Efficient, non obstructive scheduling on computing grids using virtual execution environments

Grehant Xavier

Efficient, non-obstructive scheduling on computing grids using virtual execution environments

Allocation efficace et non-contraignante des ressources de grilles de calcul à l'aide d'environnements virtuels

by

Xavier Gréhant

Thesis
Submitted to Télécom ParisTech
for the degree of
Doctor of Philosophy

Supervisors: Isabelle Demeure and Sverre Jarp

Département Informatique et Réseaux
March 2010

Résumé en français

Cette thèse envisage l'allocation efficace et non contraignante de ressources de grilles de calcul à l'aide d'environnements virtuels. Elle a été encadreée par Isabelle Demeure, directrice de thèse et Sverre Jarp, encadrant au CERN, où une grande partie de ce travail a été réalisée.

Introduction à l'allocation de ressources en grilles

Grilles de calcul

Définissons tout d'abord ce qu'est l'allocation de ressources dans une grille de calcul et quel est l'intérêt d'une allocation efficace. D'après Ian Foster, une grille est *une infrastructure matérielle et logicielle qui fournit un accès fiable, cohérent, omniprésent et économique à des capacités de calcul haut de gamme* [Fos02].

Par exemple, l'objectif d'une des expériences du CERN, CMS, est de valider par l'observation une théorie physique qui prédit l'existence du particule, le boson de Higgs. Le boson de Higgs serait la représentation particulaire de la masse, de la même manière que le photon est la représentation particulaire de la lumière. Les scientifiques de l'expérience CMS disposent d'algorithmes pour analyser des données susceptibles de contenir des traces de l'existence de cette particule. Ces données proviennent du CERN, à Genève, où le Large Hadron Collider (LHC) accélère des particules et les fait entrer en collision au sein d'un détecteur appartenant à l'expérience CMS. Ces collisions produisent des nouvelles particules dont les trajectoires sont capturées par le détecteur de CMS. Les scientifiques de l'expérience CMS vont devoir placer leurs tâches pour les exécuter sur des serveurs appropriés, dans les centres de calcul du CERN ou dans d'autres centres de calcul qui stockent les données issues du détecteur CMS.

Il existe quatre détecteurs sur le LHC, chacun correspondant à une expérience

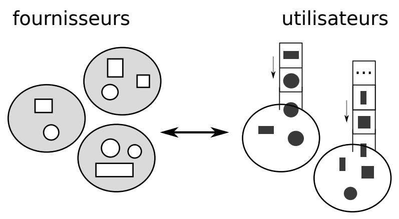


Figure 1: Fournisseurs et utilisateurs de ressources

similaire Chacun réunit une collaboration de 700 à 3000 scientifiques. D'une manière générale, on parle d'*utilisateur de ressources* d'une grille. Une grille réunit d'une part ses utilisateurs (dans notre exemple, les expériences de physique des particule) et d'autre part les *fournisseurs de ressource* : les centres de calcul qui hébergent les données et les serveurs qui servent à exécuter les tâches des utilisateurs. Ces centres de calcul, en plus du CERN, sont répartis sur trois continents.

Caractéristiques de l'allocation en grilles

L'allocation de ressources est l'action qui consiste à placer des tâches sur des serveurs appropriés. L'allocation de ressources en grilles de calcul a des caractéristiques particulières. En effet, les grilles de calcul ont été construites initialement pour ces applications d'analyse de données lourdes. Dans l'exemple précédent, les détecteurs du LHC produisent ensemble 15 peta-octets de données qui sont stockées sur des bandes magnétiques et répliquées sur des disques. Les tâches d'analyse de données ne communiquent pas entre elles et utilisent régulièrement les mêmes algorithmes sur des données différentes.

La difficulté propre à l'allocation de ressources en grilles vient du fait que les grilles réunissent des utilisateurs de ressources et des fournisseurs de ressources qui sont des entités autonomes, avec leurs propres objectifs. En particulier les serveurs sont sous le contrôle des fournisseurs, bien qu'ils exécutent les tâches des utilisateurs.

La figure 1 schématise des fournisseurs représentés par leurs centres de calcul, et des utilisateurs représentés par leurs tâches qui se renouvellent continuellement. Les serveurs et les tâches ont différentes formes pour montrer qu'ils vont plus ou moins bien ensemble.

Quel est l'intérêt d'optimiser l'allocation de ressources en grilles? Les applications du LHC vont tourner pendant 10 ans sur 40000 serveurs. Si on parvient à améliorer de 5% l'efficacité de l'allocation pour ces applications, la communauté de physique des particules gagne 6 mois, ce qui équivaut à 20000 serveur-an, soit 1 million d'euros d'électricité (10GWh) pour l'alimentation et le refroidissement des serveurs.

Contributions

Il s'agit d'abord d'identifier la métrique qu'on veut optimiser, puis de définir la politique d'allocation, c'est-à-dire le mécanismes et les responsabilités mis en jeu dans le processus d'allocation. Enfin, il s'agit de proposer une implémentation de ces mécanismes.

Sur le premier point, nous avons identifié une métrique, le *débit applicatif*, qui est intéressante à la fois pour les fournisseurs et pour les utilisateurs de ressources, et nous avons élaboré un modèle prédictif d'un paramètre déterminant du débit applicatif: la fréquence des défauts de cache. Pour prédire la fréquence des défauts de cache, nous avons proposé un algorithme plus rapide que les algorithmes de l'état de l'art sous l'hypothèsse que le cache est dédié à un processus donné. La rapidité de prédiction est importante pour pouvoir évaluer la fréquence des fautes de cache pour chaque couple (serveur, tâche) considéré. Dans un deuxième temps, nous nous sommes affranchi de l'hypothèse du cache dédié et nous avons proposé le premier algorithme de prédiction des fautes de cache en présence d'un cache partagé, c'est-à dire dans le cas habituel où un système d'exploitation ordonnance dans le temps des processus qui utilisent le même cache. Enfin nous observons que les prédictions réalisées par ces deux algorithmes encadrent la valeur exacte de la fréquence des fautes de cache.

Sur le deuxième point, nous avons d'abord observé que les grilles de calcul ont d'abord été construites pour une allocation centralisée des ressources. Cependant, une allocation décentralisée est de plus en plus utilisée de manière "cachée", c'est-à-dire sur un deuxième niveau, en détournant et sans informer les systèmes du premier niveau qui gèrent l'allocation de manière centralisée. C'est pourquoi nous avons proposé une nouvelle conception de l'architecture de grille qui vise à rendre l'allocation décentralisée "visible". Nous avons proposé un nouveau modèle de représentation des allocations de manière à pouvoir raisonner sans avoir à lister exaustivement toutes les tâches et tous les serveurs. C'est un modèle formel qui représente une allocation sous la forme d'une allocation booléenne. On peut ainsi faire des opérations sur les allocations. Ce modèle nous a servi pour définir une condition d'optimalité des allocations décentralisées.

La politique d'allocation que l'on décrit dans le deuxième point peut être réalisé

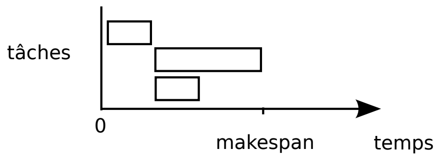


Figure 2: Makespan

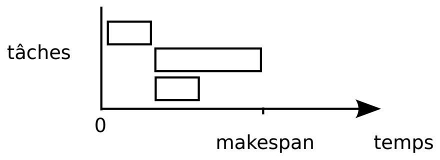


Figure 3: Somme des flôts

à l'aide de machines virtuelles à partir du moment où on dispose d'un outil pour les configurer et les déployer. La dernière partie décrit une implémentation de cet outil.

Proposition d'une métrique à optimiser et d'un modèle associé

Plusieurs objectifs, une métrique

Le débit applicatif est une métrique intéressante car elle détermine la plupart des objectifs des utilisateurs et des fournisseurs. Son expression est très simple, c'est le nombre d'instruction par seconde. Prenons des exemples d'objectifs. Du côté utilisateur, la minimisation du *makespan* et de la *somme des flôts pondérés* sont deux objectifs possibles.

– Le makespan est représenté en figure 2. C'est la date de terminaison de la dernière tâche moins la date de soumission de la première tâche. Le makespan est intéressant pour une application dont on obtient le résultat à la fin de l'exécution de la dernière tâche.
– La somme des flôts ets représentée en figure 3. C'est la somme pour chaque tâche de sa date de terminaison moins sa date de soumission. Dans l'exemple de la recherche du boson de Higgs, on ne sait pas a priori quel évennement physique issu d'une collision dans un détecteur du LHC permettra, lorsqu'on l'analysera de déceler la présence du boson de Higgs. Chaque tâche a une

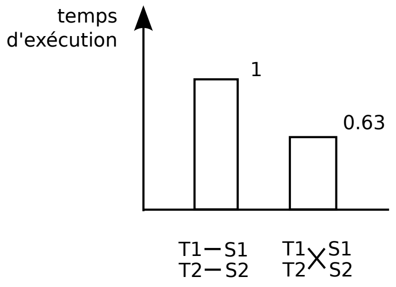


Figure 4: Effet de l'allocation sur le temps d'exécution d'après www.spec.org

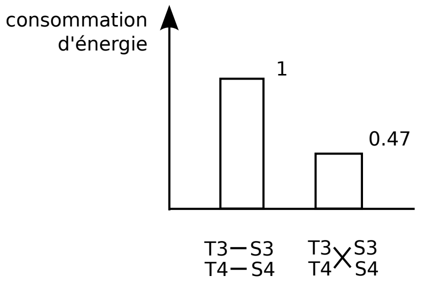


Figure 5: Effet de l'allocation sur la consommation d'énergie d'aprè [ST03]

importance indépendamment des autres, et c'est la somme des flôts qu'il faut optimiser pour qui veut être le premier à "voir" le boson de Higgs.

Les fournisseurs, de leur côté, veulent minimiser leur consommation d'énergie, et minimiser la gène occasionnée par l'exécution de tâches venant de la grille pour l'utilisation interne et l'administration de leurs ressources.

Le débit applicatif intervient dans l'expression de tous ces objectifs.

Nécessité d'un modèle prédictif

Quel est l'effet de l'allocation sur le débit applicatif?

Les figures 4 et 5 représentent des cas où on a pris deux tâches et deux serveurs et on a exécuté chaque tâche sur un serveur, puis on a interverti les combinaisons, et on a exécuté à nouveau chaque tâche sur l'autre serveur. En intervertissant le serveur associé à chaque tâche on a ainsi divisé environ par deux le makespan global dans une expérience et la consommation d'énergie globale dans une autre expérience. L'allocation de ressource a bien un effet sur le débit applicatif, qui se répercute sur les objectifs des participants d'une grille.

Pourquoi un modèle prédictif est-il nécessaire pour évaluer le débit applicatif?

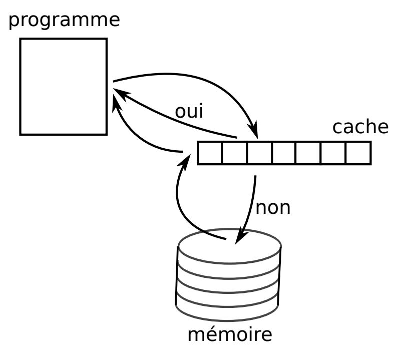


Figure 6: Accès au cache ou défaut de cache

Si on se place dans le cas où un unique centre de calcul fournit les ressources nécessaires à l'usage de ses utilisateurs. Ses serveurs sont connus et en nombre limité. On peut les regrouper en plusieurs groupes homogènes. De même pour les applications et les tâches à exécuter. Ainsi, pour prédire le débit applicatif d'une tâche sur un serveur d'avoir déjà enregistré au préalable le débit applicatif lors de l'exécution d'une tâche similaire sur un serveur du même type. C'est le principe des benchmarks. Sur une grille on ne connaît pas à l'avance toutes les tâches ni tous les serveurs. Au moment où on considère l'éventualité d'un placement d'une tâche sur un serveur il faut pouvoir prédire, à partir de certaines de leur caractéristiques, quel débit applicatif résultera de l'exécution de cette tâche sur ce serveur.

Dans l'absolu, il y a plusieurs éléments à prendre en compte pour prédire le débit applicatif.

– la capacité de la tâche à être parallélisée et la capacité du serveur à supporter ce parallélisme
– les communications entre processus
– les accès disque et en particulier les défauts de cache qui donnent lieu à des accès disque
– les accès mémoire générés par les défauts de cache.

Notre travail se limite à la modélisation des défauts de cache.

Modélisation des défauts de cache

La figure 6 montre un programme qui accède à une donnée. Si la donnée se trouve dans le cache, l'accès est rapide. Sinon, la donnée doit être accédée en mémoire, ce qui prend plus de temps. Une donnée peut se trouver dans le cache si elle y a été enregistrée par un accès précédent. Mais il se peut qu'une donnée

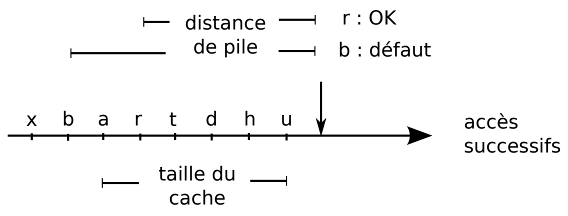


Figure 7: Distance de pile

soit effacée du cache. Cela se produit dans deux cas. Tout d'abord, si le cache est plein, l'enregistrement de l'accès à une nouvelle donnée efface une donnée du cache. Par ailleurs, les processus se succèdent dans des quanta de temps rapides sur le processeur. Entre deux quanta d'un processus, d'autres processus auront pu effacer du cache certaines de ses données. Dans tous les cas, l'accès à une donnée effacée du cache conduit à un défaut de cache.

Nous considérons successivement les deux causes d'effacement des données du cache pour prédire la fréquence des défauts de cache qui en résultent.

Variable d'intérêt

Nous partons de l'hypothèse que le cache est plein et qu'un seul processus utilise le cache. Dans ce cas un défaut de cache est du à l'accès à une donnée qui a été effacée au préalable.

On suppose que le cache est totalement associatif, et *LRU*, c'est-à-dire qu'il implémente la stratégie de remplacement *Least Recently Used*. Lorsqu'une donnée est effacée du cache, la donnée sortante est la donnée la plus anciennement utilisée. Cette hypothèse nous permet de réaliser une prédiction simple des défauts de cache. La figure 7 montre des accès successifs à des données symbolisées par des lettres. On y a représenté la taille du cache, d'une manière qui montre que le cache contient six données. La flêche verticale symbolise un nouvel accès. Deux cas se présentent. Soit, comme dans le cas de la donnée *b*, il y a eu plus de données accédées après le dernier accès à *b* que ne peut en contenir le cache. Dans ce cas *b* n'est plus dans le cache et on a un défaut de cache. Soit, comme dans le cas d'un accès à *r*, il y a eu moins de données différentes accédées depuis le dernier accès à *r* que de données dans le cache. Par conséquent aucune donnée accédée depuis le dernier accès à *r* n'a forcé à effacer *r*. La donnée *r* est toujours dans le cache, et son accès ne conduit pas à un défaut de cache.

La distance de pile est le nombre de données différentes accédées depuis l'accès

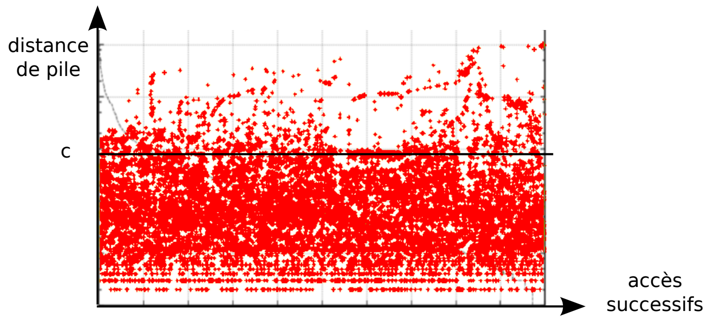


Figure 8: Accès d'un processus de Spec CPU 2006

à la même donnée. On devine sur cet exemple une propriété intéressante de la distance de pile. Si la distance de pile est supérieure à la taille du cache, on a un défaut de cache. Dans le cas contraire, il n'y a pas de défaut de cache.

Estimation par fitting

On propose d'effectuer une estimation par fitting de la loi de variable aléatoire qui décrit les observations de la distance de pile.

Sur la figure 8 on a tracé la distance de pile des accès successifs d'un processus de Spec CPU 2006, dans l'ordre chronologique. La ligne horizontale montre la taille du cache c en ordonnée. D'après les remarques précédentes, pour trouver la fréquence des défauts de cache il suffit de compter le nombre d'accès dont la distance de pile est au-dessus de cette ligne, et d'en prendre le ratio par rapport au nombre total d'accès.

Le protocole que nous proposons est le suivant:

1. Estimer la loi de la distance de pile Σ par fitting
2. Utiliser les paramètres de la loi obtenus pour calculer $\mathbb{P}(\Sigma > c)$

On pourrait très bien se limiter à cette analyse directe, mais on opte pour une analyse un peu différente, *biaisée*. Cela ne nous intéresse pas d'avoir un fitting précis pour les données basses de la distance de pile. On biaise l'estimation de manière itérative. On fait une première estimation. On calcule les moments de la loi. Ensuite on utilise cette première estimation pour générer de nouvelles valeurs, et on remplace les valeurs observées basses, celles qui ne nous intéressent pas, par les valeurs générées, et on reproduit l'estimation à partir de ce nouvel ensemble de valeurs. On montre qu'en répétant le processus l'estimation est meilleure pour les valeurs hautes de la distance de pile.

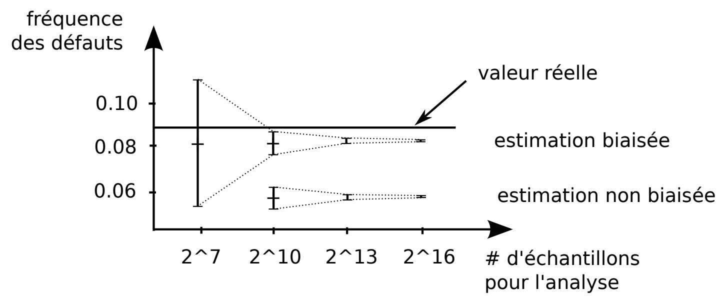


Figure 9: Qualité des prédictions

Précision, rapidité et intérêt

La figure 9 représente le résultat de plusieurs analyses et prédictions pour un benchmark de Spec CPU 2006. L'expérience a été faite sur la plupart des benchmarks de Spec CPU 2006 et celui-ci est représentatif.

Avec 2^7 échantillons, les prédictions obtenues varient beaucoup. Plus on utilise d'observations de la distance de pile, plus la variance des prédictions diminue et les prédictions convergent vers une valeur plus proche mais toujours différente de la valeur réelle. Cela signifie qu'on n'a pas réussi à représenter exactement les observations avec une loi simple de variable aléatoire. L'estimation biaisée est meilleure pour les valeurs hautes et moins bonne pour les valeurs basses. Comme ce sont les valeurs hautes qui nous intéressent (proches de la taille du cache), l'estimation biaisée nous permet d'être plus précis sur la prédiction des défauts de cache.

Cette méthode est efficace car la signature d'une tâche (les caractéristiques de la tâche que l'on transporte pour effectuer les prédictions) se réduit aux moments de la loi de probabilité de la distance de pile. Une distribution de Pareto généralisée par exemple, a seulement trois moments. Par ailleurs, la fréquence des défauts de cache est la probabilité que la distance de pile soit supérieure à la taille du cache. Ce calcul est immédiat car il s'agit de la fonction de répartition de la loi, directement calculable à partir de ses moments. C'est un calcul de complexité constante, tandis que les algorithmes à base d'histogrammes ont une complexité linéaire du nombre d'observations retenues.

On obtient un taux d'erreur qui est souvent proche de 1%. L'analyse est faite une fois pour un algorithme donné, et peut être réutilisée lorsque le même algorithms sert à analyser des données de mêmes structures.

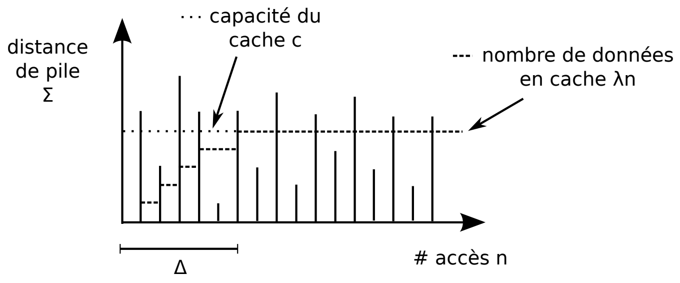


Figure 10: Accès au cours d'un quantum

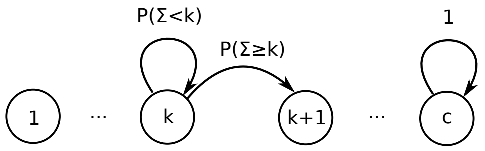


Figure 11: Nombre de données utiles dans le cache

"Cache thrashing"

Affranchissons nous de l'hypothèse selon laquelle le cache est réservé à l'usage d'un processus. Lorsque plusieurs processus se partagent le processeur, leur usage du processeur alterne selon des quanta de temps successifs. La figure 10 montre comment le cache se remplit lors d'un quantum de temps dans l'hypothèse où il ne contient aucune donnée utile au début. Chaque accès entraîne un défaut de cache et ajoute une donnée. Lorsque le cache est plein, on se retrouve dans le cas précédent.

Le nombre de défauts de cache à l'accès *n* du quantum est le nombre de données dans le cache, auquel on ajoute, si le cache est plein, la probabilité de défauts de caches dans l'état stationnaire multiplié par la durée de l'état stationnaire.

$$E[\lambda_n] + (n - E[\mathit{\Delta}|\mathit{\Delta} < n])\,\mathbb{P}(\mathit{\Delta} < n)\mathbb{P}(\mathit{\Sigma} \geq c)$$

Les différents éléments de cette équation peuvent être calculés si on considère que le nombre de données dans le cache suit un processus de Markov.

Modèle de Markov

A partir du moment où les obervations de la distance de pile sont *iid* (indépendantes, identiquement distribuées), le nombre de données du processus en cours présentes

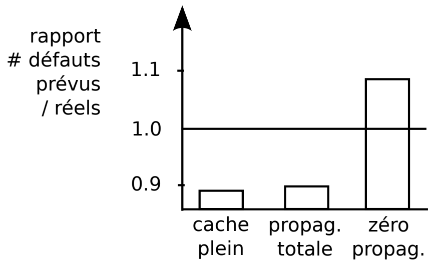


Figure 12: Qualité de prédiction sous les deux hypothèses

dans le cache est décrit par les états de la chaîne de Markov représentée en figure 11.

k est le nombre de données dans le cache. Si $k < \Sigma$, c'est qu'on accède à une donnée qui n'est pas encore dans le cache, donc on ajoute cette donnée. On passe ainsi à l'état $k + 1$ et on observe un défaut de cache.

La matrice de transition de cette chaîne de Markov est bidiagonale. On peut donc la diagonaliser et ainsi estimer directement les probabilités de "remplissage" du cache lors de tout accès, sans avoir à estimer ces probabilités pour tous les accès précédents. La complexité de prédiction de la fréquence des fautes de cache est donc quadratique de la taille du cache.

Encadrement de la valeur exacte

Nous avons proposé deux algorithmes. L'un suppose que le quantum de temps alloué au processus est infini et le cache est plein, l'autre suppose que le quantum est fini et que le cache se vide totalement entre deux quanta successifs du même processus. Ces deux hypothèses permettent d'encadrer la valeur exacte des défauts de cache. La figure 12 représente le cas d'un benchmark de Spec CPU 2006. On retrouve des valeurs similaires dans avec les autres benchmarks de Spec CPU 2006.

Défauts de cache et débit applicatif

Nous nous sommes restreints à la prédiction des défauts de cache, bien que la métrique qui nous intéresse pour le placement des tâche est le débit applicatif, et qu'il faille prendre en compte d'autres paramètres pour le prédire totalement. Une étude réalisée en 2009 évalue dans quelle mesure les défauts de cache ralentissent l'exécution [BMT09]. Dans le cas où on a un cache dédié, les techniques d'optimisation (prédiction de branche et préchargement des données) permettent

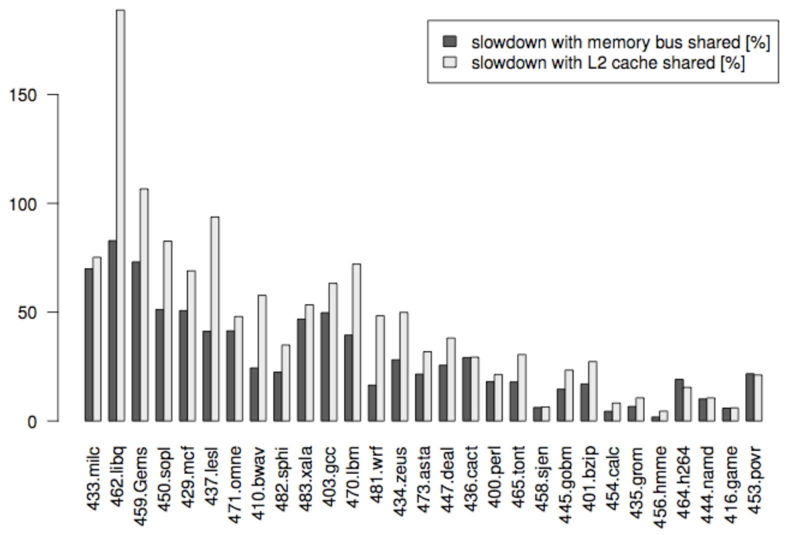


Figure 13: Effet du "cache thrashing" sur le débit applicatif - Tiré de [BMT09]

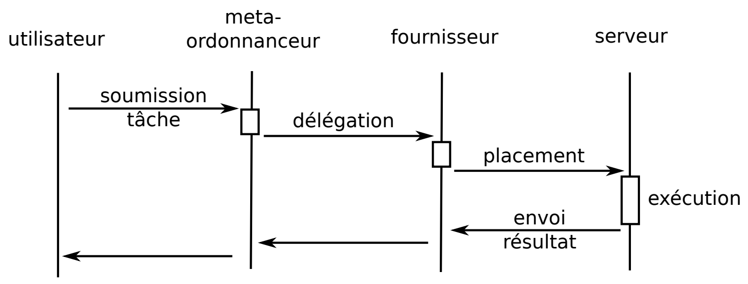


Figure 14: Allocation par le méta-ordonnanceur

de réduire à quasiment zéro l'impact des défauts de cache. Dans le cas que l'on a considéré pour la première fois dans cette thèse, où le cache est partagé entre plusieurs processus, l'impact peut être relativement fort: de l'ordre de 200% sur certains benchmarks.

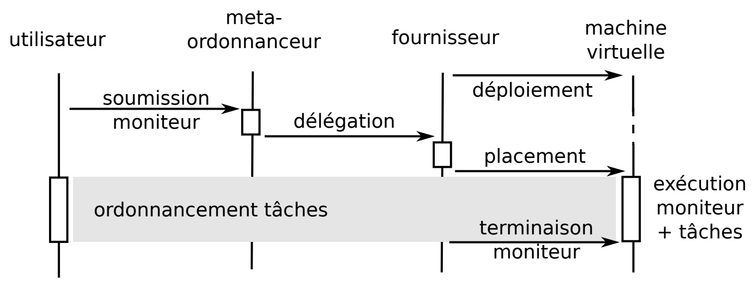


Figure 15: "Late Binding"

Proposition et caractérisation d'une politique d'allocation décentralisée

Limites de l'allocation centralisée

L'architecture actuelle des grilles de production est faite pour l'allocation centralisée. Or les fournisseurs sont les seuls à avoir un accès direct aux serveurs, ce qui pose un problème lorsqu'on veut qu'un *méta-ordonnanceur* organise l'allocation dans son ensemble. Le méta-ordonnanceur ne peut pas accéder aux serveurs, il peut seulement déléguer les tâches aux fournisseurs. De fait, le méta-ordonnanceur réalise un équilibrage de charge entre les fournisseurs.

La figure 14 illustre ce qui se produit lorsqu'un utilisateur soumet une tâche à une grille. La tâche est confiée au méta-ordonnanceur, qui lui même la déléguer à un fournisseur. Le fournisseur choisi prend la responsabilité de placer la tâche sur un de ses serveurs. La tâche s'exécute et lorsqu'elle est terminée, le résultat est renvoyé à l'utilisateur.

L'utilisateur n'a donc qu'à soumettre sa tâche. Il n'a aucun contrôle par la suite, jusqu'à ce qu'il reçoive le résultat si tout s'est bien passé. L'allocation n'est pas réalisée en vue des objectifs des utilisateurs et des fournisseurs.

Allocation décentralisée "dissimulée"

Les participants appliquent de plus en plus une nouvelle stratégie, que l'on retrouve implémentée dans de multiples projets indépendants.

Les fournisseurs déploient des machines virtuelles qui n'ont aucune différence avec des machines physiques du point de vue des utilisateurs et du méta-ordonnanceur. C'est sur ces machines virtuelles qu'ils vont placer les tâches qui leur sont confiées. Ils peuvent déplacer les machines virtuelles. Ils ont ainsi plus de liberté pour gérer

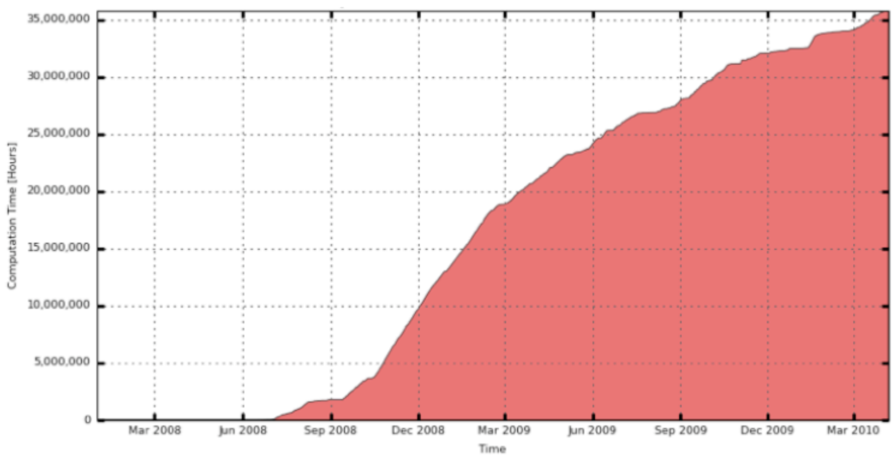


Figure 16: Allocation décentralisée par CMS sur les serveurs de Fermilab, Krista Larson, Mats Rynge *Condor Week* **2010**

leur consommation de ressources et attribuer leurs serveurs à un usage interne ou à des procédure de maintenance.

Une *machine virtuelle* est une couche logicielle basse qui prend le contrôle du matériel et émule une machine physique. Une machine virtuelle est contrôlable (elle peut être interrompue, relancée, déplacée) et isole des ressources.

Au lieu de lancer une tâche réelle les utilisateurs lancent un moniteur, une "fausse" tâche est placée sur un serveur ou une machine virtuelle comme le serait une vraie tâche selon le processus habituel d'allocation centralisée. Lorsque le moniteur commence son exécution, il initie une connection avec l'utilisateur. En effet, le moniteur "connaît" l'utilisateur et même si l'utilisateur ne peut pas initier une connection depuis l'extérieur du centre de calcul, le moniteur peut l'initier depuis l'intérieur. L'utilisateur peut ainsi ordonnancer ses propres tâches en les confiant à ses moniteurs, et utiliser les moniteurs pour contrôler leur exécution.

Importance de l'allocation décentralisée

La figure 16 donne un aperçu de l'évolution de l'allocation décentralisée sur la grille. Elle montre le temps de calcul pris par un utilisateur chez un fournisseur grâce à une allocation décentralisée. Le projet d'allocation décentralisé mené par CMS a été lancé en production en 2008. Il a permis d'allouer à CMS 25000 heures de calcul des serveurs de Fermilab, et ce sont autant d'heures pendant lesquelles les tâches réelles ont été dissimulés aux systèmes traditionnels de contrôle de la grille.

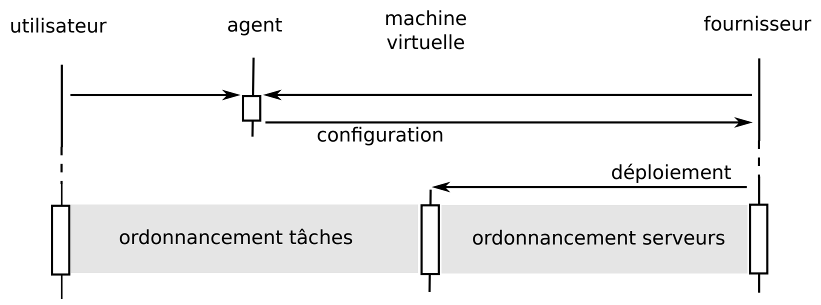


Figure 17: "Symmetric Mapping"

Allocation décentralisée "visible"

Nous proposons une architecture où le composant central a un rôle différent. Au lieu d'avoir la responsabilité de l'ensemble de l'allocation, il a pour but de mettre en contact utilisateurs et fournisseurs pour que ceux-ci s'accordent sur la mise à disposition et l'utilisation de machines virtuelles. Les fournisseurs obtiennent des configurations de machines virtuelles. Le déploiement de machines virtuelles se produit selon ces configurations. Les fournisseurs peuvent alors ordonnancer les machines virtuelles sur leurs serveurs, et les utilisateurs peuvent ordonnancer leurs tâches sur les machines virtuelles.

Cette architecture sépare les responsabilités des uns et des autres pour que chacun puisse optimiser son objectif. La question est donc: sous quelles conditions peut-on garantir que les objectifs peuvent bien être optimisés indépendamment?

Formalisation

Sous quelles conditions peut-on garantir que les objectifs peuvent bien être optimisés indépendamment? Pour répondre à cette question, nous proposons un modèle formel des allocations. Cette formalisation sert à raisonner sur les allocations de ressources sans avoir à les construire.

On représente une allocation par sa *fonction indicatrice*, c'est-à-dire l'application booléenne suivante:

$$a : Tasks \times Resources \times Time \longrightarrow \{true, false\}$$

qui renvoie $true$ si la tâche considérée est allouée à la ressource considérée à l'instant considéré. Cette représentation nous permet d'effectuer des opérations logiques et ainsi de raisonner sur les allocations. Par exemple on dispose d'opérations et et ou

sur les allocations.

Sans entrer dans les détails, on note l'opération que réalise un participant x (fournisseur ou utilisateur) comme une *projection* depuis l'espace des allocations vers un espace plus petit $\mathbf{p_x}(a)$. C'est l'action qu'il réalise en allouant soit ses tâches soit ses serveurs aux machines virtuelles.

Condition d'optimalité

On montre qu'il existe une condition suffisante pour qu'une allocation unique résulte des actions distinctes des participants et optimise chacun de leurs objectifs. Ce résultat s'écrit:

$$\arg\max \mathbf{value_x}, \ \forall x$$

Pour obtenir ce résultat, il suffit que les machines virtuelles déterminent la charge pour le fournisseur, et déterminent les ressources pour les utilisateurs. Cette condition s'écrit:

$$\mathbf{p_x}(a_1) = \mathbf{p_x}(a_2) \Rightarrow \mathbf{value_x}(a_1) = \mathbf{value_x}(a_2)$$

Cette condition signifie que chacun peut agir comme s'il était seul. Etant données deux allocations qui résultent de la même action d'un participant, la valeur perçue par ce participant est la même pour les deux allocations. En particulier, la valeur perçue est indépendante de l'action des autres participants, grâce aux contraintes qu'apportent les machines virtuelles.

Implémentation d'une architecture standard pour gérer le déploiement

Déploiement de machines virtuelles

L'architecture que nous avons présentée nécessite le déploiement automatique de machines virtuelles à partir de la donnée de leurs configurations. Nous avons implémenté un tel système.

Une machine virtuelle s'appuie sur une *image*, c'est-à-dire la zone du disque qui contient ses données et les partitions nécessaires au lancement du système d'exploitation et des programmes de la machine virtuelle. Pour lancer une machine virtuelle on utilise un moniteur de machine virtuelle. Notre implémentation repose sur Xen, un contrôleur open source. Pour obtenir une *instance* de machine virtuelle, c'est-à-dire une machine en fonctionnement, il faut d'abord installer l'image, puis

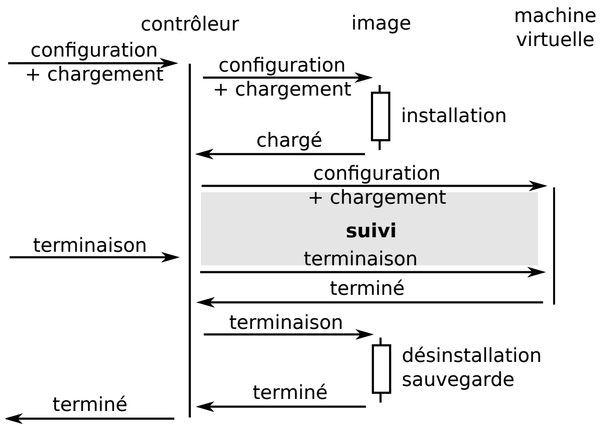


Figure 18: Déploiement d'instances Xen

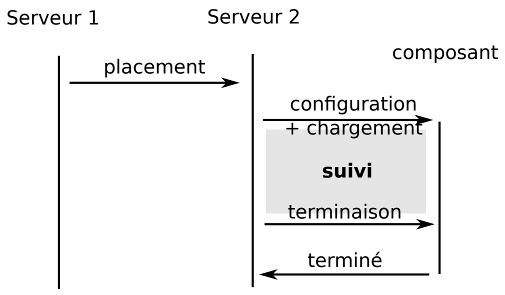


Figure 19: Orchestration par SmartFrog

commander le contrôleur. L'outil que nous avons développé effectue le chargement de l'image en lançant des opérations sur des *shells* de la machine hôte, et une fois que l'image est chargée, commande le contrôleur pour lancer l'instance, en effectuer le suivi, en détecter la terminaison ou le cas échéant, la terminer. Lors de la terminaison, il effectue les opérations nécessaires sur l'image pour soit stocker ses changements, soit les annuler.

