Mise en place d'une mécanique d'asservissement de puissance électrique au travers du gestionnaire de ressources d'un supercalculateur

Du 02 mai 2013 au 30 aout 2013

Auteur	Maël JAUNAY
Maître de stage	Matthieu HAUTREUX
Tuteur ISTIA	Mehdi LHOMMEAU
Cursus en cours	3^{e}année de cycle ingénieur
	Option AGI
	Master 2 - Spécialité SDS

Commissariat à l'énergie atomique et aux énergies alternatives

CEA, DAM, DIF, F-91297, Arpajon, France

Remerciements

Je tiens tout d'abord à remercier Matthieu HAUTREUX, mon maître de stage, pour son encadrement ainsi que ses excellents conseils tout au long de ce stage, ainsi que son collègue Francis BELOT pour son implication dans ce projet et ses opinions sur sa réalisation.

Je suis également très reconnaissant envers Mehdi LHOMMEAU, mon tuteur de stage de l'ISTIA, à la fois pour m'avoir proposé ce projet au CEA et pour son suivi et son aide durant le déroulement de ce stage.

Enfin, je souhaite remercier l'ensemble des stagiaires et des thésards que j'ai pu côtoyer durant ce stage, pour leur accueil et pour la bonne ambiance qu'ils ont su entretenir durant ces quatre mois de stage.

Table des matières

Table des figures

Définitions des termes

- **Cluster** : On appelle *cluster* une grappe de serveurs. Dans le milieu du calcul haute performance, ce terme désigne un supercalculateur.
- **Flop** : Un flop est une unité de mesure correspondant à la vitesse de traitement d'un processeur. Un flop équivaut à une opération en virgule flottante par seconde.
- **Nœud de calcul** : Dans le contexte du calcul haute performance, un nœud de calcul correspond à un serveur (autrement dit à un ordinateur).
- **Parser** : Un *parser* est un anglicisme signifiant analyseur syntaxique.
- **Plugin** : Un plugin est un module informatique qui peut être chargé dynamiquement dans un programme pour lui ajouter des fonctionnalités.
- **RPC** : Remote Procedure Call, est un terme désignant l'appel d'une fonction distante à partir d'un programme local.
- **Table de hachage** : Correspond à un tableau de structures de type clé/valeur. Chaque élément est accessible via la clé qui lui est associée.

Introduction

Le calcul scientifique haute performance, communément appelé HPC [1], est un enjeu majeur pour la recherche scientifique et technologique. Il permet notamment de simuler des expériences qui ne peuvent être menées en laboratoire pour des raisons de temps, de coûts ou de risques.
C'est dans ce cadre que s'inscrit le projet Simulation du Commissariat à l'énergie atomique et aux énergies alternatives pour lequel le calculateur TERA-100 a été conçu.

La puissance électrique nécessaire au fonctionnement des supercalculateurs devient de plus en plus importante, de l'ordre de plusieurs méga watts aujourd'hui. Les objectifs en terme de puissance de calcul à moyen terme, cinq à dix ans, sont d'atteindre une puissance exaflopique [2]. L'enveloppe énergétique prévue pour ces nouvelles générations de calculateurs est de l'ordre de 20 à 25 méga watts. Sans une rupture technologique forte, la consommation électrique associée à de telles machines dépassera largement cette prévision. Il est par conséquent possible qu'une machine de prochaine génération puisse consommer, au maximum de ses capacités, plus d'énergie que le réseau électrique ne peut en fournir. Il devient donc très important de prendre en compte cette consommation énergétique et de la contrôler, afin de la limiter en fonction de la demande, ou de l'adapter en fonction des capacités et des coûts d'approvisionnement.

Un des leviers permettant d'atteindre cet objectif consiste à modifier la façon dont sont utilisés les composants du calculateur. Actuellement, le CEA utilise le logiciel open source SLURM pour gérer l'accès à l'ensemble des nœuds de calcul qui composent ses supercalculateurs.

Dans le cadre de ce stage de fin d'études, nous nous intéressons à l'intégration dans SLURM d'un ensemble de fonctionnalités permettant d'asservir la puissance électrique globale consommée par les calculateurs. Dans une première partie, on trouvera une présentation de la structure d'accueil. Ensuite seront expliquées des généralités sur les supercalculateurs, le logiciel de gestion de ressources SLURM ainsi que la méthodologie employée pour modifier son comportement afin de prendre en compte une contrainte énergétique.

1. High Performance Computing
2. 1 exaflop = 10^{18} opérations flottantes par seconde

1 Présentation du CEA

1.1 Le CEA

Acteur majeur de la recherche, du développement et de l'innovation, le Commissariat à l'énergie atomique et aux énergies alternatives intervient dans trois grands domaines : les énergies décarbonées, la Défense et la sécurité globale, les technologies pour l'information et la santé.

Pour être au plus haut niveau de la recherche, le CEA compte plusieurs atouts :

- une culture croisée entre ingénieurs et chercheurs, propice aux synergies entre recherche fondamentale et innovation technologique ;
- des installations exceptionnelles (supercalculateurs, réacteurs de recherche, lasers de puissance...) ;
- une forte implication dans le tissu industriel et économique.

Le CEA est structuré autour de cinq pôles opérationnels (Défense, énergie nucléaire, recherche technologique, sciences de la matière et sciences du vivant) et quatre pôles fonctionnels (maîtrise des risques, ressources humaines et formation, stratégie et relations extérieures, gestion des systèmes d'information), implantés dans 10 centres répartis dans toute la France. Il développe de nombreux partenariats avec les autres organismes de recherche, les collectivités locales et les universités.

Reconnu comme un expert dans ses domaines de compétence, le CEA est pleinement inséré dans l'espace européen de la recherche et exerce une présence croissante au niveau international.

Le CEA compte actuellement 15 900 personnes pour un budget de 4,3 milliards d'euros.

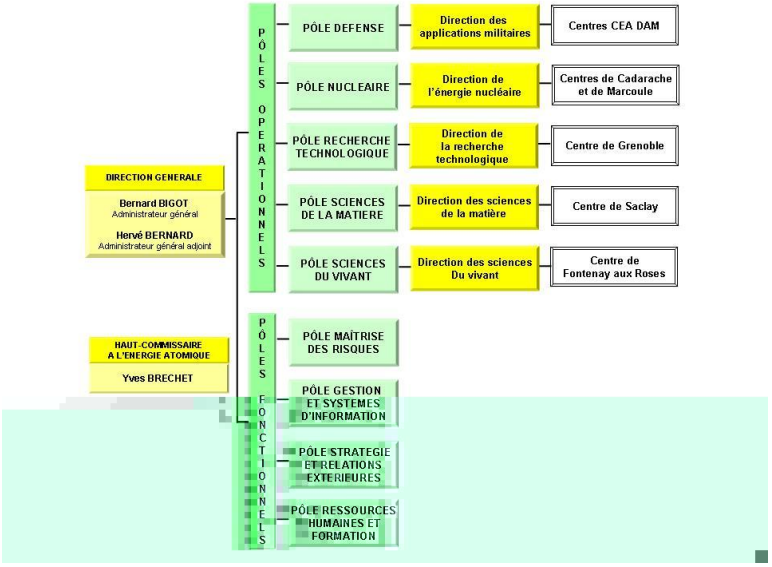

FIGURE 1 – Organigramme du CEA

1.2 La direction des applications militaires

La Direction des Applications Militaires (DAM) constitue le pôle Défense du CEA. Ce pôle a pour mission principale de concevoir, fabriquer, maintenir en condition opérationnelle puis démanteler les têtes nucléaires qui équipent les forces océaniques et aéroportées.

