
j, l, i	Le $j^{ième}$ nœud de la file d'attente l d'un site j
$\mathcal{SES}$	Ensemble des éléments de stockage
$\mathcal{F}$	Ensemble des fichiers répartis
lfn	Nom logique d'un fichier
pfn	Nom physique d'un fichier

Annexe K

Description détaillée des algorithmes présentés dans le manuscrit

Nous détaillons ici les algorithmes cités dans le chapitre 6 en pseudo-code. Nous reprenons la notation introduite dans le chapitre 5 et résumée dans le tableau de l'annexe J.

K.1 Algorithme centralisé « Join Shortest Queue » (JSQ)

Algorithme 2

Soit m_k une méta-tâche et r_k l'expression de ces contraintes et besoins
Soit r_i une ressource de calcul qui peut-être un nœud, un site ou un système de grille.
Soit $match_k = \{r_i, r_j, \ldots, r_p\}$ l'ensemble des ressources qui vérifient la description r_k méta-tâche mk
Tant que vrai **faire**
  Reçoit une méta-tâche mk
  Interroge le service correspondance
  Sélectionne l'ensemble des ressources correspondants à la description r_k de mk
  Sélectionne l'ensemble $match_k$
  Selectionne dans $match_k$ la file d'attent $f_{i,l}$ avec le service information tel que $f_{i,l} = \min_{j \in \mathcal{F}}(profondeur_j)$
  Soumet mk à $f_{i,l}$
Fin Tant que

K.2 Algorithme de l'agent *DIRAC* associé à une file d'attente d'un système de traitement par lots

Algorithme 3

Soit un site $\mathcal{C}_i$
Soit $f_{i,l}$ une file d'attente d'un site
Soit l'ensemble des nœuds associés à la file $\mathcal{N}_{f_{i,l}}$
Soit σ la période de surveillance de la ressource en seconde
Soit ε *tel que* $\varepsilon \in R_*^+$ *et* $\varepsilon \in]0,1[$
Tant que vrai **faire**
  {L'agent vérifie l'utilisation de la ressource $f_{i,l}$ avec un intervalle de temps σ}
  Tant que $\frac{profondeur_{i,l}}{card(\mathcal{N}_{f_{i,l}})} < \varepsilon$ **faire**
    {Test de disponibilité}
    Génére la description de la ressource $DR_{f_{i,l}}$
    Demande une méta-tâche mk au service correspondance $DR_{f_{i,l}}$
    Reçoit une réponse R du service correspondance
    Si R est une méta-tâche **alors**
      $mk \leftarrow R$
      Soumettre mk à la ressource $f_{i,l}$
    Fin Si
  Fin Tant que
  Attendre σ t
Fin Tant que

K.3 Algorithme du service correspondance « Fit Resource First Served » (FRFS) avec une seule file d'attente globale

Algorithme 4

Soit m_k une méta-tâche et r_k l'expression de ces contraintes et besoins
Soit $gq = \{mt_1, mt_3, \ldots, mt_p\}$ la file d'attente globale
{ Chaque mt_k dans gq possède un rang $rang_k$}
Soit r une ressource de calcul qui peut-être un nœud, un site ou un système de grille.
Soit $match_r = \{mt_i, mt_j, \ldots, mt_p\}$ l'ensemble de méta-tâches qui vérifient la description des besoins DR_r d'une ressource
Tant que vrai **faire**
  Reçoit demande de tâche sous forme d'une description de ressource (DR_r)
  Cherche une méta-tâche m_k tel que $rang_k = \min_{m \in match_r}(rang_m)$
  Si Pas de méta-tâche trouvée **alors**
    $Réponse(R) \leftarrow$ "Pas de tâche disponible"
  Sinon
    $Réponse(R) \leftarrow m_k$
  Fin Si
  Envoi de R à l'agent associé à la ressource r
Fin Tant que

K.4 Algorithme pour un agent *maître DIRAC* déployé sur la passerelle d'un site

Algorithme 5

Soit un site $\mathcal{C}_i$
Soit $f_{i,l}$ une file d'attente d'un site
Soit l'ensemble des nœuds associés à la file $\mathcal{N}_{f_{i,l}}$
Soit σ la période de surveillance de la ressource en seconde
Soit ε *tel que* $\varepsilon \in R_*^+$ *et* $\varepsilon \in]0,1[$
Tant que vrai **faire**
  {L'agent vérifie l'utilisation de la ressource $f_{i,l}$ avec un intervalle de temps σ}
  Tant que $\frac{profondeur_{i,l}}{card(\mathcal{N}_{f_{i,l}})} < \varepsilon$ et méta-tâche disponible **faire**
    {Test de disponibilité}
    Soumettre un agent *pilote* à la ressource $f_{i,l}$
  Fin Tant que
  Attendre σ t
Fin Tant que

K.5 Algorithme de l'agent *pilote* mode « run once »

Algorithme 6

Soit un site $\mathcal{C}_i$
Soit $f_{i,l}$ une file d'attente d'un site
Soit l'ensemble des nœuds associés à la file $\mathcal{N}_{f_{i,l}}$
Soit i, l, j un nœud de travail de la file et du site
Soit $puissance_{i,l,j}$ la puissance du nœud
Soit $t_{max_{i,l,j}}$ la limite temporelle maximale sur le nœud
Génére la description de la nœud $DR_{i,l,j}$
Demande une méta-tâche mk au service correspondance avec $DR_{i,l,j}$
Reçoit une réponse R à la requête au service correspondance
Si R est une méta-tâche **alors**
  $mk \leftarrow R$
  Soumettre mk au nœud i, l, j
  Attend la fin de l'exécution
  Arrêt
Sinon
  Arrêt
Fin Si