Respect des configurations

Le déploiement est réalisé sur le domaine administratif d'un fournisseur de ressources. Cependant, les machines virtuelles doivent être configurées automatiquement, sans intervention du fournisseur, afin d'assurer le respect des configurations sur lesquelles l'utilisateur s'est accordé avec lui.

Pour cela, on utilise SmartFrog, un quadriciel développé par HP Labs. SmartFrog est un système de déploiement de composants distribués qui permet de placer, configurer, charger et orchestrer des composants distribués. Le système décrit en section est en fait implémenté avec des composants SmartFrog. SmartFrog fournit un language de configuration, qu'il interprète comme un arbre de composants où un composant père configure, charge, contrôle, termine ou détecte la terminaison des composants fils.

On interprète ainsi les configurations et on les transmets aux composants spécialisés qui sont capables d'intéragir avec la machine hôte et le contrl̂eur de machines virtuelles pour déployer des machines virtuelles dans le respect de ces configurations.

A titre de démonstrateur de sa validité, ce système a été utilisé au CERN pour le déploiement de machines virtuelles préconfigurées afin d'effectuer des tests distribués sur l'intergiciel de la grille du LHC.

Conclusion

Synthèse des contributions

Nous avons identifié le débit applicatif comme métriques d'intérêt pour chacun des participants d'une grille, en ce qu'il intervient dans l'expression des différentes fonctions objectif à optimiser. Nous avons défini un modèle prédictif d'un aspect déterminant pour le débit applicatif: les fautes de cache. Il s'agit d'une première étape réputée difficile dans le problème de la prédiction de performance. Pour cela nous avons proposé le premier algorithme dont la complexité est constante sans que cela nuise à la précision des résultats par rapport à l'état de l'art; et nous avons proposé le premier algorithme réaliste, c'est à dire qui prend en compte les effets des changements de contexte lors de l'alternance de processus. Nous avons évalués ces algorithmes sur les benchmarks de Spec CPU 2006. Ils sont efficaces à condition que l'analyse soit faite sur une portion d'exécution représentative de l'ensemble. Ils évitent en particulier la simulation de l'ensemble de l'exécution, ce dont l'utilité s'étend à d'autres domaines comme la conception de programmes et de processeurs.

Nous avons identifié le besoin de mise en place d'une allocation décentralisée préalablement à toute tentative d'optimisation. Nous avons réalisé une étude formelle pour donner précisément le bénéfice des machines virtuelles et leurs "bonnes propriétés". Le modèle utilisé, où l'on représente l'allocation par sa fonction indicatrice, est probablement applicable à d'autres raisonnements sur les allocations.

Travaux futurs

Il sera avantageux d'évaluer précisément l'impact de l'hypothèse "indépendantes, identiquement distribuées" sur les obsevations de la distance de pile pour la prédiction des fréquences de défauts de cache. On peut aussi espérer que se poursuive l'effort de modélisation de la performance des calculateurs, et qu'on obtienne un modèle relativement complet.

Dans le domaine de la politique d'allocation, il reste à définir précisément le fonctionnement d'un système de négociation de machines virtuelles, qui met en relation utilisateurs et fournisseurs de ressources. On pourra également s'intéresser aux modèles et à l'implémentation des objectifs des différents participants, que nous avons seulement survolés dans la thèse. Enfin, les machines virtuelles actuelles n'ont pas les caractéristiques précises de réservation de ressources qui permettent en toute rigueur de garantir l'optimalité de l'allocation dans un système décentralisé. De nouvelles implémentations de machines virtuelles peuvent être réalisées dans ce sens.

Summary

In the last decade, *computing grids* have brought together storage and servers located across multiple institutions to support large-scale scientific applications. By analogy with power grids, the original idea is to provide with seamless computing power anyone who plugs in. However, as applications increase in demand and multiply, the efficiency of the underlying resource allocation mechanisms deserves attention.

This thesis presents the following contributions.

- We identify resource allocation patterns in grids and we compare them to resource allocation patterns on single clusters.
- We identify a common pattern (*Late Binding*) in the way that several applications have recently bypassed the mainstream grid mechanism (*Metascheduling*) in order to take more control for better perceived performance.
- We propose a new pattern (*Symmetric Mapping*) that achieves separation of control between multiple participants in resource allocation.
- We propose a new formal model to specify resource allocation strategies. This model allows to represent dynamic scheduling under multiple constraints and objectives.
- We transpose the problem of Multiple Administrative Domains (MADs) from the area of fault tolerance to the area of distributed computing; we identify it as distinctive of grids among other computing systems; and we identify *Symmetric Mapping* as a solution.
- We propose an implementation of *Symmetric Mapping* based on virtual machines. As part of the implementation, we propose a system that deploys and manages multiple virtual machines based on declarative descriptions.
- We propose a system that detects service termination and resumes discontinued services on newly elected servers, in order to maintain an implementation of *Symmetric Mapping*, or any system that requires permanent services on transient servers.
- We propose a new method to analyze tasks and predict cache performance on various servers, in order to dynamically match tasks to adequate heterogeneous computing resources such as obtained on a grid. The method relies on fitting memory access patterns with well known probability distributions. The task signature is reduced to constant size and prediction is reduced to constant time.
- We propose the first evaluation of cache thrashing, in order to make realistic

performance predictions for time-shared CPUs. The analysis is based on a new Markov model of LRU caches. It yields a lower and a higher bound of the cache miss ratio in presence of competing processes.

Dans la dernière décennie, les grilles de calcul ont permis de réunir des ressources de stockage et de calcul de multiples institutions pour pourvoir à des applications scientifiques de grande ampleur. Par analogie aux grilles électriques, l'idée d'origine est de fournir de manière transparente de la capacité de calcul selon les besoins. Cependant, alors que les applications se multiplient, l'efficacité des mécanismes sous-jacents d'allocation de ressources mérite l'attention.

Cette thèse présente les contributions suivantes.

– *Identification des patterns d'allocation de ressource, et comment ils ont évolué depuis les clusters isolés jusqu'aux grilles qui s'étendent sur plusieurs institutions autonomes.*
– *Identification d'un pattern commun (*Late Binding*) dans la façon dont plusieurs applications contournent depuis peu le méccanisme habituel (*Meta-scheduling*) dans le but d'obtenir une mainmise accrue sur l'allocation de ressources et de palier à certains manques d'efficacité.*
– *Proposition d'un nouveau pattern (*Symmetric Mapping*) qui permet d'obtenir la séparation du contrôle entre les fournisseurs et utilisateurs de ressources.*
– *Proposition d'un nouveau modèle pour spécifier des stratégies d'allocation de ressource. Ce modèle permet de représenter l'allocation dynamique, ainsi que de multiples contraintes et objectifs.*
– *Transposition du problème des Domaines Administratifs Multiples (MADs) du domaine de la tolérance aux fautes à celui du calcul distribué. Identification du problème MAD comme problème distinctif des grilles parmi les systèmes de calcul distribué. Identification de* Symmetric Mapping *comme une solution.*
– *Proposition d'une implémentation de* Symmetric Mapping *basée sur les machines virtuelles, et dont l'un des éléments déploie et contrôle des multiples machines virtuelles à partir de descriptions déclaratives.*
– *Proposition d'un système qui détecte la terminaison d'un service et relance tout service interrompu sur un serveur nouvellement sélectionné, afin de maintenir une implémentation de* Symmetric Mapping, *ou tout système qui nécessite des services permanents sur des serveurs transitoires.*
– *Proposition d'une nouvelle méthode pour l'analyse des tâches et la prédiction de performance afin d'associer de manière dynamique des tâches aux serveurs adéquats. La méthode s'appuie sur l'estimation de patterns d'accès mémoire*

par des distributions de probabilité connues. La signature des tâches est réduite à une taille constante et la prédiction est effectuée en temps constant.

- *Proposition de la première évaluation du* cache thrashing, *afin de permettre des prédictions de performance réalistes pour les CPUs partagés par plusieurs processus. L'analyse est basée sur un nouveau modèle de Markov des caches LRU. Elle donne une borne supérieure et une borne inférieure de la proportion de fautes de caches en présence de processus concurrents.*

Acknowledgements

My deepest thanks to Isabelle Demeure for being my mentor during these years; to Sverre Jarp for supervising me at CERN and for helping me to take initiatives and to take the best of this fascinating environment; to Peter Toft for his unswerving support from HP Labs; to the three of you for your patient and discerning guidance.

Special thanks to Michel Benard for having conceived this project and made it happen. Many thanks to all those at HP, including Arnaud Pierson, CERN and Telecom ParisTech, who set up and managed the project. In addition, I wish to thank François Fluckiger for his friendly interest and support.

I wish to thank the committee members, Pierre Manneback (Université de Mons), Yves Robert (ENS de Lyon), Brigitte Plateau and Jean-Marc Vincent (LIG) for accepting to assess this thesis; and also, Yves Robert and Jean-Marc Vincent, for having absorbed with me in fruitful in-depth examinations.

This work was influenced by many. The comments of Steven Hand (Cambridge Univ.) and Walfredo Cirne (Google) gave it a new orientation. Jeffrey Arnold (Intel), Alan Suleyman Ustunel and Olivier Hudry (Telecom ParisTech) gave me critics and directions. Philippe Defert (CERN), and Max Boehm (EDS), helped me with their friendly attention. Predrag Buncic, Sanjay Padhi, Jakub Moscicki and other CERN physicists and computer scientists shared with me their vision.

Many thanks to Serena Cameirano, Salman Toor and Olivier Pernet, who worked with me during their summer studentships at CERN openlab. Olivier contributed to SmartDomains, a development detailed in the thesis.

This work wouldn't be the same without my colleagues of the openlab team at CERN, and the SmartFrog and Tycoon teams at HP Labs with whom we had a privileged correspondence. My friends and colleagues gave me invaluable support. My office-mates over the years, Ryszard Jurga, Havard Bjerke and Sebastien Gardoll, deserve special credits.

Many thanks to my business partners, Stanislas Giraudet de Boudemange and Edouard Chabrol-Hubert for their confidence and support. Together we are building netsas, a startup company with a great product and a wonderful team.

Finally, I wholeheartedly thank my fiancée JooHee, my parents Agnès and Michel, my brother Olivier and his wife Alexandra, and my little sister Hélène for their patience and their love.

Contents

Introduction

CERN experiments in high energy physics attract scientists from around the world to analyze data produced by particle accelerators. In 2001, the DataGrid project began in order for physicists to execute analysis close to the data, at CERN or other computer centers that host copies. In the same year, the TeraGrid project started in the United States to support particle physics experiments at Fermilab. Progressively, more infrastructures appeared, with national or regional initiatives, to offer computational power to scientific projects of all kinds.

Today, grids allow scientists to submit their jobs and have them routed to computer centers that support their experiment and host the required data, in order to run on servers that exhibit the required configuration.

In 2009, the Large Hadron Collider (LHC), CERN's most recent particle accelerator, emitted its first beams. In 2012, it will operate at full capacity and produce 15 petabytes of data a year. The load on servers is going to increase and performance is going to be an issue. The analysis is planned for the next 10 years on 25,000 servers. A one percent performance increase will save 2,500 server-years on the LHC analysis alone.

Grid participants are resource users and resource providers. Users are scientists federated in collaborations, also known as *virtual organizations* (VOs). Members of a VO work together on the same research. Providers are institutions that support specific VOs by giving them access to their computing centers, also known as *grid sites*.

The following difficulties affect performance specifically in computing grids.

– Computing grids span multiple administrative domains. Participating institutions implement on their grid sites their own server configuration and local allocation policies.
– The objectives of independent participants differ. The difference is par-

ticularly remarkable between resource users, who want quick results, and resource providers, who want limited costs.

– VOs come with their own software and workloads. Grid sites come with their own platforms. Efficient scheduling requires to predict job execution time or carry out dynamic re-scheduling, which is difficult in these conditions.

We address the following question: *How can resource allocation on grids be efficient for resource users without being obstructive to resource providers?*

The first step in the proposed approach consists in using a new software architecture to carry out resource allocation while separating the concerns of the different participants. In this aim, we create a new resource allocation model that allows to consider dynamic scheduling under multiple objectives. Using this model, we propose an architectural pattern that allows for the separation of concerns. We present a possible implementation that consists in using virtual machines to decouple the environments where jobs execute from the backing hardware. This decoupling helps isolate participants and eliminate conflicts in the accomplishment of their objectives. We implement two elements of the architecture: a system that maintains virtual machines in correspondence to declarative descriptions; and a system that maintains permanent services on transient environments.

In the second step, we address the matching of different jobs with different resources, with limited information on both sides. In this aim, we contribute to performance prediction of memory caches. Caches and memory access patterns are responsible for performance variations that are difficult to anticipate. A stochastic analysis of these patterns gives two results. First, it simplifies the prevailing predictive approximation of cache misses frequency. Second, it reveals that this approximation is in reality a lower bound, and it produces a method to calculate a higher bound. This is the first method to predict cache performance on actual operating systems with multiple processes.

The thesis is composed with an introduction, 6 chapters decomposed in two parts, and a conclusion. The first part deals with software architecture and the second part deals with performance prediction. Figure 20 illustrates the continuity between the elements of the thesis.

– Chapter 1 is a survey of task mapping on production grids. Based at CERN openlab, we could examine the systems in place and interview their creators and users. In 2008, the main outcome of the survey's original publication was to identify the same pattern in multiple emerging

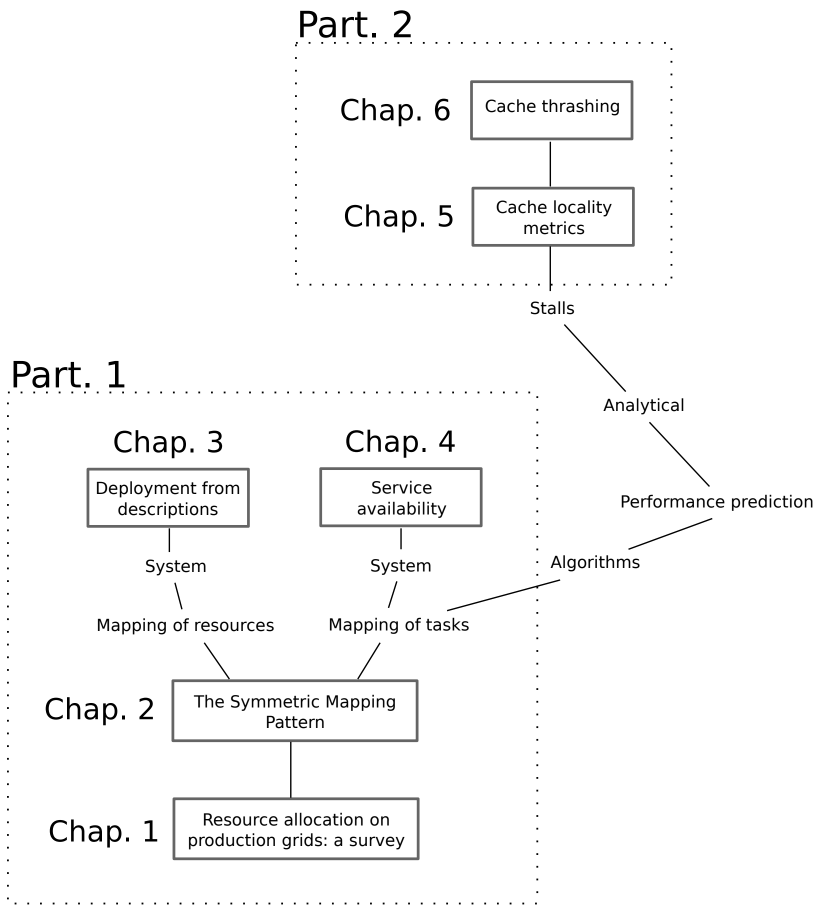


Figure 20: Thesis chapters.

projects and point it as a solution for Virtual Organizations (VOs) to take better value from their alloted grid resources. Today this pattern is known as Late Binding in grids, and is continually gaining wider adoption among large VOs. Another outcome of the survey is to identify the fact that grids span multiple administrative domains as the main difficulty for performance, and that in order to improve performance in grids, the separation of concerns between participants has to be addressed early in the architectural design.

– Chapter 2 proposes a new architectural pattern based on an analogy with MAD systems (Multiple Administrative Domains), a formalism borrowed from the study of fault tolerant systems. In order to formalize the MAD problem in grids, we introduce a new model for resource allocation that allows to consider dynamic scheduling and multiple objectives. The problem is solved with a form of containment that isolates and decouples the objectives of the participants. The solution yields a definition of the required isolation properties. We propose an implementation of the pattern, where virtual machines are inserted to decouple task placement by users from resource placement by providers.

– Chapter 3 details an element of the implementation that requires a proof of concept. It presents a mechanism that configures, deploys and maintains virtual execution environments from declarative resources descriptions. Virtual machines serve as isolated execution environments for users to freely access their alloted resources and implement their own schedulers.
– However, the distributed components of their schedulers need to be placed on the same execution environments. Chapter 4 describes the implementation of a mechanism that guarantees the continuity of user services on execution environments alloted for limited durations. The termination of a service triggers a node election and service re-deployment.
– Once collaborations of users are free to schedule their own tasks, they need to identify efficient mappings. We propose a method to evaluate the affinity between tasks and processors in terms of cache performance. The locality of memory accesses is analyzed in chapter 5 for fast cache performance prediction. It gives a method for users to sample data access patterns once for all on a binary, and lets online schedulers evaluate their affinity with processor caches. Task analysis requires the observation of a very small subset of memory accesses and is stored in a few variables. It consists in fitting a relevant memory access pattern with known statistical distributions. Scheduler prediction is reasonable to very accurate with constant complexity, both computational and in memory, while other methods are based on histograms and exhibit higher than linear complexity.
– Chapter 6 refines the analysis with a Markov model that describes how the cache fills up with data. This model allows, for the first time, to predict a higher bound of the miss frequency, taking into account cache thrashing, i.e. the adverse effect of concurrent processes and context switches on performance. Cache thrashing is of utmost importance because no single process is ever isolated on common operating systems. Besides, we identify that the "no thrashing" assumption covered in the literature and in

chapter 5 gives a lower bound of the actual rate. Therefore, the maximum prediction error is known.

Part 1.
Software Architecture

Chapter 1

A Survey of Task Mapping on Production Grids

1.1 Introduction

Grids have been in production for more than a decade. The observation of their emergence and evolution reveals actual constraints and successful approaches to task mapping across administrative boundaries. Beyond differences in distributions, services, protocols and standards, a common architecture is outlined. Application-agnostic infrastructures built for resource registration, identification and access control dispatch delegation to grid sites. Efficient task mapping is managed by large, autonomous applications or collaborations that temporarily infiltrate resources for their own benefits.

1.1.1 Scope

The focus is on task mapping in production grids. The historical perspective aims to help identify fundamental constraints behind grid resource allocation and the logic of the evolution. The survey is written as a bibliographical basis for further research towards efficiency in grids.

In this section, we define important concepts and specify the limits of our study.

Definition 1 (Grid [FKT01])**.** ***Grids*** *coordinate resource sharing and problem solving in dynamic, multi-institutional virtual organizations.*

Originally, *Virtual organizations* federate individuals to participate in grids.

> Virtual Organizations enable disparate groups of organizations and/or individuals to share resources in a controlled fashion, so that members may collaborate to achieve a shared goal [FKT01].

In practice, resource users are distinguished from resource providers.

Definition 2 (Virtual Organization)**.** *A **Virtual Organization (VO)** is a collaboration of individual users, perceived from the outside as a single user because they identify as such, who run the same applications and have common objectives.*

Definition 3 (Grid site)**.** *A **grid site** is a set of grid nodes under a single administrative domain.*

An institution or organization controls the management of its own sites, including configuration, access control, resource allocation mechanisms and policies.

Definition 4 (Production grid)**.** *A **production grid** is a solution implemented and used for the execution of large scale applications from independent collaborations (VOs) on resources that span independent institutions (grid sites).*

Definition 5 (Task mapping [KSS+07])**.** *An important research problem is how to assign resources to tasks (match) and order the execution of tasks on the resources (schedule) to maximize some performance criterion of an heterogeneous computing system. This procedure of matching and scheduling is called **mapping** or **resource allocation**.*

The mechanisms developed for resource allocation include dynamic scheduling and configuration, market simulations for utility and feedback control for performance [MDP+00, ROLV06]. This survey determines which ones are used in production. The analysis of production grids designs, their emergence and evolution helps apprehend the fundamental grid constraints that really surface in production.

The thesis is of limited relevance to connected areas:

Clusters The term *grid* is often misused to designate clusters inside a single organization. A cluster is a set of connected computers inside the same administrative domains. Definition 1 eliminates the confusion.

Testbeds Since the focus in on task mapping, the analysis concentrates on *computational grids*. It does not directly take into consideration testbeds such as PlanetLab which are not meant to run computations but to experiment the deployment of networked services [Fiu06].

Prototypes The goal is to capture constraints that govern grids, in order for the reader to build insight on the relevance of research hypotheses in this area. Therefore, the analysis does not cover grid prototypes, i.e. solutions proposed as part of research *on* grids, and not used in production.

Desktop grids A desktop grid (such as SETI@home or any system based on BOINC) is the distribution of computations from a single application to personal computers [And03]. Task mapping faces less constraints on a desktop grid than on a grid with many users and applications, and where resource providers are entire institutions. However, the reader familiar with desktop grids may understand that some elements of this analysis also apply to their context.

Enterprise clouds Companies such as Amazon Web Services sell the access to computing resources. Every paying user obtains dedicated and isolated virtual storage devices and processors, which she manages just like her own cluster. In grids, the responsibilities of resource users and providers are not so well distinguished.

1.1.2 Outline

The remainder of this chapter is organized as follows. Section 1.2 describes the aggregation of multiple sites into a grid. Distribution of control introduces a number of challenges: security, user identification, information flow, seamless resource integration, central monitoring, etc. Among them, strangely, resource allocation is a side concern and remains constrained. Section 1.3 identifies constraints on allocation and their impact on performance. To regain performance, major users centralize control on temporarily accessed resources. Section 1.4 presents this trend and characterizes the resulting environments on which efficient allocation can take place.

1.2 Federating resources

Grid infrastructures take on a distinctive challenge. They aggregate resources from different institutions to run multiple applications from multiple users. Section 1.2.1 presents major grids, their applications and their participants, and section 2.3 presents the motivation of their federation in grids. They share workload constraints (section 1.2.3) and parallelization methods (section 1.2.4). The initial idea was to replicate the structure of cluster management systems (section 1.2.5) to face the same problems in the wide area 1.2.6.

1.2.1 Infrastructures

Grids entered production around year 2000 to support scientific applications. In scientific grids, academia or other public-funded institutions voluntarily offer a certain amount of their computing resources to external scientific projects.

A few grid infrastructures scale to tens of thousands of nodes. They aggregate computing, data warehousing and networking facilities from several institutions. They distribute middleware to operate resources and manage users and applications. Grid specific components are often open-source academic projects.

The following is a non exhaustive list of grid infrastructures.

LCG is the Computing Grid for the Large Hadron Collider, CERN particle accelerator in Switzerland. It aggregates sites mainly in Europe, but also in Taiwan and Korea. It groups together over 41,000 CPUs from 240 sites [And04]. The LCG is supported by the European Commission and more than 90 organizations from over 30 countries. Participating regions identify their contributions separately like **GridPP** in the UK [tGC06]. LCG resources also support applications from different scientific domains. It forms a general-purpose European grid. **EGEE** (Enabling Grids for E-sciences) carries out its coordination at CERN from 2003 to 2009 and **EGI** (European Grid Initiatives), a federation of national projects, from 2010.

OSG The *Open Science Grid* started in 2005. It is funded by U.S. LHC software and computing programs, the National Science Foundation (NSF), and the U.S. Department of Energy. It continued **Grid3**, started in 2003 [Ave07].

TeraGrid started in 2001 with funds from the NSF to establish a Distributed Terascale Facility (**DTF**). It includes collaboration from 9 major national computer centers in the U.S. It provides 250 teraflops of computing capacity and plans to integrate a petaflop system [Pen02].

NorduGrid was funded in 2001 by the NORDUNet2 program to build a grid for countries in northern Europe. The NORDUNet2 program aimed to respond to the "American challenge" of the Next Generation Initiative (NGI) and Internet2 (I2). NorduGrid provides around 5,000 CPUs over 50 sites [EGK$^+$07].

Naregi The Japanese grid project *National Research Grid Initiative* started in 2003. It is deployed in beta on a 3,000 CPUs testbed and targets the PetaFLOPS in 2010 on national computer centers. The software development is done by private companies (Fujitsu, NEC, Hitachi, NTT). It is funded by the Ministry of Education, Culture, Sports, Science and Technology (MEXT) [Miu06].

Each of these grids is distinctive by the hardware resources integrated, by the organizations supplying these resources, by the projects supported, and by the middleware.

These infrastructures must not be confused with software development projects like **Gridbus** [1], **Globus** [2] and **VDT** [3] which distribute consistent sets of grid middleware components and are active in the standardization effort [FKNT02]. Grid infrastructures use and re-distribute some of these components [ABK$^+$04].

1.2.2 Objectives

Participants in grids usually have one of the following intents. They either want to *consolidate CPU cycles* or to *avoid data movement*.

Consolidated resources from multiple institutions make it possible to solve problems of common interest where the efforts of a single institution would require unreasonable execution time. The resolution of NUG30, a quadratic assignment problem, illustrates the use of a grid for **High Performance Computing** (HPC) [ABGL00, GLY00]. The intent in HPC is to maximize the execution speed of each application. However, supercomputers are better suited for HPC in general. They offer a lower latency between computing units and schedule processes at a lower level.

By contrast with HPC, **High Throughput Computing** (HTC) systems intend to maximize the sustained, cumulative amount of computation executed. A grid consolidates resource allocation from multiple institutions and accepts applications from external collaborations. Its overall throughput is potentially higher than the sum of the throughputs of participating institutions acting separately. However, an improvement is realized only if allocation is sufficiently

1. gridbus.org
2. globus.org
3. Virtual Data Toolkit: vdt.cs.wisc.edu

accurate and responsive.

Grids have been prominently driven by the will to analyze unprecedented amounts of data. Experiments at CERN, the European Center for Nuclear Research, and Fermilab, an American proton-antiproton collider, attract collaborations of particle physicists who account for most users of LCG, OSG, TeraGrid and NorduGrid [Ter02, GCC$^+$04]. They search for interesting events in the vast amount of data generated by detectors. The dozens of petabytes of data generated by the Large Hadron Collider (LHC) are totally replicated on a few primary sites and partially on secondary sites to avoid further data movement. Grids are interesting for **distributed data analysis** when it is beneficial to run computations on the computer centers that store the data instead of transferring data to every single user. However, the gain depends on the capability to appropriately distribute data in the first place and to map tasks according to data location.

1.2.3 Applicable workloads

In order to characterize the candidate applications to be ran on a grid, we introduce a few definitions.

Definition 6 (Divisible [BDM99])**.** *A **divisible** task is a computation which can be divided with arbitrary granularity into independent parts solved in parallel by distributed computers.*

Definition 7 (Embarrassingly parallel [Har03])**.** *A problem of size N is **embarrassingly parallel** if it is 'quite easy' to achieve a computational speedup of N without any interprocess communication.*

Definition 8 (Partially data parallel [Har03])**.** *A **partially data parallel** problem divides the input data into a number of completely independent parts. The same computation is undertaken on each part. It may require pre and post processing and redundant computations to avoid communication.*

Divisible, embarrassingly parallel applications are most relevant to grids because they can be parallelized with appropriate granularity and do not generate communication overhead [GMP07, LSV06]. More precisely, partially data parallel applications are the most frequent. For instance, a sky map is decomposed into regions of arbitrary sizes for parallel analysis [AZV$^+$02]. The output of a particle physics detector is decomposed into sets of *events*, i.e. particles created at the collision point.

1.2.4 Job: the element of an application

Definition 9 (Job [SAB+05])**.** *A* ***computational job*** *is a uniquely identifiable task, or a number of tasks running under a workflow system, which may involve the execution of one or more processes or computer programs.*

In practice, *Resource allocation* or *task mapping* (def 5) consists in placing jobs onto computers. VO members can prepare jobs manually. As an alternative, VOs *port* their applications to the grid with application-specific frameworks that spawn jobs [Mac04]. A job is presented to the grid with instructions and information necessary for its execution.

This information includes requirements that constrain the allocation in order to place the job on appropriate resources. In practice, a job is typically placed on a general purpose desktop computer or batch server. Requirements specify the operating system flavor, application software and data. No substantial data is transfered along with a job. However, if data is not present on the node, a job placement triggers a transfer from a remote storage.

1.2.5 On single administrative domains

This section looks at local resource management systems. Such systems manage grid sites internal resources. Grids initially attempted to replicate their principles.

Pure batch systems fundamentally differ from systems designed with interactive execution in mind. The former *submit jobs* while the latter *connect to resources*.

Batch job submission

Batch schedulers use mature techniques to essentially manipulate job queues [CCF+94]. They dispatch jobs to balance the load between execution nodes and order them according to precedence policies. In the following we lists some of the most popular batch scheduler systems.

LSF *Load Sharing Facility* started with Utopia, whose authors created Platform Computing, a company with now 360 employees [ZZWD93].

PBS *Portable Batch System* is the flagship product of Altair, a 1200 employees corporation [Hum06]. It was originally developed at NASA since 1993 [Hum06]. PBS proposes different scheduling policies, implemented in two

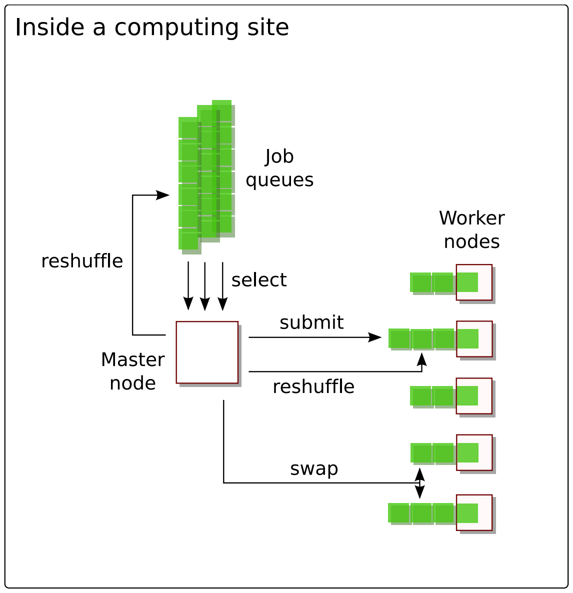


Figure 1.1: Generic batch system

different systems, *Torque* and *Maui*, and a language to configure them [BHK+00].

SGE DQS (Distributed Queuing System) started at Florida State University in 1993. In 2000, Sun acquired all rights and renamed it *Sun Grid Engine*.

Figure 1.1 shows the standard features of a generic batch system. A master assigns jobs to queues depending on priorities and expected execution times. When actual execution times are very different from expectations, the master reorders jobs within and between queues. After a job started to execute, the master can still checkpoint and migrate it if the job or the operating system provides the capability.

Connection to resources

Condor is a particular resource management system. Widely used for resource allocation in clusters, its principles contrast with the other cluster management systems. Condor does not *push* jobs to nodes. Instead, it connect users and resources.

Condor started in the 1980's with the intent to select idle CPUs for use by active users.

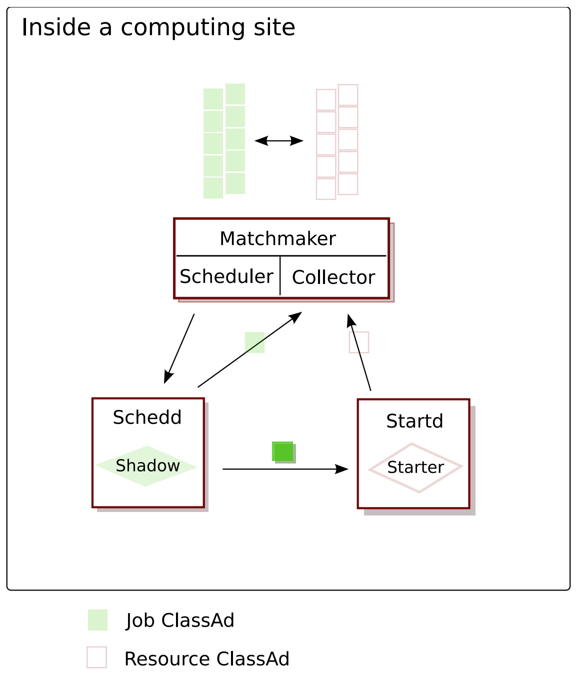


Figure 1.2: Condor

Figure 1.2 shows how this is done. When some resource is needed, a **ClassAd**[4] is emitted. On the other hand, when a CPU is idle, it also emits a ClassAd. A **Collector** collects ClassAds for the **Matchmaker** to find best matches. The **Schedd**, for Scheduler Deamon, is the process that sends the user ClassAd. The **Startd**, for Start Deamon, sends the resource ClassAd. When a match is decided, the **Scheduler** notifies the Schedd with the addresss of the Startd. The Schedd and the Startd spawn two processes responsible for the communication [RLS98].

Some grid sites use Condor as an alternative to batch schedulers. Metaschedulers use Condor matchmaking to match job requirements with grid sites capabilities [TTL02].

1.2.6 Allocation concerns

In general-purpose distributed computing, resource allocation includes the management of resource reservation, co-allocation, inter-process communica-

4. from *Classified Advertisement*

tion, priorities, dependencies, interactive jobs and process migration [CK88]. Grids must re-invent new ways to address these concerns.

Resource reservation Given that a central resource management system cannot pervade grid sites, reserving resources from the grid eventually means reserving resources from a grid site. Negotiation and reservation protocols exist [CFK+02]. However, the general consensus is that, in order to avoid intrusion, resource providers only offer *best effort*, i.e. no guarantees. Guarantees with limited intrusion to resource owners would require dynamic re-allocation inside and across grid sites [CGR+06].

Co-allocation Job co-allocation, i.e. the co-allocation of several jobs to the same node in order to improve resource utilization, is subject to grid sites policies. Resource co-allocation, i.e. the synchronous allocation of several nodes to a set of jobs would require reservation mechanisms [CFK99].

Inter-Process Communication (IPC) IPC occurs if jobs communicate at runtime. This would require resource simultaneous allocation, and a communication path with low latency between execution nodes.

Priorities Institutions prioritize the projects they support. They regulate priorities at the level of their local schedulers and by declining job submissions based on the issuer's identity.

Dependencies Some tasks use the result of others. Task dependency graphs can be specified for bulk submission on single administrative domains. To execute dependent jobs on multiple sites, users must wait for a job completion before submitting a new job that requires its result [DSS+05].

Interactive jobs Some tasks require inputs from the application or the user to proceed. Condor supports this kind of communication. However, grid sites restrict connexions initiated from the outside. In addition, users cannot in general reserve resources or predict execution windows. Therefore, the job must initiate all communications.

Process migration Condor, LSF or a grid site operator can migrate running jobs inside a single site if the operating system or the job provides the capability [MDP+00]. Production grids do not support migration by users because it would be an intrusion to the resource provider. They do not support migration across grid sites because not all sites support migration. In general, if a resource is preempted, jobs are simply discarded.

All these concerns are solved on local clusters. On grids, additional constraints make their resolution more complex: *latency between sites is inherently high;*

access to data is not uniform; the development of the infrastructure is separated from the development of applications; and grid sites are independent administrative domains. Each of these four constraints would require proper investigation. The following section shows the particular importance of the grid sites administrative independence.

1.3 Disruptions to resource allocation

This section explains how the independence of resource providers determined a *de facto* choice for a resource allocation model. Despite ideals of ubiquitous resource presence inspired by power grids, it appears that computing grids emerged with only basic batch allocation capabilities. They revive at a different scale the history of computer systems [Cer94].

Before addressing resource allocation, grids are shaped by other concerns that result from the independence of grid sites: the need for a site to screen users, apply its own policies and integrate its own systems and resources. These determinants suffice to draw a simple picture of the level of freedom left to resource allocation.

Section 1.3.1 shows how the independence of grid sites forces to delegate job management. Delegation breaks the continuity of job control and its connection to the user (section 1.3.2). In addition, it introduces unpredictable delays (section 1.3.3). To ensure interoperability of all grid sites, only remains the minimal control supported by all local management systems (section 1.3.4).

1.3.1 Delegation

Global resource allocation is disrupted by grid sites autonomy.