Apportant aux pouvoirs publics la garantie que ces têtes sont fiables et sûres, il est l'un des principaux artisans de la crédibilité de la dissuasion nucléaire française. L'objectif du Pôle Défense est de continuer à assurer sur le long terme cette capacité de dissuasion sans recourir aux essais nucléaires, définitivement arrêtés en 1996.

Ses missions portent également sur :

- l'approvisionnement en matières nucléaires pour les besoins de la Défense ;
- la propulsion nucléaire navale ;
- la lutte contre la prolifération et le terrorisme et la sécurité globale.

Le Pôle Défense publie bon nombre de ses résultats scientifiques et développe des collaborations avec d'autres acteurs de la Recherche. Il valorise ses activités auprès des industriels par le transfert de technologies et le dépôt de brevets. Il s'est également engagé dans une importante démarche de certification qualité de l'ensemble de ses activités liées aux armes nucléaires.

La DAM compte aujourd'hui 4 750 collaborateurs, menant des activités réparties entre la recherche de base, le développement et la fabrication. Son budget est d'environ 1,8 milliards d'euros.

Elle comprend :

- Un échelon Direction ;
- Cinq directions d'objectifs (DOB), qui ont pour mission la définition et la gestion des programmes et le pilotage des projets ;
- Cinq directions opérationnelles (DOP), qui ont pour mission la réalisation des produits conformément aux directeurs d'objectifs. Les directeurs opérationnels de la DAM sont les directeurs de centre ;
- Quatre directions fonctionnelles (DF), qui ont des missions de directive, de soutien et de contrôle, pour le compte du DAM.

La DAM est implantée sur cinq centres :

- Valduc, en Bourgogne ;
- Le Ripault, en Touraine ;
- le Cesta, en Aquitaine ;
- DAM - Île de France (DIF) ;
- Gramat (CEG) en Midi-Pyrénées.

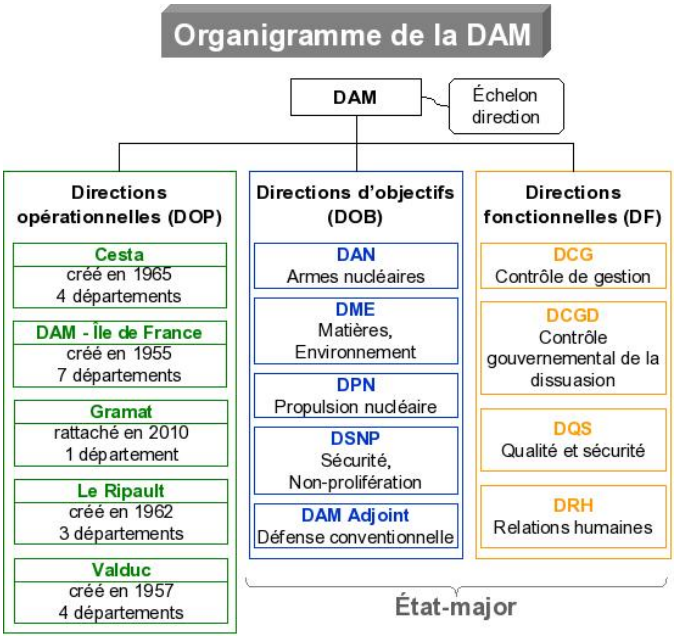

FIGURE 2 – Organisation de la DAM

1.3 Le centre DAM Ile de France

Le CEA/DAM - île de France (DIF) est l'une des directions opérationnelles de la DAM. La site de la DIF compte environ 2000 salariés CEA et accueille quotidiennement environ 600 salariés d'entreprises extérieures.

Elle comprend deux sites :

- Le site de Bruyères le Châtel, situé à environ 40 km au sud de Paris, dans l'Essonne ;
- Le site du Polygone d'expérimentation de Moronvilliers (PEM), situé à 20 km de Reims, dans la Marne.

Les missions de la DIF comprennent :

- La conception et garantie des armes nucléaires, grâce au programme Simulation. L'enjeu consiste à reproduire par le calcul les di□érentes phases du fonctionnement d'une arme nucléaire et à confronter ces résultats aux mesures des tirs nucléaires passés et aux résultats expérimentaux obtenus sur les installations actuelles (machine radiographique Airix, lasers de puissance, accélérateurs de particules) ;

- La lutte contre la prolifération et le terrorisme, en contribuant notamment au programme de garantie du Traité de Non Prolifération et en assurant l'expertise technique française pour la mise en œuvre du Traité d'Interdiction Complète des Essais Nucléaires (TICE) ;

- L'expertise scientifique et technique, dans le cadre de la construction et du démantèlement d'ouvrages complexes ainsi que pour la surveillance de l'environnement et les sciences de la terre ;

- L'alerte des autorités, mission opérationnelle assurée 24h sur 24, 365 jours par an, en cas d'essai nucléaire, de séisme en France ou à l'étranger, et de tsunami dans la zone Euro-méditerranéenne. La DIF fournit aux autorités les analyses et synthèses techniques associées.

Depuis 2003, le centre DAM-Île-de-France héberge le complexe de calcul scientifique du CEA, qui regroupe l'ensemble des supercalculateurs du CEA, et qui comprend :

- Le supercalculateur TERA-100 pour les besoins Défense, premier calculateur européen à dépasser la barre du petaflops, c'est à dire capable d'e□ectuer un million de milliards d'opérations par seconde ;

- Les ordinateurs du Centre de Calculs pour la Recherche et la Technologie (CCRT), ouverts à la communauté civile de la recherche et de l'industrie, pour une puissance globale de 500 teraflops ;

- Le supercalculateur Curie, d'une puissance de 2 petaflops, deuxième élément d'un réseau de supercalculateurs de classe pétaflopique destiné aux chercheurs de la communauté scientifique européenne. Ce supercalculateur est hébergé au TGCC (Très Grand Centre de Calcul) et exploité par les équipes du CEA, qui apporte ainsi sa contribution à la participation de la France au projet PRACE (Partnership for Advanced Computing in Europe), dans le cadre de la recherche européenne.

2 La notion de puissance consommée

2.1 Les supercalculateurs et le gestionnaire de ressources

Quelques généralités

Un supercalculateur est composé d'un ensemble d'ordinateurs, aussi appelés nœuds de calcul, reliés entre eux par un réseau haute performance. Ces machines interconnectées permettent de constituer un calculateur très performant d'un point de vue puissance de calcul. Ces calculateurs atteignent actuellement des performances pétaflopiques [1].

D'une manière générale, les supercalculateurs sont architecturés d'une façon analogue à celle décrite par la figure 3. Cette architecture comporte un nœud maître qui va communiquer avec l'ensemble des autres nœuds. La plus grande partie de ces nœuds sont dédiés aux calculs. Certains d'entre eux sont cependant réservés à l'administration et d'autres permettent aux utilisateurs de se connecter sur le calculateur. Cette configuration permet d'obtenir une puissance de calcul importante, directement liée au nombre de serveurs composant le calculateur. Cependant, pour que les calculs soient réellement performants, ils doivent être fortement parallélisés.