K.6 Algorithme de l'agent *pilote* mode « filling »

Algorithme 7

Soit un site $\mathcal{C}_i$
Soit $f_{i,l}$ une file d'attente d'un site
Soit l'ensemble des nœuds associés à la file $\mathcal{N}_{f_{i,l}}$
Soit i, l, j un nœud de travail de la file et du site
Soit $puissance_{i,l,j}$ la puissance du nœud
Soit $t_{max_{i,l,j}}$ la limite temporelle maximale sur le nœud
$t \leftarrow t_0$ {Initialise le compteur de consommation de temps}
Tant que *temps courant* $< t_{max_{i,l}}$ **faire**
  Génére la description de la nœud $DR_{i,l,j}$
  Demande une méta-tâche mk au service correspondance avec $DR_{i,l,j}$
  $\{\frac{taille_{i,l,j}}{puissance_{i,l,j}} < (t_{max} - (t - t_0))\}$
  Reçoit une réponse R à la requête au service correspondance
  Si R est une méta-tâche **alors**
    $mk \leftarrow R$
    Soumettre mk au nœud i, l, j
    Attendre la fin de l'exécution mk
  Sinon
    Arrêt
  Fin Si
  $t \leftarrow$ *temps courant* {Mise à jour du compteur de temps}
  {or $t \leftarrow t + \frac{taille_k}{puissance_{i,l,j}}$ }
Fin Tant que

K.7 Algorithme pour le service correspondance appliquant la statégie « Fit Resource First Served » (FRFS) avec priorité et plusieurs files d'attente globales

Algorithme 8

Soit m_k une méta-tâche et r_k l'expression de ces contraintes et besoins
Soit $\mathcal{GQ} = \{gq_1, gq_3, \ldots, gq_p\}$ l'ensemble des files d'attente globales
Soit $gq_l = \{mt_1, mt_3, \ldots, mt_p\}$ une file d'attente globale
{ Chaque mt_k dans gq_l possède un rang $rang_k$}
Soit $priorité_l$ la priorité associée à la file d'attente gq_l dans l'ensemble $\mathcal{GQ}$
Soit $requirement_l$ l'expression de ces contraintes et besoins des méta-tâches dans gq_l
Let $matchqueue_r$ the set of global queues which match well a Resource Requirement RR_r
Tant que vrai **faire**
  Reçoit demande de tâche sous forme d'une description de ressource (DR_r)
  Trouve une file d'attente global non vide tel que $priorité_l = \max_{m \in matchqueue_r}(piorité_m)$
  Trouve une méta-tâche m_k tel que le $rang_k = \min_{m \in gq_l}(rang_m)$
  Si pas de méta-tâche trouvée **alors**
    $Réponse(R) \leftarrow$ "Pas de tâche disponible"
  Sinon
    $Réponse(R) \leftarrow m_k$
  Fin Si
  Envoi de R à l'agent associé à la ressource r
Fin Tant que

K.8 Algorithme d'un agent *pilote DIRAC* sur un nœud mode « préemptif »

Algorithme 9

Soit un site $\mathcal{C}_i$
Soit $f_{i,l}$ une file d'attente d'un site
Soit l'ensemble des nœuds associés à la file $\mathcal{N}_{f_{i,l}}$
Soit i, l, j un nœud de travail de la file et du site
Soit $puissance_{i,l,j}$ la puissance du nœud
Soit $t_{max_{i,l,j}}$ la limite temporelle maximale sur le nœud
Soit $charge_{max_{i,l,j}}$ la charge maximale du nœud i, l, j, c-à-d le nombre maximum de tâche pouvant s'exécuter en concurrence
Soit *pile* la pile de tâches préemptées
Soit $pile_{max}$ la taille maximum de la pile de tâche
Soit τ l'intervalle de temps entre deux requêtes de tâche
$t \leftarrow t_0$ {Initialise le compteur de temps à 0}
Soit $tâche_{courantes}$ l'ensemble des tâches en exécution sur le nœud
$tâche_{courantes} \leftarrow \{\emptyset\}$ {Initialise l'ensemble des tâche à l'ensemble vide}
Tant que *current time* $< t_{max}$ **faire**
  Si *taille de la pile* $< pile_{max}$ **alors**
    Demande une méta-tâche mk au service correspondance avec $DR_{i,l,j}$
    Reçoit une réponse R à la requête au service correspondance
    Si R est une méta-tâche **alors**
      $mk \leftarrow R$
      Si *charge courante* $> charge_{max_{i,l,j}}$ **alors**
        Préemption de la tâche k tel que $priorité_k = \min_{m \in tâche_{courante}}(priorité_m)$ et $état(k) = running$
        empile $(k, pile)$
      Fin Si
      $t \leftarrow t + \frac{taille_k}{puissance_{i,l,j}}$
      $tâche_{courante} \leftarrow tâche_{courante} \cup \{mk\}$
      Soumettre mk au nœud i, l, j
    Sinon
      Si $card(tâche_{courante}) = 0$ **alors**
        Stop
      Fin Si
    Fin Si
  Fin Si
  Tant que ($\exists$ une tâche $k / k \in tâche_{courante}$ and $état(k) = done$) **faire**
    $task_{current} \leftarrow task_{current} \backslash \{k\}$
    Si *pile* non vide **alors**
      tâche $\leftarrow$ dépile (*pile*)
      Réveil et reprise de tâche *tâche*
    Fin Si
  Fin Tant que
  Attendre τ s
  Si $t <$ *temps courant* **alors**
    $t \leftarrow$ *temps courant*
  Fin Si
Fin Tant que