In spite of their different nature, production grids reproduced the central control designed for single administrative domains. Under a single administrative domain a central scheduler assigns each job to a node (see section 1.2.5). Similarly, in a grid, a central *meta-scheduler* or *broker* assigns each job to a grid site [CFK04].

Definition 10 (Broker [ABK$^+$04])**.** A grid resource **broker** is a service with which *the end users interact* and *that performs resource discovery, scheduling, and the processing of application jobs on the distributed Grid resources.*

The authority of the meta-scheduler ends at the boundaries of independent

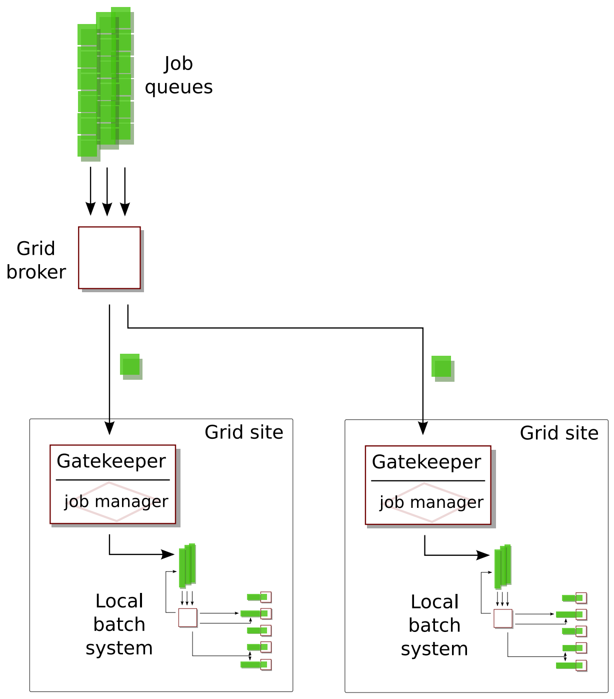


Figure 1.3: Delegation of job submission to Globus Resource Allocation Manager (GRAM)

resource providers. Since interference is proscribed, resource aggregation is reduced to a simple superposition of grid sites.

On a grid site a *gatekeeper* screens jobs and their owners identity.

Definition 11 (Gatekeeper [Gra02])**.** *The **gatekeeper** is a process that exists before any request is submitted. When the gatekeeper receives an allocation request from a client, it mutually authenticates with the client, maps the requester to a local user, starts a job manager on the local host as the local user, and passes the allocation arguments to the newly created job manager.*

Grid sites screen jobs because they only accept jobs submitted by trusted users, and provide differentiated *service levels* to different users. Job allocation is simply delegated by the broker to a grid site via its site gatekeeper because this is the simplest way to let grid sites screen jobs and enforce their own access policies.

In most infrastructures, authentication and delegation is processed by *Globus Resource Allocation Manager* (**GRAM**, fig 1.3) [Fos06]. Submitted jobs queue at the **grid broker**. The broker never submits to a worker node directly. Instead it finds an appropriate site and submits a job to its **gatekeeper**, along with a certificate to identify the job owner. If the gatekeeper refuses the job, it sends it back to the broker. If the broker accepts it, it starts a **job manager** process. The job manager submits the job to the local batch system and collects job status information.

As a consequence of grid sites autonomy, no single component in a grid controls resource allocation from job submission to end node assignment. The set of grid nodes is not considered as a whole: a node is never compared for assignment with another node from another site.

Resources are segregated between grid sites. They are also disconnected from users.

1.3.2 User and job disconnected

In single administrative domains, interactive jobs and runtime job migration is possible using a direct connection between user and execution node. In grids, jobs are delegated to sites. Execution nodes are not directly accessible from outside of their sites. The connection is lost.

From Condor to Condor-G

With Condor (section 1.2.5 and fig 1.2), daemons take responsibility for a task on both user node (the *shadow*) and execution node (the *starter*). These daemons maintain a connexion between each other through which users or applications control remote computations at runtime. If the code is linked with appropriate libraries, both sides communicate seamlessly. In addition, the starter can checkpoint the task while it is running, and send checkpoints to the shadow, possibly for migration on another execution node.

This connexion, runtime management and migration is lost with Condor-G. Condor-G was produced in 2001 to introduce the matchmaking mechanism in grid brokers [FTF^{+}02]. *G* initially meant *Globus* because Condor-G was designed to integrate with GRAM. Now it means *Grid* to denote the system's generality.

With Condor-G (fig 1.4) the **Grid broker** not only includes a **matchmaker** but has also taken over the **schedd** that would be run by the user in Condor. The *schedd* launches a **grid manager**, the process responsible for a job. This means that the user fully delegates the job to the broker, including potential

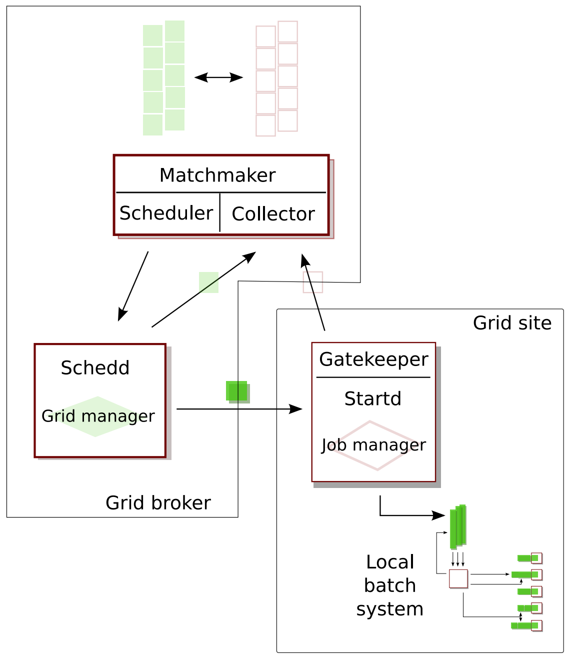


Figure 1.4: Condor-G

localization, control and communication at execution time. However, the *grid manager* also drops these capabilities. It simply submits the job to the **grid site**'s **gatekeeper**. As on standard GRAM, the gatekeeper decides to accept or return the job to the broker. If it accepts the job, it forwards it to the **startd**, which deploys a **job manager** process. The job manager, in turn, submits to the grid site's **local batch system** of choice, which can be other than Condor.

Grid manager and Job manager in Condor-G are the counterparts of Shadow and Starter in Condor. Their functionality is reduced. They submit jobs and do not manage them at runtime. Communication between each other is reduced to notifications of job status.

gLite, a middleware distribution

Condor-G illustrates the loss of user control on their jobs. Condor-G is integrated in distributions like gLite, LCG middleware [Lit07].

In a deployment of gLite, the broker is replicated to handle the load. On job submission, a broker replica submits a Condor *schedd* to the selected site. Once running on the site, the *schedd* sends a simple ClassAd to the broker to request the associated job. The *schedd* then forwards the job to the local batch system. It knows which broker replica sent the job. Therefore it can communicate back the job status. In this case the Condor matchmaker is not used to select the appropriate site but only to identify the request of a single schedd. This construction makes little use of its components individual functionalities.

On that account, gLite also distributes a simpler system as a more recent alternative. It consists in two simple java servlets, ICE and CREAM[5]. ICE implements the practical functionality of a gLite grid broker and CREAM interfaces a local batch system on a grid site [And06]. ICE/CREAM reproduces the functionalities of task mapping that have been achieved in practice in a central component that federates multiple administrative domains. The result is a simple batch job submission to grid sites with access control and site selection.

1.3.3 Out-of-date matching

A grid broker does not map jobs on execution nodes. Instead, it matches job requirements against site resource advertisements. Advertised resources may not hold when the job is scheduled to run.

Job description

A job carries a description of hardware and software flavor and configuration, resources and data that it expects to find on the execution node. This is done in a format chosen by the infrastructure. gLite uses Condor ClassAds (see section 1.2.5), renamed **JDL** (*Job Description Language*). NorduGrid middleware, ARC[6], has its own format called **RSF** (*Resource Selection Language*). The Open Grid Forum, a standardization consortium assembling representatives from industry and academia, recommends another variant: **JSDL** (*Job Submission Description Language*) [SAB+05].

Resource description

Sites, on the other hand, advertise their resources. They do not publish specific information for every node. Instead, they group their nodes into homogeneous clusters called *computing elements* [Chi04].

5. Computing Resource Execution And Management
6. Advanced Resource Connector

Definition 12 (Computing element [ABD$^+$07])**.** *As a common abstraction, the* ***computing element*** *(CE) refers to the characteristics, resource set and policies of a single queue of the underlying management system.*

At the Grid level, computing capabilities appear as ***computing elements*** *(each being a set of job slots to which policies and status information are associated) that are reachable from a specific network endpoint.*

A CE description includes information on operating system flavor, number of processes and queue length. The same configuration is maintained on all nodes of the CE. A grid broker does not make a difference between them. Therefore each computing element has its own batch system. A site usually contains one or a few computing elements.

There are several CE description formats: the **GLUE** schema, which stands for *Grid Laboratory Uniform Environment* is the most used by early grids. GLUE is specialized in grid resources [ABD$^+$07]. The *Common Information Model* (**CIM**) applies to distributed computing in general. It is recommended by DMTF[7] and used notably in Naregi[8]. Different formats can be converted to ClassAds for processing by Condor matchmaker [BGK$^+$03].

The grid broker delegates a job to the grid site that has the least loaded matching CE at time of submission. Each computing element typically updates its information every five to fifteen minutes. Added to a job's queuing time on the site, it gives the obsolescence of the broker's resource perception.

Consequences

Out-of-date matching is not fault tolerant. For instance, some CE failures result in jobs being considered terminated as they arrive. Before the malfunction is acknowledged, jobs keep flowing to the CE and failing. This situation is called a *black hole*.

Out-of-date matching proscribes fine-grained resource allocation. CEs and jobs describe themselves in very broad terms. Indeed, the state of transient resources such as cache, memory, disk and CPU cannot be communicated. Therefore, their utilization cannot be optimized.

To summarize, a central decision system that preserves the independence of grid sites fails to improve the efficiency of their aggregation. The following shows that it cannot either provide better functionalities than those of individual sites.

7. Distributed Management Task Force: www.dmtf.org
8. National Research Grid Initiative: www.naregi.org

1.3.4 Intersecting the capabilities of local systems

A grid broker must deal with any possible type of local resource management system present on grid sites.

There are two ways to integrate interfaces: *standards* and *translators*.

Standards

In a first approach, grid brokers and grid sites must implement standard interfaces [9] [Fos05]. Once a consensus is reached and all systems comply with it, functionality is not lost along the path [FKNT02]. However, the process is long and standards requires wide acceptance.

Translators

The second approach is pragmatic. Before standard interfaces are implemented for all components in use, these components have to communicate anyways. Instructions from the grid broker, forwarded to the site, must be understood by the variety of local batch systems available.

GAHP (*Grid ASCII Helper Protocol*) is a translation protocol originally developed as part of Globus Toolkit [Fos06]. It translates instructions from the grid broker to various implementations of a site's gatekeeper, and from the site's job manager, i.e. the process that controls a job on a grid site (section 1.3.2), to various local batch systems [NYI$^+$05].

Unfortunately, the vocabulary of translators intersects the capabilities of the systems they interface. With a few variants, it is reduced in practice to: **submit** to submit a job, **cancel** to cancel a job submission and **status** to get the status of a job submission [Reb05, NLJ$^+$05].

1.3.5 Partial conclusion

The attempt to centrally orchestrate grid resources results in a simple superposition of grid sites, no communication or control from the user at runtime, a delayed, broad-grained knowledge of the resources, and the simplest existing batch functionalities.

9. Standardization organizations involved in grid computing include OGF (Open Grid Forum: ogf.org), IETF (Internet Engineering Task Force: ietf.org), OASIS (Organization for the Advancement of Structured Information Standards: oasis-open.org), DMTF (Distributed Management Task Force: dmtf.org).

In practice, grid users have the option to skip the broker and submit directly to grid sites. Experienced users often choose this option. The grid broker brings little advantage outside of identifying grid sites.

In the next section, a recent trend shows that a decentralized allocation is possible.

1.4 User-driven allocation

This section studies a decentralized approach that has recently gained momentum. In this approach, a large part of the allocation is processed by federated users and application frameworks.

This new trend is motivated by user concerns that do not find satisfaction with traditional infrastructures (section 1.4.1). A way to bypass the standard allocation process is identified in section 1.4.2. Section 1.4.3 shows how a framework for large applications use this bypass, as well as VO allocation systems (section 1.4.4). A Condor mechanism was found to be well suited for this purpose (section 1.4.5). Section 1.4.6 shows how major VOs start to change their way of allocating their tasks. This turn of events redefines the paradigms of grid resource allocation (section 1.4.7).

1.4.1 User concerns

Users are concerned with *problem-solving* and *performance*.

Problem-solving Users and applications have constraints on task mapping: job dependencies, interactivity, inter-process communication, prioritization, etc. Given the loose coupling of grid resources, best suited applications are divisible into independent jobs. The absence of support for task mapping constraints further restricts the class of applications that can be handled directly.

Performance Low-level resources allocated to individual jobs determine the performance, even with independent, standalone jobs [KSS+07]. A grid assembles heterogeneous nodes. Task mapping must take into account their differences in conjunction with job profiles. For example, a processor with little cache would not handle well memory-intensive tasks, but it could handle together a CPU intensive task and a task that only downloads data [XZQ00].

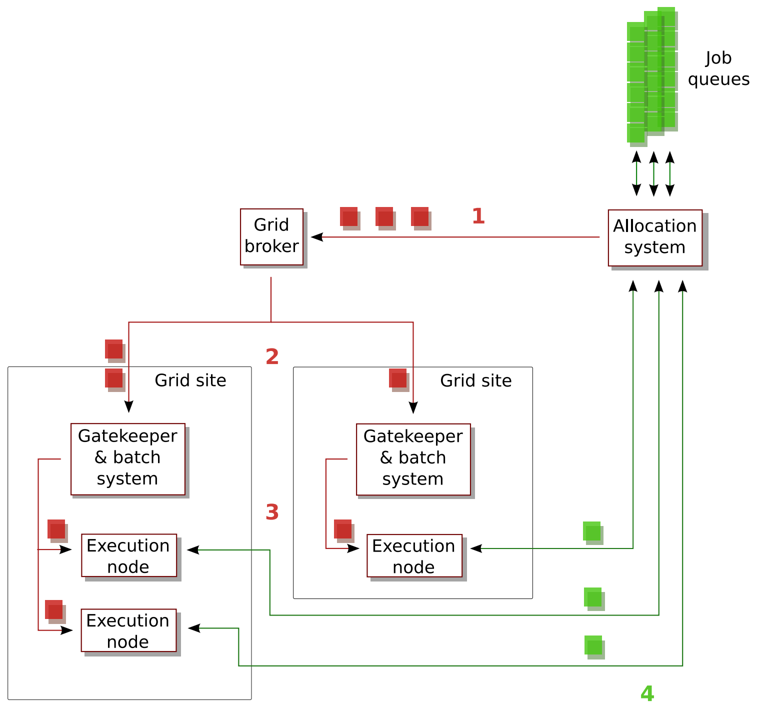


Figure 1.5: Late binding on a grid

Attempts for centralized task mapping do not achieve support for problem-solving and performance. Between the authority of resource users and resource providers, little remains to be controlled by a third party. Still, grid infrastructures centralize the access to different sites. Starting there, some large VOs and application frameworks have the capacity to address their own concerns by themselves.

The following identifies a common model, implementations, benefits, and the new constraints in these environments.

1.4.2 Late binding

The initial intent of grids is to consolidate resources from multiple administrative domains for the benefit of multiple independent applications and users. Serving all users as a whole has not yield substantial benefits. However, a growing number of VOs and large applications successfully address their own concerns with late binding.

Definition 13 (Late binding)**.** ***Late binding*** *is an allocation strategy in which a task is selected to run only once a placeholder is scheduled for it on identified resources.*

Figure 1.5 illustrates late binding on a grid. An **allocation system** is responsible for a set of jobs. It submits monitors to the **grid broker** (1). Each monitor follows the normal job flow. The grid broker delegates it to a site **gatekeeper** (2). The gatekeeper forwards it to a **batch system**. The batch system assigns it to an **execution node** (3). Once running on the execution node, the monitor connects back to the allocation system for actual job submissions (4).

Late binding allows for *controlled start time, placement, and runtime communication.*

Controlled start time Jobs start as they are submitted.

Placement Monitors inspect resources, their location and configuration.

Runtime communication Communications initiated from inside a grid site cross the site's firewall.

1.4.3 Allocation by applications

An application that runs on a grid uses a module to spawn jobs. This module simply submits jobs to the grid or schedules them with late binding.

In the absence of late binding, unpredictable delays before execution must be expected. A study on a major infrastructure shows that half of the jobs wait for more than five minutes between submission and execution, and 5% wait for more than 15 minutes [GLMR07].

With late binding, application schedulers control start time and directly interact with end nodes. They address job dependencies, interaction with the user, interaction between jobs, real-time control, priorities and fault tolerance [J.T06].

DIANE[10] is a framework to schedule applications on grids [Mos03]. The project started in 2002 to port CERN applications to the DataGrid, the ancestor of LCG. Other notable applications include genome sequencing and *in-silico* drug discovery against malaria and bird flu [MHS$^+$04, LHC$^+$06].

DIANE uses late binding to minimize **makespan**, i.e. the time until the last job completes. It keeps queues in front of each execution node and dynamically re-assigns jobs to different queues. Re-assignment is triggered when new

10. DIstributed ANalysis Environment

monitors start execution and bring new resources to the framework or when the progress on a node is unexpectedly slow [J.T06].

The more jobs an application spawns, the more useful DIANE is. A high job liquidity results in substantial gains from coordinated scheduling. Comparable or greater liquidity is reached by centralizing jobs from the members of a scientific collaboration.

1.4.4 Allocation by collaborations

A growing number of virtual organizations (VOs, def 2) take care of scheduling for their members.

Examples

Particle physics is the area with the most grid users. In particle physics, A VO is a collaboration that builds a detector and analyzes its data.

- At Fermilab, in the area of Chicago, **CDF**[11] and **D0**[12] study the results of Protons-Antiprotons collisions, scheduled to analyze data until 2009. **MINOS**[13] analyzes Neutrino oscillations.
- At SLAC, *Stanford Linear Accelerator Center*, **BaBar**[14] analyzes the violation of charge and parity (CP) symmetry in the decays of B mesons.
- At CERN, the European Center for Nuclear Research in Geneva, four VOs are finishing to build detectors and preparing for data analysis for the coming years. **ATLAS**[15] and **CMS**[16] are two general-purpose detectors to analyze proton-proton and heavy ions collisions. **Alice**[17] studies Pb-Pb collisions generating a quark-gluon plasma as in the early universe, and **LHCb**[18] studies collisions of baryons containing the Beauty quark for CP violation measurements and rare decays observations.

Other scientific domains are also represented. For example, **BIOMED** is a VO that covers 3 domains: Medical Imaging, Bio-informatics and Drug Discovery. BIOMED maintains application software for use on grids and domain-specific grid portals [LHC+04, GSM+07].

11. *Collider Detector at Fermilab.* www-cdf.fnal.gov
12. www-d0.fnal.gov
13. *Main Injector Neutrino Oscillator Search.* www-numi.fnal.gov
14. from $B\bar{B}$. www-public.slac.stanford.edu/babar
15. *A Toroidal LHC ApparatuS.* altas.ch
16. *Compact Muon Solenoid.* cms.cern.ch
17. *Large Ion Collider Experiment.* aliceinfo.cern.ch
18. *Large Hadron Collider beauty.* lhcb.web.cern.ch

Why scheduling by VOs is appropriate

There are several reasons why the VO is a natural authority to schedule the jobs of their users. Users identify to grid services as members of a VO. In the same VO, users collaborate and share the same goals. They put in common their knowledge, their efforts and their solutions.

Identity VOs provide grid users with an identity recognized by resource providers. Grid sites identify VOs to define bulk resource supply contracts and authenticate job owners [FKT01].

Cooperation VO members collaborate for a shared goal. Competition between individual users is supervised for the benefit of the whole. For example, a job can be delayed to improve overall performance, or to satisfy priorities.

De-multiplied effort Job submission is often naturally delegated to a few expert VO members. The gap is narrow between centralized skills to manage grid jobs and the development of a single allocation system for the whole VO.

Knowledge VOs use a limited number of applications and algorithms. Their jobs have known resource requirements. VO administrators know which software configuration works on the execution node. They can evaluate resources to accurately map their jobs.

Liquidity Finally, VOs can be large. For example, 1900 physicists collaborate in ATLAS. The number of users and jobs, and the amount of resources used raises opportunities for performance optimization.

Since VOs do not control the hardware, late binding is currently their only option for scheduling.

Late binding by VOs

VOs request the highest possible/useful number of grid nodes and maximize the computing throughput on these nodes.

Grid nodes are requested by submitting monitors as described in section 1.4.2. A monitor pretends to be a job and runs as long as allowed. Grid sites set maximum durations, typically of 48 or 72 hours, after which the monitor must terminate.

For this time period, monitors receive actual jobs from the VO and control their execution. It appear to resource owners that jobs run for days. In fact, actual jobs last for a few minutes to a few hours. They bypass the queues of traditional grid submission to be directly allocated on end nodes following a strategy defined by the collaboration.

VO schedulers are relatively recent. Despite their performance they are not widely identified in the literature. The following is a brief survey.

1.4.5 Sudden success of an old Condor mechanism.

Condor is designed to use temporary idling nodes, a strategy known as **CPU scavenging**. To cope with transient resources, Condor implements late binding. Besides, since 2001, Condor offers a mechanism called **glide-ins** to form personal pools with nodes controlled by external batch schedulers. Since 2006, glide-ins are used to implement late binding in grids.

Glide-ins for personal pools

Glide-ins are portable shell scripts that a user submits to external batch schedulers. Once running on an end node, a glide-in launches a process equivalent to a Condor startd that makes itself known to a matchmaker.

A standard Condor pool takes care of an organization's resources and users (section 1.2.5 and fig 1.2). The matchmaker serves all members of the organization. A Condor pool constituted with glide-ins is intended for personal use. The matchmaker matches classads from the sender's schedd with classads from the glide-ins' startd's.

Glide-ins for late binding on grids

Only recently, a few allocation systems equipped with a Condor matchmaker started to systematically send glideIns for late binding to grid nodes.

Once resources are identified, the remaining difficulty is to provide inbound connectivity to grid sites. The starter initiates a connection with a proxy outside of the grid site, the GCB (*Generic Connexion Broker*), and receives an identifier. To communicate with the starter, the shadow sends a message to the GCB using this identifier and the GCB redirects the packets [SL03, BSK05].

glideCAF

The Central Analysis Farm (CAF) of CDF, the particle physics VO and Fermilab experiment, was extended in 2005 to use grid resources with glideCAF, which integrates late binding with Condor glideIns [BHL+06].

Cronus

An individual initiative in Atlas gradually gained momentum and led to Cronus in 2006 [PW07]. In 2007, Cronus allocates a substantial part of Atlas jobs and controls a dynamic pool of about 8500 CPUs infiltrated through LCG, OSG and NorduGrid.

In addition to short-circuiting grid submission delays, Cronus manages *data distribution* and *load takeover*.

Data distribution Considering that Atlas jobs do not consume network bandwidth, Cronus downloads data in the background from major storage systems. Succeeding jobs do not wait for data transfers. Instead they are placed where their data is already present.

NorduGrid plans in advance data downloads before jobs are scheduled for execution, but LCG does not: jobs start execution by requesting the data and stay idle until the download is complete. Cronus saves this idle time.

Load takeover Cronus lets glide-ins take over the jobs of others whose lease is about to expire. This saves from 80% of the jobs failures.

glideinWMS

glideinWMS is a project started in 2007 by US CMS, the American part of the CMS collaboration. It extends glideCAF and Cronus information system [Sfi07].

1.4.6 An evolution of VO strategies

AliEn and job agents

Alice Environment, has been since 2001 the distributed data analysis environment of ALICE, a CERN detector and collaboration [BPS03, SAB$^+$03]. In 2004 it serves as a basis for the gLite middleware for LCG [LHA$^+$04]. It keeps evolving besides for ALICE and introduces the *pull model*. In this model, instead of having the broker *push* jobs to computing elements (CE, section 12), *job agents* that monitor CEs *pull* jobs when appropriate.

Definition 14 (Job agent [BPSGO04]). *[The Job Agent is a] Web Service allowing users to interact with the running job, send a signal or inspect the output. Prior to job execution, the Job Agent can automatically install the software packages required by the job.*

Pull model preserves from faulty configurations and black holes, i.e. the situation where jobs keep flowing to a faulty CE (section 1.3.3). Pull model is only a step away from late binding. Another VO, LHCb, takes this step together with AliEn.

DIRAC and job pilots

DIRAC, *Distributed Infrastructure with Remote Agent Control*, started in 2003 for LHCb, another CERN detector and collaboration [vHCF$^+$03, TGSR04]. In 2004, DIRAC follows AliEn's pull model and "glides in" job agents on execution nodes, obtaining so-called **pilots agents** [PSP06].

Jobs are classified on a central server in a number of queues. When a job pilot requests a job, a queue is selected, known to contain the most appropriate class of jobs, and one of the first jobs found on this queue is sent.

The pilot/glide-in mechanism is becoming the rule for large VOs. AliEn quickly made the move to job pilots. In late 2005, the US Department of Energy funded a project, **Panda**, that uses the same architecture as DIRAC to manage ATLAS jobs on OSG [WLrW06, Nil07].

1.4.7 Specific constraints

Late binding on grids, as opposed to the use of dedicated clusters, presents specific constraints for the deployment of task mapping systems.

Convoluted communication TCP or UDP communication can be instantiated only from inside a grid site. Solutions are asynchronous, via messages passing and polling, or involve registration to a proxy and packet forwarding [SL03].

Limited node control A job is unprivileged on a grid site. The same is true for glide-ins and pilot jobs. Isolating jobs inside virtual machines might allow for more control from the VOs [KDF04, GPJ$^+$07].

Using late binding, computers accessed on the wide area may appear as a dedicated, local cluster, although a few differences remain. First, grid resources are alloted for a limited period. Second, communication between remote nodes exhibit some latency.

Transience Nodes leave the pool when their maximum lease period expires. Other nodes join the pool when newly submitted job pilots / glide-ins

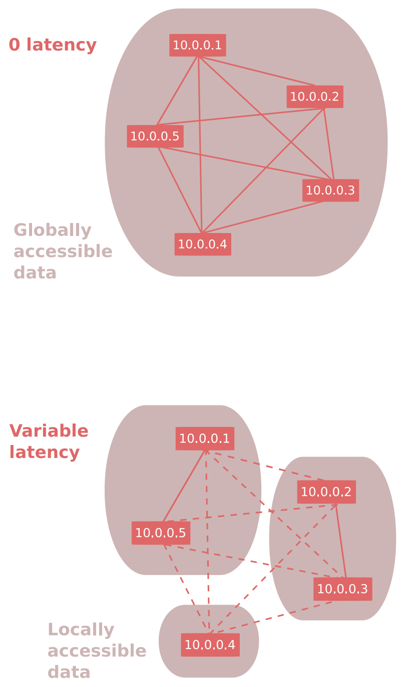


Figure 1.6: Wide area (bottom) compared to local clusters (top).

start execution. VOs operate on pools where nodes constantly come and go.

Latency Both data access and communication between peers of a distributed system are affected. Problems arise that were not present in the local area context. Access to data is heterogeneous. Therefore, allocation must be data driven. In addition, no single point has a complete knowledge of the whole system state.

The upper part of figure 1.6 represents a network of computers taken from a local cluster (top). Data is accessible everywhere, and communication is seamless. The lower part shows a network of computers taken from remote grid sites. Some are tightly connected, while other links exhibit some latency; and large data is not accessible everywhere. Late binding convert the cross-organization barrier into these few constraints.

1.5 Conclusion

Grids are world-wide aggregation of computational resources and demand from multiple institutions. Systems that attempt to centrally orchestrate together grid resources and users did not count with the autonomy of administrative domains. The need to avoid obtrusiveness confined them to a simplistic superposition of grid sites, instead of an efficient consolidation of resource management.

However, some applications generate enough liquidity to graft their own opportunistic task mapping mechanisms. They have interest and capabilities to check acquired resources against their tasks. They operate fine-grained, dynamic mapping. They realize the move from resource allocation across autonomous applications to usage-centric allocation across autonomous resource providers, without *a priori* knowledge of resource topology.

Grids were designed to be usable by scientists inexperienced with resource management. It turns out that large, experienced and powerful organizations of users take more control that initially expected. With late binding, they turn grids into on-demand resources, and they use them the same way companies use resources from "the cloud".

Chapter 2

MAD Approach to Grid Resource Allocation

2.1 Introduction

In grids, resource users and resource providers are distinct. The placement of a task on a server involves both. They are bound in a contract, explicit or tacit, that motivates their co-operation. However, since in general they report to different institutions, they have distinct objectives.

– Users are generally interested in computing speed. Computing speed has different definitions for a single large scale application and for a collaboration of individuals who constantly execute new tasks. User concerns also include adaptation of software configuration, optimization of computing throughput, compliance with task priorities, minimization of resource oversubscription if subscription is limited [CGV07], and prevention of resource undersubscription.
– Providers favor minimization of resource supply costs, i.e. minimization of server activity and power consumption [CIL$^+$07, JB07], optimization of cooling efficiency [BF07], minimization of obstruction to servers maintenance, to local use and local policies.

These objectives may conflict in the resource allocation process. Initially based on voluntary collaborations, grids have not focussed on mitigating conflicts. As a result, compromises in performance and flexibility are made on both sides. We propose to start with the separation of concerns. In this aim, we introduce an architectural pattern, Symmetric Mapping.

Design patterns were originally introduced for object oriented programming by the *Gang of Four* in OOPSLA meeting [GHJV95]. They identify best practices to solve recurrent software design problems. They are defined in terms of relationships between objects that compose the software. Similarly, in distributed systems, architectural patterns define component structures to solve recurrent architectural problems [BMR$^+$96b]. Famous examples include *Model-view-controller* that isolates application logic from interface [Gre07], and *Peer-to-peer* that decentralizes control and resources to all elements in a system, making them functionally equivalent [CvR05].

Symmetric Mapping has his foundations in a new model of resource allocation. The model is an analogy with the model developed for the study of fault tolerance in distributed systems [AAC$^+$05]. In a MAD[1] distributed system, participants' behaviors are not controlled. Instead, the system builds on assumptions. Namely, a participant is rational or not, altruist or egoistic, or byzantine (entirely unpredictable). In our case, we assume that participants are selfish and rational, and the objective is to satisfy each of them independently. To the best of our knowledge, our formalism is the first to use MADs principles for resource allocation.

Section 2.2 details previous work. Section 2.3 identifies several participants objectives. Section 2.4 introduces Symmetric Mapping. Section 2.5 simulate its implementation on synthetic examples. The rest of the chapter presents the model more formally. Section 2.6 introduces the fundamentals. Section 2.7 formalizes the problem. Section 2.8 establishes separation of responsibilities as a solution.

2.2 Previous work

In this chapter, we propose a new architectural pattern with its foundations in a new formal model. Relevant previous work includes the identification of existing architectural patterns in the same area, and the existing formal models that underly their emergence. This discussion is largely based on chapter 1. Chapter 1 gives a historical perspective on resource allocation approaches in grids. This section synthesizes the observed patterns and the underlying models.

Models based on queuing theory and game theory are considered, as well as

1. Multiple Administrative Domains

models that formalize the notion of resource containment. Existing architectural patterns are organized around Metascheduling, Late Binding, and economic patterns.

2.2.1 Queuing models

The study of grid resource allocation is traditionally based on queuing theory [CK88, All90, MAS$^+$99, BBC$^+$04, Van08]. So are grid systems in production today [GDJ08]. The model we introduce in this chapter departs from queuing theory and allows to consider resources that do not use queues, and events that occur outside of queues, including co-allocation and live-migration. This fine-grained, dynamic view is necessary to capture the level at which users and providers incentives collide or settle.

Matchmaking or resource brokering systems use requirements and preferences from both sides [FTF$^+$02, CFK04, CIL$^+$07]. The design obtained in this chapter builds on comparable requirements and preferences from decision-taking participants. It can use matchmaking or brokering to implement some of its elements. Section 2.2 gives detailed comparisons with related designs.

In systems not primarily designed with participants autonomy in mind, services are added in the attempt to centrally handle every possible constraint participants might find important [CFK04]. Instead, we separate responsibilities between participants, so that they deal with their own concerns, as opposed to having them managed by a third party.

2.2.2 Economic models

Economic models have received increasing attention recently. They study pricing strategies that achieve user objectives such as performance or fairness with income guarantees for providers [HWZ05, BMB$^+$08]. The problem we address is relevant when the contract between users and providers does not entirely determine the resource allocation process. This is mostly the case in grids where high level contracts bind together large organizations. Such contracts may or may not use monetary compensation.

Game theory includes the study of possible strategies and resulting equilibria in *games* with competing participants. It has yield various protocols in distributed computer systems [CS00, JZ09]. In this area, Multiple Administrative Domains (MADs) were introduced to design fault tolerant systems. Participants' behaviors are given, and the goal is to design the *game* so that the equilibrium coincides with the objective. MADs principles have been used to design cooperative backup services, peer-to-peer data streaming, Internet and wireless routing [AAC$^+$05]. Our formalism is the first to use MADs principles to design grid resource allocation architectures.

2.2.3 Containment in other models

Containment can be used to separate concerns. Various systems introduce containment with virtual machines to separate users from providers [KFFZ05, RMX05, RIG$^+$06, GPJ$^+$07]. They evaluate the benefits of virtual machines in addressing the problems that users may want a different software configurations than the native software of the host, and providers may want to move execution environment while in use. Our approach yields a new definition of containment, of which virtual machines are potential implementations, as well as a new definition of responsibilities.

Abstract State Machines (ASMs) are formal tools that can be understood as generalizations of turing machines [Gur03]. They are primarily used to specify complex systems at every level of abstraction and generate tests. High level ASMs have been proposed as a basis for the specification of grid systems [NS06]. For generality, the authors introduced generic entities called *abstract resources*, *user mapping* and *resource mapping*. ASMs start by modeling a design. By contrast we start by modeling the specific problem of reconciling the participants diverging objectives. The solution we obtain yields a definition of *containers* consistent with *abstract resources*, that divide resource allocation in the same two parts, one carried out by users and the other by providers. It is consistent with their model and provides a semantic refinement of their components. However, we do not express it in ASM terms.

2.2.4 Metascheduling Pattern

The *Master-Slave* pattern specifies that a master process allocates tasks to slave processes [BMR$^+$96a]. In addition, the *Broker* pattern defines a class that acts on behalf of requesters and hides the details of its action [BMR$^+$96a]. Inspired from these two design patterns, *Metascheduling* was an early architectural pattern for grid resource allocation [Wei98]. It specifies that tasks are submitted to a global meta-scheduler, which in turn submits to local schedulers. As detailed in chapter 1, this is the pattern used in most grid systems until recently.

The mainstream infrastructure behind LCG, the Large Hadron Collider Computing Grid, is an implementation of Metascheduling [BBB+05]. LCG is a collaboration of academia and scientific communities that aggregates resources for use primarily by CERN experiments.

In LCG, a task comes with requirements in terms of the software that must be present on the execution environment. Computing resources are not guaranteed. Their type and amount varies according to site policies, their servers, and the load of other tasks possibly collocated on the same servers.

The central component is the *workload management system*, which is not under the user's nor the provider's control. It selects a cluster that satisfies the task requirements, on a site that accepts contracts with the VO [2] involved, based on access-right policies [And04].

Although the workload management system does not pervade a grid site, it sets strong terms on site resource management. In practice, the only freedom left to the resource provider is the choice of a local batch system such as Condor, LSF [3], PBS [4] or SGE [5] [BHK+00, IGF05, Hum06]. The middleware provides interfaces to supported batch systems. The only effort to minimize allocation cost is done by having the batch system statically assign a new task to the least loaded server on the cluster, that is, the server with shortest task queue. Providers have difficulties to perform maintenance operations to their resources because running tasks are directly bound to resources. For critical security upgrades, user tasks are abruptly discontinued.

The allocation of tasks is a static assignment. Tasks and allocated resources remain bound until the expiration of one of the two. Resources are divided into *server slots* that typically consists in three CPUs, without data isolation, and assigned by the provider's batch system.

Other grids such as EGEE [6], NorduGrid, BaltiGrid, Naregi and OSG [7] have a similar design [Ave07].

Virtual Workspaces isolates user resource in a Metascheduling implementation, the Globus Toolkit [KFFZ05]. Virtual Workspaces provides a Web-Service interface to deploy and configure Xen virtual machines (VMs) [BDF+03]. The targeted users and providers are the same as in scientific grids. Instead of a

2. Virtual Organization, identified as a single user.
3. Load Sharing Facility
4. Portable Batch System
5. Sun Grid Engine
6. Enabling Grids for E-Science
7. Open Science Grid

server slot, a user is given a *workspace*, i.e. a VM.

The VM deployment interface is presented to Globus middleware. VM management is assigned to the user or a third party. This is comparable with Condor's recent *VM Universe*, which lets the user define the deployment of a VM on a remote host.

The Broker pattern specifies that resources are hidden to the user and task management is delegated. This is not compatible with Symmetric Mapping.

2.2.5 Pull Mechanism and Late Binding Patterns

Master-Slave was later referred to as *Push Mechanism* by contrast with *Pull Mechanism* where idle resources request tasks [SBP03]. Pull Mechanism makes allocation resilient to broken resources and removes queues in front of end resources.