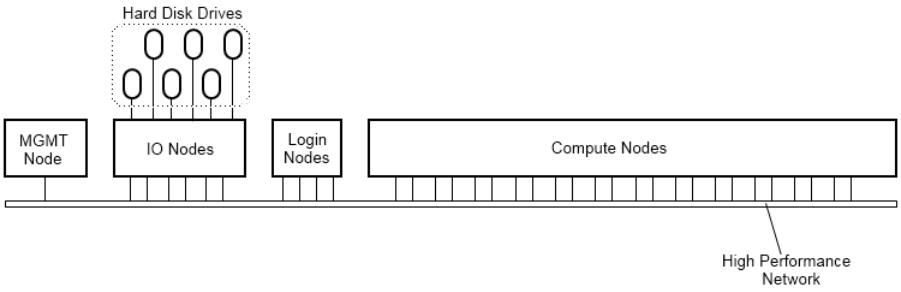

FIGURE 3 – Architecture d'une grappe de serveurs

Il est à noter que les nœuds de calcul qui constituent un cluster ne sont pas nécessairement constitués des mêmes composants. Utiliser un calculateur hétérogène permet notamment de le rendre plus adaptable aux di□érentes

1. 1 pétaflop correspond à 10^{15} calculs sur des opérandes à virgule flottante par seconde

tâches qu'on peut lui allouer. Il est par exemple possible d'optimiser certaines tâches pour qu'elles soient exécutées sur des accélérateurs matériels.

Pour gouverner l'ensemble des nœuds de calcul d'un calculateur, il est nécessaire de recourir à un type de logiciel appelé gestionnaire de ressources. Ce logiciel remplit trois fonctions principales :

- il alloue de façon exclusive ou non des ressources pour des durées précises ;
- il fournit un environnement permettant de démarrer, d'exécuter, et de suivre l'avancement des tâches de calcul sur l'ensemble des nœuds alloués ;
- il arbitre les conflits de ressources en gérant une file de tâches en attente d'exécution.

Dans le cadre de ce projet, c'est au gestionnaire de ressources open source SLURM que nous allons nous intéresser. Ce logiciel, développé en C, est conçu selon une architecture relativement simple. Il est constitué d'un

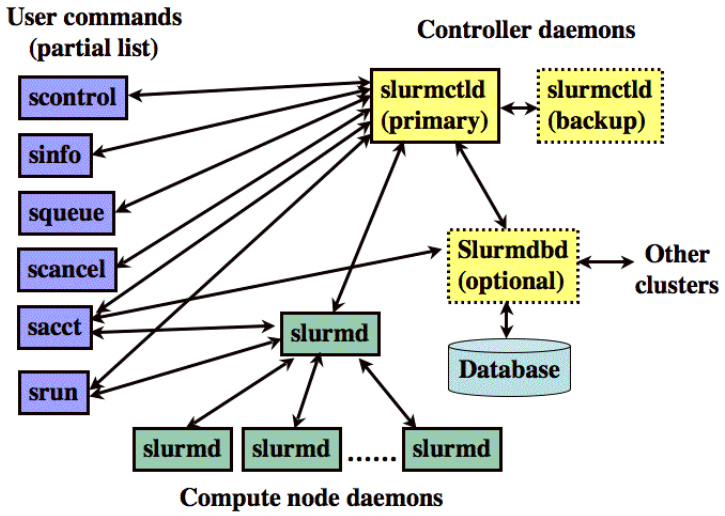

FIGURE 4 – Architecture de SLURM

contrôleur central, représenté par *slurmctld* sur la figure 4, ainsi que d'un ensemble d'agents résidant sur les di□érents nœuds de calcul, représentés par

les *slurmd*. Ces processus sont accompagnés d'un ensemble de commandes permettant d'agir soit sur le contrôleur, soit directement sur les nœuds ou sur les calculs en cours.

La problématique

Les performances d'un seul nœud de calcul étant limitées, l'augmentation des performances globales des calculateurs passe par une hausse du nombre de nœuds de calcul. Ce type d'architecture implique d'adapter les codes de calcul en les parallélisant plus. Aussi, cet accroissement du nombre de nœuds augmente considérablement l'impact énergétique des calculateurs. Puisque la consommation énergétique est déjà de plusieurs méga watts, et qu'elle va continuer à augmenter, il est indispensable d'optimiser les coûts d'exploitation des clusters. Leurs performances seront éventuellement bridées, mais cette optimisation permettra d'obtenir un ratio performance-coût plus intéressant.

L'approche que nous proposons dans ce rapport, pour mieux contrôler la consommation du cluster, consiste à intégrer une mécanique d'asservissement dans le gestionnaire de ressources. C'est en e□et le lieu idéal pour maîtriser au mieux le calculateur puisque ce logiciel dispose d'une vue d'ensemble de ses composants. L'objectif de cet asservissement est de pouvoir contrôler en temps réel la puissance électrique utilisée par le calculateur dans sa globalité.

La méthode retenue dans un premier temps consiste à entraver certains nœuds de calcul. Ils ne pourront donc pas être sélectionnés pour des tâches de calcul. En procédant de la sorte, seule la quantité d'énergie nécessaire pour maintenir ces nœuds en état de repos sera considérée.

En considérant le graphe de la figure 5 comme l'objectif à atteindre, il est nécessaire de définir certaines notions et certaines notations.

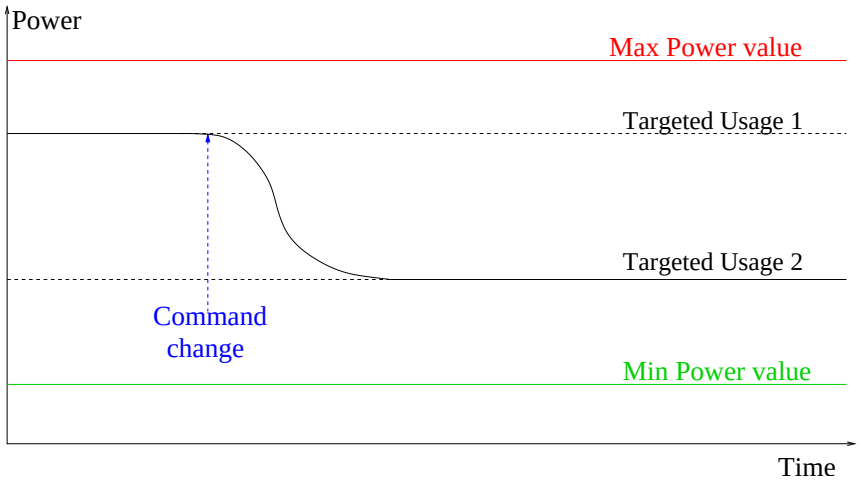

FIGURE 5 – Comportement de consommation souhaité

2.2 Notion de puissance électrique consommée

Un des problèmes majeurs de l'ordonnancement sous contraintes (comprendre le placement de tâches sur des nœuds de calcul) est qu'il n'est pas solvable dans un temps raisonnable, et est la plupart du temps pris en charge par des heuristiques.

Plutôt que de modifier l'heuristique d'ordonnancement, nous allons réduire le nombre de nœuds de calcul utilisés afin de respecter la contrainte énergétique. Ces nœuds inutilisés doivent donc être placés dans un état leur permettant de consommer moins d'énergie. SLURM permet d'attribuer différents états aux nœuds, les plus courants sont les suivants :

- $DOWN$, signifiant que le nœud est actuellement inutilisable par le gestionnaire ;
- $IDLE$, signifiant que le nœud est actuellement en état de repos, il est donc alimenté mais n'est pas utilisé pour des calculs ;
- $ALLOCATED$, signifiant que le nœud est actuellement utilisé pour des calculs ;

L'état qui nous intéresse le plus dans un premier temps est l'état $idle$.
Nous introduisons un nouvel état pour les nœuds, l'état exclu nommé $EXCLUDED$. Cet état peut être donné à un nœud en plus des états mentionnés

ci-dessus. Il est, par exemple, possible de placer un nœud dans un état $ALLOCATED$ et d'indiquer dans le même temps qu'il est $EXCLUDED$. Cette combinaison permettra d'exclure le nœud une fois que la tâche qui lui est allouée sera terminée.