BIBLIOGRAPHIE

[14] I. Foster et C. Kesselman. Globus : A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2) :115–128, 1997.

[15] Michalis Faloutsos, Petros Faloutsos, et Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[16] Emir Halepovic et Ralph Deters. Building a p2p forum system with jxta. In *P2P '02 : Proceedings of the Second International Conference on Peer-to-Peer Computing*, page 41, Washington, DC, USA, 2002. IEEE Computer Society.

[17] Ian Foster, Jonathan Geisler, Carl Kesselman, et Steven Tuecke. Managing multiple communication methods in high-performance networked computing systems. *Journal of Parallel and Distributed Computing*, 40(1) :35–48, 1997.

[18] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, et S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, pages 365–375. IEEE Computer Society Press, 1997.

[19] Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, et Steven Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Comput.*, 28(5) :749–771, 2002.

[20] Ian Foster, Carl Kesselman, Gene Tsudik, et Steven Tuecke. A security architecture for computational grids. In *CCS '98 : Proceedings of the 5th ACM conference on Computer and communications security*, pages 83–92, New York, NY, USA, 1998. ACM Press.

[21] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, et Steven Tuecke. A resource management architecture for metacomputing systems. *Lecture Notes in Computer Science*, 1459 :62–??, 1998.

[22] Bill Allcock, Lee Liming, et Steven Tuecke. GridFTP : A Data Transfer Protocol for the Grid. `http://www.gridforum.org/meetings/ggf6/ggf6_wg_papers/GridFTP.doc`.

[23] Henri Casanova et Jack Dongarra. NetSolve : a network server for solving computational science problems. In *Supercomputing '96 : Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, page 40, 1996.

[24] Hidemoto Nakada, Mitsuhisa Sato, et Satoshi Sekiguchi. Design and implementations of ninf : towards a global computing infrastructure. *Future Gener. Comput. Syst.*, 15(5-6) :649–658, 1999.

[25] Ian Foster et Carl Kesselman, editors. *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.

[26] Ian Foster et Carl Kesselman, editors. *The Grid 2 : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 2004.

[27] N. Nisan, S. London, O. Regev, et N. Camiel. Globally distributed computation over the internet - the popcorn project. In *ICDCS '98 : Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 592, Washington, DC, USA, 1998. IEEE Computer Society.

[28] Bernd O. Christiansen, Peter R. Cappello, Mihai F. Ionescu, Michael O. Neary, Klaus E. Schauser, et Daniel Wu. Javelin : Internet-based parallel computing using java. *Concurrency - Practice and Experience*, 9(11) :1139–1160, 1997.

[29] Tim Brecht, Harjinder Sandhu, Meijuan Shan, et Jimmy Talbot. Paraweb : towards world-wide supercomputing. In *EW 7 : Proceedings of the 7th workshop on ACM SIGOPS European workshop*, pages 181–188, New York, NY, USA, 1996. ACM Press.

[30] Klaus E. Schauser, Chris J. Scheiman, Gyung-Leen Park, Behrooz Shirazi, et Jeff Marquis. Superweb : Towards a global web-based parallel computing infrastructure. In *11th International Parallel Processing Symposium (IPPS '97), 1-5 April 1997, Geneva.*, pages 100–106, 1997.

[31] Hernâni Pedroso, Luis M. Silva, et João Gabriel Silva. Web-based metacomputing with JET. *Concurrency : Practice and Experience*, 9(11) :1169–1173, 1997.

[32] A. Baratloo, M. Karaul, Z. M. Kedem, et P. Wyckoff. Charlotte : Metacomputing on the web. In *Proc. of the 9th Int'l Conf. on Parallel and Distributed Computing Systems (PDCS-96)*, 1996.

[33] G. Pape. runit Service Supervision Toolkit. `http://smarden.org/runit/`.

[34] C. Percival. PiHex : A distributed effort to calculate Pi. `http://www.cecm.sfu.ca/-projects/pihex/index.html`.

[35] Distributed.net. `http://http://www.distributed.net/`.

[36] Globus et IBM. `http://www.globus.org/wsrf/#announcement`, January 2004.

[37] XML-RPC. `http://www.xmlrpc.com/`.

[38] OSG Project . The Open Science Grid OSG. `http://www.opensciencegrid.org/`.

[39] LCG Project. The LHC computing grid project - LCG. `http://lcg.web.cern.ch/LCG/`.

[40] Gilles Fedak. *XtremWeb : une plate-forme générique pour l'étude expérimentale du Calcul Global et Pair-a-Pair*. PhD thesis, Laboratoire de Recherche Informatique - Université Paris XI, 2003.