The *Late Binding* pattern allows the introduction of Pull Mechanism on a system designed with Push Mechanism. In Late Binding, monitors *pushed* to an end resource check the resource status before *pulling* actual tasks [BHL$^+$06]. The pattern was first implemented by Condor Glide-In in 2001 [TTL02]. Since 2003, major grids are moving towards Late Binding [GDJ08].

AliEn is an example of Late Binding. AliEn is the Analysis Environment for ALICE, a virtual organization (VO). AliEn uses its own scheduling system on LCG infrastructure [SBP03].

As a consequence from using LCG, members of ALICE rely on the sites' best effort. However, tasks are processed immediately when submitted because a task is pulled when relevant resources are avaiable. AliEn gives more flexibility to the ALICE collaboration for task mapping. A *job agent* monitors every resource and triggers task selections from ALICE *job queue*. This mechanism is designed both to cope with lack of guarantees in LCG SLAs, and to prioritize tasks.

AliEn initiated the emergence of a proper front mapping via Late Binding. Most major VOs are now implementing a similar system: CDF with GlideCAF, ATLAS with Cronus and Panda, LHCb with DIRAC, CMS with Glidein-WMS and other independent large-scale applications may use DIANE [GDJ08].

Symmetric mapping decomposes the allocation in two parts and Pull Mechanism is a possible design for both parts. Late Binding separates resource subscription from task allocation. A system that implements Late Binding implements Symmetric Mapping if the underlying Metascheduling implementation does not constrain or obstruct the provider.

2.2.6 Market Based Control and Peer-to-Peer Matching Patterns

Market-Based Control simulates markets to share resources efficiently among competing users [Cle96, LB06].

For example Tycoon balances the load across and inside servers according to user payment [Lai05]. A contract involves the provider with the least expensive offer. The price of a provider's resources depends on the load. The provider does not intervene in the allocation. Commercial services are analogous. Amazon EC2 [8] is a web-service to sell Amazon's resources on demand. The only provider is Amazon. In both examples, users have direct access to virtual machines. By contrast with Market-Based Control, Symmetric Mapping also supports the cases where the contract between user and provider does not entirely determine the allocation.

Peer-to-peer Matching is another alternative for contract mapping. In the absence of a market, participants match and negotiate contacts in a collaborative manner. For example, a Condor flock is the assembly of computer centers that borrow hardware resource from one another when needed [ELvD$^+$96]. A component called the *Gateway* is hosted by each participant. If a user cannot assign tasks to its own resources the Gateway examines its pairs. An important part of Condor is designed specifically for matchmaking and negotiation. The *Matchmaking* mechanism is provided through the use of *ClassAds* [Ram00]. Users and providers define themselves with relevant attribute and write their requirements with regular expressions on the other participant's attributes.

2.3 Objectives

This section examines typical user and provider concerns. We identify a **value** function in each case. This function quantifies a benefit that a given participant can expect from the allocation. We discuss the ability of a participant to predict or measure **value**, and optimize it with a schedule. The following concerns are defined : *makespan, sum of weighted flows, energy consumption* and *obstruction.*

In the following we consider allocations defined as follows: An allocation is an application that takes a task, a resource and a time, and returns *true* if the

8. Elastic Compute Cloud, amazon.com

task is allocated to the resource at that time, $false$ otherwise.

$$a : Tasks \times Resources \times Time \longrightarrow \{true, false\}$$

2.3.1 Minimum makespan

The makespan is the time between submission of the first task and termination of the last. A user who wants all tasks to finish as soon as possible wants to minimize the makespan. This is the case if the user launches a single embarrassingly parallel application.

We write $m(a)$ the makespan of allocation a. For a user concerned with makespan, $\mathbf{value}(a) = -m(a)$ is a valid value function.

With identical tasks and identical resources, a simple online greedy algorithm finds the optimal schedule with constant complexity. It buffers tasks as they come. Whenever a processing unit is free, it assigns a waiting task to run until termination.

In a model often used for scheduling, a task $t \in Tasks$ has a *length* l_t, a processing unit $r \in Resources$ has a *performance* p_r, and the execution time of t on r is l_t/p_r. The resulting problem is the Multiprocessor Scheduling Problem. No tractable algorithm finds the exact optimal schedule. However, achieving near-optimality with known error is tractable [AMZ03, MKK$^+$05].

With less loss of generality a task $t \in Tasks$ has a number of instructions $s(t) \in \mathbb{N}$. An allocation $a \in A$ yields an instruction rate $\rho(t, a, \tau)$ for task t at time τ. The instruction rate is the number of instructions per second. The makespan $m(a)$ is written

$$m(a) = \max_{t \in Tasks} \min\{\tau' \in Time | \int_0^{\tau'} \rho(t, a, \tau) d\tau \geq s(t)\}$$

The estimation of $\rho(t, a, \tau)$ is reputedly a difficult problem. Precise analysis of computing performance, that takes into account the affinity between tasks and resources, is in its inception. It is addressed in areas of real time systems and heterogeneous computing [XZQ00, KL01, SOBS04, PRV08].

2.3.2 Minimum sum of weighted flows

A task flow is the time between task submission and termination. A user $u \in X$ who wants every task to finish as soon as possible wants to minimize a sum of weighted flows [LSV06]. This is the case when the user is actually a collaboration of individuals who launch batch computations over time.

For $t \in Tasks$ submitted at time τ_t, if $F(t,a)$ is its flow given allocation $a \in A$:

$$F(t,a) = \min\{\tau' \in Time | \int_{\tau_t}^{\tau_t+\tau'} \rho(t,a,\tau)d\tau \geq s(t)\}$$

If w_t the priority of task t, the sum of weighted flows is written

$$w(a) = \sum_{t \in Tasks} w_t F(t,a)$$

$\mathbf{value_u}(a) = -w(a)$ is a valid value function for user u.

Sum of weighted flows minimization suffers the same problems of tractability and modeling as makespan minimization.

2.3.3 Minimum energy consumption

The operation of a computing resource affects its power consumption. Power-manageable processors and devices exhibit different states which limit at different levels the range of their operation and their power consumption. State transitions require some time and consume energy, too. Power management policies determine the conditions of state transitions in an attempt to minimize the energy consumption of a computing resource under workload constraints [BR04, RRT+08]. We write $\mu(r,a,\tau)$ the resulting power consumed by $r \in Resources$ at time τ given the allocation $a \in A$ and the power management policy.

Part of the consumed power dissipates in heat. Cooling devices are operated according to the temperature in their zone of influence. The operation of a cooling device also consumes power. Thermal management policies determine the level of operation of each cooling device in an attempt to maintain acceptable temperatures with minimum energy consumption [BF07, TGV08]. We write $\mu(d,a,\tau)$ the resulting power consumed by cooling device $d \in Devices$ at time τ. We write $Z_d \subset Resources$ the resources that have thermal transfers with d.

A provider $p \in X$ concerned with energy consumption wants to minimize $e(a)$, or maximize $\mathbf{value_p}(a) = -e(a)$, where

$$e(a) = \int_0^{+\infty} \left(\sum_{r \in Resources} \mu(r,a,\tau) + \sum_{d \in Devices} \mu(d,a,\tau) \right) d\tau$$

$\mu(d, a, \tau)$ depends on $\mu(r, a, \tau')$ $\forall \tau' \leq \tau$ and for all resource r in the zone of influence of d.

$\mu(d, a, \tau)$ is difficult to know because thermal transfers are involved. In a simple representation, thermal transfers are considered instantaneous. Under this hypothesis, the power consumed by a cooling device is a function of the power consumed in its zone of influence at the same time. If this function is replaced with its linear approximation, $\exists \alpha_d$, $\forall a \in A$, $\forall \tau \in Time$

$$\mu(d, a, \tau) = \alpha_d \sum_{r \in Z_d} \mu(r, a, \tau)$$

With this simplification,

$$e(a) = \int_0^{+\infty} \sum_{r \in Resources} \mu(r, a, \tau) \left(1 + \sum_{d \in Devices | r \in Z_d} \alpha_d \right) d\tau$$

In the simplest formulation, a resource r is a server. Its states are *up* or *sleep*. On up mode, r consumes power $\mu_u(r)$. A resource r has a power efficiency $\eta(r)$ where

$$\eta(r) = \mu_u(r) \left(1 + \sum_{d \in Devices | r \in Z_d} \alpha_d \right)$$

In the simplest greedy algorithm, tasks are taken in order of starting time. The server r with highest $\eta(r)$ which is not already busy is assigned the next task. A server which is not assigned a task for a sufficiently long time is put in sleep mode.

In fact the energy consumption of a server does not only depend on its run/sleep status. It depends on the workload applied to its processors and each of its devices. A provider who already carries out power management of its resources and thermal management of its cooling devices will probably want to carry out energy-aware resource scheduling because it is the last piece of control to get full responsibility of energy expenses.

2.3.4 Minimum obstruction

We call *obstruction* the event in which an external user blocks resources that the provider wants to use internally or access for maintenance. It is not always possible for the provider to pre-empt resources or foresee when resources will be needed internally. The cost of obstruction can be quantified based on the eagerness of provider p to free some resource at a given time:

$$\mathbf{eager_p} : \begin{cases} Resources \times Time \longrightarrow \mathbb{R}^+ \\ \qquad\qquad (r, \tau) \longmapsto \mathbf{eager_p}(r, \tau) \end{cases}$$

$\mathbf{eager_p}(r, \tau) = 0$ if the provider does not need r at τ. Otherwise, $\mathbf{eager_p}(r, \tau) > 0$ and $\mathbf{eager_p}(r, \tau)$ indicates the relative benefit from monopolizing r at τ. The measure of eagerness is entirely up to the provider.

A provider p who wants to minimize obstruction wants to maximize the following value function.

$$\mathbf{value_p}(a) = \sum_{r \in Resources} \int_0^{+\infty} \mathbf{eager_p}(r, \tau)(1 - \delta_a(r, \tau))d\tau$$

Where $\delta_a(r, \tau) = 1$ if $\exists t \in Tasks | a(t, r, \tau) = true$, 0 otherwise.

Very often $\mathbf{eager_p}(r, \tau)$ is not known before τ, and the algorithm to maximize $\mathbf{value_p}(a)$ is an online algorithm.

2.4 The Symmetric Mapping pattern

In order to separate the concerns of resource providers and resource users, we define an architectural pattern: Symmetric Mapping.

2.4.1 Overview

Symmetric Mapping is intended to design architectures that perform resource allocation in a way that satisfies both resource providers and resource users when their interests differ.

The allocation of tasks to resource determines the satisfaction of both providers and users. In general, their incentives conflict. For example the most power-efficient server is not always the fastest. An allocation directed by a decision system under user control can result in high resource supply costs and an allocation directed by a decision system under provider control can result in low user-perceived resource value. Instead of compromising with them, Symmetric Mapping builds on these differences from the system design.

The principle of Symmetric Mapping is to divide resource allocation in three parts: **contract mapping**, **front mapping** and **back mapping** (fig 2.1). *Contract mapping* consists in the match between users and providers and the

$$\begin{tikzcd}
\text{Tasks} \arrow[rrrr, leftrightarrow] & & & & \text{Resources} \\
& \text{Users} \arrow[rr, leftrightarrow, "\textit{Contract}", "\textit{mapping}"'] & {} \arrow[d] & \text{Providers} & \\
\text{Tasks} \arrow[rr, leftrightarrow, "\textit{Front}", "\textit{mapping}"'] & & \text{Containers} \arrow[rr, leftrightarrow, "\textit{Back}", "\textit{mapping}"'] & & \text{Resources}
\end{tikzcd}$$

Figure 2.1: Mapping decomposition.

issue of containers that specify elementary resource transactions and subdivide higher level contracts. Mapping a task to a resource contributes to fulfilling a contract. It involves a container. Symmetric Mapping specifies that providers map their endorsed containers to their physical resources (*back mapping*) and users map their tasks to their subscribed containers (*front mapping*).

2.4.2 Definition

The Open Group Architecture Framework [9], a standardization consortium, identifies a terminology that can be used to define architectural patterns. According to the consortium, a pattern is defined with a *name*, an *intent*, *preconditions*, *forces* that play a role towards the intent, the *solution*, *postconditions* and *rationale*.

– Name: Symmetric Mapping
– Intent: Separate the concerns of providers and users and dispatch their responsibilities accordingly, so that their actions do not conflict and yield to the independent optimization of their respective concerns.
– Preconditions: Participants are administratively independent. Users need to run tasks and providers propose resources for tasks to run. They are bound by implicit or explicit contracts which justify their exchange. They are supposed to have rational and selfish behaviors. They have some idea of their objective and some knowledge on how to reach it.
– Forces: The greater the amount and heterogeneity of resources, tasks, and participants objectives, the better the opportunity to optimize the objectives [KSS+07].
– Solution:

1. Insert intermediate entities between tasks and resources. These entities are called **containers**. A container holds attributes of a resource transaction. In conjunction with the actual resources involved in the

9. opengroup.org

transaction, it determines a cost for the provider. In conjunction with the tasks involved in the transaction, it determines a revenue for the user.

2. Decompose the mapping of tasks to resources into the mapping of tasks to containers and the mapping of resources to containers.

– Postconditions: Users and providers match and issue containers. This is the first part of the allocation called **contract mapping**. Providers map resources to containers. This is another part of the allocation called **back mapping**. Users map tasks to containers. This is the third part of the allocation called **front mapping**.
– Rationale: If given appropriate responsibility, a participant will reach his own objective better than anyone else on his behalf.

2.4.3 Practical perspective

Participants

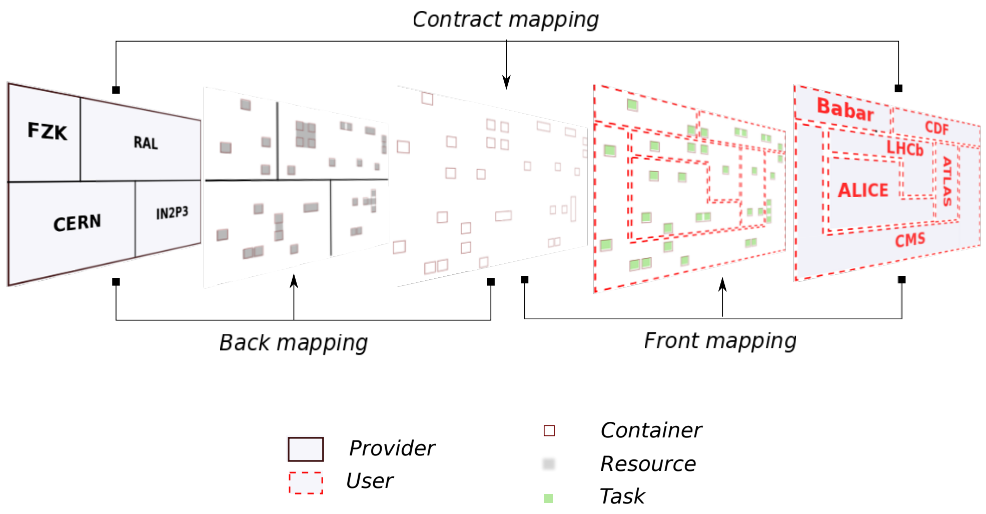


Figure 2.2: Participants, tasks, resources and containers.

Figure 2.2 features participants in a large particle physics grid. Resource providers are represented on the left hand side plane. The names correspond to institutions involved in the analysis of high energy physics data: CERN (Switzerland) is called the *Tier0*. It is where data is generated and stored in the first place. FZK (Germany), IN2P3 (France) and RAL (Great Britain) are example *Tier1*'s, where data is replicated [Rob06]. Tier0 and Tier1's are computer centers where a large part of the analysis takes place. At the time of writing, CERN has 7192 CPUs, IN2P3 3356, FZK 8340, and RAL 4016.

Resource users are represented on the right hand side plane. These are instances of virtual organizations (VOs): Babar, CDF, LHCb, ALICE, ATLAS, CMS. Each of them is a community of researchers whose data is generated by a particle physics detector. The VO names are detector names. Members of a VO analyze data and therefore generate tasks that run on the LCG. A VO is considered a single user because its members report to the same institution, have common objectives and the same applications.

In Contract mapping, users and providers define containers. In Back mapping, providers select resources to back their containers, and in Front mapping, users select tasks to use their containers. A container is backed by one or an assembly of resources from a unique provider. It supports one or several tasks from a unique user.

Containers

Resource exchange between two autonomous institutions is the result of a contract between them. Grid sites engage in long term support for chosen virtual organizations. Symmetric Mapping requires that the contract that sets the terms of this support is made explicit. In addition, these contracts must be divisible into non-redundant subcontracts called *containers*. Containers must not allow resource oversubscription or undersubscription, and must be described with enough precision to have a determined value as perceived by each participant.

Therefore containers are specific kinds of *Service Level Agreements* (SLAs) [CIL$^+$07]. SLAs typically specify the type and amount of subscribed resources and their lifetime, as well as constraints on the workload. Sometimes SLAs can be defined directly in terms of Quality of Service guarantees, which directly determine perceived values [RCAM06].

If relevant to determine the expected perceived value, container descriptions may include specifications on dedicated memory and cache hierarchy, the number of dedicated cores, their frequency, their optimization logic, network bandwidth, disk space, bandwidth and latency. Software configuration also belongs to resource specifications, potentially including operating system flavor, compilers and interpreters and their versions, and available administrative utilities.

The implementation of a container may or may not force participants into complying with its specifications. A Condor sandbox pins to one processor but does not restrict the use of memory, whereas platform-level virtual machines do. Both do not constrain the provider [TTL02, BDF+03]. At least each participant must be able to check for compliance with specifications. Trust is usually necessary and mechanisms like reputation facilitate it.

Contract mapping

The process by which users and providers agree on containers requires search and decisions supervised by a neutral third party or performed by a peer-to-peer mechanism [RLS98]. It can be based on a market, real or simulated.

Contract mapping is the match between a user and a provider and the issuing of containers that bind them. Symmetric Mapping specifies that contract mapping is separated from the rest of the allocation. Contract mapping is a function:

Front and back mappings

Containers leave adequate flexibility on both sides. A user independently schedules its tasks and a provider its resources.

Symmetric Mapping respects the possibility that users schedule their tasks according to their objective and their knowledge on how to achieve it. In addition, users can adapt to unplanned task behaviors and faulty resources by dynamic reallocation, or checkpointing and migration.

To the provider, a container is non obstructive. It clearly isolates user access and limits the provider's commitment. In addition, a container is loosely-coupled to the provider's resources. While containers constrain resource types, providers choose physical resources. Since commitments are known in advance, resource consolidation can be planned [PZU+07]. Since back mapping is dynamic, providers can freely administrate their resources, grant access to local users or for regular maintenance operations, and react to some light cases of resource faults.

2.4.4 Relevance of Symmetric Mapping

Whether Symmetric Mapping should be used instead of another design must be decided on a case by case basis. Relevant considerations include:

Autonomy. The more autonomous providers are from users, the more useful Symmetric Mapping is with regards to Metascheduling or Late Binding.

Expertise. Whether users allocate their tasks manually or use a decision system, the relevance of Symmetric Mapping or Late Binding with regards to Metascheduling depends on their ability to perform allocations that impacts their perceived value of the resources. Similarly, a provider who does not manage energy consumption or maintain servers will not have strong diverging requirements.

Liquidity. The more tasks and the more resources, the higher the gain from optimized allocation on each side.

Trust. Participants must trust a system that addresses their concerns on their behalf. Otherwise, the use of Symmetric Mapping is relevant.

Sensitivity to cost and value. If users can obtain enough resources at no charge, and if they do not make a difference between heterogeneous resources, they do not have the incentives that justify the use of Symmetric Mapping. The situation is similar if providers have fixed operating budgets and if they do not make other use of their resources.

2.5 Accuracy and benefits

In reality, users and providers do not know the precise information and mechanisms that determine the value of their objective functions. In addition, accurate optimization might be intractable. These limitations may weigh against an architecture that gives participants the responsibility of their objectives. The following simulation suggests that even with approximate knowledge and basic algorithms, participants benefit from the separation of responsibilities that we propose.

We consider a user interested in minimum makespan and a provider interested in minimum obstruction.

2.5.1 Resources and tasks

For simplicity, a resource is a server, and two tasks do not run together on a server. $\forall t \in Tasks$, $\forall r \in Resources$, $\exists \rho(t, r) \in \mathbb{R}^+$, $\forall \tau \in Time$,

$$\rho(t, a, \tau) = \begin{cases} \rho(t, r) & \text{if } a(t, r, \tau) = true \\ 0 & \text{otherwise} \end{cases}$$

The throughput, $\rho(t, r)$ is the number of instructions per second.

In order to reflect correlations between throughputs on the same server, we write:

$$\rho(t, r) = \rho_{c/s}(r)\rho_{i/c}(t, r)$$

$\rho_{c/s}(r)$ is cycle rate of r, i.e. the number of hardware threads times the frequency. $\rho_{c/s}(r)$ reflects the maximum absolute performance of r. $\rho_{i/c}(t, r)$ is the number of instructions of t per cycle of r. $\rho_{i/c}(t, r)$ reflects the relative width of the bottleneck, or affinity between r and t.

In the simulation, we generate a random cycle rate $\rho_{c/s}(r)$ for each resource r, a random task size $s(t)$ in number of instructions for each task t, and a random affinity $\rho_{i/c}(t, r)$ for each couple (t, r).

In practice, participants have an approximate understanding of the mechanisms that determine performance. To account for it, we suppose that the participants do not know $\rho_{c/s}(r)$, $\rho_{i/c}(t, r)$ and $s(t)$. Instead, they have a notion of task length l_t and server performance p_r obtained by observation. l_t is the observed execution time of task t in average. The user who owns t knows l_t.

$$l_t = \frac{1}{|Resources|} \sum_{r \in Resources} \frac{s(t)}{\rho(t, r)}$$

p_r is the observed average ratio between a task length and its actual execution time on the server. The provider who owns r knows p_r, the average server "performance".

$$p_r = \frac{1}{|Tasks|} \sum_{t \in Tasks} \frac{l_t \rho(t, r)}{s(t)}$$

To simulate obstruction to the provider, for every server r we pick random time periods $T_r \subset Time$ such that $\mathbf{eager_p}(r, \tau) = 1$ if $\tau \in T_r$ and 0 otherwise, and such that $\mathbb{P}(\tau \in T_r)$ is specified in the input of the simulation, for example 10%.

2.5.2 Algorithms

For a user u and a provider p, the contract $C_{u,p}$ says: *Starting at t_0, u obtains the monopoly on up to N_p servers of p as long as u uses these servers to process any of the N_u specified tasks.*

We compare makespan and obstruction in the following cases.

1. A third party controls the allocation, independently from user and provider objectives, and with less information on resources and tasks.
2. The user controls the allocation, in a rational and selfish manner, with less dynamic control as the provider would have had.
3. The provider controls the allocation, in a rational and selfish manner.
4. Provider and user control their respective schedules on containers compatible with their contract, and such that a schedule determines the perceived value.
5. Previous cases are compared with a theoretically attainable measure that uses exact values of performance, affinity and task size.

The allocation carried out by a third party is implemented as a static random assignment of the tasks to N_p random servers.

With full control, user u picks each task t in order of decreasing l_t and assigns it to a server of minimum cumulated lengths $\sum_{t' \in T_r} l_{t'}$. $T_r \subset Tasks$ is the set of tasks assigned to server r.

With full control, p starts with the same random allocation as a third party. When willing to preempt a busy server, p moves its tasks to a random free server if there is one.

A valid set of containers has N_p containers. Each of them says: *Starting at t_0, p provides u with the possibility to compute one of the N_u specified tasks with a performance p_c that does not vary so much that it affects the value perceived by the user.* We consider that p_c must remain within the minimum standard deviation of performance σ.

$$\sigma = \min_{r \in Resources} \sqrt{\frac{1}{|Tasks|} \sum_{t \in Tasks} \left(\frac{l_t \rho(t,r)}{s(t)} - p_r \right)^2}$$

Given these containers, user u picks each task t in order of decreasing l_t and assigns it to a container of minimum predicted load $(\sum_{t' \in T_c} l_{t'})/p_c$. $T_c \subset Tasks$ is the set of tasks assigned to container c.

For $r \in Resources$, we write its neighborhood $\mathcal{N}_r$.

$$\mathcal{N}_r = \{r' \in Servers | r' \neq r \text{ and } |p_{r'} - p_r| \leq \sigma\}$$

p can move containers inside the same neighborhood.

Theoretical attainable values are measured from the following allocation. Initially, each task t is picked in order of decreasing $s(t)$ and assigned to a container of minimum size $\sum_{t' \in T_c} s_{t'}$. Whenever a task t starts or the provider wants to preempt a server r, the corresponding container is moved to the free server of best throughput $\rho(t, r)$.

2.5.3 Results

Simulations are written in Python using test-driven development methods. The code is released under Artistic License 2 and available on a public repository[10].

Each figure shows the makespan and obstruction on five runs. Each run corresponds to a set of tasks and resources, and a number of containers. On all figures, all five runs have the same number of tasks, resources, containers, statistical distribution of task sizes, cycle rates, instructions per cycle.

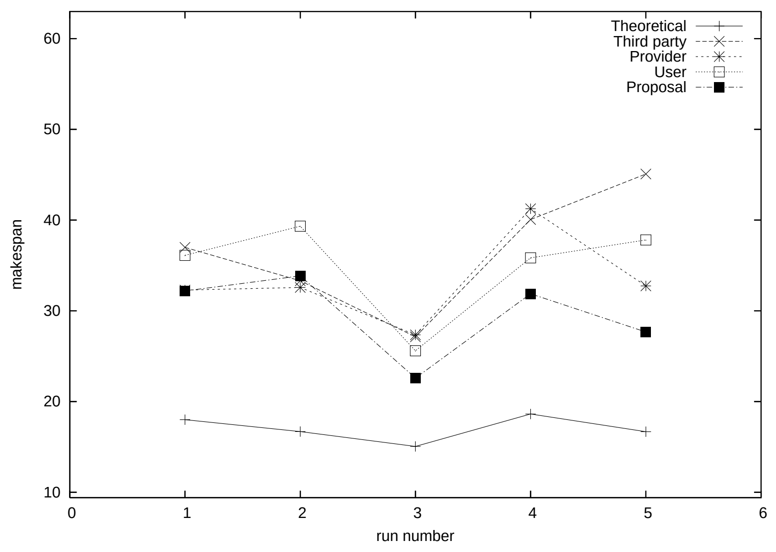
Eagerness is identically generated in all cases. On every server, periods of availability ($\mathbf{eager_p}(r, \tau) = 0$) follow a normal distribution of average 10 hours and standard deviation 5 hours. This is a reasonable period, e.g. night time, during which a server is available without interruption to external use. Periods of providers potential occupation ($\mathbf{eager_p}(r, \tau) = 1$) follow a normal distribution of 5 hours in average and 2 hours in standard deviation, which is the time typically needed for server maintenance or interactive use.

Figures 2.3, 2.4 and 2.5 are taken with heterogeneous tasks. $s(t)$ is uniform from $7,200$ to $18,000$ billion instructions. This corresponds to two to five hours on a one giga-ops system, which is common to observe for a single submission on a CPU.

Figures 2.3, 2.4 and 2.6 are taken with heterogeneous resources. $\rho_{c/s}(r)$ is uniform from 1 to 6 billion cycles per second. $\rho_{i/c}(t, r)$ is uniform from 0.5 to 0.75 instructions per cycle. This can be observed with performance monitoring software on commodity servers.

Figure 2.5 is taken with homogeneous resources. Server performance $\rho_{c/s}(r)$

10. code.google.com/p/symmetric-mapping



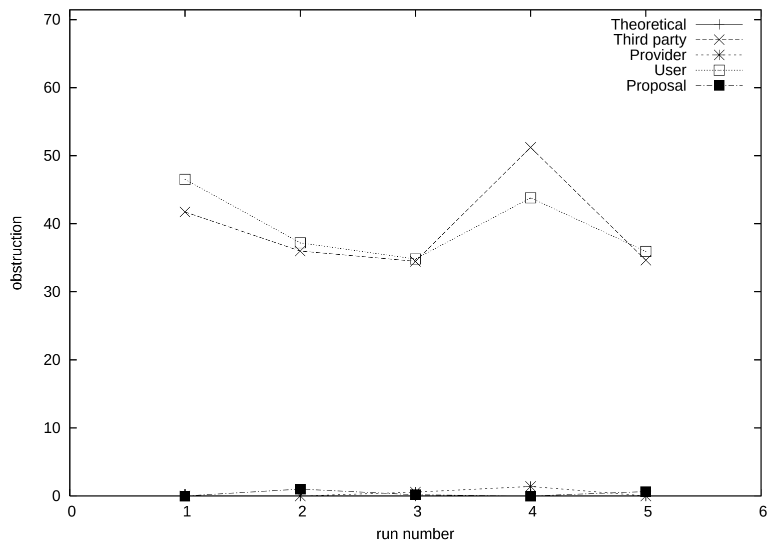
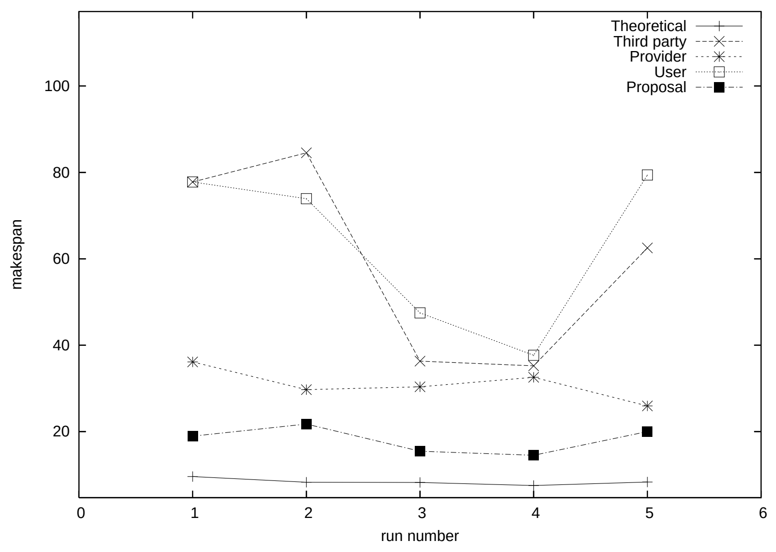


Figure 2.3: Small liquidity



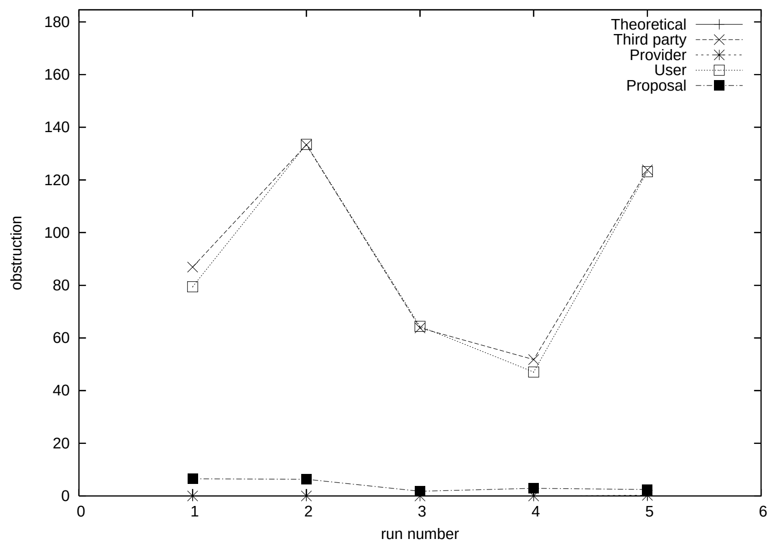
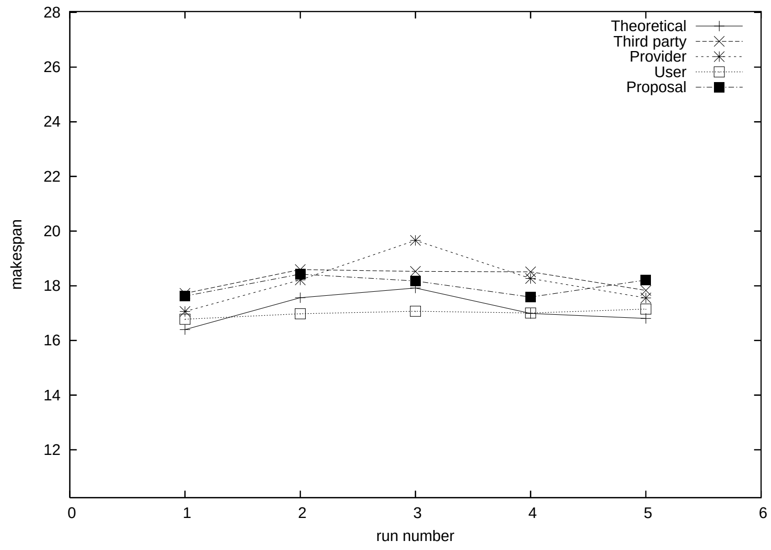


Figure 2.4: Large liquidity



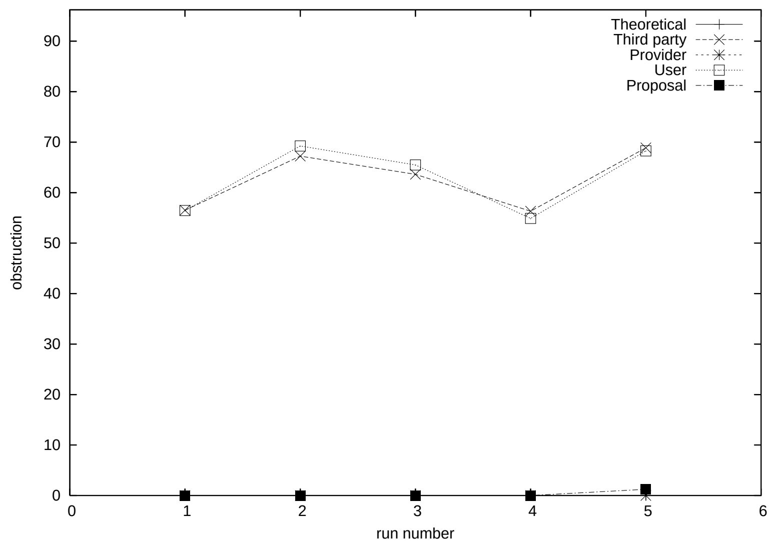
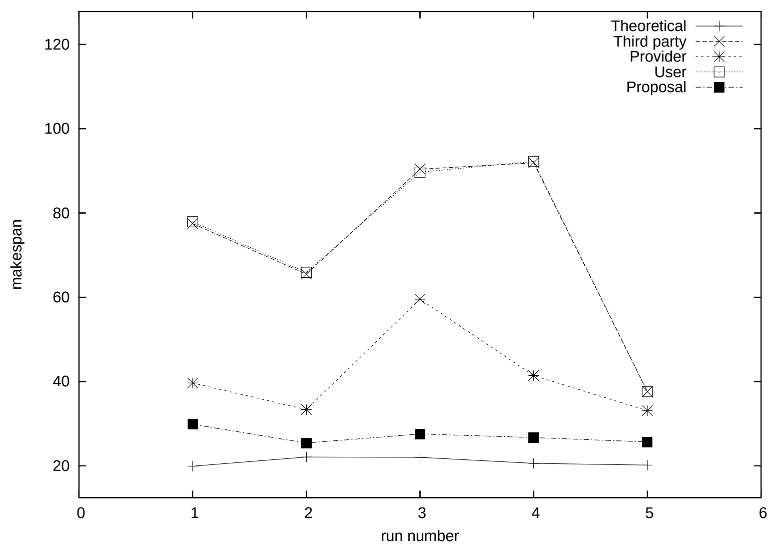


Figure 2.5: Identical resources, large liquidity



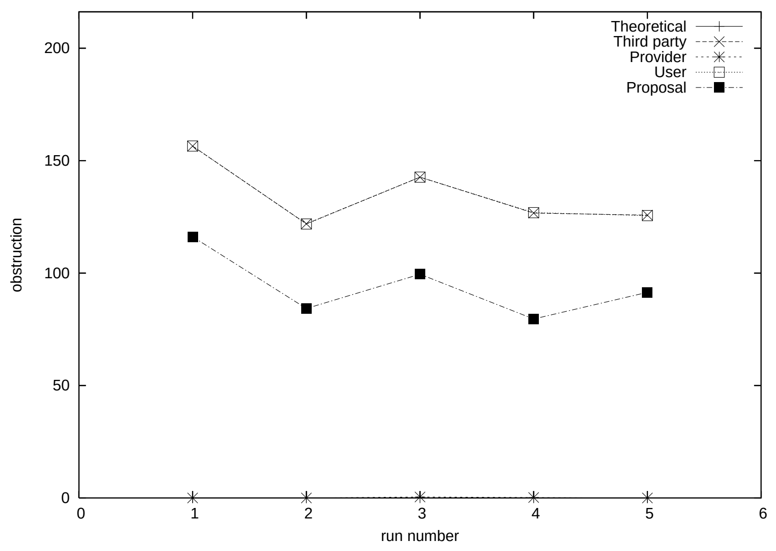


Figure 2.6: Identical tasks, large liquidity

is constant $= 8$ and $\rho_{i/c}(t,r)$ is uniform from 0.5 to 0.55. $\rho_{i/c}(t,r)$ denotes the affinity between task t and server r. Since the processor is always the same, the task is the only factor of variations and the variation is lower than with heterogeneous resources.