Nous considérons les notations suivantes :

- $PowerCap$, la puissance maximale de fonctionnement imposée au calculateur ;
- N, le nombre de nœuds constituant le calculateur ;
- $Nodes = \{n_i \mid 0 \leqslant i \leqslant N\}$, représente l'ensemble des nœuds du calculateur
- Chacun de ces nœuds contient les valeurs suivantes :
  - $IdleWatts$, la puissance de fonctionnement du nœud en état $idle$;
  - $MaxWatt$, sa puissance de fonctionnement en pleine charge ;
  - $CapPrio$, son poids d'exclusion, une valeur 0 indiquant qu'il est impossible d'exclure ce nœud ;
  - $State$ son état courant, potentiellement composé d'un état de base, par exemple $ALLOCATED$, et d'un état secondaire tel que l'état $EXCLUDED$

Nous définissons également une fonction indicatrice générique pour faciliter l'écriture des équations suivantes :

$$\mathbb{1}_A = \begin{cases} 1 \text{ si A est vrai} \\ 0 \text{ sinon} \end{cases}$$

A partir de ces notations, nous pouvons définir les di□érentes contraintes qui vont influencer l'asservissement de la puissance électrique du calculateur :

La consommation maximale du calculateur,

$$ClMaxWatts = \sum_{i=0}^{N} n_i \rightarrow MaxWatts \quad (1)$$

La consommation maximale réelle du calculateur,

$$ClCurrentMaxWatts = ClMaxWatts - \sum_{i=0}^{N} (n_i \rightarrow MaxWatts) \times \mathbb{1}_{n_i \rightarrow State=DOWN} \quad (2)$$

Sa consommation minimale,

$$ClMinWatts = \sum_{i=0}^{N} (n_i \rightarrow MaxWatts) \times \mathbb{1}_{n_i \rightarrow CapPrio=0} + \sum_{i=0}^{N} (n_i \rightarrow IdleWatts) \times \mathbb{1}_{n_i \rightarrow CapPrio>0} \quad (3)$$

La consommation maximale courante du calculateur,

$$\begin{aligned} ClEnforcedMaxWatts = & \sum_{i=0}^{N} (n_i \rightarrow MaxWatts) \times \mathbb{1}_{n_i \rightarrow CapPrio=0} + \\ & \sum_{i=0}^{N} (n_i \rightarrow IdleWatts) \times \mathbb{1}_{n_i \rightarrow CapPrio>0} \times \mathbb{1}_{n_i \rightarrow State\ \&\ EXCL} \times \mathbb{1}_{n_i \rightarrow State\ \&\ \overline{ALLOC}} + \\ & \sum_{i=0}^{N} (n_i \rightarrow MaxWatts) \times \mathbb{1}_{n_i \rightarrow CapPrio_i>0} \times \mathbb{1}_{n_i \rightarrow State\ \&\ \overline{EXCL}} \times \mathbb{1}_{n_i \rightarrow State\ \&\ ALLOC} \end{aligned} \quad (4)$$

Nous considérons également la contrainte suivante qui doit être respectée par le système d'asservissement :

$$ClMinWatts \leqslant ClEnforcedMaxWatts < PowerCap \leqslant ClCurrentMaxWatts \leqslant ClMaxWatts \quad (5)$$

Dans le cas de l'équation 4, l'état $DOWN$ des nœuds devra être considéré

en plus de l'état $EXCLUDED$ si un paramètre de la configuration globale est déclaré. En e□et, les nœuds que nous considérons $DOWN$ peuvent être relancés de façon manuelle par les administrateurs système, auquel cas leurs consommations correspondraient aux valeurs $IdleWatts$. Dans ce cas, il serait nécessaire de comptabiliser ces valeurs dans le bilan énergétique du calculateur.
La contrainte 5 indique la relation d'ordre entre les di□érentes valeurs décrites précédemment. La valeur estimée de la consommation de puissance du cluster est $ClEnforcedMaxWatts$. Cette valeur sera toujours supérieure à la consommation réelle puisque les valeurs de consommation des nœuds sont elles mêmes légèrement supérieures aux valeurs réelles. Le fait de considérer une valeur majorée permet d'avoir la certitude que le système n'utilisera jamais plus de puissance électrique que ce que la consigne impose.

En adaptant la première courbe de consommation (figure 5), on peut mettre en avant les di□érentes notions présentées ci-dessus. Ce graphe introduit aussi la notion d'écart à la consigne, ou erreur statique, noté $Static\ error$ en figure 6. Cette valeur apparaît à cause de l'emploi de valeurs entières pour la consommation de chacun des nœuds et de la consigne qui ne sera pas nécessairement égale à la somme des valeurs de consommation unitaires des nœuds.

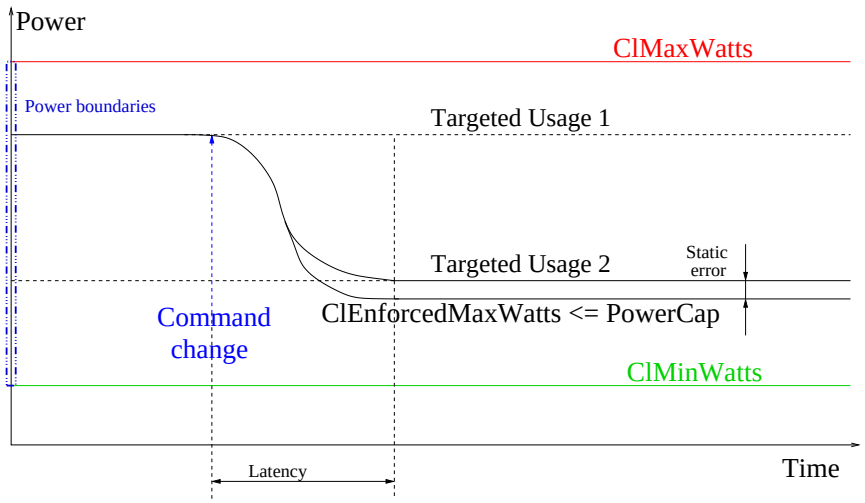

FIGURE 6 – Comportement de consommation recherché

Il est également à noter l'apparition d'une latence entre le changement de consigne et son application réelle. Ce temps de réponse est lié au fait que certaines tâches en cours d'exécution ne pourront pas nécessairement être arrêtées immédiatement. Cette latence sera exprimée dans le fichier de configuration de SLURM sous la forme d'un délai de grâce accordé aux tâches de calcul.
Sachant qu'au CEA, la plupart des tâches sont programmées de façon à intégrer une mécanique permettant de vérifier leur durée et leur date de fin d'exécution de façon autonome, ce délai de grâce permettra, depuis le contrôleur, d'indiquer si certaines tâches doivent se terminer plus tôt automatiquement, quitte à être replacées en file d'attente pour achever leurs calculs ultérieurement.

2.3 Intégration de cette notion dans SLURM

Pour intégrer cette notion de puissance consommée dans le gestionnaire de ressources SLURM, il y a plusieurs points à considérer. Premièrement, puisque la méthode s'appuie sur une utilisation di□érente des nœuds de calcul, il faut définir un nouvel état et adapter le reste du produit pour le prendre en compte. Cet état est $NODE\_STATE\_EXCLUDED$. Il permet d'indiquer qu'un nœud ne doit pas être utilisé par le gestionnaire pour effectuer des calculs.