[41] Yves Caniou. *Ordonnancement sur une plate-forme de métacomputing*. PhD thesis, Thèse d'université, Université Henri Poincaré, 2004.

[42] R. Buyya. *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, Monash University, Melbourne, Australia, 2002.

[43] H. Casanova. Simgrid : A toolkit for the simulation of application scheduling. In *Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid2001)*. IEEE Computer Society, May 2001.

[44] A. Legrand et J. Lerouge. Metasimgrid : Toward realistic scheduling simulation of distributed applications. Technical Report 2002-28, Laboratoire de l'Informatique du Parallélisme (LIP), July 2002.

[45] I. A. Howes, M. C. Smith, et G. S. Good. *Understanding and deploying LDAP directory services*. Macmillian Technical Publishing, 1999.

[46] Michael Litzkow, Miron Livny, et Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[47] D. Anderson et al. A new major SETI project based on project serendip data and 100,000 personal computers. In *Proceedings of the 5th Intl. Conf. on Bioastronomy*, June 1997.

[48] The MathWorks, Inc. *MATLAB, High-performance Numeric Computation and Visualization Software : Reference Guide*. MATHWORKS, 1992.

[49] German Cancio, Steve M. Fisher, Tim Folkes, Francesco Giacomini, Wolfgang Hoschek, Dave Kelsey, et Brian L. Tierney. The DataGrid Architecture Version 2. In *EDMS 439938*. CERN, Feb 2004.

[50] Ian Foster. The Anatomy of the Grid : Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[51] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov, et Francine Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Heterogeneous Computing Workshop*, pages 349–363, 2000.

[52] Guido Gehlen et Linh Pham. Mobile web services for peer-to-peer applications. In *Proceedings of the Consumer Communications and Networking Conference 2005*, page 7, Las Vegas, USA, Jan 2005.

[53] IETF. Extensible Messaging and Presence Protocol. `http://www.ietf.org/html.charters/xmpp-charter.html/`.

[54] sqlite. `http://www.sqlite.org/`.

[55] I Foster, C Kesselman, J Nick, et S Tuecke. The physiology of the grid : An open grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure WG*. Global Grid Forum, 22 June 2002.

[56] P. Saiz et al. Alien - alice environment on the grid. *Nucl. Instrum. Meth.*, A502 :437–440, 2003.

[57] P. Krishna Gummadi, Stefan Saroiu, et Steven D. Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. *SIGCOMM Comput. Commun. Rev.*, 32(1) :82–82, 2002.

[58] Ian T. Foster et Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*, pages 118–128, 2003.

[59] Ian Foster, Carl Kesselman, Jeffrey M. Nick, et Steven Tuecke. The Physiology of the Grid : An Open Grid Services Architecture for Distributed Systems Integration. http ://www.globus.org/research/papers/ogsa.pdf, January 2002.

[60] Nathaniel S. Good et Aaron Krekelberg. Usability and privacy : a study of kazaa p2p file-sharing. In *CHI '03 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 137–144, New York, NY, USA, 2003. ACM Press.

[61] Ian Clarke, Oskar Sandberg, Brandon Wiley, et Theodore W. Hong. Freenet : A distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies : Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.

[62] J. Dongarra, H. Meuer, et E. Strohmaier. Top500 supercomputer sites. `http://www.top500.org/`.

[63] P. Eerola et al. The NorduGrid Architecture and Tools. In *Proceedings of Computing for High Energy Physics 2003*, 2003.

[64] E. Laure. EGEE Middleware Architecture. Technical report, June 2004.

[65] LHC. Architectural Roadmap Towards Distributed Analysis - Final Report. Technical report, CERN, November 2003.

[66] Ian T. Foster et all. Paul Sheldon. The grid2003 production grid : Principles and practice. In *13th International Symposium on High-Performance Distributed Computing (HPDC-13 2004), 4-6 June 2004, Honolulu, Hawaii, USA*, pages 236–245, 2004.

[67] Particle physics data grid. `http://www.ppdg.org/`.

[68] The grid2003 project. `http://www.ivdgl.org/grid2003/`.

[69] P. Avery et I. Foster. Towards petascale virtual data grids (griphyn project). `http://www.griphyn.org/`.

[70] The virtual data toolkit(vdt). `http://www.lsc-group.phys.uwm.edu/vdt/`.

[71] Dieter Kranzlmuller. EGEE Project - A Multidisciplinary, Production-Level Grid. *Lecture Notes in Computer Science*, 3726, October 2005.

[72] Larry Smarr et Charles E. Catlett. Metacomputing. *Commun. ACM*, 35(6) :44–52, 1992.

[73] Douglas Thain et Miron Livny. Building reliable clients and servers. In Ian Foster et Carl Kesselman, editors, *The Grid : Blueprint for a New Computing Infrastructure(Second Edition)*. Morgan Kaufmann, 2004.

[74] Jon Crowcroft, Tim Moreton, Ian Pratt, et Andrew Twigg. *The Grid : Blueprint for a New Computing Infrastructure (Second Edition)*, chapter Peer-to-peer technologies. Morgan Kaufmann, 2004.

[75] Andrew A. Chien. *The Grid : Blueprint for a New Computing Infrastructure (Second Edition)*, chapter "Computing Elements". Morgan Kaufmann, 2004.