Figure 2.6 is taken with homogeneous tasks. $s(t)$ is constant $= 18,000$ billion instructions.

Figure 2.3 is taken with 100 tasks, 20 servers and 5 containers, and the others figures with 300 tasks, 60 servers and 15 containers.

In all cases, the provider obtains a better obstruction than the user or the third party. The user obtains a better makespan on homogeneous resources only. In other cases user efforts are not evidently better than random because task length has little correlation with actual task execution time. The user advantage in the case of homogeneous resources is marginal. It suggests that the gain from *load balancing* in homogeneous computing is marginal compared to the gain from *mapping* in heterogeneous computing, taking into account affinities between tasks and resources.

Data marked *Proposal* corresponds to the use of containers and the dispatch of responsibilities between user and provider. The obstruction is similar to the obstruction obtained by a provider, except with homogeneous tasks. The makespan is the best, except on homogeneous resources, where user makespan is the best. In cases where tasks or resources are homogeneous, σ is small and the number of servers in the neighborhood is limited.

The difference with theoretical values illustrates the effect of an approximate performance model. These experiments suggest that even when the underlying models of the participants are approximate, it is valuable to appropriately divide the allocation and dispatch responsibilities. Benefits increase with heterogeneity and liquidity.

2.6 The model

This section introduces a new model of resource allocation. This model provides the foundations of Symmetric Mapping. Its novelty is to consider dynamic scheduling under constraints.

Section 2.6.1 defines an allocation as the association of a resource and a task at a given time, specified with a boolean function. Section 2.6.2 splits an allocation into two functions, one that specifies how resources are scheduled, and one that specifies how tasks are scheduled. Section 2.6.3 introduces a specifications as a predicate that a given allocation must satisfy. Section 2.6.4 defines the value of an allocation as perceived by a participant.

2.6.1 Allocations

An allocation is an application that takes a task, a resource and a time, and returns $true$ if the task is allocated to the resource at that time, $false$ otherwise.

$$a : Tasks \times Resources \times Time \longrightarrow \{true, false\}$$

We write $\mathcal{F}(E, G)$ the set of applications from set E to set G. Let A be the set of allocations.

$$A = \mathcal{F}(Tasks \times Resources \times Time, \{true, false\})$$

We call X the set of participants. If $u \in X$ is a resource user that owns $Tasks_u \subset Tasks$ and $p \in X$ is a resource provider that owns $Resources_p \subset Resources$, we define $A_{u,p}$ the set of allocations that involve u and p.

$$A_{u,p=}\mathcal{F}(Tasks_u \times Resources_p \times Time, \{true, false\})$$

We define the null allocation $\emptyset \in A$ such that $\forall(t, r, \tau) \in Tasks \times Resources \times Time$:

$$\emptyset(t, r, \tau) = false$$

We define an operation on allocations, that takes two allocations and returns the allocation that is the combination of the two. We call it their *merger*. Operator $\vee$ between booleans is logical *or*. For all $(a_1, a_2) \in A^2$, the merger of a_1 and a_2 is $a_1 \vee a_2$ such that $\forall(t, r, \tau) \in Tasks \times Resources \times Time$:

$$(a_1 \vee a_2)(t, r, \tau) = a_1(t, r, \tau) \vee a_2(t, r, \tau)$$

The merger of two sets of allocations is the set of all possible mergers with one allocation from each set. If E is a set, we write $\mathcal{P}(E)$ the set of subsets of E. For two sets of allocations $(A_1, A_2) \in \mathcal{P}(A)^2$, we define their merger $A_1 \vee A_2$.

$$A_1 \vee A_2 = \{a \in A | \exists(a_1, a_2) \in A_1 \times A_2, a = a_1 \vee a_2\}$$

2.6.2 Schedules

In this section we divide allocations in parts. Each part is called a *schedule*. A schedule involves either tasks or resources but not both. A resource schedule tells if a given resource is involved at a given time, and a task schedule tells if a given task is involved at a given time.

A resource schedule returns $true$ for a couple (r, τ) where resource r is assigned at time τ; and a task schedule returns $true$ for a couple (t, τ) where task t is assigned at time τ.

The goal is to isolate every part of the allocation that interests a single participant. In this section we introduce a notation that allows to represent participants as mathematical objects. In fact, a participant is represented with an function of allocations, or a *projector* that, given an allocation, returns the part of the allocation of interest to the participant.

Let u be a user. We define its projector $\mathbf{p_u}$

$$\mathbf{p_u} : \begin{cases} A \longrightarrow \mathbf{p_u}(A) \\ a \longmapsto \mathbf{p_u} \circ a \end{cases}$$

such that:

$$\mathbf{p_u}(A) = \mathcal{F}(Tasks_u \times Time, \{true, false\})$$

and $\forall a \in A, \forall (t, \tau) \in Tasks_u \times Time$,

$$\mathbf{p_u} \circ a(t, \tau) = \begin{cases} true & \text{if } \exists r \in Resources | a(t, r, \tau) \\ false & \text{otherwise} \end{cases}$$

$\forall a \in A$, $\mathbf{p_u} \circ a$ is the task schedule of allocation a.

Let p be a provider. We define its projector $\mathbf{p_p}$.

$$\mathbf{p_p} : \begin{cases} A \longrightarrow \mathbf{p_p}(A) \\ a \longmapsto \mathbf{p_p} \circ a \end{cases}$$

Such that:

$$\mathbf{p_p}(A) = \mathcal{F}(Resource_p \times Time, \{true, false\})$$

and $\forall a \in A, \forall (r, \tau) \in Resources_p \times Time$,

$$\mathbf{p_p} \circ a(r, \tau) = \begin{cases} true & \text{if } \exists t \in Tasks | a(t, r, \tau) \\ false & \text{otherwise} \end{cases}$$

$\forall a \in A$, $\mathbf{p_p} \circ a$ is the resource schedule of allocation a.

Projectors allow to extract the part of the allocation of interest to a user or a provider. It is also possible to reconstruct an allocation from two projections. The following shows how a schedule of tasks and a schedule of resources together define an allocation.

Let u be a user and p a provider. $\forall a_u \in \mathbf{p_u}(A), \forall a_p \in \mathbf{p_p}(A)$, we define[11] $a_u \wedge a_p$ the expansion of a_u and a_p such that $\forall (t, r, \tau) \in Tasks_u \times Resources_p \times Time$,

$$(a_u \wedge a_p)(t, r, \tau) = a_u(t, \tau) \wedge a_p(r, \tau)$$

An allocation is the expansion of a resource schedule and a task schedule. If a resource is assigned at a given time in the allocation, it is assigned at the same time in the corresponding resource schedule. If a task is assigned at a given time in the allocation, it is assigned at the same time in the corresponding task schedule.

2.6.3 Specifications

Specifications represent "anything that can be said about an allocation". A specification is a predicate. A specification applies to an allocation if the specification and the allocation are said to *match*. The **match** function returns *true* when applied to them.

In fact we introduce specifications to represent the contracts between users and providers that justify the existence of allocations. The allocation matches the specification that represents the contract between involved users and providers.

We call *Specs* the set of specifications. We introduce a function **match** that says if an allocation is compatible with a specification.

$$\mathbf{match} : A \times Specs \longrightarrow \{true, false\}$$

A specification can be split into constraints on the schedule of each partici-

11. Operator $\wedge$ between booleans is logical *and*.

pant. $\forall s \in \mathit{Specs}, \forall x \in X$, $\exists s_x \in \mathit{Specs}$ such that $\forall a \in A$

$$\mathbf{match}(a, s) \Leftrightarrow \bigwedge_{x \in X} \mathbf{match}(\mathbf{p_x}(a), s_x)$$

If A' is a set of allocations, for readability we define $A'|s$ the subset of A' in which all allocations satisfy s. $\forall A' \in \mathcal{P}(A), \forall s \in \mathit{Specs}$:

$$A'|s = \{a \in A' | \mathbf{match}(a, s)\}$$

We introduce the notion of compatibility of a specification with a set of specifications. A specification can be decomposed into a *compatible* set of specifications.

Let $s \in \mathit{Specs}$, $S \in \mathcal{P}(\mathit{Specs})$. We say that S is compatible with s, and we write $S \vDash s$ if and only if

$$\bigvee_{s' \in S} A|s' = A|s$$

This formula says that by choosing an allocation for each specification of the compatible set, such that the allocation satisfies the specification, and by merging all these allocations, we obtain an allocation that satisfies the initial specification.

2.6.4 Value

In this section we define the *value* of an allocation. It is the function illustrated in section 2.5. It quantifies the outcome of an allocation as perceived by a participant, in terms of how well the allocation reaches the participant's objective and how beneficial it is for the participant.

Let x be a participant. x is a user or a provider. We write $\mathbf{value_x}$ the value of an allocation for participant x, i.e. how much x gains from the allocation.

$$\mathbf{value_x} : A \longrightarrow \mathbb{R}$$

The value of the null allocation is null: $\forall x \in \{u, v\}$,

$$\mathbf{value_x}(\emptyset) = 0$$

The inclusion-exclusion principle applies: $\forall x \in \{u, v\}$, $\forall (a_1, a_2) \in A^2$,

$$\begin{aligned}\mathbf{value_x}(a_1 \vee a_2) = \mathbf{value_x}(a_1) + \mathbf{value_x}(a_2) \\ - \mathbf{value_x}(a_1 \wedge a_2)\end{aligned}$$

If x is a user, $\mathbf{value_x}$ is generally positive because a user gains in having their tasks processed. If x is a provider, $\mathbf{value_x}$ is generally negative because processing tasks generates a cost.

We write $\mathbf{value_{x\downarrow}}$ the guaranteed value of a set of allocations, knowing one allocation of the set will be effective.

$$\mathbf{value_{x\downarrow}} : \begin{cases} \mathcal{P}(A) \longrightarrow \mathbb{R} \\ \quad A' \longmapsto \min\limits_{A'} \mathbf{value_x} \end{cases}$$

The guaranteed value is the minimum value on the set of allocations that contain the effective allocation.

2.7 MAD resource allocation problem

This section formalizes the hypotheses and the objective of grid resource allocation. The formulation is an analogy with the MAD [12] model for fault tolerance in distributed systems. We propose reasonable hypotheses on the behavior of autonomous participants in resource allocation. The objective is to independently optimize the value as perceived by every participant.

2.7.1 Hypotheses

1. Participants are selfish and rational.
2. A user can only schedule its own tasks and a provider can only schedule its own resources.
3. Participant $x \in \{u, p\}$ is bound in a contract $S_{u,p}$.

This summarizes as follows. $\forall S \in \mathcal{P}(Specs)$ compatible with $S_{u,p}$ ($S \vDash S_{u,p}$ as defined in section 2.6.3), $\forall s \in S$, $x \in \{u, p\}$ chooses the schedule:

$$x(s) = \arg \max_{p_x \in \mathbf{p_x}(A|s)} \mathbf{value_{x\downarrow}} \left(\mathbf{p_x}^{-1}(\{p_x\}) | s \right)$$

12. Multiple Administrative Domains

It means that a participant chooses a schedule that satisfies her constraints, and that guarantees the best value as she perceives it, provided specifications are satisfied.

This hypothesis is a statement of responsibility rather than capability. The efforts of a participant will be based on her perception of what is or is not valuable, whether her perception is correct or not. However, participants must have enough confidence in their own assessments in order to find it worthwhile to perform their own scheduling.

$S_{u,p}$ is normally negotiated to make sure that some requirements are fulfilled. If there is a minimum acceptable value $vmin_x$ for each participant $x \in \{u, p\}$

$$\forall a \in A|S_{u,p}, \mathbf{value_x}(a) \geq vmin_x$$

2.7.2 Objective

We want to find an allocation $\bar{a}$ that independently maximizes all perceived values among allocations that satisfy the contracts. For all participant x bound in contract S_x,

$$\bar{a} = \arg\max_{A|S_x} \mathbf{value_x}$$

2.8 A solution to the MAD problem for grids

We propose a solution to the problem of multiple administrative domains in resource allocation. This solution relies on the existence of containers. A container determines the perceived value of the part of the allocation that it contains.

2.8.1 Containers

Our proposal relies on the existence of a set of specifications that follows certain properties. We call *Containers* this set. $Containers \subset Specs$. The following hypotheses define a notion of containment that allows to solve the MAD problem for grids.

1. A container forbids over- or under-subscription.
   $\forall c \in Containers,\ \exists T_c \subset Time,\ \forall a \in A|c,$

$$\forall \tau \in T_c,\ \exists (t,r) \in Tasks \times Resources, a(t,r,\tau)$$
$$\forall \tau \in Time \backslash T_c,\ \forall (t,r) \in Tasks \times Resources, \neg a(t,r,\tau)$$

T_c is called the container lifetime.

2. Among allocations that satisfy a container, schedules determine the perceived value.
   $\forall x \in \{u, p\}$, $\forall c \in \mathit{Containers}$, $\forall (a_1, a_2) \in (A_{u,p}|c)^2$,

$$\mathbf{p_x}(a_1) = \mathbf{p_x}(a_2) \Rightarrow \mathbf{value_x}(a_1) = \mathbf{value_x}(a_2)$$

3. For every specification, there is a compatible non redundant set of containers. $\forall s \in \mathit{Specs}$, $\exists C \in \mathcal{P}(\mathit{Containers})$ such that

$$A|s = \bigvee_{c \in C} A|c$$

   and

$$\forall (c_1, c_2) \in C^2 | c_1 \neq c_2, \quad \forall (a_1, a_2) \in A|c_1 \times A|c_2, a_1 \wedge a_2 = \emptyset$$

2.8.2 Protocol

If *Containers* exists, it is possible to independently optimize the value perceived by each participant. This is done by giving participants the possibility to pick a set of non redundant containers $C_{u,p} \in \mathcal{P}(\mathit{Containers})$ compatible with their contract: $C_{u,p} \vDash S_{u,p}$. From property 3 of *Containers*, $C_{u,p}$ exists.

In the following, we show that this protocol yields a unique allocation, which satisfies the contract and the optimality objective.

2.8.3 Correctness

Theorem 1. *The protocol yields a unique allocation.*

$$\bar{a} = \bigvee_{c \in C_{u,p}} u(c) \wedge p(c)$$

Proof. Let $c \in C$, $H = \mathbf{p_u}^{-1}(\{u(c)\}) \cap \mathbf{p_p}^{-1}(\{p(c)\})$. We will prove that $u(c) \wedge p(c) = \bigvee_{a \in H} a$.

Part 1 We show that $u(c) \wedge p(c) \in H$.

First, we show that $u(c) \wedge p(c) \in \mathbf{p_u}^{-1}(\{u(c)\})$, i.e. $\mathbf{p_u}(u(c) \wedge p(c)) = u(c)$. Let $(t, \tau) \in \mathit{Tasks} \times \mathit{Time}$.

Case $\mathbf{p_u}(u(c) \wedge p(c))(t,\tau) = true$. $\exists r \in Resources$ such that $(u(c) \wedge p(c))(t,r,\tau) = true$. *A fortiori*, $u(c)(t,\tau) = true$.

Case $\mathbf{p_u}(u(c) \wedge p(c))(t,\tau) = false$; *ad absurdum*. Suppose $u(c)(t,\tau) = true$. Necessarily, $\forall r' \in Resources$, $p(c)(r',\tau) = false$, i.e. $\exists a \in A_{u,p}|c$, $\forall (r',t') \in Resources \times Tasks$, $a(t',r',\tau) = false$. It means that $\tau \notin T_c$, and therefore $u(c)(t,\tau) = false$.

Second, the proof of $u(c) \wedge p(c) \in \mathbf{p_p}^{-1}(\{p(c)\})$ is analogous.

Part 2 We show that $\forall a \in H, a \vee (u(c) \wedge p(c)) = u(c) \wedge p(c)$. Let $a \in H$, $(t,r,\tau) \in Tasks \times Resources \times Time$.

Case $(a \vee (u(c) \wedge p(c)))(t,r,\tau) = true$; *ad absurdum*. Suppose $(u(c) \wedge p(c))(t,r,\tau) = false$. It yields $a(t,r,\tau) = true$. It follows $(\mathbf{p_u} \circ a)(t,\tau) = true$. Since $a \in \mathbf{p_u}^{-1}(\{u(c)\})$, $\mathbf{p_u} \circ a = u(c)$. Therefore $u(c)(t,\tau) = true$. Similarly, $(\mathbf{p_p} \circ a)(r,\tau) = true$ and therefore $u(p)(r,\tau) = true$. Finally, $(u(c) \wedge p(c))(t,r,\tau) = true$.

Case $(a \vee (u(c) \wedge p(c)))(t,r,\tau) = false$. It follows $a(t,r,\tau) = false$ and $(u(c) \wedge p(c))(t,r,\tau) = false$.

Part 3 $u(c) \wedge p(c) \in A|c$ because

$$\mathbf{p_u}(u(c) \wedge p(c)) = u(c) \in \mathbf{p_u}(A)|c_u$$
$$\mathbf{p_p}(u(c) \wedge p(c)) = u(c) \in \mathbf{p_p}(A)|c_p$$

□

Theorem 2. *The obtained allocation satisfies a set of specifications compatible with the contract that binds the participants.*

$$\bar{a} \in A|C_{u,p} \text{ and } C_{u,p} \vDash S_{u,p}$$

Proof.

$$\bar{a} = \bigvee_{c \in C_{u,p}} u(c) \wedge p(c) \in \bigvee_{c \in C_{u,p}} A|c = A|C_{u,p}$$

□

Theorem 3. *The obtained allocation is optimal.*

$$\forall x \in \{u,p\}, \bar{a} = \arg \max_{A|S_{u,p}} \mathbf{value_x}$$

Proof. Let $c \in C_{u,v}$, $x \in \{u, v\}$. From hypothesis,

$$x(c) = \arg \max_{p_x \in \mathbf{p_x}(A|c_x)} \mathbf{value_{x\downarrow}} \left(\mathbf{p_x}^{-1}(\{p_x\})|c\right)$$

$$\mathbf{value_{x\downarrow}} \left(\mathbf{p_x}^{-1}(\{x(c)\})|c\right) = \max_{p_x \in \mathbf{p_x}(A|c_x)} \mathbf{value_{x\downarrow}} \left(\mathbf{p_x}^{-1}(\{p_x\})|c\right)$$

Let $p_x \in \mathbf{p_x}(A|c_x)$,

$$\mathbf{value_{x\downarrow}} \left(\mathbf{p_x}^{-1}(\{p_x\})|c\right) \leq \mathbf{value_{x\downarrow}} \left(\mathbf{p_x}^{-1}(\{x(c)\})|c\right)$$

$$\min_{a_2 \in \mathbf{p_x}^{-1}(\{p_x\})|c} \mathbf{value_x}(a_2) \leq \min_{a_1 \in \mathbf{p_x}^{-1}(\{x(c)\})|c} \mathbf{value_x}(a_1)$$

$\forall a_1 \in \mathbf{p_x}^{-1}(\{x(c)\})|c$, $\exists a_2 \in \mathbf{p_x}^{-1}(\{p_x\})|c$,

$$\mathbf{value_x}(a_2) \leq \mathbf{value_x}(a_1)$$

From containers property 2, $\forall a \in \mathbf{p_x}^{-1}(\{p_x\})|c$,

$$\mathbf{value_x}(a) = \mathbf{value_x}(a_2)$$

Therefore, $\forall a_1 \in \mathbf{p_x}^{-1}(\{x(c)\})|c$, $\forall a \in \mathbf{p_x}^{-1}(\{p_x\})|c$,

$$\mathbf{value_x}(a) \leq \mathbf{value_x}(a_1)$$

It simplifies as $\forall a_1 \in \mathbf{p_x}^{-1}(\{x(c)\})|c$, $\forall a \in A|c$,

$$\mathbf{value_x}(a) \leq \mathbf{value_x}(a_1)$$

Since $u(c) \wedge p(c) \in \mathbf{p_x}^{-1}(\{x(c)\})|c$,

$$\mathbf{value_x}(a) \leq \mathbf{value_x}(u(c) \wedge p(c))$$

It means that:

$$\mathbf{value_x}(u(c) \wedge p(c)) = \max_{A|c} \mathbf{value_x}$$

Since $\forall (c_1, c_2) \in C_{u,v}^2 | c_1 \neq c_2$, $u(c_1) \wedge p(c_1) \in A|c_1$ and $u(c_2) \wedge p(c_2) \in A|c_2$,

$$(u(c_1) \wedge p(c_1)) \wedge (u(c_2) \wedge p(c_2)) = \emptyset$$

Therefore, $\forall x \in \{u, v\}$,

$$\begin{aligned}&\mathbf{value_x}((u(c_1) \wedge p(c_1)) \vee (u(c_2) \wedge p(c_2)))\\ &\qquad = \mathbf{value_x}(u(c_1) \wedge p(c_1)) + \mathbf{value_x}(u(c_2) \wedge p(c_2))\end{aligned}$$

It yields:

$$\mathbf{value_x}\left(\bigvee_{c \in C_{u,v}} u(c) \wedge p(c)\right) = \sum_{c \in C_{u,v}} \mathbf{value_x}(u(c) \wedge p(c))$$

Finally, let $x \in \{u, p\}$

$$\begin{aligned}&\mathbf{value_x}\left(\bigvee_{c \in C_{u,v}} u(c) \wedge p(c)\right)\\ &= \sum_{c \in C_{u,v}} \mathbf{value_x}(u(c) \wedge p(c))\\ &= \sum_{c \in C_{u,v}} \max_{A|c} \mathbf{value_x}\\ &= \sum_{c \in C_{u,v}} \mathbf{value_x}\left(\arg\max_{A|c} \mathbf{value_x}\right)\\ &= \mathbf{value_x}\left(\bigvee_{c \in C_{u,v}} \arg\max_{A|c} \mathbf{value_x}\right)\\ &= \mathbf{value_x}\left(\arg\max_{A|S_{u,v}} \mathbf{value_x}\right)\\ &= \max_{A|S_{u,v}} \mathbf{value_x}\end{aligned}$$

□

2.9 Conclusion

The fact that grids span multiple administrative domains is commonly acknowledged as their distinctive feature among other distributed computing systems. However, prior to this work, the diverging objectives of the participants to grid resource allocation were not taken into account in the architectural design.

A new model for grid resource allocation permits to apply the MAD principles and formalize the problem. It yields a definition of containment that allows to separate the concerns of the participants, and independently optimize their diverging objectives. Specifically, the contracts that bind resource users and providers are decomposed into *containers*. Containers split the allocation into task schedules and resource schedules. A schedule carried out by a participant determines her perceived value of the container.

The outcome of this model is translated in terms of a new architectural design pattern, Symmetric Mapping. Symmetric Mapping separates the concerns of resource users and providers. As a side effect, Symmetric mapping allows to carry out the mapping of tasks to heterogeneous resources according to affinities between tasks and resources, instead of the traditional load balancing on homogeneous resources.

For modeling and tractability issues, participants can only approximately optimize their objective functions. Still, experiments suggest that the proposed separation of concerns yields better outcome than other distributions of responsibilities, and that the gain from traditional load balancing on homogeneous resources can be marginal compared to the gain from dynamically mapping heterogeneous tasks and resources.

The next chapter describes a software framework to implement the Symmetric Mapping pattern based on virtual machines.

Chapter 3

Deploying virtual machines from descriptions

3.1 Introduction

The Symmetric Mapping pattern divides the allocation in two parts. In one part resource providers map containers to resources, and in the other part, users map tasks to containers. While the users part is analogous to the classical task mapping problem, the providers part is not as usual.

This chapter describes SmartDomains, a system for resource providers to carry out their part of the allocation, according to the Symmetric Mapping pattern, and if containers are implemented with virtual machines.

Virtual machines embody allocation constraints that containers can specify. SmartDomains deploys and controls virtual machines from declarative descriptions. SmartDomains takes care of the operational aspect of back mapping. It does not cover the decisional aspect, that consists in the choice of physical resources to back containers. SmartDomains inputs are obtained from the resolution of container descriptions, where abstract resources descriptions are resolved in terms of pointers to physical resources.

Section 3.2 compares SmartDomains with related systems. Section 3.3 presents the choices of underlying software. Section 3.4 explains the operations involved in managing virtual machines. The following sections present how the system handles declarative descriptions to carry out virtual machines placement and configuration (section 3.5), lifecycle management (section 3.6) and configuration attributes lookup (section 3.7). Section 3.8 presents development matters.

3.2 Related systems

A number of advanced enterprise resource management systems interface virtual machine monitors to make VMs centrally and remotely controllable. They address the problems of high availability, configurability and differentiated attribution that a computer center administrator faces to serve local users.

Platform VM Orchestrator [1] manages a wide range of virtualization technologies to maintain policies between users, groups or applications;

Cassat Collage [2] lets the administrator define high-availability goals and manages Xen and VMWare to take action in the case of divergence of the system from its goal state;

OpenQRM , the only open source system in its category, does the same for Xen [Qlu06].

Fusion Dynamics [3] implements SOA (Service Oriented Architecture) standards, enforces predefined high-availability service models and lets the administrator organize a computing center's resources via a drag-and-drop interface.

Several systems present a web based interface for resource users to manage virtual machines, which is not compatible with the principles of Symmetric Mapping. Such systems include Virtual Workspaces, and Enomalism. **Virtual Workspaces**' web-service interface implement grid related protocols for user identification and file transfers [KFFZ05]. **Enomalism**'s web interface is used notably in Amazon EC2 (Elastic Compute Cloud). Other systems let a remote third party operate virtual machines. This is the case for Shirako and COD that support grid services on virtual machines for the benefits of isolation [ICG+06]. This is also the case for XenoServers and PlanetLab that develop a network of isolated, distributed environments to enable research on distributed systems [KMP+04, Fiu06].

To our knowledge, SmartDomains is the only system that manages virtual machines in the background based on declarative descriptions of their configuration and lifecycle.

1. www.platform.com/resources/datasheets/vmov4-ds.pdf, **2008**
2. www.cassatt.com/prod_virtualization.htm, **2007**
3. www.fusiondynamics.com, **2007**

3.3 Choices

Virtualization is the simulation of resources on software. A virtual machine exploits actual resources in the back end and presents them as if they were constitutive of a different system. It presents the interfaces of a different system to users and other systems.

There are basically two types of virtualization. The lighter is also known as emulation or operating system level virtualization. It reproduces the application programming interface of a different operating system. Such emulators include QEMU and User Mode Linux for example [Dik01, Bel05]. Platform virtualization is more constrained. It isolates virtual machine resources at a low abstraction level so that no data exchange between virtual machines can bypass the simulated interface. Typically, platform virtualization requires to partition memory and disk. Such systems are called *Virtual Machine Monitors*. They include for example Xen and VMWare [SVL01, BDF$^+$03].

Platform virtualization provides appropriate isolation to implement containers for Symmetric Mapping [BDF$^+$03, GD07]. Virtual machines (VMs) provide the following benefits.

Software compatibility: By creating a library of customized filesystem images, virtual machines can easily replicate a wide range of resource configurations, satisfying the specific needs of a wide range of applications.

Resource sharing and performance isolation: By running multiple virtual machines on the same physical machine, fractional resources can be allocated, with fine-grained control over the resource consumption of each virtual machine.

Failure isolation: Guest virtual machine failures do not affect the physical node nor other guests.

Decoupling from back-end hardware: Virtual machines are quickly deployed and teared down. They can be migrated without substantial effect on the running tasks they support.

Platform virtualization currently allows for the most varied and precise software and hardware resources encapsulation. Precision and diversity are important for containers to divide contracts that bind resource users and providers with determined perceived value on each side. In addition, resource decoupling and isolation allow for the most significant freedom in mapping resources to containers.

In order to preserve the benefits of the pattern, resource virtualization must not substantially penalize performance. The overhead of a Xen virtual machine is practically null on pure CPU performance, and generally acceptable on I/O performance [MST$^+$05].

Back mapping can be seen as the combination of two processes: resource selection and container deployment. Resource selection is the process of choosing appropriate and available physical resources to endorse specified resource descriptions. Container deployment consists in actually launching and monitoring containers on selected resources. The system we developed, SmartDomains, implements the deployment mechanism. It reads combined container descriptions and resource identifications, configures and deploys pools of Xen virtual machines, and manages their lifecycle according to the descriptions.

SmartDomains is based on SmartFrog, a framework for the configuration, deployment and lifecycle management of distributed software systems, developed by HP Labs [GGL$^+$03, GPJ$^+$07, GD07]. SmartFrog provides:

- A rich description language to express the configuration of distributed software components and to specify their orchestration at run-time using composition and lifecycle components.
- A deployment engine run by a network of SmartFrog daemons that distribute and resolve component descriptions, register components, initiate their deployments, check their liveness and propagate liveness and configuration changes.

SmartDomains provides specialized components to enable the management of virtual machines with SmartFrog.

3.4 Managing virtual machines

SmartDomains looks up container descriptions and resource selections and deploys virtual domains on Linux servers running the Xen hypervisor [BDF$^+$03]. While other VM deployment systems present management interfaces, SmartDomains runs in the background from declarative resource descriptions.

Xen hypervisor adds features to the hardware to support paravirtualization. Figure 3.1 symbolizes the virtual guest domain on top of a physical host, and illustrates the terminology. A server is depicted on the top of the figure, and a virtual domain running on a physical server is shown on the bottom. A virtual domain is an instance of a virtual machine, namely, a running guest virtual

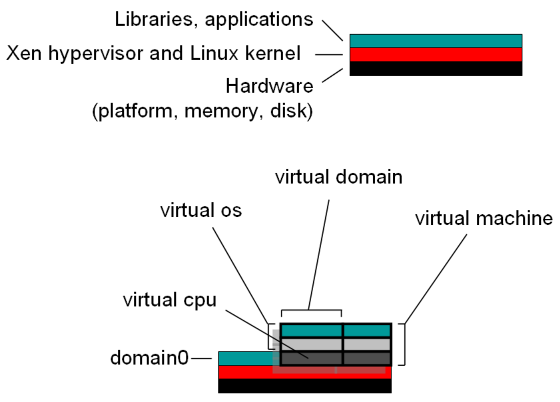


Figure 3.1: A layered view of virtual domains

machine. *domain0* stands for the native operating system of the physical host.

Using virtual machines as containers and SmartDomains for their deployment, back mapping takes place as follows.

– A decision mechanism translates generic container descriptions into descriptions of actual resources that the provider owns. The latter includes names of physical hosts, where to find appropriate OS images, where to find software packages, virtual IP addresses, reserved amounts of disk space, memory, CPU share, etc. This decision relies on the provider's resource cost model [CIL$^+$07, JB07, BF07]. The decision mechanism is outside of the scope of this work.

– SmartDomains triggers appropriate actions. These include transfer, expansion, and configuration of operating system images, mounting images on appropriate devices, running VMs, notifying when ready and monitoring their liveness, and then terminating them, saving and compressing the images. SmartDomains also monitors VM liveness, and sends the information to the allocation system. It updates configurations when the mapping changes.

Once back-end resources are identified, SmartDomains places and configures virtual machines from resolved container specifications.

3.5 Placement and configuration

A deployable system is seen as a tree of components. A leaf of the tree controls a piece of software that needs to be configured, placed, deployed, started and terminated.

Smartfrog language provides a syntax to define the attributes of a component. In the following example, `XenDomain` defines a generic Xen domain. `MyDomain` instantiates a Xen domain by extending `XenDomain` and overriding/defining its attributes.

```
MyDomain extends XenDomain {
    sfProcessHost "oplaslim9.cern.ch";
    ip "123.45.678.90";
    kernel "/boot/vmlinuz-2.6-xen";
    ...
}
```

`sfProcessHost`, `ip` and `kernel` are attribute names. `sfProcessHost` refers to the backing server, `ip` refers to the IP address of the guest domain, and `kernel` gives the location on the host of the guest's kernel.

The `sfProcessHost` attribute is used to place components. When a component description is resolved, the component is committed to the daemon running on `sfProcessHost`.

A component points to the class that defines its behavior. The class implements methods invoked when the component deploys, starts, and terminates.

```
XenDomain extends Prim {
    sfClass "ch.cern.openlab.smartdomains.XenDomainManager.class";
}
```

The code is stored on a code server, accessible from all nodes susceptible to host a component.

Figure 3.2 illustrates how placement and configuration take place.

1. SmartFrog daemons run on the physical hosts.
2. The administrator submits a description to a daemon.
3. Component descriptions are sent to the daemons on the appropriate hosts.
4. Daemons load the code of local components, configure and deploy them (here virtual domains).

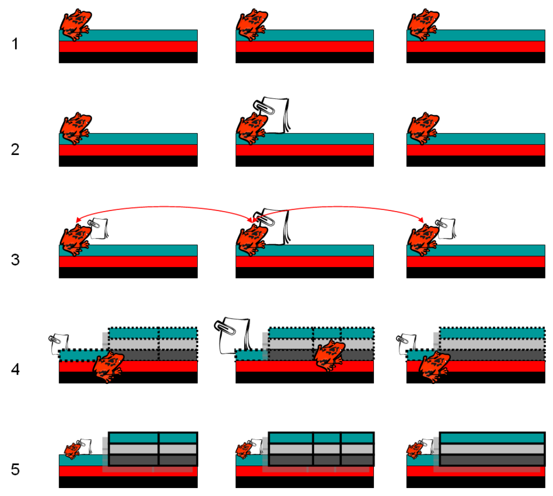


Figure 3.2: Domains placement and deployment by SmartFrog daemons.

5. Daemons wait for liveness checks and possible description updates by other daemons.

Daemons place, configure and run software according to attributes of associated component descriptions and methods of linked objects. Components are organized in a tree to enable appropriate lifecycle management.

3.6 Lifecycle management

In the component's class, methods that run at different stages of its lifecycle are called lifecycle methods. They include `sfDeploy()`, `sfStart()`, and `sfTerminateWith()`. They are invoked respectively at deployment time, start time, and for termination. In the component tree structure, root and intermediate components control the lifecycle of their children components. To do so, their lifecycle methods invoke their children's lifecycle methods.

The `Compound` component is used in the following example to deploy a synchronized pool of Xen domains. When a `Compound` is deployed, it deploys all its children in sequence. When it is started, it starts all its children in sequence. When it is terminated, it terminates all its children in sequence and then completes its own termination.

```
Pool extends Compound {
    domain1 extends XenDomain {...}
    domain2 extends XenDomain {...}
    ...
}
```

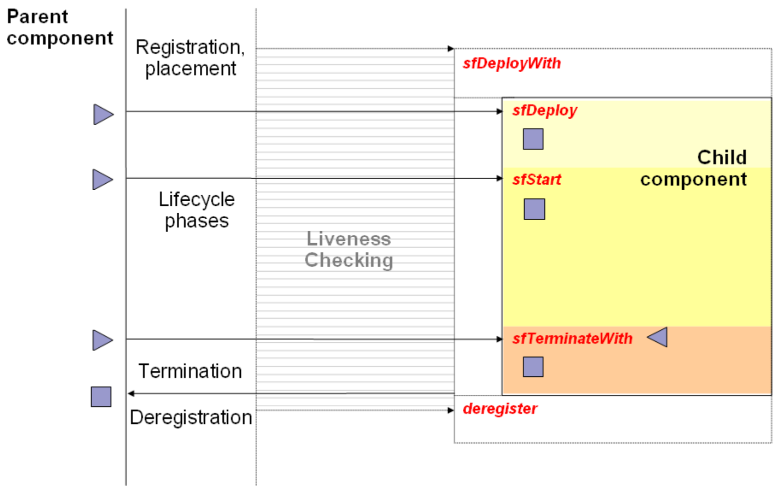


Figure 3.3: Lifecycle management with parent and child components.

Figure 3.3 illustrates the parent-child relationship. A square stands for a method implementation, and a triangle stands for a method call. Since components are generally located on different servers, method calls are carried out with Java Remote Method Invocation (RMI).

SmartDomains components invoke shell commands to perform deployment, starting and termination actions. A filesystem component uses logical volume and file management tools common in Linux. A Xen domain component uses the management interface exported by the Xen hypervisor.

In addition to liveness checks carried out between components, a `XenDomain` component spawns a monitoring thread that regularly pings the virtual machine to check its status and availability. The monitoring thread triggers termination of the `XenDomain` component when the virtual machine appears to be out of reach.