Ensuite, les valeurs représentant la consommation énergétique du cluster, que ce soit la consommation globale ou celles des nœuds de calcul, sont codées dans les structures de données de SLURM. La valeur correspondant à la consigne d'asservissement (i.e. la contrainte de puissance électrique globale) est intégrée dans la structure informatique du contrôleur de SLURM. Ensuite, de nouveaux attributs sont incorporés dans la structure de données qui représente les nœuds de calcul :

- max_watts, représente la consommation maximale d'un nœud ;
- $idle\_watts$, représente sa consommation en état de repos ;
- $perf\_watts$, représente le ratio flops/watt du nœud ;
- $power\_cap\_priority$, représente la priorité d'exclusion d'un nœud.

La dernière valeur est un peu plus particulière que les autres puisqu'elle va indiquer si un nœud est apte ou non à être exclu. Si elle est renseignée à 0, alors l'état de ce nœud ne pourra en aucun cas être altéré. Typiquement, cette valeur sera assignée pour les nœuds du calculateur qui sont réservés à l'administration, ou encore à ceux qui sont utilisés pour se connecter au calculateur.
Si cette valeur n'est pas nulle, elle indiquera dans quel ordre sélectionner les nœuds à exclure. Plus la valeur sera forte, plus le nœud aura de chances d'être exclu pour économiser de l'énergie. Dans le cas d'un calculateur dont les nœuds sont homogènes, cette valeur sera identique pour tous les nœuds de calcul.

Afin de rendre ces nouveaux attributs réellement exploitables, il est également nécessaire de les rendre visibles depuis les di□érentes commandes de contrôle de SLURM. Il y a principalement deux commandes à modifier : *sinfo*, qui permet à tout utilisateur d'a□cher les données relatives à l'état du calculateur, et *scontrol* qui permet aux administrateurs de contrôler le fonctionnement du produit SLURM.

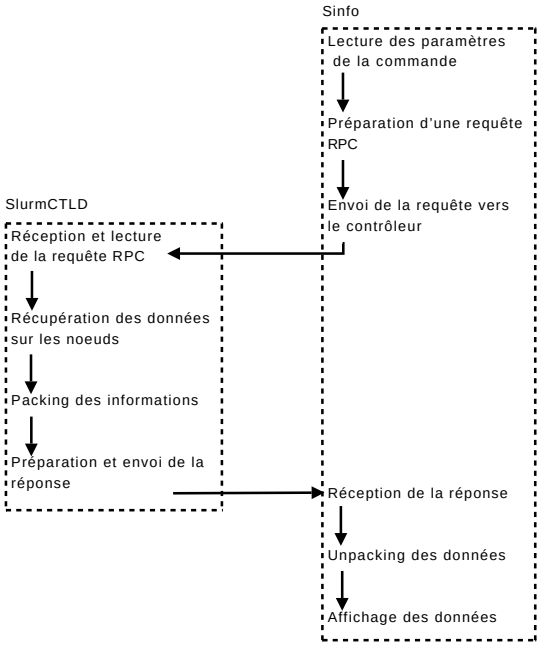

FIGURE 7 – Communication entre le contrôleur et *sinfo*

Un point essentiel du fonctionnement de SLURM est sa mécanique de communication. Que ce soit des nœuds vers le contrôleur, ou bien des commandes externes vers le contrôleur ou les nœuds, SLURM utilise un système de communication à base de RPC.
Toutes les données transitant entre les processus sont donc transmises sous forme de paquets de données binaires. Toutes les nouvelles informations intégrées dans SLURM doivent donc être référencées dans ce protocole de communication en prenant soin d'en respecter les versions.

Dans le cas de la commande *scontrol*, les appels de fonction se font de façon analogue au comportement décrit en figure 7.
Cette mécanique de communication repose essentiellement sur les fonctions de *packing/unpacking*. Ces fonctions permettent d'encoder (fonction de *packing*) et de décoder (fonction d'*unpacking*) les données présentes dans les structures du programme afin de les transmettre aux différents processus actifs de SLURM.

Pour permettre au système de lire ces nouvelles données depuis le fichier *slurm.conf*, il a fallu faire évoluer les différents *parsers* de l'application. Le principe de fonctionnement des *parsers* de SLURM est décrit par la figure 8. Une fois le fichier de configuration lu, les données sont stockées dans une table de hashage temporaire, qui permet d'effectuer différents traitements sur les valeurs lues. La plupart de ces traitements servent à transformer les valeurs textes en valeurs numériques (fonction de *cast*).

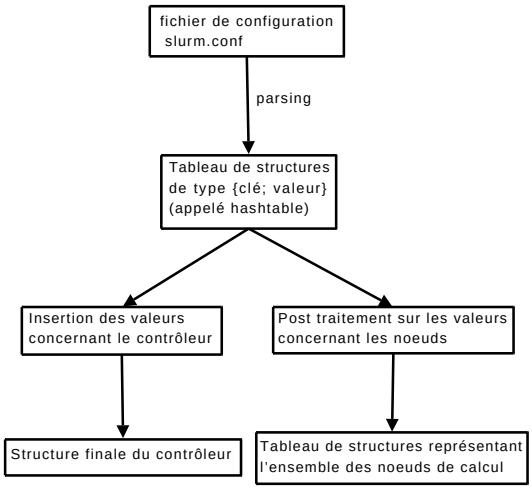

FIGURE 8 – Parsing de la configuration

Enfin, une dernière partie de SLURM doit être modifiée, la gestion des nœuds. Pour faciliter cette gestion, SLURM utilise des $bitmaps$. Succinctement, les $bitmaps$ sont des tableaux de bits présents dans la mémoire globale du contrôleur permettant de référencer les nœuds selon leur état. L'intérêt de ces tableaux, est qu'ils permettent de connaître et de faire évoluer rapidement l'état de l'ensemble des nœuds de calcul. Par exemple si nous nous intéressons aux nœuds utilisables pour le calcul, on peut lire le $bitmap$ correspondant dans SLURM. Pour un cluster de dix nœuds, un tableau semblable à celui présenté en figure 9 signifierait qu'il est possible d'utiliser six nœuds.

0	0	0	0	1	1	1	1	1	1

FIGURE 9 – Exemple d'un bitmap de dix éléments

Puisque nous proposons, pour l'asservissement, de masquer les nœuds au système d'ordonnancement, il est nécessaire d'ajouter un $bitmap$ représentant les nœuds exclus pour cette raison. Cela permettra par la suite de dé-référencer les nœuds à exclure des autres $bitmaps$. C'est grâce à ce mécanisme qu'il sera possible d'altérer la façon dont l'algorithme d'ordonnancement placera les calculs sur les ressources disponibles.

3 L'heuristique de contrôle de l'asservissement

3.1 L'heuristique mise en place

Une fois les attributs ajoutés au sein du gestionnaire de ressources, il est indispensable d'intégrer une heuristique permettant de sélectionner les di□érents nœuds du système devant être exclus pour respecter la consigne énergétique. Cette heuristique s'appuie sur deux points essentiels.
Premièrement, l'asservissement se base sur une fonction de coût. Comme expliqué dans le chapitre précédent, à chaque nœud de calcul est associé un poids. On pourrait se contenter dans un premier temps d'une valeur uniquement calculée à partir des valeurs au repos et en pleine charge de la consommation des nœuds. Cependant, à cause de l'homogénéité (au moins partielle) des calculateurs, on se retrouverait avec beaucoup de valeurs identiques, ce qui compliquerait la sélection.