[76] I. Foster, J. Geisler, B. Nickless, W. Smith, et S. Tuecke. Software infrastructure for the i-way high-performance distributed computing experiment. In *HPDC '96 : Proceedings of the High Performance Distributed Computing (HPDC '96)*, page 562. IEEE Computer Society, 1996.

[77] Avery, P. Foster, I. Newman, H. Szalay, et A. An international Virtual-Data Grid Laboratory for Data Intensive Science. Technical report, Technical Report : GriphyN-2001-2, January 2001.

[78] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8) :114–117, 1965.

[79] Lawrence G. Roberts. Beyond moore's law : Internet growth trends. *Computer*, 33(1) :117–119, 2000.

[80] David P. Anderson. Boinc : A system for public-resource computing and storage. In *5th International Workshop on Grid Computing (GRID 2004), 8 November 2004, Pittsburgh, PA, USA, Proceedings*, pages 4–10, 2004.

[81] Efstratios Gallopoulos, Elias Houstis, et John R. Rice. Computer as thinker/doer : Problem-solving environments for computational science. *IEEE Comput. Sci. Eng.*, 1(2) :11–23, 1994.

[82] James Frey, Todd Tannenbaum, Miron Livny, Ian T. Foster, et Steven Tuecke. Condor-g : A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3) :237–246, 2002.

[83] Balázs Kónya. Advanced resource connector (arc) - the grid middleware of the nordugrid. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 19-22, 2004, Proceedings*, page 10, 2004.

[84] R. Alfieri, Roberto Cecchini, Vincenzo Ciaschini, Luca dell'Agnello, Ákos Frohner, A. Gianoli, Károly Lörentey, et Fabio Spataro. Voms, an authorization system for virtual organizations. In *Grid Computing, First European Across Grids Conference, Santiago de Compostela, Spain, February 13-14, 2003, Revised Papers*, pages 33–40, 2003.

[85] A. Chien, B. Calder, S. Elbert, et K. Bhatia. Entropia : Architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5) :597–610, 2003.

[86] Jeff Linderoth, Sanjeev Kulkarni, Jean-Pierre Goux, et Michael Yoder. An enabling framework for master applications on the computational grid. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed computing (HPDC9), Pittsburgh, Pennsylvania.*, pages 43–50, August 2000.

[87] Neil Gunton. Soap : simplifying distributed development. *Dr. Dobb's J.*, 26(9) :89–ff, 2001.

[88] Andrew Hanushevsky et Heinz Stockinger. A proxy service for the xrootd data server. In *Scientific Applications of Grid Computing, First International Workshop, SAG 2004, Beijing, China, September 20-24, 2004, Revised Selected and Invited Papers*, pages 38–49, 2004.

[89] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, et Hari Balakrishnan. Chord : A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[90] Miron Livny, Jim Basney, Rajesh Raman, et Todd Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1), June 1997.

[91] Universal Description, Discovery and Integration. http ://www.uddi.org/.

[92] Byrom et Others. R-GMA : A Relational Grid Information and Monitoring System. In *"Proceedings of of Cracow 2002 Grid Workshop"*, Dec 2002.

[93] Pasta - The LHC Technology Tracking Team for Processors, Memory, Architectures, Storage and Tapes Brief. http ://lcg.web.cern.ch/LCG/PEB/PASTAIII/pasta2002Report.htm.

[94] Matthew L. Massie, Brent N. Chun, et David E. Culler. The ganglia distributed monitoring system : design, implementation, and experience. *Parallel Computing*, 30(5-6) :817–840, 2004.

[95] Harvey B. Newman, I. C. Legrand, Philippe Galvez, R. Voicu, et C. Cirstoiu. Monalisa : A distributed monitoring service architecture. *CoRR*, cs.DC/0306096, 2003.

[96] Pierre-François Dutot, Lionel Eyraud, Grégory Mounié, et Denis Trystram. Bi-criteria algorithm for scheduling jobs on cluster platforms. In *SPAA '04 :*

Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures, pages 125–132, New York, NY, USA, 2004. ACM Press.

[97] Klaus Krauter, Rajkumar Buyya, et Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *International Journal of Software : Practice and Experience (SPE)*, (2), 2002.

[98] Y. Wang et R. Morris. Load sharing in distributed systems. *IEEE Trans. Comput. C-34*, (3) :204–217, 1985.

[99] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, et A. Chien. The microgrid : a scientific tool for modeling computational gridsr. In *Supercomputing '00 : Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 53, Washington, DC, USA, 2000. IEEE Computer Society.

[100] Atsuko Takefusa, Satoshi Matsuoka, Kento Aida, Hidemoto Nakada, et Umpei Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *HPDC '99 : Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing*, page 11, Washington, DC, USA, 1999. IEEE Computer Society.

[101] David G. Cameron, A. Paul Millar, Caitriana Nicholson, Rubén Carvajal-Schiaffino, Kurt Stockinger, et Floriano Zini. Analysis of scheduling and replica optimisation strategies for data grids using optorsim. *J. Grid Comput.*, 2(1) :57–69, 2004.

[102] W. M. Jones, L. Pang, et D. Stanzione. Technical report, Computational Mini-Grid Research at Clemson University. Technical report, Parallel Architecture Research Lab, Clemson University, November 2002.