The filesystem image of a virtual domain is defined in a separate component. This allows for multiple ways to mount a filesystem image. The filesystem component is not a child of the virtual domain component because the filesystem exists prior to the virtual domain. However, the virtual domain needs to refer to a filesystem. The following description shows how this is done.

```
LVMStorageBackend extends Prim {
    sfClass "ch.cern.openlab.smartdomains.LVMStorageBackend.class";
}
VM extends Compound {
    filesystem extends LVMStorageBackend {
        baseImage "/data/xen/slc4-smartfrog.img";
    }
    domain extends XenDomain {
        ip "123.45.678.90";
        filesystem LAZY PARENT:filesystem;
    }
}
```

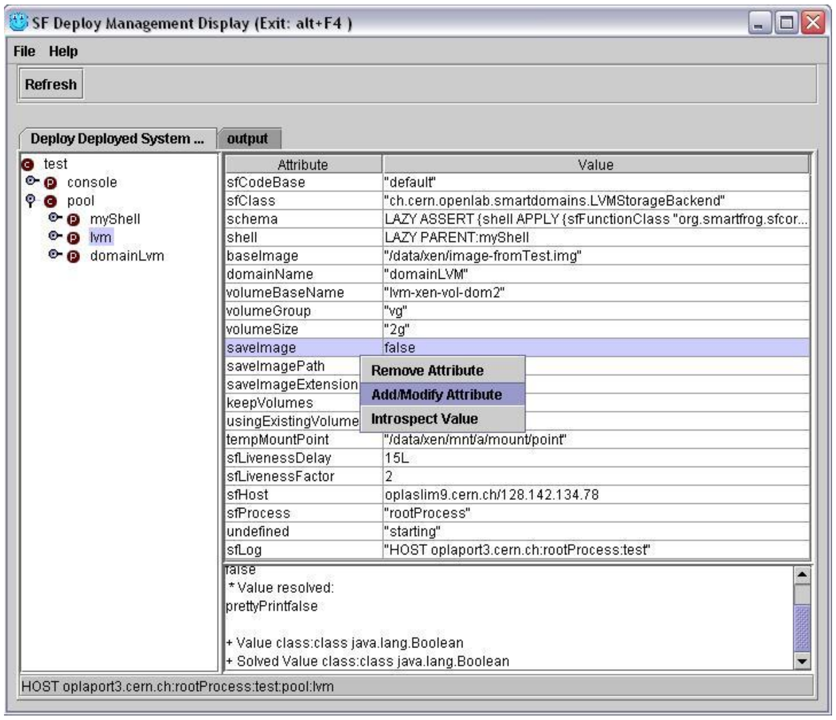
SmartFrog provides the `PARENT:` keyword to refer to an attribute of the parent component. Alternatively, `ATTRIB:` looks up in the hierarchy until the following attribute is found. Keyword `LAZY` prevents a plain copy of the attribute's value or description. Instead, an attribute marked `LAZY` is resolved at deployment time only. Since `Compound` deploys its children in sequence, `VM:filesystem` is deployed before `VM:domain`. Therefore, `VM:domain:filesystem` refers to the deployed filesystem.

The same mechanisms that allow for component referencing across the tree are used to lookup resolved container descriptions.

3.7 Description lookup

A container obtained between a user and a provider makes references to abstract resources. It is resolved when combined with references to selected physical resources that the provider chooses to endorse it.

- Container description attributes include names of virtual hosts, assigned virtual IP address, amounts of memory, disk and swap space, number of virtual CPUs, paths to store images, type of compression.
- Resource selection attributes include addresses of physical hosts, paths to filesystems and mount points.
- Permanent attributes configure the inner workings of deployments.



Attribute	Value
sfCodeBase	"default"
sfClass	"ch.cern.openlab.smartdomains.LVMStorageBackend"
schema	LAZY ASSERT {shell APPLY {sfFunctionClass "org.smartfrog.sfcor...
shell	LAZY PARENT:myShell
baseImage	"/data/xen/image-fromTest.img"
domainName	"domainLVM"
volumeBaseName	"lvm-xen-vol-dom2"
volumeGroup	"vg"
volumeSize	"2g"
saveImage	false
saveImagePath	
saveImageExtension	
keepVolumes	
usingExistingVolume	
tempMountPoint	"/data/xen/mnt/a/mount/point"
sfLivenessDelay	15L
sfLivenessFactor	2
sfHost	oplaslim9.cern.ch/128.142.134.78
sfProcess	"rootProcess"
undefined	"starting"
sfLog	"HOST oplaport3.cern.ch:rootProcess:test"

```
false
 * Value resolved:
prettyPrintfalse

+ Value class:class java.lang.Boolean
+ Solved Value class:class java.lang.Boolean
```

HOST oplaport3.cern.ch:rootProcess:test:pool:lvm

Figure 3.4: SmartFrog console

The following flat, fully resolved description of a virtual domain lists some attributes directly under the root of a tree.

```
sfConfig extends Compound {
    vm extends VM {
        sfProcessHost "physicalHost.cern.ch";
        domainName "physicalHost-virtualDomainName";
        hostname "virtualHost.cern.ch";
        ip "123.45.678.147";
        gateway "999.999.1.1";
        netmask "255.255.0.0"
        ramdisk "no initrd";
        memory 512;
        volumeSize 5g;
        swapSize 512m;
        vcpus 2;
        extra "fastboot nousb";
        baseImage "/data/xen/slc3-smartfrog.tar";
        volumeBaseName "xen-domain-virtualDomainName";
        usingExistingVolumes false;
        keepVolumes true;
        saveImage true;
        saveImageName "saved-image.tar.gz";
        volumeGroup "vg";
        domainLivenessDelay 2000;
        domainLivenessFactor 3;
    }
}
```

Figure 3.4 shows the console provided to the resource administrator to view an up-to-date status of the domains. The administrator can manually modify attributes to override resource selection or permanent configurations. When necessary, changes are propagated across the tree.

3.8 Development and tests

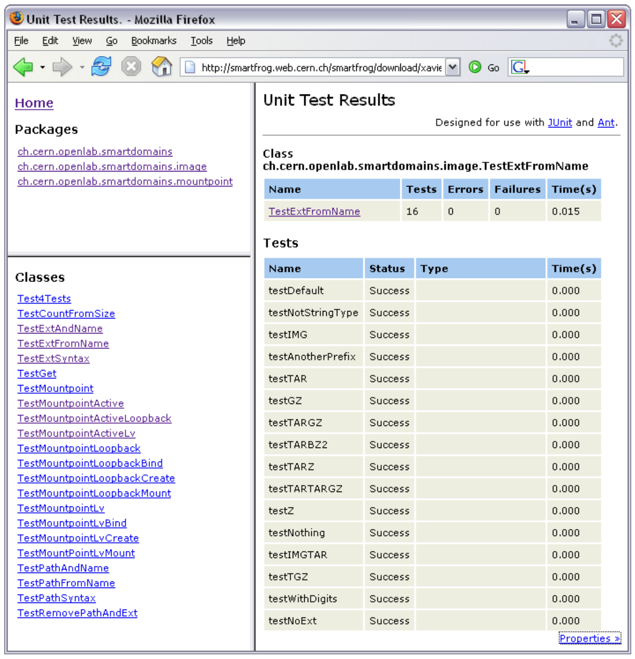
SmartDomains is released under Library General Public License (LGPL) and available on Sourceforge.net, a web based source code repository.

SmartDomains has been used since its early development to create distributed virtual testbeds for the integration tests of gLite services. gLite is a major grid middleware distribution [BBB$^+$05]. gLite testers have used a series of predefined descriptions for every test configuration of interest. Test environments for grid services do not differ from execution environments for grid jobs. This use case has driven us to maintain the code, develop several convenient features, and keep the system operational.

SmartDomains' own code is tested with both unit tests and system tests.

Unit tests do not involve actual deployments. They verify assumptions on methods and classes. Unit tests are the fastest tests to run. Since SmartDomains is all about controlling systems, we refactored its code to separate system interaction from internal logic, and we simulated server responses with *mock objects*, for maximum coverage by unit tests.

System tests involve actual deployments. SmartFrog provides logger components to capture logs from the whole tree. SmartFrog also provides Ant


Unit Test Results. - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://smartfrog.web.cern.ch/smartfrog/download/xavie
Go

Home

Packages

ch.cern.openlab.smartdomains
ch.cern.openlab.smartdomains.image
ch.cern.openlab.smartdomains.mountpoint

Classes

Test4Tests
TestCountFromSize
TestExtAndName
TestExtFromName
TestExtSyntax
TestGet
TestMountpoint
TestMountpointActive
TestMountpointActiveLoopback
TestMountpointActiveLv
TestMountpointLoopback
TestMountpointLoopbackBind
TestMountpointLoopbackCreate
TestMountpointLoopbackMount
TestMountpointLv
TestMountpointLvBind
TestMountpointLvCreate
TestMountPointLvMount
TestPathAndName
TestPathFromName
TestPathSyntax
TestRemovePathAndExt

Unit Test Results

Designed for use with JUnit and Ant.

Class
ch.cern.openlab.smartdomains.image.TestExtFromName

Name	Tests	Errors	Failures	Time(s)
TestExtFromName	16	0	0	0.015

Tests

Name	Status	Type	Time(s)
testDefault	Success		0.000
testNotStringType	Success		0.000
testIMG	Success		0.000
testAnotherPrefix	Success		0.000
testTAR	Success		0.000
testGZ	Success		0.000
testTARGZ	Success		0.000
testTARBZ2	Success		0.000
testTARZ	Success		0.000
testTARTARGZ	Success		0.000
testZ	Success		0.000
testNothing	Success		0.000
testIMGTAR	Success		0.000
testTGZ	Success		0.000
testWithDigits	Success		0.000
testNoExt	Success		0.000

Properties »

Figure 3.5: Html display of unit and system test results

tasks to organize tests and display results on an html page (fig. 3.5).

Performance tests with 48 CPU-intensive benchmark runs did not show significant performance overhead. SmartDomains uses CPU time mostly for liveness checks. We found 0.25% difference in minimum elapsed times between no liveness checks and liveness check every 2 seconds, and between checks every 2 and 10 seconds.

We monitored memory consumption in different scenarii (figure 3.6).

The top left hand side diagram shows that an idle daemon uses about 20MB of memory, and booting and terminating a first VM requires about 3 more MB.

The top right hand side diagram shows that booting 5 VMs simultaneously uses about 24 MB total, and about 40 seconds.

The bottom diagrams show that part of the memory used for a deployment is never released. Every successive deployment adds an additional 300kB to memory usage. We presume that Java will release this memory after a certain threshold, but we did not test it. Otherwise the daemon should be restarted after a few hundred deployments.

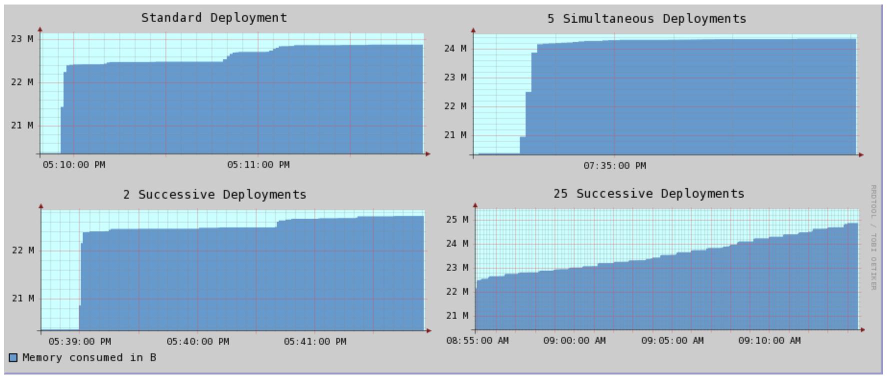


Figure 3.6: Memory measurements.

3.9 Conclusion

The Symmetric Mapping pattern described in chapter 2 specifies that providers deploy isolated resources for users to map their tasks on their own alloted containers. Once decision algorithms are in place to select physical resources in compliance with container specifications, it is possible to deploy and manage virtual machines in the background, as SmartDomains performs this function. Intervention from the resource provider is possible but not required.

Virtual machines deployed in this manner embody certain specifications that constrain resource utilization. They help establish the conditions that separate the concerns of resource providers and resource users. Future virtualization techniques can be expected to increase precision and maneuverability at a lower performance cost, and thus increase the number of cases where adequate containers can be implemented.

Chapter 4

Deploying Permanent User Services

4.1 Introduction

A grid user accesses a potentially vast set of resources. A typical major particle physics collaboration can use at any time tens of thousands of nodes distributed across grid sites anywhere on Earth. To schedule tasks as part of the Symmetric Mapping pattern, a user needs services to run on neighbor containers. These services include:

– Configuration nodes that hold data and instructions to configure and verify user software.

– Allocation nodes replicated for scalability and / or because the allocation algorithm is inherently distributed.

– Monitors placed close to the tasks to avoid delays when monitoring runtime information.

– User-specific proxies and communication services installed on every provider site where containers are located. Such services allow communication through firewalls [SL03].

Subscribed containers are the only support for user services. However, containers are alloted for a limited time. As a result, users must cope with their transience.

In fact, the need for permanent services on transient nodes predates the use of Symmetric Mapping. Powerful grid users, and especially large particle physics collaborations, commonly use the Late Binding pattern [GDJ08]. They submit monitors prior to submitting tasks. The monitors report to user services that operate actual task submission. Currently, there is no viable solution to support user services on grid sites. Providers often dedicate a server, the *VOBox*[1], to each major user.

To demonstrate that permanent services can be run over transient resources, we developed SmartCitizens, a deployment system with an integrated election mechanism.

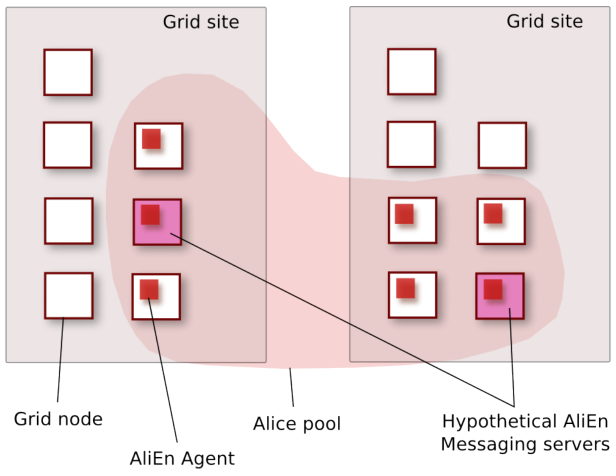


Figure 4.1: User services distributed across grid sites.

SmartCitizens was formed in consultation with ALICE, a physics collaboration and major grid user. With AliEn (ALICE Environment), ALICE introduced user control for task scheduling on grids [SAB$^+$03]. Figure 4.1 symbolizes a pool of AliEn resources, supported by nodes from different grid sites. In order for AliEn agent to seamlessly communicate, a messaging server would run at the intersection of Alice pool and each grid site.

Section 4.2 identifies prior occurrences of the same problem in distributed systems. Section 4.3 presents the SmartCitizens mechanisms in terms of distributed components and their roles. Section 4.4 shows how SmartCitizens components are placed and services deployed.

1. VO means Virtual Organization. A VO is a collaboration of individuals sharing the same goals and usage. In Symmetric Mapping, a VO is considered a single resource user.

4.2 Analogies

The leader election problem occurs when systems following the client-server paradigm are deployed on nodes subject to disconnection. It applies for instance to fault tolerant systems and mobile ad-hoc networks.

- Fault tolerant systems are intended to be resilient to node failures. They re-elect a server node when the previous server fails [FB98].
- In mobile ad-hoc networks, a connection between any of two nodes is inherently transient. Applications are distributed on groups that respect desirable connectivity. The resulting grouping is dynamic, and so are the responsibilities in supporting the applications [MWV00].

Grid computing is a new application as grid users started to take temporary control over remote nodes exposed to pre-emption. In this case, however, although node failure is a possibility, the normal process involves notifying the user before pre-emption. Notifications allow users to designate a new node in advance to take over responsibility, and migrate the service without experiencing noticeable downtime. Also, nodes can be allocated for several hours or days before pre-emption. Election is not meant to be as frequent as with mobile networks.

A leader election algorithm involves participants running the same local algorithm. Any of them can initiate the election, and the algorithm terminates on a consensus under any circumstances. In addition, it is desirable that a election algorithm exhibits fault tolerance itself, and a minimal time and message complexity [Lan77, TS92, PLL00].

The following sections shows how SmartCitizens integrates node election with service redeployment for use in implementations of the Symmetric Mapping pattern. Responsibilities in the election algorithm are dispatched between communicating components.

4.3 Components logic

SmartCitizens components are implemented with SmartFrog [2]. SmartFrog processes component descriptions, deploys components, and configures and deploys associated software. In component hierarchies, parent components deploy their children at specified stages of their lifetime.

2. A framework by HP Labs. www.smartfrog.org.

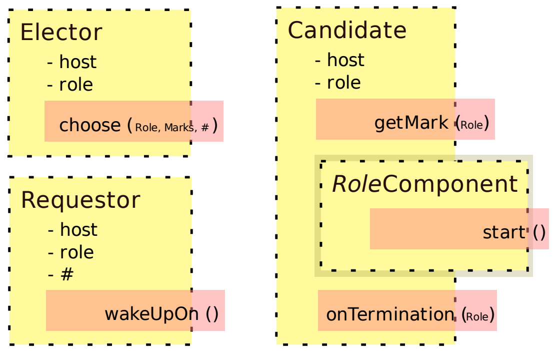


Figure 4.2: SmartCitizens components.

Figure 4.2 represents SmartCitizens components responsible for the election. They consist in **Requestors**, **Candidates** and **Electors**.

Requestors initiate an election. SmartCitizens supports the election of several nodes to run a specified service. A requestor holds a **Role** attribute, that tells the kind of service to place, and a **Number** attribute, that tells the number of nodes that must run the service.

Candidates assess the resources on which they run and make the result known to Electors. A candidate is defined for a specific role. It holds a child component (***Role*Component**), or a reference to it, that configures and deploys the service once elected. Candidates have the ability to act as requestors when notified of incoming pre-emption, or when they detect that the service is terminated. Indeed, the *role*component monitors the service liveness and terminates when the service terminates. Its termination triggers a method of its parent candidate component. Optionally, candidates request new elections when they start, in order to challenge the current service bearers.

Electors evaluate candidates and vote. An elector is operative for a specified role only. Electors also have the ability to act as requestors when they notice that a service for which they are qualified is unresponsive for too long. Optionally, evaluators request new elections when they start. By doing so, they contribute their new observation to revise the current configuration.

SmartCitizens components broadcast messages on selected ports. Messages do not go through the resource provider's firewall. In addition, they are meant

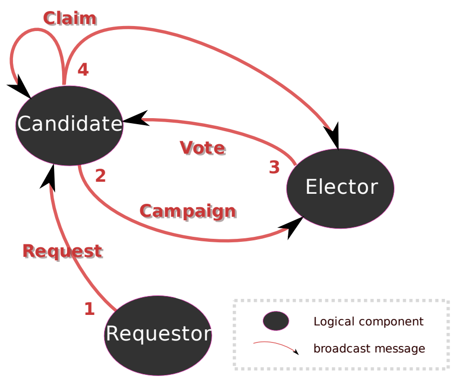


Figure 4.3: Workflow between components.

to be understood only by processes of the same user. They can be encrypted to avoid interferences. Only components that run on resources of the same provider, and alloted to the same user, are meant to communicate. This is where users need to maintain permanent services.

An election involves the communication depicted on figure 4.3.

1. a requestor component broadcasts a **Request** message. The request message carries the *role* and *number* attributes.

2. Every candidate component, on reception of the request message, checks if its role is requested. If yes, it permanently stores n, the requested number of service instances, and assesses its own resources. Resource assessment takes into account their appropriateness for the service. The candidate broadcasts the assessment, along with n, in a **Campaign** message.

3. Electors collect campaigns for roles on which they are qualified. They permanently store n, the requested number of services instances. They rank candidates according to the order in which they received campaigns, candidates self assessments, and appropriate prior knowledge on the candidates. Every elector broadcasts a **Vote** containing its n favorite candidates, the corresponding assessments, and a weight. The weight typically tells how extensively the host of the elector has used the service in the past.

4. Candidates collect votes and calculate the weighted sums of received assessments. Each candidate checks its rank against the number of nodes requested, and deduces whether it is elected or not. Elected candidates

notify electors with a **Claim** message.

Since communicating components are controlled by the same user, they are supposed to collaborate. Resource assessment by candidates and candidates assessment by electors are supposed correct. To ensure that only one candidate is elected, a resource assessment and a candidate assessment are integers added to a high precision uniform random number between 0 and 1.

Requestor, candidate and elector components implement the election of newly subscribed hosts to take over existing services, or other available hosts to take over services from terminating hosts. Requestors, candidates and electors must themselves be carefully placed and deployed on subscribed resources.

4.4 Components placement

The following details a test configuration.

In AliEn, a *Computing Element* (CE) is an ALICE service that takes care of ALICE tasks scheduling on a single grid site. A *Proxy* is a generic name for a service that allows communications to pass through the grid site firewalls. We assume ALICE wants to keep one CE and two proxies on every grid site.

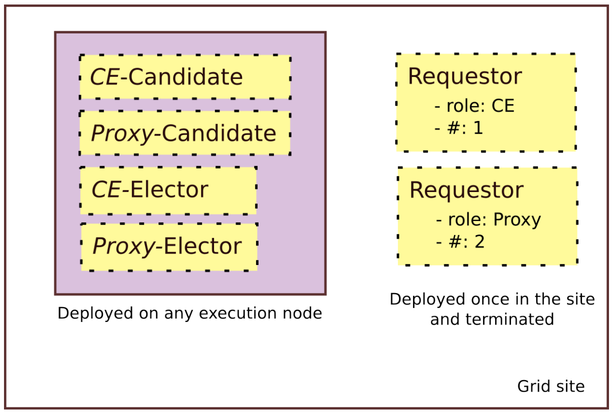


Figure 4.4: Components placement on a grid site.

A simple solution shown on figure 4.4 consists in running four components on every container alloted to ALICE and endorsed by the grid site.

- One CE-Candidate: a candidate component with *CE* role, with a linked child component able to configure and deploy a CE.

– One Proxy-Candidate: a candidate component with *Proxy* role, with a linked child component able to configure and deploy a proxy.
– One CE-Elector: an elector qualified to elect CE's.
– One Proxy-Elector: an elector qualified to elect proxies.

Additionally, two requestors are deployed on the first container that ALICE obtains on the site, and teared down immediately after they trigger the first election.

– One CE-Requestor, that requests one CE.
– One Proxy-Requestor, that requests two proxies.

Candidates and electors store service types and instance numbers. They trigger new elections when containers appear and services gracefully or abruptly disappear. We tested it by bringing up new nodes and killing services or nodes using the SmartFrog console. Services are accurately re-deployed. The requested number of services of each type remains. New requestors update the instance number for given service types.

The code and deployment tests are distributed under Library General Public License (LGPL) in the *election* package of SmartDomains development repository [3].

4.5 Conclusion

With the Symmetric Mapping pattern, grid users perceive resources as inherently transient. They appear and disappear on globally distributed grid sites. SmartCitizens demonstrates a solution to maintain permanent services on transient pools of resources by integrating an election mechanism within a service deployment system.

The need for a system like SmartCitizens has existed prior to the Symmetric Mapping pattern, since the use of Late Binding, where grid users attempt to take more control on their subscribed resources in order to operate their own efficient task scheduling. SmartCitizens also relieves resource providers from having to keep dedicated servers under the control of established user, previously the only solution for users to run their own permanent services on a grid.

The practice of Late Binding is bypassing traditional grid middleware to deliver higher performance. This lesson led us to rethink the design of traditional grid architectures and propose the Symmetric Mapping pattern. SmartDomains

3. sourceforge.net/projects/smartdomains/develop

and SmartCitizens prove that its inherent difficulties can be solved with adequate frameworks.

The adoption of Late Binding suggests that resource users are best qualified to schedule their own tasks, and Symmetric Mapping builds on this assumption.

The resulting architecture is beneficial if both resource users and providers are able to elaborate and implement a strategy for their own benefit. It is a valid question whether accurate methods exist for both sides. In the following of this thesis, we do not address computer center administration or power management, or other concerns that belong to resource providers. Instead, we focus on the resource users side, whose concern is mainly processing speed.

Due to the independence of providers, subscribed resources appear to users as heterogeneous clusters. Heterogeneity brings an opportunity for optimizations [PMP^{+}04, SOBS04]. However, efficient task mapping in this most general case requires fast and accurate performance prediction. This is the issue that we address in the second part of this document.

Part 2.
Performance Prediction

Chapter 5

Fast and Light Cache Performance Prediction

5.1 Introduction

Program performance is affected by the number and performance of available processing units, program parallelism and data distribution. The performance exhibited on a single processing unit is determined by its frequency and the stall time ratio, i.e. the number of cycles spent waiting for data divided by the total number of cycles. Cache memories temporarily store small amounts of data with the processor for quick fetching when required. In addition, many processors look ahead on the code. They predict branches with mostly high success rates, and they fetch data in advance to registers and cache [Smi98, SCL06b].

Reference streams and caches have been studied extensively in the last decades [Rau77, Smi82, SSkP^{+}07]. However, performance prediction has been hindered by failure to quickly and accurately predict cache misses, i.e. the event that a data is not in cache when required by the processor.

In order to enhance schedulers by taking into account cache resources, programs must be analyzed quickly. The program analysis overhead must not overpass the gain in scheduling efficiency.

We present a novel characterization of how a program stresses cache. This characterization permits fast performance prediction in order to simulate and assist task scheduling on heterogeneous resources. It is based on the estimation of stack distance probability distributions. The analysis requires the observation of a very small subset of memory accesses, and yields a reasonable to very accurate prediction in constant time.

Related characterizations are presented in section 5.2. The scope of this work is defined in section 5.3. The design of the new characterization is explained in section 5.4 and evaluated in section 5.5.

5.2 Related work

Cycle-accurate simulators return a cache event in response to each instruction. They require a handle on the application being executed [SSR01] or an exhaustive trace of the execution [Rot95, Smi82]. Although trace compression methods exist, these simulators are slow compared to other predictors [JIPH07].

How well a program behaves relative to cache has been explained in the literature with the notions of program locality [PG95, Mil00, FACA03]. Program locality has a variety of descriptions. Reducing the description size has always been a challenge for performance prediction. Programs can be decomposed into building blocks [LMW99, WE00, YMM05]. Resulting descriptions are still substantial and they do not apply to all kinds of caches.

Monte Carlo performance models represent a program as inter-dependent statistical generators of stall conditions [KS04, SCL06a, SCL06b]. These models are fast. The average number of cache misses in a run is correct even for complex processors. However, the cache misses generators used in these works are still specific to a cache configuration.

Fast cross-platform cache analysis is usually based on *stack distance*. **Stack distance** is the number of different memory lines accessed between two accesses to the same line. Stack distances are suited to evaluate fully associative caches with *Least Recently Used* (LRU) replacement policy. In this cases, and in the absence of pre-fetching, cache misses occur for stack distances greater than the cache size. In addition, stack distances have shown to accurately extend to set-associative caches with various cache line sizes and replacement policies [Rau77, Smi82, HS89, GAFN94, BE99].

For prediction, stack distances are usually recorded in a *stack distance histogram*. A stack distance histogram contains the number of occurrences of each stack distance. Stack distance histograms are widely used for cross-platform performance prediction [MMC04, PMP^{+}04, HHTE07]. They are lighter than application traces when the cache line size is known. However, their size is still substantial and the whole trace still needs to be collected.

5.3 Scope of the contribution

This section explains the limitations and novelty of the characterization.

Limitations. This characterization aims to predict the number of cache misses. The cost of a cache miss and the impact of pre-fetching are not studied here, although they are important to simulate and assist cache-aware scheduling. They must be addressed separately.

Cost of a cache miss. In modern processor architectures the cost of a cache miss on the process execution time depends on memory latency and bandwidth, the number of hardware threads, the quality of branch prediction, other platform characteristics, and on whether it occurs during direct or speculative execution. Evaluating the cost of a cache miss is not the concern of this work, which focuses on their number.

Pre-fetching. Modern processors use pre-fetching, a strategy that consists of loading data to cache before it is required in the program stack. Pre-fetching takes advantage of spatial locality. Along with efficient branch prediction, pre-fetching dramatically reduces the number of cache misses. However pre-fetching is externally scheduled by processors. It does not belong to cache configuration. The evaluation of how well it filters out cache misses can be done separately, as in [SCL06b, KS04].

Compulsory cache misses correspond to first-time accessed memory addresses, that is, to infinite stack distances. Compulsory instruction misses are given by the binary size and compulsory data misses are given by the data size. The characterization predicts *capacity* and *conflict misses* according to the standard taxonomy [HP06].

Cache thrashing. occurs when multiple processes share a processor in time. Each newly scheduled process erases lines from other processes. Cache thrashing is addressed in the next chapter.

Line size. Stack distance depends on the line size. Prediction is valid on computers with same line size as in the analysis. To our knowledge, this limitation has not been overcome yet. We are working on an analysis method independent from the line size. It still requires experimental validation and therefore is not included in the thesis.

Novelty. We propose a new characterization of how a program stresses cache. This characterization outperforms current methods for description size, analysis and prediction speed. It accounts for constant prediction complexity and for the fastest analysis since only small subsets of the application trace need to be extracted. It permits cross-platform cache performance prediction with reasonable to very good accuracy.

These performances are required to provide on the fly performance prediction in order to simulate and assist task scheduling on heterogeneous resource pools.

5.4 A characterization

We propose a characterization based on the estimation of the stack distance probability distribution. Stack distance is seen as a random variable X. It is fitted to a combination of well known probability distributions. The obtained distribution has a cumulative distribution function $cdf(x) = \mathbb{P}(X \leq x)$. If the estimation is correct, the cache misses ratio is $\mathbb{P}(X > c) = 1 - cdf(c)$ where c is the cache size in number of lines. Therefore, prediction exhibits constant computational complexity.

In addition we propose a method to refine a simple fit. Cache misses prediction requires to fit correctly only the upper values of random variable X. Indeed, prediction is only useful for realistic cache sizes. If it can be determined that no considered cache is smaller than a minimal cache size m, then X must fit the distribution correctly for values greater than m.

The refinement algorithm is as follows. X is a random variable, in fact a list of samples. *dist* represents the parameters of a distribution, i.e. the result of a random variable fit. Function fit is a regular fit. Function fit' is the refined fit.

```
function bias(X, dist, m) :
   for each s in X such that s < m
      do
         s' := randomly generated from dist
      loop until s' < m
      s := s'
   end for
   return X
end function
```

```
function fit'(X, m) :
   dist := fit(X)
   X' := X
   for each refinement
      X' := bias(X', dist, m)
      dist := fit(X')
   end for
   return dist
end function
```

At each refinement, randomly generated values based on the previous estimation replace the lower samples.

Proof. Suppose that an estimation minimizes the Mean Squared Error ϵ. $\epsilon_{i,j}$ is the error of estimation at ith refinement on data at jth refinement. ϵ^{down} and ϵ^{up} are the contributions of lower and upper samples to the error. Since estimation $n+1$ minimizes the error on data $n+1$,

$$\epsilon_{n+1,n+1} \leq \epsilon_{n,n+1}$$

It yields

$$\epsilon^{up}_{n+1,n+1} + \epsilon^{down}_{n+1,n+1} \leq \epsilon^{up}_{n,n+1} + \epsilon^{down}_{n,n+1}$$

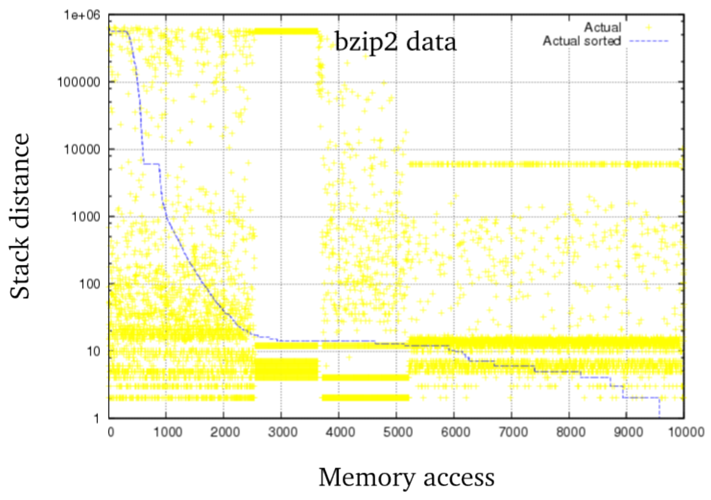
Since $\epsilon^{down}_{n,n+1} = 0$ and $\epsilon^{up}_{n,n+1} = \epsilon^{up}_{n,n}$ by construction, it yields

$$\epsilon^{up}_{n+1,n+1} \leq \epsilon^{up}_{n,n}$$

The upper samples are better fitted after each refinement. □

The remaining of this chapter is an evaluation of the characterization based on the analysis of SPEC[1] CPU2006 benchmarks. The objective of SPEC CPU2006 benchmarks is to represent with a limited number of benchmarks the whole spectrum of modern applications and workloads. We used most of them in our experiments. The figures of this chapter represent gromacs, lbm, libquantum, gemsFDTD, soplex, dealII, bzip2, gobmk, leslie3d, perlbench, specrand, libquantum, tonto, h264ref, hmmer, omnetpp, sjeng, calculix. More information on the benchmarks can be found on the SPEC website.

1. www.spec.org - SPEC is a non profit corporation that maintains relevant benchmarks to analyze the performance of modern computers.