On définit par conséquent la fonction de coût en prenant en compte quatre facteurs :

- la di□érence de consommation du nœud, i.e. max_watts - $idle\_watts$;
- le ratio de performance du nœud, i.e. son rapport $\frac{flops}{watt}$;
- la priorité du nœud, la valeur renseignée dans l'attribut $power\_cap\_priority$;
- et un poids topologique représentant la quantité de nœuds disponibles au voisinage réseau du nœud concerné.

Chacun de ces facteurs ayant un impact di□érent sur le système, ils sont pondérés par un coe□cient déclaré dans le fichier de configuration de la même manière que les autres attributs (cf section 2.3). Pour la suite de ce rapport, nous noterons ces coe□cients de la façon suivante :

- $C_{\Delta Power}$, l'impact de la puissance consommée ;
- $C_{Performance}$, l'impact du ratio $\frac{flops}{watt}$;
- $C_{Priority}$, l'impact de la priorité d'exclusion ;
- $C_{Topology}$, l'impact de la topologie.

Ces notations nous permettent de définir la fonction de coût comme étant :

$$\forall\ node\ i\ n_i\ \in\ Nodes$$

$$Cost_i\ =\ C_{\Delta Power} * \Delta ScaledPower\ +\ C_{Performance} * ScaledPerf_i\ +$$

$$C_{Priority} * ScaledPrio_i\ +\ C_{Topology} * ScaledTopo_i \tag{6}$$

$$avec$$

$$\Delta ScaledPower_i\ =\ \frac{n_i \rightarrow MaxWatts - n_i \rightarrow IdleWatts}{\max_J(n_J \rightarrow MaxWatts - n_J \rightarrow IdleWatts)}$$

$$PerfValue_i\ =\ \frac{Frequency_i * CoreNumber_i * InstructionNumber_i}{n_i \rightarrow MaxWatts}$$

$$ScaledPerf_i\ =\ \frac{PerfValue_i}{\max_J(PerfValue_J)}$$

$$ScaledPrio_i\ =\ \frac{n_i \rightarrow PowerCapPriority}{\max_J(n_j \rightarrow PowerCapPriority)}$$

$$ScaledTopo_i\ =\ \frac{TopoWeight_i}{\max_J(TopoWeight_J)}$$

avec $J \in [1, N]$

Outre cette fonction de coût, la sélection des nœuds doit également tenir compte de certains facteurs qui ne sont pas liés directement à la configuration des nœuds.

Pour commencer, il est nécessaire de s'assurer qu'aucun calcul n'est en cours sur le nœud que l'on souhaite exclure. Cependant, la plupart des programmes de calculs qui sont traités par les calculateurs, du moins dans le cadre du CEA, sont conçus en intégrant un système de vérification qui permet de contrôler le temps final d'exécution prévu. En procédant de la sorte, si un calcul n'est pas réellement prioritaire, il est envisageable de raccourcir son allocation de temps et de le pousser à se terminer proprement. Si les

calculs ne sont pas terminés dans le nouveau laps de temps qui leur est accordé, il sera toujours possible de les arrêter.
Un des principaux avantages d'autoriser l'arrêt de certains calculs est de pouvoir exclure des nœuds proches, soit physiquement dans la salle machine, soit sur le réseau. En procédant de la sorte, la prise en compte du critère topologique est facilitée.

La constitution des nœuds est un autre élément qui doit être pris en compte. La plupart des calculateurs n'étant pas homogènes, des nœuds de calcul peuvent avoir des composants qui répondent mieux aux besoins de certains programmes. L'exemple le plus courant permettant d'illustrer cette proposition, est l'accélération graphique. En e□et, certains calculs peuvent requérir de la puissance graphique pour s'exécuter, soit pour générer des a□chages particuliers, soit pour tirer parti de la puissance de calcul importante des cartes graphiques.
Sachant cela, il est possible de prendre en compte les besoins des prochains programmes qui vont être exécutés. Il su□t pour cela de retirer du $bitmap$ d'exclusion des nœuds qui vont correspondre aux besoins du programme. Il est nécessaire dans le même temps d'ajouter d'autres nœuds dans le $bitmap$ d'exclusion afin de continuer à respecter la consigne de consommation de puissance.

Un dernier point doit être pris en compte lors de la sélection des nœuds à exclure, la contrainte globale. Cette consigne ayant été conçue pour être dynamique et planifiable, le volume de nœuds exclus doit en anticiper les variations, en relâchant ou en excluant des nœuds selon le sens de variation.

Cet ensemble de contraintes nous permet de définir la fonction de sélection des ressources en s'appuyant sur l'algorithme décrit en figure 10.

Le principe de fonctionnement de cet algorithme est relativement simple. Dans un premier temps, si la consigne n'est pas déjà respectée, une liste va être créée à partir des nœuds qui peuvent être exclus, car inutilisés (lignes 4-10). Ensuite, on exclut du système les nœuds de cette liste qui ont le coût d'utilisation le plus fort (lignes 11-22). Enfin, si la consigne n'est toujours pas respectée, l'algorithme va rechercher les calculs qui peuvent être préemptés et les forcer à s'arrêter (ligne 23-28). Ces nœuds sont également exclus mais

```
1: Declare MaxState, ExcludedPower, CommandedValue As float
2: Declare nodesToExclude As Node list

3: If MaxState − ExcludedPower > CommmandedValue Then
4:   For Each node_i ∈ Nodes Do
5:     If node_i is not allocated and is not reserved Then
6:       // Allocated nodes will be excluded next call if necessary
7:       nodesToExclude → append({node_i, Cost(node_i)})
8:       // Cost() is calculated with equation 6
9:     End If
10:  End For
11:  If nodesToExclude → Count > 0 Then
12:    sort nodesToExclude by Cost(node_i) DESCENDING
13:    While MaxState − ExcludedPower > CommmandedValue
       Do
14:      // We exclude heaviest nodes first
15:      n = nodesToExclude → pop()
16:      Exclude(n)
17:      Get ExcludedPower new value
18:      If nodesToExclude → Count == 0 Then
19:        Break While loop
20:        // No more nodes to exclude from list
21:      End If
22:    End While
23:  End If
24:  If MaxState − ExcludedPower > CommmandedValue Then
25:    // Look for preemptable jobs to respect the global constraint
26:    Find Preemptable Jobs
27:    Preempt Jobs
28:    Exclude(n)
29:    Get ExcludedPower new value
30:  End If
31: End If
```

FIGURE 10 – Pseudo code de la fonction de sélection de nœuds

vont bénéficier d'un laps de temps pour que les programmes qu'ils accueillent aient le temps de se terminer (le délai de grâce mentionné dans les parties précédentes de ce rapport). On peut également noter qu'à chaque itération la valeur de la puissance électrique déjà économisée est recalculée pour prendre en compte les nœuds qui viennent d'être exclus.

3.2 L'intégration dans l'ordonnanceur de SLURM

Avant d'envisager de modifier le système d'ordonnancement de SLURM, il est nécessaire de comprendre son fonctionnement. La figure 11 décrit les appels de fonctions qui sont e□ectués dans le gestionnaire de ressources pour placer des calculs sur les nœuds. A l'initialisation du contrôleur du gestionnaire de ressources, un processus détaché est créé par le biais du système de $plugin$ de SLURM. Ce processus va travailler en tâche de fond, et régulièrement appeler la fonction de placement correspondant à l'algorithme du $Backfilling$, définit en annexe A. Pour chacune des tâches de la file d'attente, cet algorithme va e□ectuer di□érents tests permettant de connaître les nœuds de calcul aptes à démarrer la tâche.
Un des tests e□ectués par l'ordonnanceur de SLURM concerne les réservations de ressources. Ces réservations sont définies par des dates de début et de fin, un ensemble de ressources, et des droits d'utilisation pour indiquer qui peut les exploiter. Pour assurer le bon fonctionnement de l'exclusion de nœuds, il est nécessaire d'ajouter, au moment de la création de ces réservations, un test pour vérifier que des nœuds exclus ne sont pas réservés pour des calculs.