[103] The Chicago Grid Simulator. http ://people.cs.uchicago.edu/ krangana/-ChicSim.html.

[104] Wan in Lab. http ://netlab.caltech.edu/.

[105] Grid eXplorer. http ://www.lri.fr/ fci/GdX/.

[106] Luigi Rizzo. Dummynet : a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1) :31–41, 1997.

[107] Mohamed A. Kerasha et Ian Greenshields. Huskysim : a simulation toolkit for application scheduling in computational grids. In *WWW Alt. '04 : Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 380–381, New York, NY, USA, 2004. ACM Press.

[108] Rajkumar Buyya et M. Manzur Murshed. Gridsim : a toolkit for the modeling and simulation of distributed resource management and scheduling for grid

computing. *Concurrency and Computation : Practice and Experience*, 14(13-15) :1175–1220, 2002.

[109] Thilo Kielmann, Henri E. Bal, Jason Maassen, Rob van Nieuwpoort, Lionel Eyraud, Rutger Hofman, et Kees Verstoep. Programming environments for high-performance grid computing : the albatross project. *Future Gener. Comput. Syst.*, 18(8) :1113–1125, 2002.

[110] P.A. Crosby. *Measurement and Simulation of the Performance of High Energy Physics Data Grids.* PhD thesis, University College London, 2003.

[111] Ramin Yahyapour. *Design and Evaluation of Job Scheduling Strategies for Grid Computing.* PhD thesis, Universität Dortmund, Germany, 2002.

[112] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, et Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[113] Fred Howell et Ross McNab. Simjava : a discrete event simulation package for java with applications in computer systems modeling. In *First International Conference on Web-based Modeling and Simulation*, San Diego, CA, January 1998. Society for Computer Simulation.

[114] Rajive Bagrodia, Richard Meyer, Mineo Takai, Yu an Chen, Xiang Zeng, Jay Martin, et Ha Yoon Song. Parsec : A parallel simulation environment for complex systems. *IEEE Computer October 1998*, 31(10) :77 – 85, 1998.

[115] J. Davis II, M. Goel, C. Hylands, B. Kienhuis, E.A. Lee, J. Liu, X. Liu, L. Muiadi, S. Neuendorffer, J. Reekie, N. Smyth, et Y. Xiong. Heterogeneous Concurrent Modeling and Design in Java, PtolemyII Design Document. Technical report, EECS, University of California at Berkeley, 1998. http ://ptolemy.eecs.berkeley.edu/publications/papers/98.

[116] L. Kleinrock, editor. *Queueing Systems Volume I : Theory.* John Wiley and Sons, 1975.

[117] L. Kleinrock, editor. *Queueing Systems Volume II : Computer Applications.* John Wiley and Sons, 1976.

[118] Sun Microsystems. *Sun Grid Engine 5.3 Manual Sun :Grid Engine,Enterprise Edition 5.3 annual*, 5.3 edition, July 2001.

[119] Platform Computing Corporation. *Platform LSF Version 5.0 Documentation*, 31 May 2002.

[120] Veridian Information Solutions. *Portable Batch System Administrator Guide*, August 2000.

[121] Rajkumar Buyya. *High Performance Cluster Computing : Architectures and Systems.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[122] Srividya Srinivasan, Rajkumar Kettimuthu, Vijay Subramani, et P. Sadayappan. Selective Reservation Strategies for Backfill Job Scheduling. In *Proceedings of 8th Workshop on Job Scheduling Strategies for Parallel Processing*, July 2002.

[123] T. L. Casavant et J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2) :141–154, 1988.

[124] H.G. Rotithor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. In *Proceedings of Computer Digital Technology*, volume 141, pages 1–10, January 1994.

[125] Nirav H. Kapadia et Jose A. B. Fortes. On the design of a demand-based network-computing system : The purdue university network-computing hub. In *HPDC*, pages 71–80, 1998.

[126] Rich Wolski, Neil T. Spring, et Jim Hayes. The network weather service : a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6) :757–768, 1999.

[127] Fabio Kon, Roy H. Campbell, M. Dennis Mickunas, Klara Nahrstedt, et Francisco J. Ballesteros. 2K : A Distributed Operating System for Heterogeneous Environments. Technical Report UIUCDCS-R-99-2132, Department of Computer Science, University of Illinois at Urbana-Champaign, December 1999.

[128] Derek L. Eager, Edward D. Lazowska, et John Zahorjan. A comparison of receiver-initiated and sender-initiated adaptive load sharing. *Perform. Eval.*, 6(1) :53–68, 1986.

[129] Paul Stelling, Cheryl DeMatteis, Ian T. Foster, Carl Kesselman, Craig A. Lee, et Gregor von Laszewski. A fault detection service for wide area distributed computations. *Cluster Computing*, 2(2) :117–128, 1999.

[130] PACMAN site, saul youssef. `http://physics.bu.edu/~youssef/pacman/index.html`.

[131] Christian Arnault, Bruno Mansoux, et Antoine Pérus. Experiencing CMT in software production of large and complex projects. In *Proceedings of Computing for High Energy Physics 2001*, 2001.

[132] O. Lodygensky, G. Fedak, F. Cappello, V. Neri, M. Livny, et D. Thain. Xtremweb & condor sharing resources between internet connected condor pools. In *CCGRID '03 : Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 382, Washington, DC, USA, 2003. IEEE Computer Society.