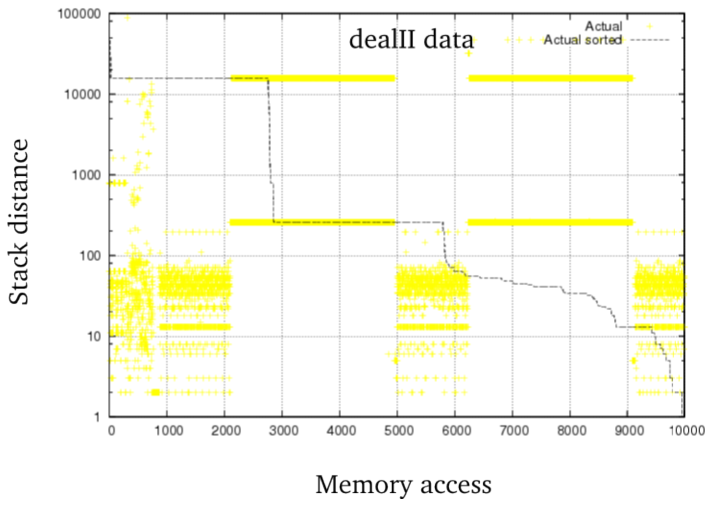
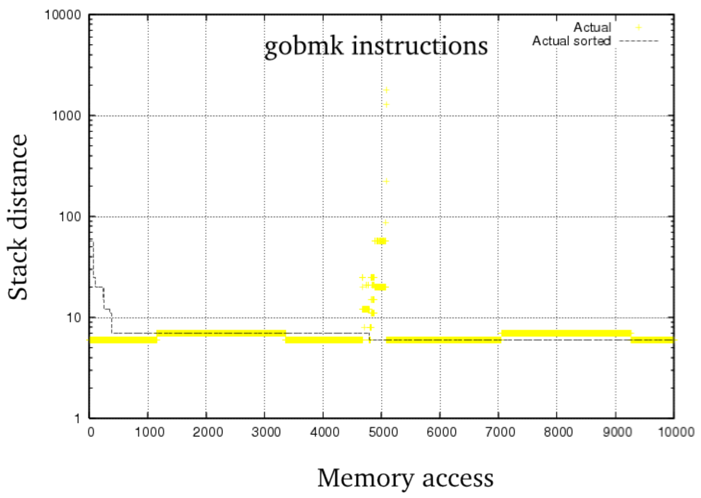


Figure 5.1: Stack distance patterns



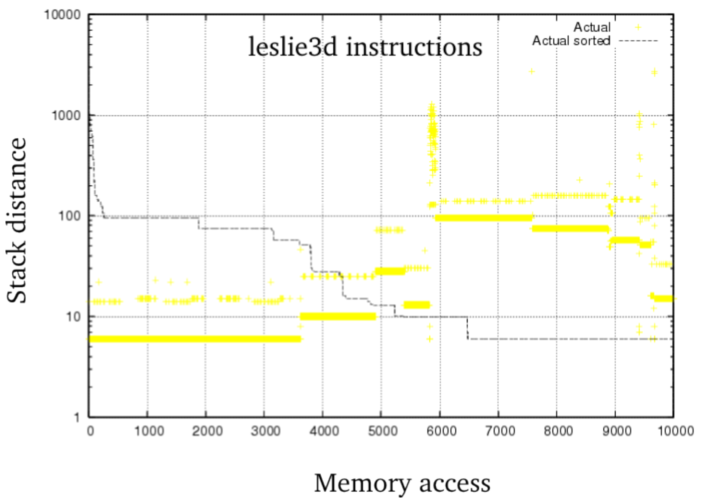
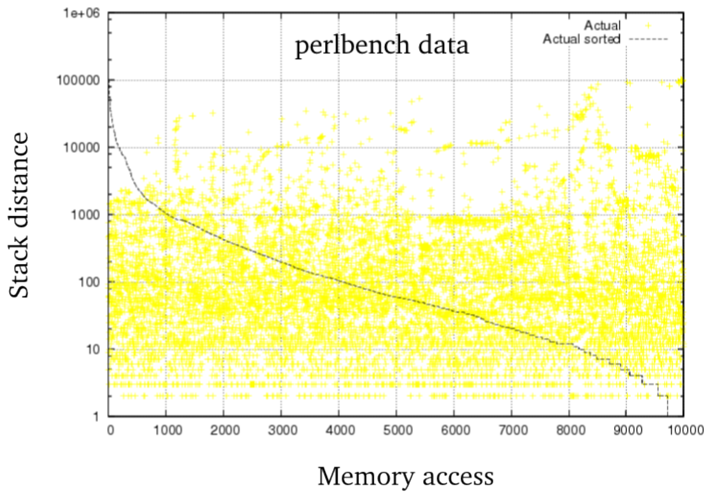


Figure 5.2: Stack distance patterns



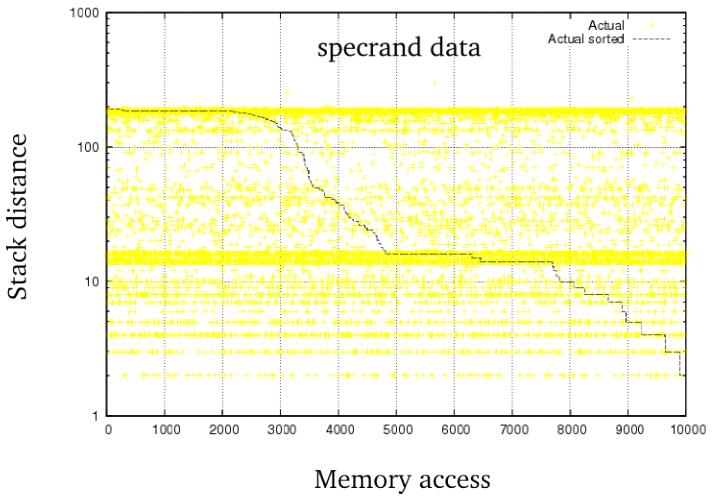
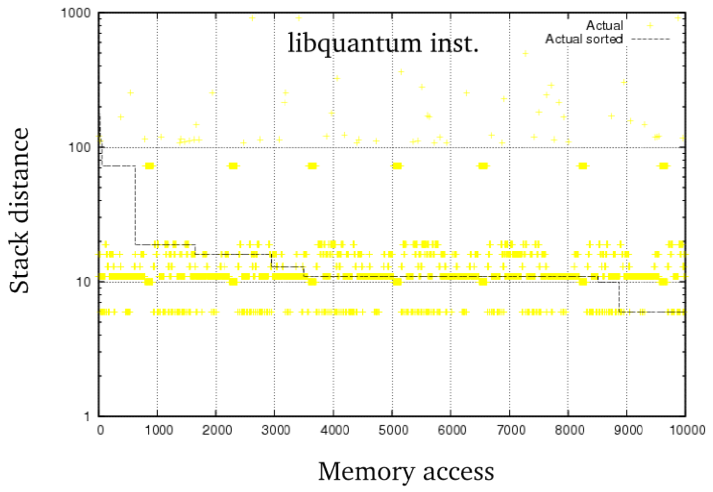


Figure 5.3: Stack distance patterns



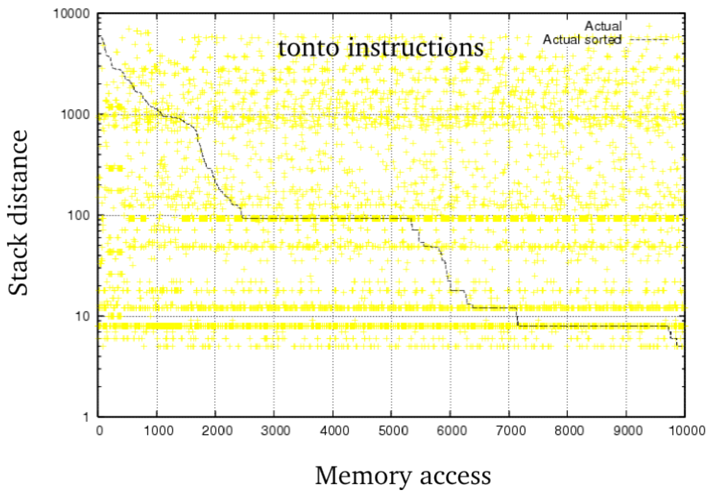
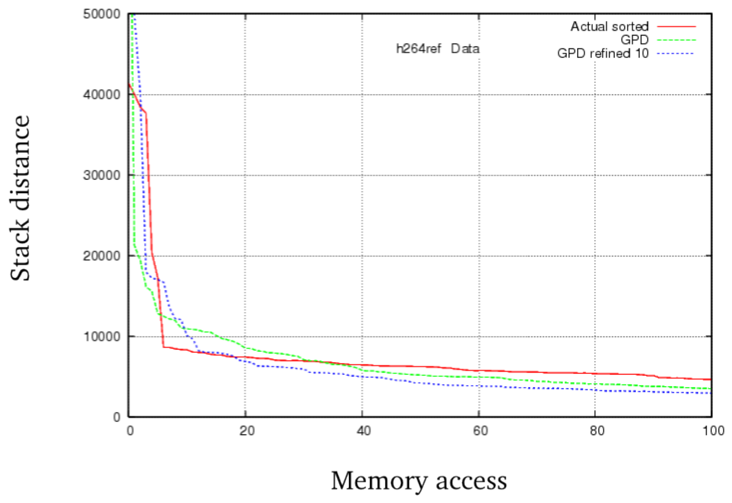


Figure 5.4: Stack distance patterns



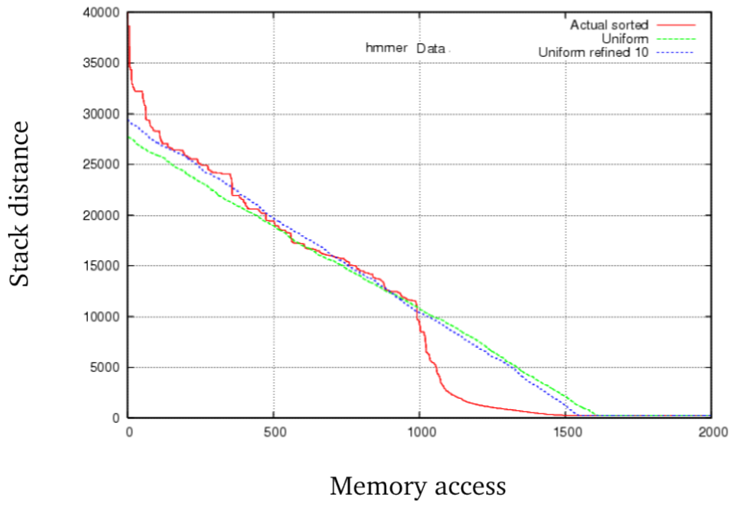
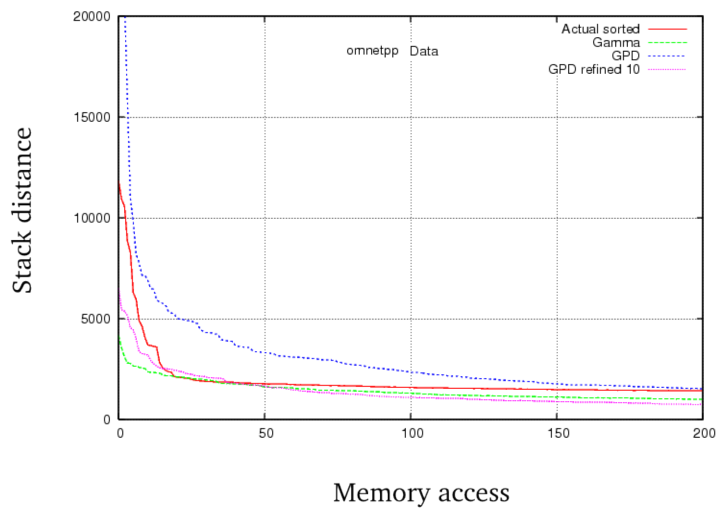


Figure 5.5: Stack distance fits.



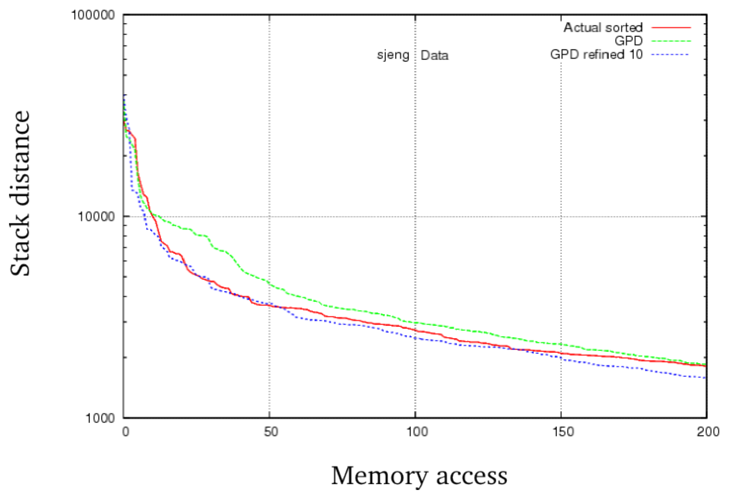
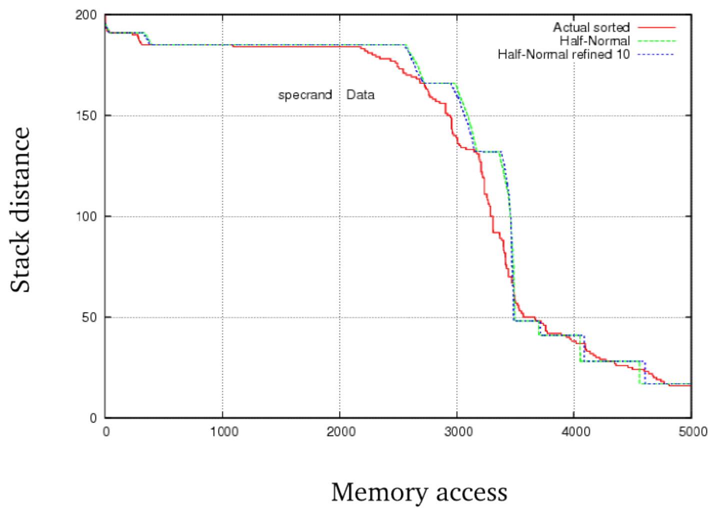


Figure 5.6: Stack distance fits.



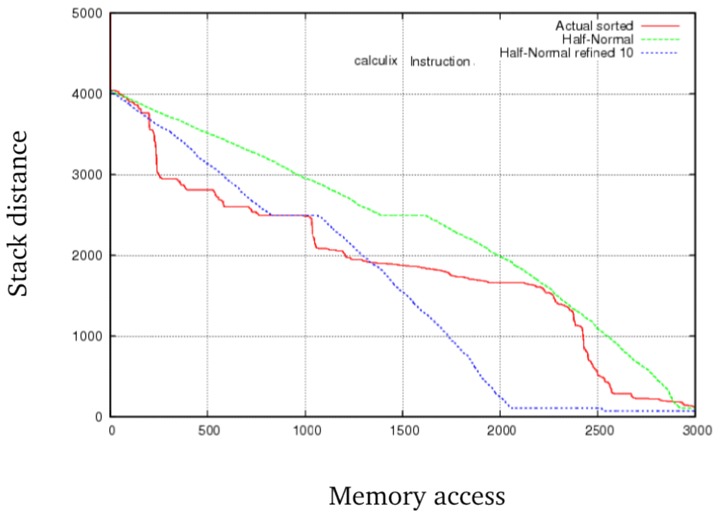


Figure 5.7: Stack distance fits.

5.5 Evaluation

We instrumented SPEC binaries with PIN [PACL05] to obtain instructions and data load traces. Stack distances are extracted using the *trace profiling algorithm* [HHTE07]. For simplicity we set line sizes to one, and we instrumented stack distances between individual references. We developed necessary analysis tools in Java. They include trace analysis, estimators based on the Method of Moments for a large spectrum of distributions: Discrete, Uniform, Gamma, Generalized Pareto (GP) and Half Normal (HN). They include a random number generator for each of these distributions, and the estimation refinement. They are released under Artistic License version 2. They come with the whole data presented in this chapter [2].

The evaluation has three steps. The first step shows the precision to which stack distances fit a probability distribution. The second step shows the effect of collecting a limited number of stack distance samples. The third step is a discussion on using the analysis to predict cache misses of other parts of the program and with other input data. Figures 5.1 to 5.4 show stack distances on a representative set of SPEC CPU2006 benchmarks. Diagrams on the left-hand-side show stack distances between data accesses, and diagrams on the right-hand-side show stack distances between instruction accesses. Light dots are stack distances in chronological order. Dashed lines are the **outlines**. Outlines are the same values in descending order. The plots of figure 5.1 to 5.4 differ from histograms. On a histogram, values are on the x axis and the y axis measures the number of occurrences. On figure 5.1 to 5.4 the x axis is a list of memory accesses and the y axis measures corresponding stack distances.

In general, outlines are composed of curves and straight segments. An outline exclusively composed of straight segments indicates that the variable perfectly fits a discrete distribution. In this case the characterization is equivalent to estimating the histogram. It results in a compressed histogram where empty bins are removed [PMP$^+$04]. To the contrary, a curve indicates that a histogram would require a high number of bins. When curves exist, fitting a continuous distribution dramatically reduces the characterization size, for continuous distributions are determined with typically two or three parameters. Among the 28 SPEC CPU2006 benchmarks, 11 have discrete instruction stack distances, and three (gromacs, lbm and libquantum) have discrete data stack distances. In general, a stack distance distribution is the sum of a discrete distribution and

2. code.google.com/p/mtc-project

continuous distributions.

5.5.1 Stack distance distribution fit

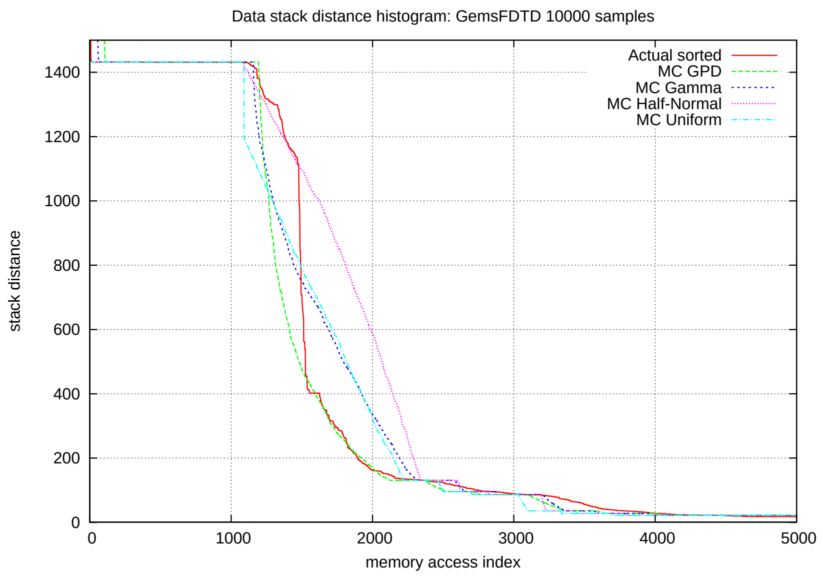


Figure 5.8: A problematic fit.

Figure 5.8 illustrates the analysis of GemsFDTD. The outline is shown along with Monte Carlo simulations based on different analysis. For analysis, discrete parts are filtered out and fitted separately. The remaining samples are fitted to a continuous distribution. HN fits well the upper part of the curve and GPD the lower part. However, the whole curve does not fit any single distribution alone. Gamma and Uniform average the trends. The characterization does not accurately account for all stack distances in a program whose outline has an inflexion point. 8 data traces out of the 28 benchmarks fall into this category. In these worst cases, the refined fit permits to concentrate on the higher stack distances that account for cache misses.

Figures 5.5 to 5.7 illustrate six representative analysis scenarios. For each benchmark the best distribution is selected, and the fit is refined. The selection can be done automatically by picking the distribution that accounts for the smallest estimation error. For the three first benchmarks, the estimation is quite accurate. Refined estimations give the best results. Indeed, outlines have long tails that bias the estimation of upper values in the absence of refinement. The precision on the fourth benchmark is impaired by the precision of the discrete fit. The two last benchmarks are the worst cases, one because of its heavy tail and the other because of its irregular outline.

In conclusion, the characterization is accurate for most SPEC CPU2006 benchmarks. Stack distances predicted in Monte-Carlo simulations perfectly match actual values. However, there are impracticable scenarios where the precision is the order of magnitude.

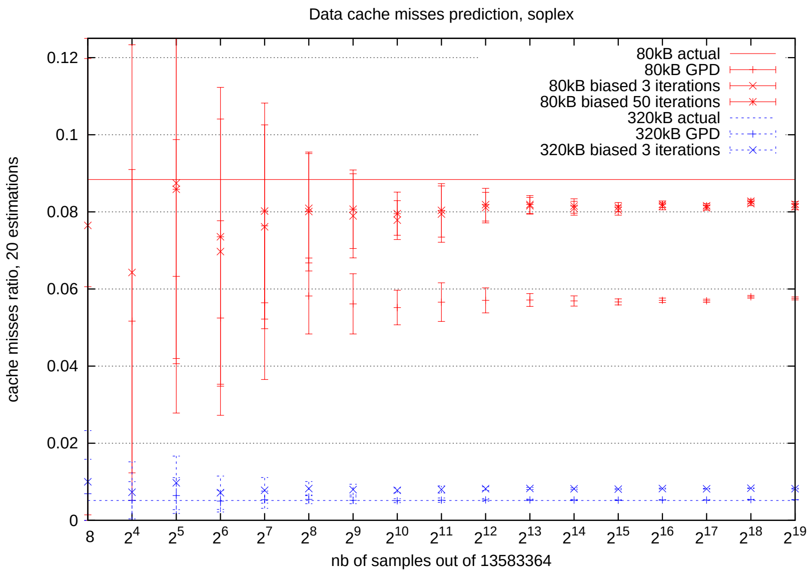
5.5.2 Analysis speed and prediction accuracy

In section 5.5.1 we considered the ability of a probability distribution to accurately reproduce stack distances and thus predict cache misses for any kind of cache. The difference between actual stack distances and the best distribution is a first contribution to the prediction error. In this section we evaluate the number of samples required to fit such a distribution. The limited number of stack distance **samples** introduces another contribution to the prediction error. There must be just enough samples to obtain an estimation as close to the best estimation as the best estimation to the real data. In the following this number of samples is called **adequate**.

On figure 5.9, two benchmarks are examined. Actual cache misses ratios with two different caches are compared to predictions based on different sample sets. The same characterization is used to predict cache misses for the two cache sizes.

Although refined fits are better *in general*, they are not better for all cache sizes. For soplex data misses prediction with refined fits, the adequate number of samples is around 2^8. One sample must be collected every 50,000 data accesses in memory. This number yields a prediction accuracy of 99%. For dealII instruction misses prediction, the adequate number of samples is around 2^{11}. One sample must be collected every 13,000 instructions, for an accuracy of 99.6%.

In conclusion, accurate predictions are obtained with fast analysis. Less accurate predictions can be done faster.



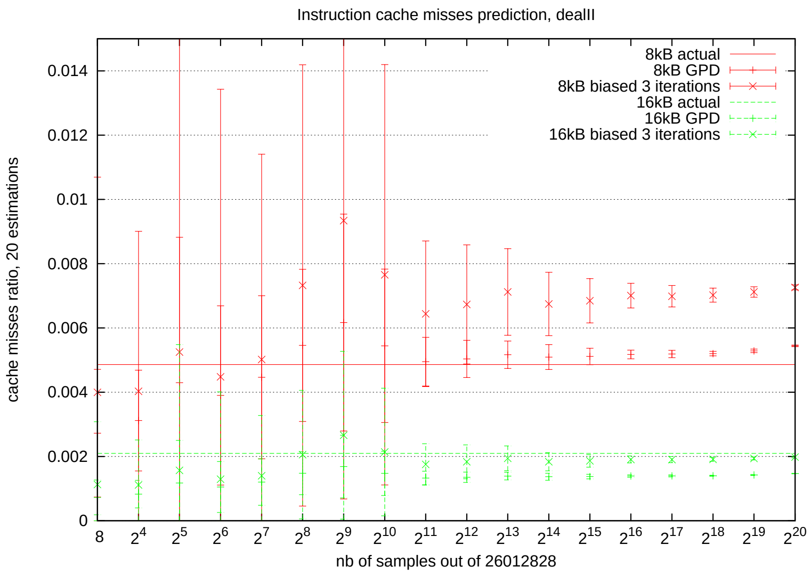
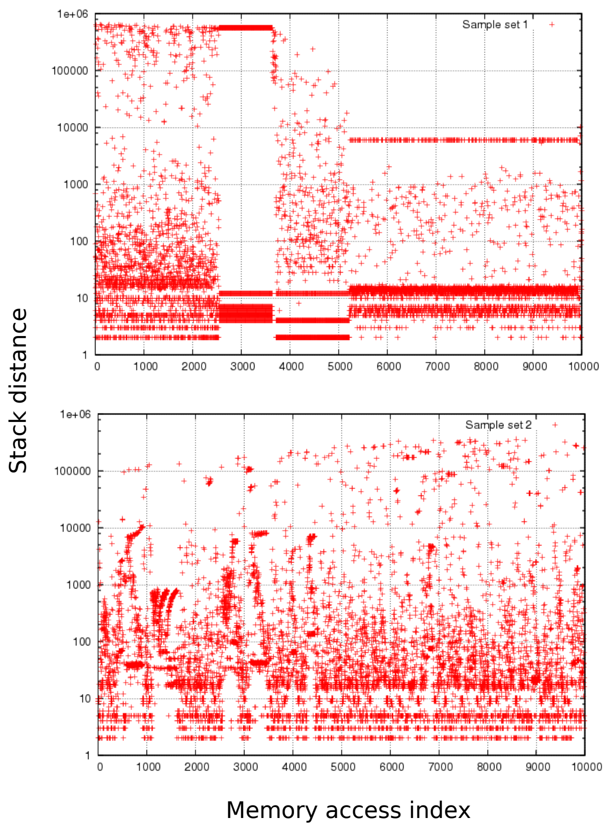


Figure 5.9: Cache misses prediction on two benchmarks.



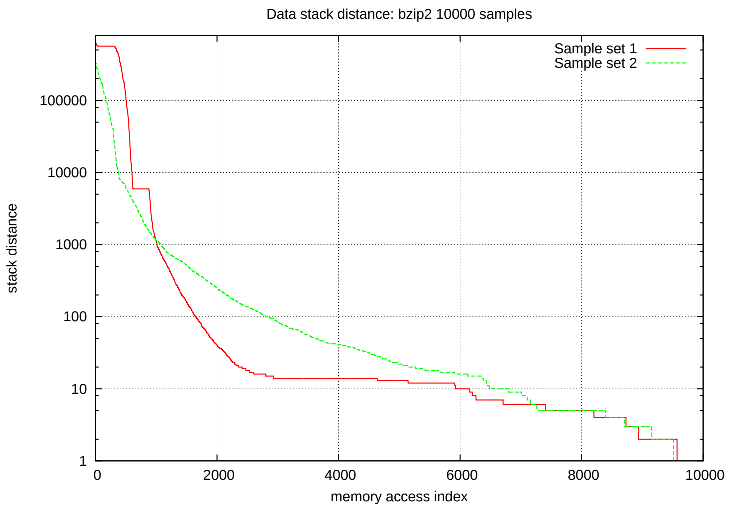


Figure 5.10: Two sample sets of bzip2.

5.5.3 Prediction robustness

In this section we briefly discuss the characterization accuracy to predict cache misses in the future and with different input data.

Figure 5.10 illustrates the variation of bzip2 stack distances outline on chronological and sorted views. Samples are taken from millions of consecutive memory accesses, and the two sample sets are separated by a few seconds. Samples are represented in chronological order on separate figures (top). Outlines are represented on the same picture (bottom). With bzip2, outlines show a qualitative resemblance, but precise prediction is not possible for future cache misses.

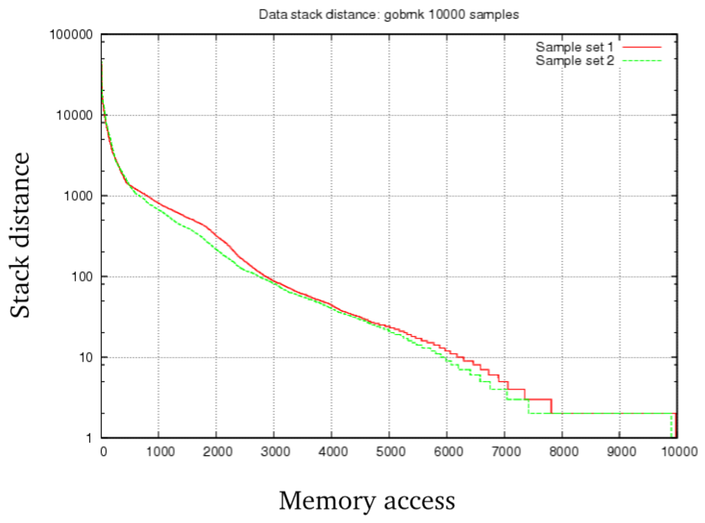
Figures 5.11 and 5.12 show the evolution of the outline for four benchmarks. sjeng does not change its memory access pattern in time. gobmk does not change with different input data. To the contrary, astar instruction access pattern changes in time, as well as wrf data access pattern.

In conclusion, future behaviors can be predicted only if the program is known to follow a certain regularity. For example, scientific computations often involve the repetitive execution of the same routines [YMM05]. In some cases, as with gobmk, different input data do not change memory access patterns. The analysis, unnoticeable on the first run of a routine or on the first input data, provides at worst a rough indication of the cache misses ratio, useful for cache-aware scheduling. In other cases it predicts future cache misses with very high precision.

5.6 Conclusion

The contribution of this chapter is a novel characterization of how a program stresses cache, in terms of the stack distance fit to a probability distribution. The characterization has a very small size and provides cache misses predictions in constant time. Its evaluation distinguishes three contributions to the prediction error. One is relative to the appropriateness of a probability distribution to describe stack distances. The second is relative to the number of samples used for the fit, and the third is relative to the changes in program behavior. The worst cases yield reasonable accuracy to simulate or assist scheduling systems. Many application behaviors are very accurately described by probability distributions and have enough regularity for the prediction to apply under different circumstances. Fitting a distribution requires the extraction of a very small subset of the trace. This makes the analysis extremely fast, which is a requirement to simulate and assist task scheduling systems.

While this chapter improves the complexity of analysis under the usual hypothesis that context switches do not introduce additional cache misses, the next chapter removes this assumption and presents a more complex algorithm that also accounts for cache thrashing.



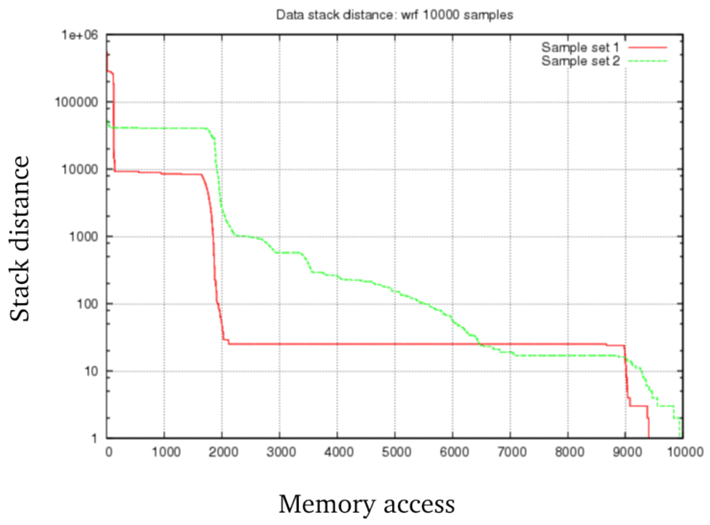
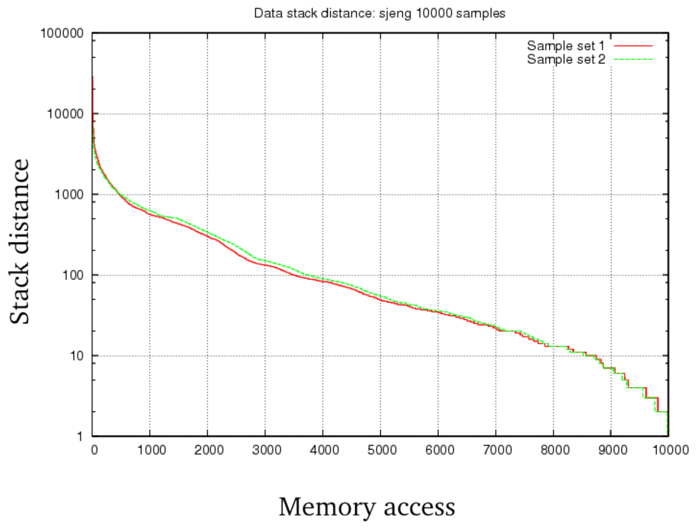


Figure 5.11: Different inputs and observation segments.



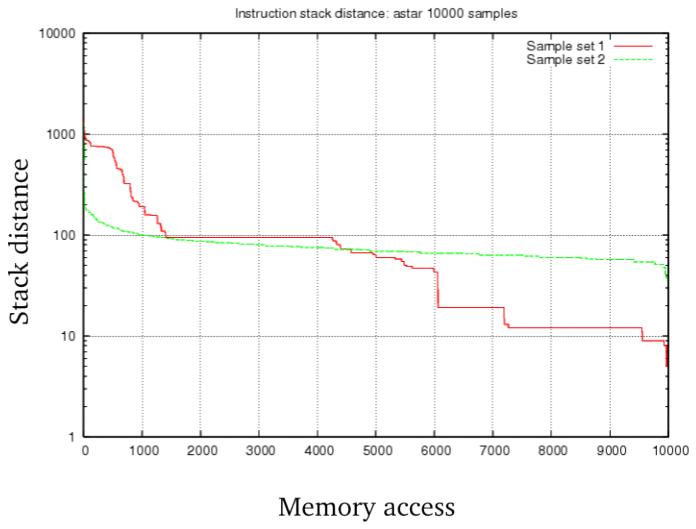


Figure 5.12: Different inputs and observation segments.

Chapter 6

Stochastic Analysis of Cache Thrashing

6.1 Introduction

Processor caches are normally shared by multiple processes. A cache stores a limited amount of memory for fast access. Processes scheduled on successive time quanta affect each other's performance by erasing each other's cached items. This produces extra *cache misses*, i.e. failures to fetch items from cache. The adverse effect on performance is called *cache thrashing*. It is in some cases overwhelming, although reputedly difficult to quantify.

Performance prediction is particularly relevant to application, compiler and platform developers, and to task schedulers. Developers evaluate cache performance with expensive simulations. Online task schedulers require a faster alternative to map tasks to heterogeneous platforms. Cache misses are to a great extent responsible for performance variability between heterogeneous platforms. However, former prediction algorithms make simplifying assumptions that question their applicability to real cases, as with multiple concurrent processes.

This chapter presents a stochastic approach to cache misses prediction in presence of multiple concurrent processes time-sharing a fully associative LRU[1] *cache.*

The method gives a higher bound of the *cache miss rate*, i.e. the ratio of cache misses per memory access.

1. A cache with Least Recently Used replacement policy

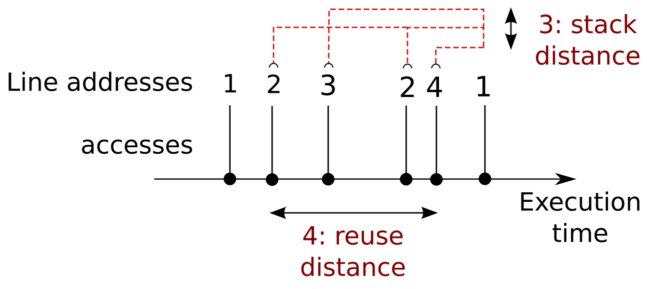


Figure 6.1: Stack and reuse distance

Section 6.2 places this work with current research. Section 6.3 introduces useful concepts: *age, rank, span, propagation*. Section 6.4 shows their relationships with the expected number of cache misses in a time quantum. Section 6.5 presents their calculation based on the stack distance distribution of the process. Section 6.6 summarizes the algorithm and its complexity. Section 6.7 measures the cost of its hypotheses on a set of benchmarks.

6.2 Position of this work

This section places the present work in its context. Section 6.2.1 introduces the metric that forms the basis of the analysis. Section 6.2.2 presents related work on cache simulation and performance prediction. Section 6.2.3 identifies our contributions.

6.2.1 Stack distance

In order to prepare for the complexity of the analysis, this section goes into more details than chapter 5 on the definition of the metrics.

The method relies on the following cache design principles. Given that contiguous memory items are often accessed in a row, multiple contiguous items are fetched from memory at a time, in a *line* or *block*. Moreover, given that the same memory accesses often occur repeatedly, every line is stored on cache (is *cached*) when accessed, in order to be available on subsequent accesses. Several strategies may apply to determine the position of a new line on cache.

Associativity is a design choice. With a n-associative cache, a memory line can only be stored on n different positions based on its memory address. A fully-associative cache of size c lines is a c-associative cache. Fully-associative

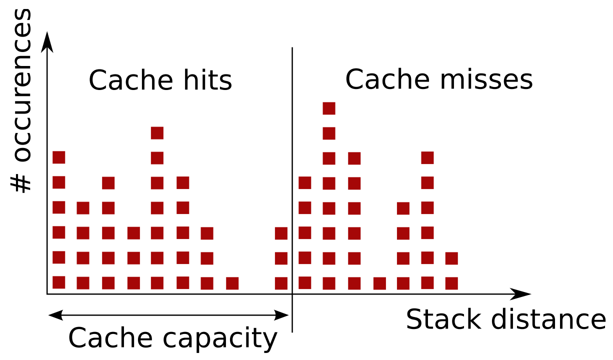


Figure 6.2: Misses under cache monopoly

caches make it possible to place a new line anywhere. When the cache is full, the replacement policy determines which line is selected for eviction.

On fully associative caches with Least Recently Used (LRU) replacement policy, every new line overwrites (*evicts*) the least recently used line, considered one of the least likely to be reused in the future.

The method is based on the estimation of its *stack distance* probability distribution. This distribution describes the temporal locality of memory accesses. An observation can be made for every memory access.

Definition 15 (Stack distance)**.** *The* ***stack distance*** *of a memory access is the number of different lines accessed since the last reference to the requested line, present and initial references excluded. Stack distance is defined on* $\mathbb{N} \cup \{+\infty\}$*. A stack distance of* $+\infty$ *occurs on the first request of any line [BD01].*

A few authors call stack distance *circular sequence* [CGKS05]. Others call it *reuse distance* [SSkP$^+$07]. However, for most authors, reuse distance is incremented for every distinct access to the same line. On figure 6.1, the stack distance of the last access to line 1 is 3 while its reuse distance is 4 because line 2 was accessed twice between the two accesses to line 1.

Definition 16 (Reuse distance)**.** *the* ***reuse distance*** *of a memory access is the number of memory references since the same line was previously accessed, present and initial references excluded. Reuse distance is defined on* $\mathbb{N} \cup \{+\infty\}$*. A reuse distance of* $+\infty$ *occurs on the first request of any line [BH04].*

Stack distance allows to predict the number of cache misses for a process that runs forever with a dedicated LRU, fully associative cache. An access is a miss if and only if its stack distance is greater than the cache capacity. Figure 6.2 represents a stack distance histogram. For each stack distance, the histogram stores the number of corresponding accesses. The miss rate appears on the histogram as the ratio of the number of misses by the number of hits.

There is indecision in current literature whether stack distance is a number of different *lines* accessed between two successive accesses to the same *line*, or a number of different *items* accessed between two successive accesses to the same *item*. Counting lines is useful for prediction but cannot be done by looking at the process only. Counting items can be done once for all for a process but is not directly useful for performance prediction. We deliberately choose the first alternative and assume that stack distance is known. Strictly speaking, it depends on line size and on spatial distribution of data on memory. To obtain stack distance from independent analysis of a process and a platform is a work in progress.

6.2.2 Previous work

Agarwal, Hennessy and Horowitz show in [AHH89] that the impact of multiprogramming on cache misses is at least substantial and can be predominant. In [LGS$^+$08], Liu et al. confirm this observation on recent benchmarks and processors.

A LRU, fully associative cache is comparable to a FIFO (First In First Out) queue with perturbations. In the absence of memory reads, the first line written (first in the stack) is the first evicted (first out). Perturbations are caused by memory accesses that reorder line rankings. After every access, the accessed line is ranked first and the others are shifted down in the rankings. In this perspective, the present work is related to queuing theory as described in [All90]. However, queueing theory is concerned with the time spent in the queue - the queueing delay, while the present work is concerned with the probability that a line is in the cache when the program needs to access it.

Strictly speaking, a *stack* is a LIFO (Last In First Out) queue. Caches resemble stacks because of the programming artifact that the most recently used lines (last in) generally have a high probability to be re-used soon (first *out*). This observation motivates the use of the LRU replacement policy, that dictates that the least recently used line in cache is the first line evicted, i.e. overwritten by a new entry.

Gecsei, Slutz, and Traiger introduced stack distances in 1970 [GST70]. In

[GAFN94], Grimsrud et al. show that stack distances describes a typical memory access pattern better than other models still in use today. In 1999, Brehob and Enbody define the *stack distance of a reference as the depth in the stack from which it was fetched*; and use it for cache misses prediction [BE99]. They exhibit prediction errors of a few percents when running a single process at a time.

In [SDR01], Suh, Devadas and Rudolph calculate the average miss probability on a finite time quantum, from the given miss probability as a known, convex function of the number of cached lines. More recently, in [LGS$^+$08], Liu et al. use a Markov model to simulate the effect of cache thrashing. A three-states Markov chain describes the last step to obtain a given distribution of data in cache. Recursion on transition probabilities yields a simple cache simulation algorithm. To the best of our knowledge, these two works are representative for the few attempts to predict or simulate context switch misses. Among these references, other works in cache performance prediction are restricted to the monotasking case, where the cache is dedicated to a single process.