Pour intégrer la fonction de sélection et d'exclusion de nœuds, ce test va particulièrement nous intéresser. L'intérêt de ce test est qu'il est systématiquement appelé lorsqu'un calcul doit être lancé sur le calculateur. Lorsqu'il est e□ectué, tous les $bitmaps$ sont testés afin de s'assurer que les nœuds censés être utilisés pour le calcul sont bien disponibles. C'est ici qu'interviendra le $bitmap$ référençant les nœuds exclus du système.

Comme on peut le voir sur la figure 11, lorsqu'une tâche est testée avant d'être placée, il y a deux possibilités. Premièrement, la tâche est déjà dotée d'une réservation. Dans ce cas, il n'y aura pas de tests supplémentaires à e□ectuer puisque le système de réservation de SLURM a déjà été modifié

pour ne pas créer de réservations sur des nœuds exclus.
Deuxièmement, la tâche ne dispose d'aucune réservation. Dans ce cas, la fonction de placement va rechercher des espaces d'allocation potentiels pour la tâche. Lors de cette recherche, un test est ajouté afin de ne pas allouer cette tâche sur des nœuds référencés dans le $bitmap$ des nœuds exclus.

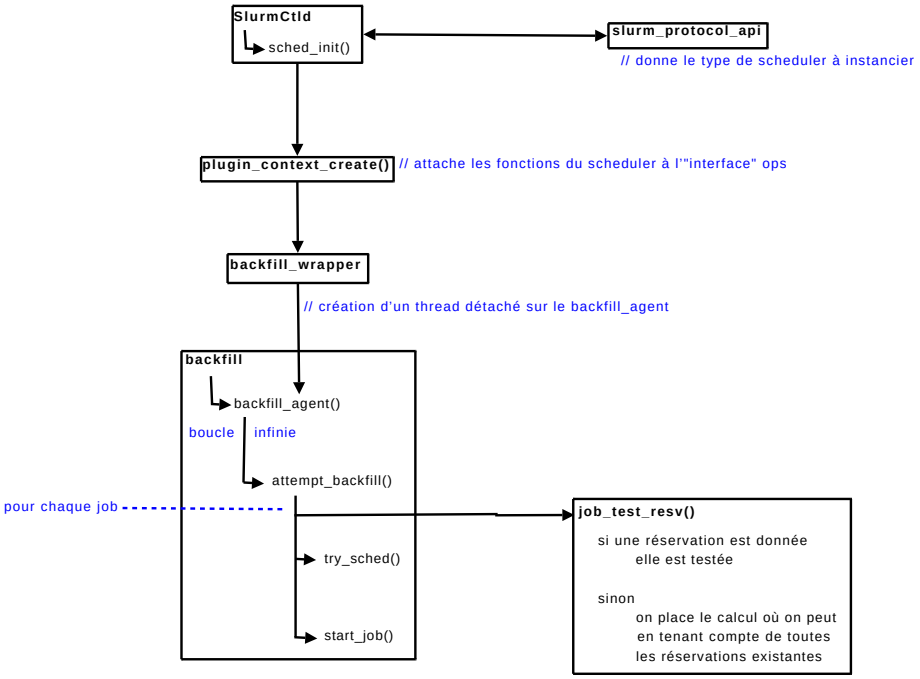

FIGURE 11 – Fonctionnement de l'ordonnanceur

Afin de compléter l'adaptation de l'ordonnanceur à la mécanique d'exclusion de nœuds, il reste un élément à modifier. SLURM inclut une fonctionnalité permettant de notifier le contrôleur des différents événements qui se produisent dans le calculateur. La gestion de ces événements permet de rendre le gestionnaire de ressources plus réactif. Par exemple, si un nœud est placé en état $DOWN$ suite à une défaillance, SLURM sera capable de réagir et d'essayer de le remettre en état de marche. Ces événements vont permettre de prendre en compte différentes éventualités relatives à l'asservissement en puissance :

- Dans le cas d'un changement d'état d'un nœud, surtout le cas dans

lequel un nœud devient $DOWN$, il est important que le contrôleur soit notifié pour réagir en conséquence. Dans le cadre de ce projet, on peut envisager d'exclure ces nœuds et de libérer un nœud fonctionnel.

- Dans le cas du $bitmap$ d'exclusion de nœuds, chaque modification doit être notifiée pour synchroniser les autres $bitmaps$ du système et éviter des conflits.

- La fonction permettant de bloquer les nœuds pour respecter la consigne doit être appelée régulièrement, cet appel sera idéalement effectué suite à une notification concernant un changement de la consigne énergétique, ou l'apparition d'un nouveau programme de calcul.

3.3 Ajout de la notion de planification de contrainte

Cette dernière fonctionnalité est encore en cours d'étude. L'objectif de ce projet est d'adapter la charge du calculateur en fonction de la quantité de puissance électrique disponible. Pour faciliter la gestion de l'asservissement en puissance électrique du calculateur, nous souhaitons également intégrer la possibilité de planifier cette contrainte.

Considérons l'exemple de la courbe représentée en figure 12. Entre 18h et 22h, le calculateur est contraint de n'utiliser que 40% de ses capacités. Admettons qu'il n'en utilise que 35% sur cette fenêtre de temps. Toute tâche de calcul nécessitant plus de 5% de puissance de calcul ne pourra donc pas être planifiée sur cette période. Une fois la consigne atteinte, plus aucune tâche ne pourra être démarrée.

L'intérêt de cette méthode est que le calculateur peut être parfaitement autonome vis à vis de la consigne énergétique. En e□et dans le cas où les variations de la contrainte sont renseignées dans SLURM, le système d'exclusion pourra alors ajuster automatiquement le $bitmap$ d'exclusion. Les tâches de calcul devront donc attendre qu'il y ait su□samment de nœuds de calcul disponibles, mais également qu'il y ait su□samment de puissance électrique disponible.
En revanche, si les variations de la contrainte ne sont pas connues du gestionnaire de ressources, il y aura systématiquement un temps d'adaptation entre l'instant où la consigne change et celui où elle est respectée.

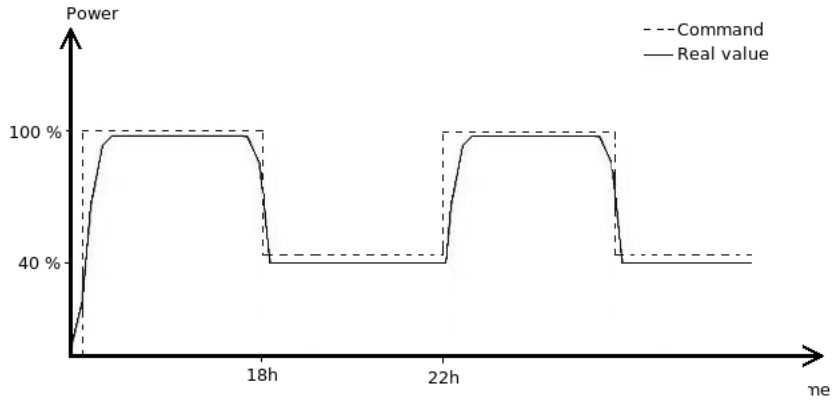

FIGURE 12 – Réponse du système aux changements de consigne

Une autre utilisation des réservations est à envisager dans le cadre de la gestion de la puissance électrique du calculateur. Des réservations renseignées par une certaine quantité de puissance électrique permettraient de bloquer des nœuds, sans nécessairement savoir lesquels, dont la somme des puissances unitaires serait égale à la puissance indiquée dans cette réservation. Cette approche pourra être utilisée pour anticiper les changements de contrainte, mais également pour imposer des comportements particuliers au calculateur. Il serait par exemple possible d'imposer une consommation de puissance cyclique. Sachant que les coûts d'approvisionnement en électricité varient au cours du temps, le calculateur pourrait augmenter ou diminuer son utilisation de puissance électrique selon ces coûts.