[133] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, et R. Neugebauer. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003.

[134] Martin Quinson. *Découverte automatique des caractéristiques et capacités d'une plate-forme de calcul distribué*. PhD thesis, École normale supérieure de Lyon, December 2003.

[135] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, et D. Zagorodnov. Adaptive computing on the grid using apples. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 14(4) :369–382, April 2003.

[136] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau Bni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, et Bin Yao. A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. In *Symposium on Reliable Distributed Systems*, pages 330–335, 1998.

[137] The Globus Project, The Globus Resource Specification Language RSL vl.0. `http://www.globus.org/gram/rsl_spec1.html`.

[138] A. Anjomshoaa, F. Brisard, R. L. Cook, D. K. Fellows, A. Ly, S. McGough, et D. Pulsipher. Job submission description language (jsdl) specification v0.3. Technical report, Global Grid Forum 2004., 2004.

[139] DataTag. The glue schema effort. http ://www.cnaf.infn.it/ sergio/datatag/-glue/.

[140] Fabrizio Pacini. Job Description Language HowTo. Technical report, DataGRID project, December 2001.

[141] Dan Gunter, Brian Tierney, Brian Crowley, Mason Holding, et Jason Lee. Netlogger : A toolkit for distributed system performance analysis. In *MASCOTS '00 : Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Washington, DC, USA, 2000. IEEE Computer Society.

[142] DongWoo Lee, Jack Dongarra, et R. S. Ramakrishna. visperf : Monitoring tool for grid computing. In *International Conference on Computational Science*, pages 233–243, 2003.

[143] M.J. Gonzalez. Deterministic Processor Scheduling. *ACM Computing Surveys*, 9(3) :173–204, 1997.

[144] Gary Shao, Fran Berman, et Rich Wolski. Using effective network views to promote distributed application performance. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.

[145] Dong Lu et Peter A. Dinda. Synthesizing realistic computational grids. In *SC '03 : Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 16, Washington, DC, USA, 2003. IEEE Computer Society.

[146] Waxman B M. Routing of multipoint connections. *IEEE Journal of Selected Areas in Communications*, 6(3) :1617–1622, 1988.

[147] Alberto Medina, Ibrahim Matta, et John Byers. On the origin of power laws in internet topologies. Technical Report 2000-004, 20, 2000.

[148] Jared Winick et Sugih Jamin. Inet-3.0 : Internet topology generator. Technical Report UM-CSE-TR-456-02, EECS, University of Michigan, 2002.

[149] Alberto Medina, Anukool Lakhina, Ibrahim Matta, et John Byers. Brite : An approach to universal topology generation. In *MASCOTS '01 : Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*, page 346, Washington, DC, USA, 2001. IEEE Computer Society.

[150] Christopher R. Palmer et J. Gregory Steffan. Generating network topologies that obey power laws. In *Proceedings of GLOBECOM '2000*, November 2000.

[151] William Aiello, Fan Chung, et Linyuan Lu. A random graph model for massive graphs. In *STOC '00 : Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180, New York, NY, USA, 2000. ACM Press.

[152] Jared Winick et Sugih Jamin. Inet-3.0 : Internet topology generator. Technical Report UM-CSE-TR-456-02, EECS, University of Michigan, 2002.

[153] Kenneth L. Calvert, Matthew B. Doar, et Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6) :160–163, June 1997.

[154] Kenjiro Taura et Andrew A. Chien. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In *Heterogeneous Computing Workshop*, pages 102–115, 2000.

[155] Hongsuda Tangmunarunkit, Ramesh Govindan, Sugih Jamin, Scott Shenker, et Walter Willinger. Network topology generators : degree-based vs. structural. *SIGCOMM Comput. Commun. Rev.*, 32(4) :147–159, 2002.

[156] Doar. A better model for generating test networks. In *IEEE GLOBECOM*, 1996.

[157] V. Hamscher, U. Schwiegelshohn, A. Streit, et R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 191–202. Springer-Verlag, 2000.

[158] Uri Lublin et Dror G. Feitelson. The workload on parallel supercomputers : modeling the characteristics of rigid jobs. *J. Parallel Distrib. Comput.*, 63(11) :1105–1122, 2003.

[159] Hui Li, David L. Groep, et Lex Wolters. Workload characteristics of a multicluster supercomputer. In *Job Scheduling Strategies for Parallel Processing,*

10th International Workshop, JSSPP 2004, New York, NY, USA, June 13, 2004., pages 176–193, 2004.

[160] Hui Li, Jeff Templon, et Lex Wolters. "predicting job start times on clusters". In *In proceedings of 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004), Chicago, Illinois, USA.*, april 2004.

[161] James Basney, Marty Humphrey, et Von Welch. The MyProxy Online Credential Repository. In *Software : Practice and Experience*, volume 35(8), 2005.

[162] Ahuva W. Mu'alem et Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6) :529–543, 2001.

[163] Warren Smith, Ian T. Foster, et Valerie E. Taylor. Predicting application run times using historical information. In *IPPS/SPDP '98 : Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 122–142, London, UK, 1998. Springer-Verlag.

[164] S. Vadhiyar et J. Dongarra. A metascheduler for the grid. In *Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing*, 2002.

[165] J. F. C. M De Jongh. *Share Scheduling in Distributed Systems*. PhD thesis, Technische Universiteit Delft, February 2002.