Conclusion

A quelques semaines de la fin de ce stage, une première maquette logicielle a été intégrée dans le gestionnaire de ressources SLURM. Cette maquette propose trois fonctionnalités indispensables à l'asservissement de la puissance électrique nécessaire au fonctionnement d'un calculateur. Tout d'abord, une méthode permet d'exclure des nœuds. Ensuite une heuristique permet de sélectionner les nœuds à exclure selon une fonction de coût dépendant de différents critères (consommation électrique, priorité arbitraire, topologies physique et réseau, ...). Enfin, une boucle d'asservissement basée sur l'heuristique de sélection et la méthode d'exclusion permet de respecter la contrainte énergétique en permanence.

L'aspect planification de contrainte doit encore être étudié et implémenté. Cette partie du projet s'appuiera sur les différentes fonctionnalités déjà mises en place. Une prochaine étape intéressante consistera à diversifier les états des nœuds de calcul. On pourra notamment donner la possibilité de régler la fréquence et le voltage des composants des nœuds pour réduire la vitesse de calcul et atteindre les valeurs optimales des ratios entre la puissance de calcul et la consommation.

Du point de vue personnel, ce stage m'aura permis de participer à l'évolution d'un projet open source et de mieux comprendre les différents enjeux et méthodes de travail du développement libre. Ce projet m'a également donné l'occasion de travailler sur un langage beaucoup moins abstrait que ceux pour lesquels j'ai été formé, et de mieux prendre conscience du fonctionnement à la fois des machines sur lesquelles j'ai pu travailler et sur les langages de programmation avec lesquels je développerai au cours de ma carrière.

Enfin, la clarté des objectifs de ce projet, ainsi que l'encadrement qui a été proposé tout au long de sa réalisation m'ont permis de travailler dans un cadre idéal pour compléter ma formation d'ingénierie informatique.

Références

[1] Calcul Haute Performance, CEA.
http://www-hpc.cea.fr/index.htm.

[2] Classement Top 500 de juin 2013.
http://www.top500.org/list/2013/06/.

[3] Club des développeurs.
http://www.developpez.com/.

[4] Simple Linux Utility for Resource Management.
http://slurm.schedmd.com.

[5] Stack Overflow.
http://stackoverflow.com/.

[6] Pierre LECA and Sophie HOUSSIAUX. Le TERA 100 brille côté rendement. *La Recherche*, Novembre 2011.

[7] Florent REYNIER. Ordonnancement de tâches pour minimiser la consommation maximale d'énergie d'un ordinateur. Master's thesis, ISTIA, 2011.

A Pseudo code de l'algorithme Backfilling

1: **In :** P the job list (with t the execution time and proc the number of processors required for this job)
2: **Out :** M the scheduling matrix, size mxn

3: $DeadLine \leftarrow 0$
4: $TemporaryDeadLine \leftarrow 0$
5: $task \leftarrow Pop(P)$
6: **While** P is not empty **Do**
7: // Search for empty space in matrix M
8: **For** $i = 0$ *to* n **Do**
9: **For** $j = 0$ *to* m **Do**
10: **If** M[n,m] *is not allocated* **Then**
11: // Is the matrix large enough ?
12: **If** $n + t \geq DeadLine$ **Then**
13: $TemporaryDeadLine \leftarrow DeadLine$
14: $DeadLine \leftarrow DeadLine + (n + t - DeadLine)$
15: $ExpandMatrix(M, task \rightarrow proc)$
16: **End If**
17: **End If**
18: **For** $k = 0$ *to* $task \rightarrow proc$ **Do**
19: **For** $l = 0$ *to* $task \rightarrow t$ **Do**
20: **If** M[m+k, n+l] is allocated **Then**
21: // Back up DeadLine to last value
22: $DeadLine \leftarrow TemporaryDeadLine$
23: $goto : newTask$
24: **End If**
25: **End For**
26: **End For**
27: $AddTask(task, m, n)$
28: **End For**
29: **End For**
30: $label : newTask$
31: $task \leftarrow Pop(P)$
32: **End While**

Résumé

Titre : Mise en place d'une mécanique d'asservissement de puissance électrique au travers du gestionnaire de ressources d'un supercalculateur

Mots-clés : Gestionnaire de ressources, Consommation d'énergie, SLURM, Asservissement, Calculateur

Résumé : Les quantités d'énergie nécessaires au fonctionnement des calculateurs ne cessant d'augmenter, il devient déterminant de prendre en compte la puissance électrique dans l'ordonnancement des tâches.
Pour répondre à ce besoin, il a été proposé de modifier le gestionnaire de ressources SLURM, actuellement utilisé par une grande proportion des calculateurs référencés dans le top 500. Le but de ces modifications est d'intégrer dans SLURM une notion de puissance consommée et de s'en servir en tant que critère pour asservir l'utilisation globale du calculateur.
La solution proposée consiste à travailler en amont de l'algorithme d'ordonnancement du gestionnaire. Les nœuds de calculs sont bloqués avant de pouvoir être sélectionnés pour exécuter une tâche de calcul. Dans la première maquette logicielle, les nœuds bloqués pour l'économie d'énergie sont simplement mis au repos. La seconde version de cette maquette permettra d'éteindre complètement les nœuds de calcul, afin d'économiser l'énergie consommée lorsqu'ils sont au repos, ou d'appliquer un système de *dynamic voltage and frequency scaling* pour utiliser les nœuds à leur fréquence de fonctionnent optimale pour les travaux visés.
Les intérêts de la mise en place de cette technique sont l'adaptation automatique de la charge du calculateur en fonction des maintenances anticipées, des défaillances, des capacités du réseau électrique ou du coût de l'électricité en fonction des plages horaires.

Abstract

Title : Integration of an electrical control in the workload manager of a supercomputer.

Keywords : Workload manager, Energy consumption, SLURM, Control, Supercomputer.

Abstract : The energy quantity required by supercomputers is still increasing, taking into consideration the electrical power in scheduling algorithms in supercomputer is becoming a real requirement.
To overcome this problem, it has been proposed to modify the SLURM workload manager software, which is used by several supercomputers referenced in the Top 500. The aim of those modifications is to implement in SLURM a way to represent the electrical power used, and to use it to control the global power required by the supercomputer.
The idea in this project is not to create a new scheduling algorithm but to prevent compute nodes from being selected as computational resources. The first version of this control logic will rely on the idle state of the nodes. Then intermediate states will be added to this control and allow nodes to be considered as powered down to save a larger amount of energy. A dynamic voltage and frequency scaling mechanism will be used to use the optimal frequencies of the compute nodes.
This power control approach will allow cluster administrators to regulate the electrical power depending on failures, maintenances or electrical power transmission.