[166] Michael Litzkow, Todd Tannenbaum, Jim Basney, et Miron Livny. Checkpoint and migration of UNIX processes in the Condor distributed processing system. Technical Report UW-CS-TR-1346, University of Wisconsin - Madison Computer Sciences Department, April 1997.

[167] Andrew S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, et Paul F. Reynolds Jr. Legion : The next logical step toward a nationwide virtual computer. Technical Report CS-94-21, University of Virginia, 8, 1994.

[168] Richard West et Jason Gloudon. User-Level Sandboxing : a Safe and Efficient Mechanism for Extensibility. Technical Report BUCS-TR-2003-014, CS Department, Boston University, June 1 2003.

[169] F. Carminati et al. Technical report, HEPCAL II : Common Use Cases for a HEP Common Application Layer for Analysis. Technical report, CERN, June 1 2003.

[170] LHCb. Technical report, LHCb Computing Model. Technical report, CERN, June 1 2005.

[171] LCG Editorial Board. Technical Design Report, LHCb Computing Grid. Technical report, CERN, June 1 2005.

[172] G. Bosilca, F. Cappello, A. Djilali, G. Fedak, T. Herault, et F. Mangiette. Technical report, Performance evaluation of sandboxing techniques for peer-to-peer computing. Technical report, LRI-CNRS and Paris-Sud University. Technical report, June 2003.

[173] D. Adams, D. Barberis, C. Bee, R. Hawkings, S. Jarp, R. Jones1, D. Malon, L. Poggioli, G. Poulard, D. Quarrie, et T. Wenaus. The atlas computing model. Technical report, CERN, January 2005. CERN-LHCC-2004-037/G-085.

[174] S. Paterson. Software distribution for lhcb using pacman. Technical report, CERN, August 2004. LHCb Note 2004-066.

[175] Guido Haefeli. *Contribution to the development of the acquisition electronics for the LHCb experiment*. PhD thesis, 2004.

[176] P. R. Barbosa-Marinho et al. Lhcb online system technical design report : Data acquisition and experiment control. CERN-LHCC-2001-040.

[177] Jack J. Dongarra, Cleve B. Moler, J. R. Bunch, et George W. Stewart. *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[178] E. Caron et F. Desprez. DIET : A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 2005. To appear.

[179] Guido van Rossum. Python reference manual. Report CS-R9525, April 1995.

[180] Aaron R. Watters. Tutorial article no. 005 : The what, why, who, and where of Python. *UnixWorld Online*, 1995.

[181] D. M. Beazley. SWIG : an easy to use tool for integrating scripting languages with C and C++. pages 129–139, 1996.

[182] Franck Cappello, Frederic Desprez, Michel Dayde, Emmanuel Jeannot, Yvon Jegou, Stephane Lanteri, Nouredine Melab, Raymond Namyst, Pascale Primet, Olivier Richard, Eddy Caron, Julien Leduc, et Guillaume Mornet. Grid'5000 : A large scale, reconfigurable, controlable and monitorable grid platform. In *Grid2005 6th IEEE/ACM International Workshop on Grid Computing*, 2005.

[183] Iosif C. Legrand et Harvey B. Newman. The monarc toolset for simulating large network-distributed processing systems. In *WSC '00 : Proceedings of the 32nd conference on Winter simulation*, pages 1794–1801, San Diego, CA, USA, 2000. Society for Computer Simulation International.

[184] I. Foster. Globus toolkit version 4 : Software for service-oriented systems. In *IFIP, International Conference on Network and Parallel Computing*, pages 2–13. Springer-Verlag LNCS 3779, 2005.

[185] Guy Rixon. http ://wiki.astrogrid.org/bin/view/Astrogrid/GlobusToolkit3Problems, December 2003.

[186] Gabriel Antoniu, Luc Bougé, et Mathieu Jan. Juxmem : Weaving together the p2p and dsm paradigms to enable a grid data-sharing service. *Kluwer Journal of Supercomputing*, 2005. To appear. Preliminary electronic version available as INRIA Research Report RR-5082.

[187] David Abramson, Rajkumar Buyya, et Jonathan Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource brok. *CoRR*, cs.DC/0111048, 2001.

[188] Abhijit Bose, Brian Wickman, et Cameron Wood. Mars : A metascheduler for distributed resources in campus grids. In *5th International Workshop on Grid Computing (GRID 2004), 8 November 2004, Pittsburgh, PA, USA, Proceedings*.

[189] John Viega, Matt Messier, et Pravir Chandra. *Network Security with OpenSSL : Cryptography for Secure Communications*. 2002.

[190] M. Myers, R. Ankney, A. Malpani, S. Galperin, et C. Adams. RFC 2560 : X.509 Internet Public Key InfrastructOnline Certificate Status Protocol - OCSP. IETF RFC Publication, June 1999.

[191] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, et John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *SOSP '03 : Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 314–329, New York, NY, USA, 2003. ACM Press.

[192] Michael P. Wellman, Jeffrey K. MacKie-Mason, Daniel M. Reeves, et Sowmya Swaminathan. Exploring bidding strategies for market-based scheduling. In *EC '03 : Proceedings of the 4th ACM conference on Electronic commerce*, pages 115–124, New York, NY, USA, 2003. ACM Press.