Stack distance histograms are often used as tasks *signatures*, i.e. to store memory access patterns. They can be computed at compile time, by analyzing loops, and under some assumptions on the regularity of the references within a loop, as is done in [CP03]. An alternative is to monitor memory accesses at runtime on a short time period, and suppose that the extracted stack distance histogram is representative of the full run [MMC04]. In chapter 5, presented in [GJ08], we fit *stack distance* to known probability distributions to reduce the size of the task signature to a few parameters and the complexity of the miss prediction to a constant complexity, with measured risks on accuracy.

Results are often obtained for LRU, fully associative caches and said to extrapolate to other types of cache. The impact of replacement policy is measured for example in [GS06]. The impact of cache associativity is detailed in [HS89]. In [LZ09], Liu et al. generalize to *associative* caches the conditions on stack distance under which an access yields a cache miss.

6.2.3 Assumptions and contributions

The present work builds on the assumption that stack distance observations are *independent, identically distributed.* To the best of our knowledge, this assumption is always done for non cycle accurate cache performance prediction and simulation. It underlies for example the use of stack distance histograms.

However, it removes some information contained in a stack distance trace. For example, it hides the fact that because of program loops, successive accesses often exhibit regular stack distance patterns. An alternative would be to consider stack distance as a stochastic process, at the expense of a probably more complex model.

In addition, the stack distance distribution Σ of the process of interest is supposedly known, and representative of an identified phase of the execution. Its cumulative distribution function $F(k) = \mathbb{P}(\Sigma \geq k)$ is given at any point. The computational complexity of the proposed algorithm is the number of calls to F.

The method gives the first higher bound of the cache miss ratio in presence of context switches. It complements prediction methods such as [LZ09] that do not account for context switches and, therefore, present a lower bound of the cache miss ratio in multitasking. It is consistent with other analysis of cache thrashing ([SDR01] and [LGS$^+$08]). However, by contrast with [SDR01], it does not require prior knowledge or assumptions on the miss probability function, and by contrast with [LGS$^+$08], it results in a prediction algorithm faster than a simulator.

We prove that the computation can be done with the initial approximation of the singular values of a bidiagonal matrix, followed by a computation with cubic complexity of the cache size. We also indicate, but we fail to prove in the general case, an exact method, quadratic of the cache size.

A detailed terminology is developed to elaborate from the observation of stochastic processes. It cements a theoretical frame for the development of more accurate or faster algorithms.

6.3 Metrics

This section introduces metrics to model the state of a cache as seen by a process in competition with other time-shared processes. Memory accesses increment a measure of *time* relative to the process. At a given time, *age* and *rank* characterize cache lines, and *span*, characterizes the process by its number of cached lines. *Propagation* measures consistency between two time quanta.

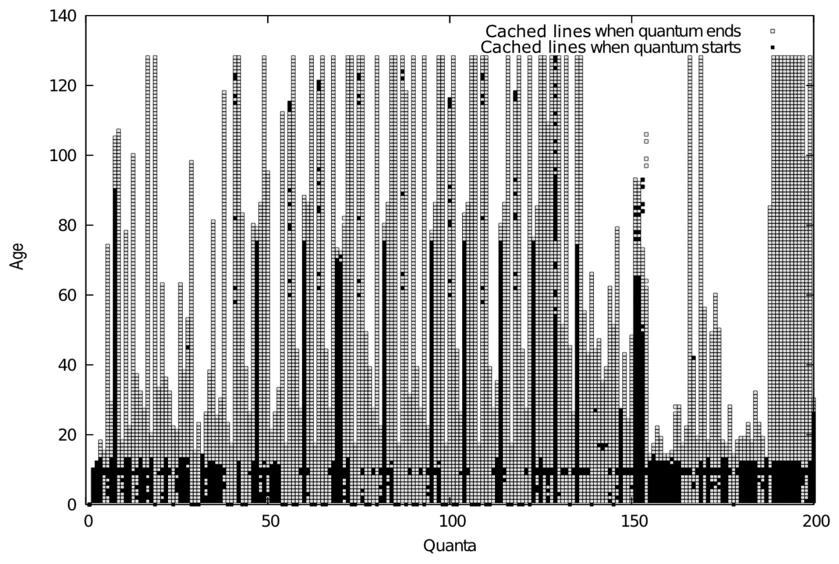


Figure 6.3: Cached lines with their age at the beginning and the end of successive time quanta.

6.3.1 Age

This section defines *time* and a line's *age* to describe usage recency, and connects age to reuse distance: the reuse distance of a hit is the age of the requested line.

Definition 17 (Time)**.** *The* ***time*** *of a memory access is the number of previous memory accesses by the process since the beginning of the time quantum. Time is defined on* $\mathbb{Z} \cup \{+\infty, -\infty\}$*.* $+\infty$ *is the time of a non existent future access, and* $-\infty$ *is the time of a non existent past access.*

Definition 18 (Duration)**.** *The* ***duration*** *of a period is the number of memory accesses of the process on this period. Duration is defined on* $\mathbb{N}$*.*

Definition 19 (Age)**.** *The* ***age*** *of a line is the time minus the time of its last request (fig. 6.4). Age is defined on* $\mathbb{N} \cup \{+\infty\}$*.*

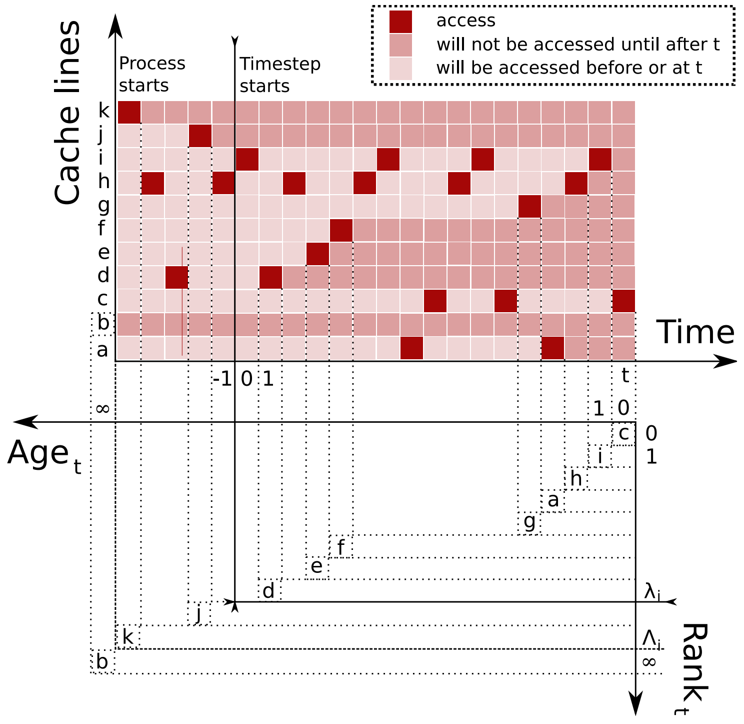


Figure 6.4: Relationship between time, age, rank and span.

Theorem 4. *An access i of reuse distance $r \in \mathbb{N}$ is a hit on the line of age r at $i-1$ if it exists and a miss otherwise.*

Proof. If i is the time of the access of reuse distance r, the previous access to the same line is at time $i-r-1$. Therefore the age of the requested line at $i-1$ is r. □

6.3.2 Rank

This section defines *rank*, that indexes lines according to their age, and connects rank to stack distance: the stack distance of a hit is the rank of the requested line.

Definition 20 (Rank)**.** *A line's **rank** is the number of younger lines (fig. 6.4).*

Definition 21 (Cache capacity)**.** *The cache **capacity** is the number of different lines that the process is allowed to have in cache at a given time.*

Lemma 1 (Necessary condition for eviction)**.** *If C is the cache capacity, a line of rank r at i can be evicted at $i+1$ only if*

$$r = C - 1$$

Proof. LRU replacement policy ensures that r is the maximum rank. Full associativity ensures that the allocated cache is fully occupied before eviction. Therefore, the maximum rank is $C - 1$. □

Theorem 5. *An access i of stack distance s hits the line of rank s at $i-1$ if it exists and is a miss otherwise.*

Proof. Suppose access i is a hit. Let s be its stack distance, l the line hit, and x its rank l at access $i-1$. There were s different lines $l_1, \ldots, l_s$ accessed since last access n_0 to the same line l, x of which have not been evicted at $i-1$. Therefore, $x \leq s$.

Ad absurdum, suppose $x < s$. In this case $\exists k \in [1, s]$ such that l_k was accessed at time i_2 and evicted a time i_3 with $i_0 < i_2 < i_3 < i$. Therefore, if $r_{k,3}$ is the rank of l_k at time $i_3 - 1$, $r_{k,3} < x$. In addition, since i_3 is an eviction and i is not, from lemma 1, $r_{k,3} > x$. By contradiction, $x = s$. Therefore, if i is a hit, the requested line has rank s.

By contraposition, if there is no line of rank s, n is a miss. □

6.3.3 Span

This section introduces the *span*, the number of lines that have been accessed by the process, and its evolution in the quantum.

Definition 22 (Active line)**.** *A cached line of age a is **active** at time i if $a \leq i$, i.e. if it has been accessed in the time quantum.*

Definition 23 (Active span)**.** *The **active span** is the number of active lines (fig. 6.4).*

Lemma 2 (Active span and rank)**.** *If r_{max} is the maximum rank of active lines and λ is the active span,*

$$\lambda = r_{max} + 1$$

Proof. Let l be the active line of maximum rank r_{max}. All $\lambda - 1$ other active lines are younger, therefore $r_{max} \geq \lambda - 1$. All r_{max} younger lines are also active, therefore $\lambda \geq r_{max} + 1$. □

Definition 24 (Maximum span)**.** *With C the cache capacity and Σ the stack distance distribution of the process (only the finite values), the* ***maximum span*** *c is defined by*

$$c = \min(C, \max \Sigma + 1)$$

Theorem 6 (Convergence of active span)**.** *In a time quantum, the active span is monotonic increasing and converges into the maximum span almost surely.*

Proof. Let c be the maximum span, and $\forall i \in \mathbb{N}$, λ_i the active span at time i. The sequence $(\lambda_i)_{i\in\mathbb{N}} \in \mathbb{N}^{\mathbb{N}}$ gives the evolution of active span.

Let $i \in \mathbb{N}$.

– If access $i + 1$ requests a line active at i, $\lambda_{i+1} = \lambda_i$.

– If access $i + 1$ requests a line not active at i, $\lambda_{i+1} > \lambda_i$.

Therefore, (λ_i) is monotonous increasing.

We show that $\forall i \in \mathbb{N}$, $\lambda_i \leq \max \Sigma + 1$ by recursion on time. $\lambda_0 = 1 \leq \max \Sigma + 1$. We suppose that $\lambda_i \leq \max \Sigma + 1$ and examine every case for λ_{i+1}.

- If $i + 1$ requests a line active at i, $\lambda_{i+1} = \lambda_i$ and $\lambda_i \leq \max \Sigma + 1$ by hypothesis. Therefore, $\lambda_{i+1} \leq \max \Sigma + 1$.
- If $i + 1$ requests a line not active at i, let s be its stack distance at i.
  - If $i + 1$ is a hit, let l be the requested line and r its rank at i. From theorem 5, $s = r$. Since l is not active at i, l is older than all active lines. If r_{max} is the maximum rank of active lines at i, $r > r_{max}$. From lemma 2, $r_{max} = \lambda_i - 1$ and therefore $r \geq \lambda_i$. Since $r = s$ and $s \leq \max \Sigma$, $\lambda_i \leq \max \Sigma$ and therefore $\lambda_{i+1} \leq \max \Sigma + 1$.
  - If $i + 1$ is a miss, from theorem 5, there is no line of rank s at i. We show that $\lambda_i \leq s$ *ad absurdum*. Suppose that $\lambda_i > s$. From lemma 2, there is a line l of rank $r_l = \lambda_i - 1$. Therefore, $r_l \geq s$. Since there is no line of rank s, $r_l > s$. By definition 20, there are r_l lines younger than l, and the $(s + 1)$th youngest is of rank s. By contradiction, $\lambda_i \leq s$ and therefore $\lambda_{i+1} \leq s + 1$.

We showed that $\forall i \in \mathbb{N}$, $\lambda_i \leq \max \Sigma + 1$. In addition, $\forall i \in \mathbb{N}$, $\lambda_i \leq C$ by definition 23. By the monotone convergence theorem, (λ_i) converges and $\lim\limits_{k\to\infty} \lambda_k \in [0, c]$.

Let $\lambda \in [0, c - 1]$ and $i \in \mathbb{N}$ such that $\lambda_i = \lambda$. By theorem 5, an access of stack distance c is a miss, and by lemma 1, it does not yield an eviction. Therefore,

$$\forall n \in \mathbb{N}, \mathbb{P}(\lambda_{i+n} = \lambda) \leq (1 - \mathbb{P}(\Sigma = c))^n$$

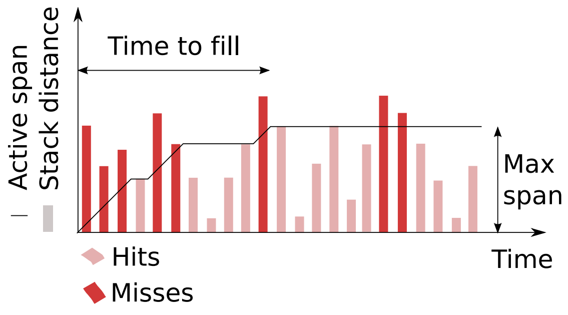


Figure 6.5: Memory accesses in a single, isolated time quantum.

By taking the limit:

$$\mathbb{P}\left(\lim_{k\to\infty} \lambda_k = \lambda\right) = 0$$

It remains that:

$$\mathbb{P}\left(\lim_{k\to\infty} \lambda_k = c\right) = 1$$

□

Definition 25 (Time to fill)**.** *The **time to fill** is the time at which the active span reaches the maximum span. If λ_i is the active span at time i and $\varDelta$ is the time to fill:*

$$\varDelta = \min\{i \in [1, +\infty] | \lambda_i = c\}$$

The time to fill may be greater than the quantum duration. In this case the active span does not reach the maximum span during the quantum.

Figure 6.5 represents memory accesses in a single, isolated time quantum. The x coordinate enumerates accesses in chronological order. The y coordinate shows the stack distance of each access with a vertical bar and the active span with a horizontal line.

6.3.4 Propagation

This section introduces *propagation* to measure cache consistency between two time quanta.

Definition 26 (Propagation)**.** *The **propagation** is the proportion of cached lines that are not erased between two time quanta of the process of interest.*

The evaluation of propagation is not detailed in this chapter.

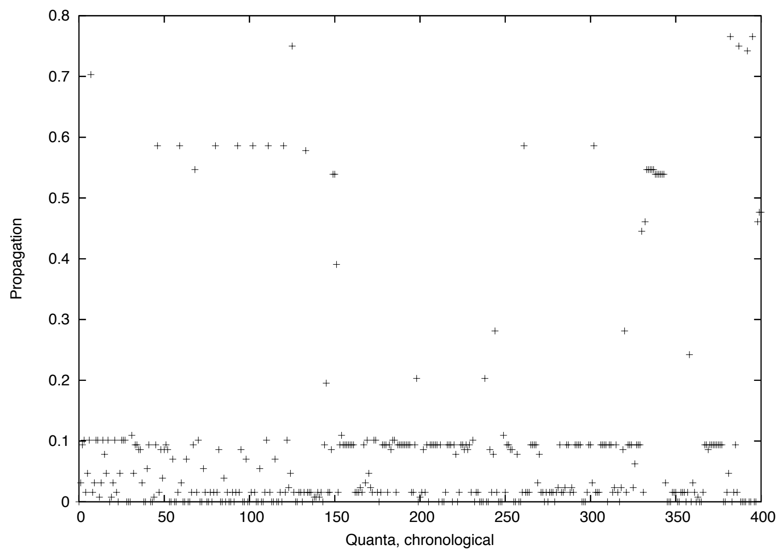


Figure 6.6: Propagation observed between successive quanta of a process running "alone" on Linux.

Definition 27 (Propagated line)**.** *A cached line of age a is* ***propagated*** *at time i if $a > i$, i.e. if it was last accessed by the process of interest in a previous time quantum and not accessed ever since.*

On a common operating system, many services run together with user processes. Even with a single user process, the propagation between its time quanta is low, as shown by figure 6.6.

6.4 Cache misses

This section shows how the probability of occurrence of cache misses in a time quantum is linked to the cumulative distribution function of stack distance and the metrics defined in section 6.3.

6.4.1 A typology of cache misses

Lines cached by the process were last accessed either in the current time quantum or in a previous time quantum. They are either active or propagated. The request of a line which is neither active nor propagated is a miss.

Definition 28. *A* ***blind access*** *is the request of a non active line.*

Definition 29. *A* ***blind hit*** *is the request of a propagated line.*

Lemma 3. *If M is the set of misses over a time period, B_A the set of blind accesses and B_H the set of blind hits,*

$$M = B_A \backslash B_H$$

with $B_A \backslash B_H = \{x \in B_A | x \notin B_H\}$.

Proof. A miss is the request of a line which is not in cache. Active lines are in cache, therefore a miss is a request of a non active line, i.e. $M \subset B_A$ (1). Propagated lines are in cache, therefore the request of a propagated line is not a miss, i.e. $M \cap B_H = \emptyset$ (2). (1) and (2) yield $M \subset B_A \backslash B_H$.

Conversely, a line which is neither active nor propagated is not in cache. Therefore, its request is a miss, i.e. $B_A \backslash B_H \subset M$. □

6.4.2 Blind accesses count

Theorem 7. *Let λ_i be the active span at i and s_{i+1} the stack distance of $i+1$. $i+1$ is a blind access if and only if*

$$s_{i+1} \geq \lambda_i$$

Proof. By lemma 2, all active lines at i have ranks in $[0, \lambda_i - 1]$. Then theorem 5 applies. □

Corollary 1. *If Σ is the stack distance distribution, i is an access and λ_i is the active span at i, the probability that $i+1$ is a blind access is*

$$\mathbb{P}(i+1 \in B_A) = \mathbb{P}(\Sigma \geq \lambda_i)$$

Corollary 2. *We write $[0, i]$ the $i+1$ first accesses of the time quantum, $[0, i] \cap B_A$ the corresponding blind accesses, λ_k the active span at k, Σ the stack* *distance of the process, c the maximum span, $\mathit{\Delta}$ the time to fill. The expected number of blind accesses on $[0, i]$ is:*

$$\begin{aligned} &E\left[|[0,i] \cap B_A|\right] \\ &= E[\lambda_i] + (i - E[\mathit{\Delta}|\mathit{\Delta} < i])\mathbb{P}(\mathit{\Delta} < i)\mathbb{P}(\mathit{\Sigma} \geq c) \end{aligned}$$

Proof. We use the indicator variable $\mathbf{1}_X$. $\mathbf{1}_X = 1$ in the event X and 0 otherwise. For all $k \in [0, i]$, let s_k be the stack distance at access k. From corollary 1:

$$\begin{aligned} E[|[0,i] \cap B_A|] &= E\left[1 + \sum_{k=1}^{i} \mathbf{1}_{s_k \geq \lambda_{k-1}}\right] \\ &= E\left[1 + \sum_{k=1}^{\min(i,\mathit{\Delta})} \mathbf{1}_{s_k \geq \lambda_{k-1}}\right] \\ &+ E\left[\mathbf{1}_{i>\mathit{\Delta}} \sum_{k=\mathit{\Delta}+1}^{i} \mathbf{1}_{s_k \geq \lambda_{k-1}}\right] \end{aligned}$$

By theorem 6, $k < \mathit{\Delta} \Leftrightarrow \lambda_k < c$ and $k \geq \mathit{\Delta} \Leftrightarrow \lambda_k = c$.

$$\begin{aligned} &E\left[\mathbf{1}_{i>\mathit{\Delta}} \sum_{k=\mathit{\Delta}+1}^{i} \mathbf{1}_{s_k \geq \lambda_{k-1}}\right] \\ &= E\left[\mathbf{1}_{i>\mathit{\Delta}} \sum_{k=\mathit{\Delta}+1}^{i} \mathbf{1}_{s_k \geq c}\right] \\ &= E\left[\mathbf{1}_{i>\mathit{\Delta}} \sum_{k=\mathit{\Delta}+1}^{i} 1\right] \mathbb{P}(\mathit{\Sigma} \geq c) \\ &= E\left[\mathbf{1}_{i>\mathit{\Delta}}(i - \mathit{\Delta} + 1)\right] \mathbb{P}(\mathit{\Sigma} \geq c) \\ &= (i - E[\mathit{\Delta}|\mathit{\Delta} < i])\mathbb{P}(\mathit{\Delta} < i)\mathbb{P}(\mathit{\Sigma} \geq c) \end{aligned}$$

By recursion on i, we show that

$$1 + \sum_{k=1}^{\min(i,\mathit{\Delta})} \mathbf{1}_{s_k \geq \lambda_{k-1}} = \lambda_i$$

If $i = 1$, $\lambda_0 = 1$. Suppose that the hypothesis is true for i, we examine case

$i+1$. If $i+1 \leq \mathit{\Delta}$

$$\begin{aligned} 1 + \sum_{k=1}^{\min(i,\mathit{\Delta})} \mathbf{1}_{s_k \geq \lambda_{k-1}} &= \lambda_i + \mathbf{1}_{s_k \geq \lambda_{k-1}} \\ &= \lambda_{i+1} \end{aligned}$$

If $i+1 > \mathit{\Delta}$

$$\begin{aligned} 1 + \sum_{k=1}^{\min(i,\mathit{\Delta})} \mathbf{1}_{s_k \geq \lambda_{k-1}} &= \lambda_i \\ &= \lambda_{i+1} \end{aligned}$$

□

6.4.3 Blind hits count

Theorem 8. *Let $i \in \mathbb{N}$. If $i+1$ is a request of line l with stack distance s, λ_i the active span at i and Λ_i the total span at i,*

$$\lambda_i \leq s < \Lambda_i \Leftrightarrow l \textit{ is propagated and not active at } i$$

Proof. This implication is straightforward:

$$l \text{ is propagated and not active at } i \Rightarrow \lambda_i \leq s < \Lambda_i$$

We prove that:

$$\lambda_i \leq s < \Lambda_i \Rightarrow l \text{ is propagated and not active at } i$$

Since $s < \Lambda_i$, there is a cached line of rank s at i. From theorem 5, this line is l. Since its rank $s \geq \lambda_i$, l is not active. However, l is in cache, and therefore, l is propagated. □

Corollary 3. *Let $i \in \mathbb{N}$. If $i+1$ is an access of stack distance s, λ_i the active span at i, Λ_i the total span at i, the probability that $i+1$ is a blind hit is:*

$$\mathbb{P}(i+1 \in B_H) = \mathbb{P}(s < \Lambda_i)\mathbb{P}(s \geq \lambda_i)$$

And for $i = 0$, with ρ the propagation:

$$\mathbb{P}(0 \in B_H) = \rho\mathbb{P}(s < \Lambda_{-1})$$

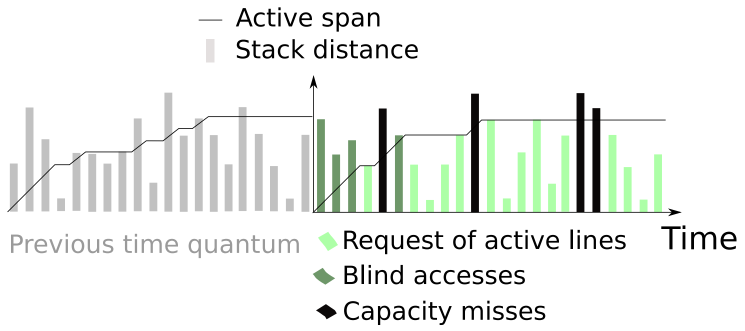


Figure 6.7: Accesses in successive time quanta

Proof. The general case follows from theorem 8. We show the case $i = 0$. Access 0 is blind by definition. Supposing that $\rho = 1$, From theorem 5 it is a hit if and only if the line l of rank s at -1 is still in cache before the access. l is in cache at -1 with probability $\mathbb{P}(s < \Lambda_{-1})$ and l is still in cache before access 0 with probability $\rho\mathbb{P}(s < \Lambda_{-1})$. □

Corollary 4. *For $i \in \mathbb{N}$, the expected number of blind hits on $[0, i]$ is*

$$\begin{aligned} &E\left[|[0,i] \cap B_H|\right] \\ &= \rho\mathbb{P}(s < \Lambda_{-1}) + \sum_{s=1}^{c}\sum_{k=0}^{i-1} \mathbb{P}(\Sigma = s)\mathbb{P}(\Lambda_k > s)\mathbb{P}(\lambda_k \leq s) \end{aligned}$$

Proof. From corollary 3,

$$\begin{aligned} &E\left[|[0,i] \cap B_H|\right] \\ &= \rho\mathbb{P}(s < \Lambda_{-1}) + \sum_{k=1}^{i} \mathbb{P}(\Sigma < \Lambda_{k-1})\mathbb{P}(\Sigma \geq \lambda_{k-1}) \end{aligned}$$

□

Figure 6.7 shows accesses in non-isolated time quanta. The x axis is time. On the y axis, vertical bars represent stack distances and the horizontal line represents active span. Positive x's are accesses in current time quantum, and negative x's (left panel) are accesses in previous time quantum of the same process. On current time quantum, accesses of stack distance greater than cache capacity (also known as **capacity misses** in [HP06]) are shown with dark bars. Blind accesses are shown with medium-light bars. Depending on propagation,

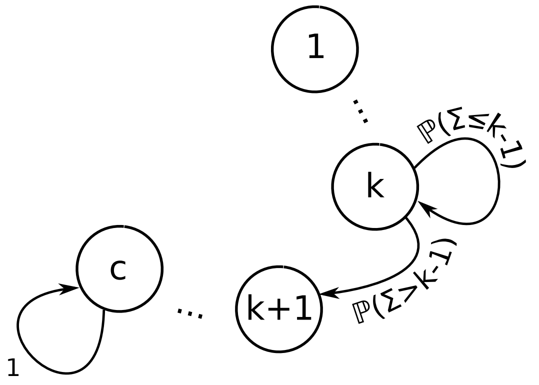


Figure 6.8: State diagram of active span indexed by time $(\lambda_i)_{i\in\mathbb{N}}$

some are hits and others are misses. Requests of active lines are shown with light bars. These are hits.

6.5 Stochastic analysis

The metrics defined and used in previous sections are related to stack distance. This section explains how. They are based on a Markov process. The stack distance distribution Σ appears in transition probabilities.

6.5.1 Active span

Theorem 9. *The active span indexed by time in a quantum, $(\lambda_i)_{i\in\mathbb{N}}$, is a Markov chain with the following transitions:*

– $\forall k \in [0, c-1]$, $\forall i \in \mathbb{N}$,

$$\mathbb{P}(\lambda_{i+1} = k | \lambda_i = k) = \mathbb{P}(\Sigma \leq k-1)$$

$$\mathbb{P}(\lambda_{i+1} = s+1 | \lambda_i = s) = \mathbb{P}(\Sigma > k-1)$$

– $\forall i \in \mathbb{N}$, $\mathbb{P}(\lambda_{i+1} = c | \lambda_i = c) = 1$

Proof. Let $i \in \mathbb{N}$. From theorem 7, $i+1$ is a blind access with probability $\mathbb{P}(\Sigma \leq \lambda_i - 1)$. While the active span is not maximal, blind accesses increment it (proof of theorem 6). Theorem 6 also states that when the active span is maximal, it remains constant. □

Figure 6.8 shows the state diagram of $(\lambda_i)_{i\in\mathbb{N}}$.

Let P_f be the transition matrix of $(\lambda_i)_{i\in\mathbb{N}}$. Diagonal elements are supposed non zero. However, they can be chosen as small as necessary.

We write $F_k = \mathbb{P}(\Sigma \leq k)$ and $\bar{F}_k = \mathbb{P}(\Sigma > k)$

$$P_f = \begin{pmatrix} F_0 & \bar{F}_0 & & \\ & \ddots & \ddots & \\ & & F_{c-2} & \bar{F}_{c-2} \\ & & & F_{c-1} \end{pmatrix} = \begin{pmatrix} T_f & \mathbf{T_f^0} \\ \mathbf{0} & 1 \end{pmatrix}$$

Since $c \geq \max \Sigma + 1,\ F_c = \mathbb{P}(\Sigma \leq c) = 1$. This defines the $1 \times (c-1)$ vector $\mathbf{T_f^0}$ and the $(c-1) \times (c-1)$ substochastic matrix T_f.

Since P_f is triangular, P_f is diagonalizable and its eigenvalues are its diagonal values. Therefore, there is a passage matrix A_{P_f} such that $A_{P_f}^{-1} P_f A_{P_f} = D_{P_f}$ with:

$$D_{P_f} = \begin{pmatrix} F_0 & & \\ & \ddots & \\ & & F_{c-1} \end{pmatrix}$$

Since A_{P_f} is invertible, $\forall n \in \mathbb{N},\ {P_f}^n = A_{P_f} {D_{P_f}}^n {A_{P_f}}^{-1}$.

Let $\boldsymbol{\tau_f}$ be the initial distribution of $(\lambda_i)_{i\in\mathbb{N}}$, i.e. the list of probabilities of each state at access 0. Since $\mathbb{P}(\lambda_0 = 0) = 1$ and $\mathbb{P}(\lambda_0 > 0) = 0$, $\boldsymbol{\tau_f} = [1, 0, \ldots, 0]$.

Corollary 5. *The expected active span is written:*

$$\begin{aligned} E[\lambda_i] &= \sum_{k=1}^{c} k\mathbb{P}(\lambda_i = k) \\ &= \boldsymbol{\tau_f} {P_f}^i [1, \ldots, c]^t \end{aligned}$$

6.5.2 Time to fill

Lemma 4. *$(\lambda_i)_{i\in\mathbb{N}}$ is terminating and c is an absorbing state.*

Proof. This is a direct consequence of theorem 6. It can also be seen on P_f. $\forall i \in [0, c-1], 0 \leq F_i < 1$ and $F_c = 1$ therefore,

$$\exists x \in \mathbb{Q} | \lim_{n\to\infty} {P_f}^n = \begin{pmatrix} 0 & & & 0 \\ & \ddots & & \vdots \\ & & 0 & 0 \\ 0 & \ldots & 0 & x \end{pmatrix}$$

Since P_f is stochastic $[0, \ldots, 0, 1] P_f^n \mathbf{1} = 1$. By taking the limit, $x = 1$.

As a consequence,

$$\lim_{n\to\infty} {D_{P_f}}^n = \begin{pmatrix} 0 & & & 0 \\ & \ddots & & \vdots \\ & & 0 & 0 \\ 0 & \dots & 0 & 1 \end{pmatrix} \text{with exponential speed}$$

□

Theorem 10. *The time to fill, Δ, is a discrete phase type distribution.*

Proof. Δ takes values in $\mathbb{N}$. $(\lambda_i)_{i\in\mathbb{N}}$ is a terminating Markov chain with finitely many states. The maximum span, c, is its absorbing state. Δ is the first passage time to c, therefore, Δ, is a discrete phase type distribution. □

Corollary 6. *The f_Δ be the probability mass function and F_Δ the cumulative distribution function of Δ, i.e. $\forall k \in \mathbb{N}$,*

$$f_\Delta(k) = \mathbb{P}(\Delta = k)$$
$$F_\Delta(k) = \mathbb{P}(\Delta \leq k)$$

As a characterization of a discrete phase type distribution:

$$f_\Delta(k) = \boldsymbol{\tau_f} {T_f}^{k-1}\mathbf{T^0_f}$$
$$F_\Delta(k) = 1 - \boldsymbol{\tau_f} {T_f}^{k}\mathbf{1}$$

Corollary 7. *If Δ is the time to fill, $\forall k \in \mathbb{N}$, P_f the transition matrix of active span, T_f its sub-stochastic matrix, and $T_f = A_{T_f} D_{T_f} A_{T_f}^{-1}$ the diagonalization of T_f, The expected value of Δ under the condition that $\Delta \leq i$, is:*

$$E[\Delta|\Delta \leq i] = \boldsymbol{\tau_f} A_{T_f} \left(\sum_{k=0}^{i} k {D_{T_f}}^{k-1} \right) A_{T_f}^{-1} \mathbf{T^0_f}$$

Proof. Let $f_\Delta(k) = \mathbb{P}(\Delta = k)$.

$$\begin{aligned}
E[\Delta|\Delta \le i] &= \sum_{k=0}^{i} k f_\Delta(k) \\
&= \sum_{k=0}^{i} k \boldsymbol{\tau_f} {T_f}^{k-1} \mathbf{T_f^0} \\
&= \boldsymbol{\tau_f} \left(\sum_{k=0}^{i} k {T_f}^{k-1} \right) \mathbf{T_f^0} \\
&= \boldsymbol{\tau_f} A_{T_f} \left(\sum_{k=0}^{i} k {D_{T_f}}^{k-1} \right) A_{T_f}^{-1} \mathbf{T_f^0}
\end{aligned}$$

□

6.6 Algorithm and complexity

The following expression gives the number of cache misses in a time quantum based on knowledge of the stack distance distribution, the number of accesses in the quantum, and the assumption that no line was propagated from previous quantum.

Theorem 11. *Let n be the duration of a time quantum, λ_n the active span at n, Δ the time to fill, and Σ the stack distance distribution. We write P_f the transition matrix of the active span, T_f its sub-stochastic matrix, $\boldsymbol{\tau_f} = [1, 0, \ldots, 0]$ of size c, $\mathbf{T_f^0} = [0, \ldots, 0, \bar{F}(c-2)]^t$ of size $c-1$, $\mathbf{1} = [1, \ldots, 1]^t$ of size $c-1$, and $\forall k \in \mathbb{N}$, $F(k) = \mathbb{P}(\Sigma \le c)$ and $\bar{F}(k) = \mathbb{P}(\Sigma > k)$.*

In the absence of propagation, the number of cache misses in the quantum is:

$$\begin{aligned}
M &= E[\lambda_n] + (n - E[\Delta|\Delta < n])\mathbb{P}(\Delta < n)\mathbb{P}(\Sigma \ge c) \\
&= \boldsymbol{\tau_f} {P_f}^n [1, \ldots, c]^t \\
&+ \left(n - \sum_{k=0}^{n} k \boldsymbol{\tau_f} {T_f}^{k-1} \mathbf{T_f^0} \right) (1 - \boldsymbol{\tau_f} {T_f}^n \mathbf{1})(1 - F(c-1))
\end{aligned}$$

In addition, the computational complexity is the complexity of the implicit-shifted QR algorithm [DK90] on P_f and T_f, and the rest of the computation is done in $O(c^3)$.

Proof. From corollary 2, M is the number of blind accesses in the time quantum. Since there is no propagation, there is no blind hit by definition. Therefore, from lemma 3, M is the number of cache misses in the time quantum.

From corollary 5,

$$E[\lambda_n] = \boldsymbol{\tau_f} {P_f}^n [1, \dots, c]^t$$

From corollary 6,

$$\mathbb{P}(\Delta < n) = 1 - \boldsymbol{\tau_f} {T_f}^n \mathbf{1}$$

From corollary 7,

$$E[\Delta | \Delta < n] = \boldsymbol{\tau_f} A_{T_f} \left(\sum_{k=0}^{n} k {D_{T_f}}^{k-1} \right) A_{T_f}^{-1} \mathbf{T_f^0}$$

P_f and T_f are bidiagonal. The implicit-shifted QR algorithm can be used to approximate their singular values.

Once P_f and T_f are diagonalized, P_f^n and T_f^n are computed in $O(c^3)$. Therefore, $E[\lambda_n]$, $\mathbb{P}(\Delta < n)$ and $E[\Delta|\Delta < n]$ are computed in $O(c^3)$. □

The QR algorithm approximates the singular value decomposition of a bidiagonal matrix. It is iterative, and new iterations are done until enough precision is reached. In fact, we suspect that use of the QR algorithm is not necessary and that the overall complexity is in fact in $O(c^2)$. This is due to the fact that the transition matrix of active span P_f and its sub-stochastic matrix T_f are bidiagonal stochastic. The following presents this faster alternative.

We consider a bidiagonal right-stochastic matrix M_n of size n and we write its diagonal elements $F_0 \dots F_{n-1}$ and $\forall k \in [0, n-1]$, $\bar{F}_k = 1 - F_k$.

$$M_n = \begin{pmatrix} F_0 & \bar{F}_0 & & \\ & \ddots & \ddots & \\ & & F_{n-2} & \bar{F}_{n-2} \\ & & & F_{n-1} \end{pmatrix}$$

If $\forall k \in [0 \dots n-1]$, $F_k = \mathbb{P}(\Sigma \leq k)$, then $P_f = M_c$ and $T_f = M_{c-1}$. We write the diagonalization of M_n:

$$M_n = A_n D_n A_n^{-1}$$

with D_n diagonal and A_n invertible.

A pattern systematically appears in A_n and A_n^{-1}, for every n. If ${A_n}_{i,j}$ is the element at row i and column j of A_n, and ${A_n^{-1}}_{i,j}$ is the element at row i and column j of A_n^{-1}, $\forall(i,j) \in [0, n-1]^2$, we observe that:

$$A_{n\,i,j} = \prod_{k=i}^{j-1} \frac{\bar{F_k}}{F_j - F_k} \delta_{i \leq j}$$

$$A_n{}^{-1}{}_{i,j} = \frac{\prod_{k=i}^{j-1} \bar{F_k}}{\prod_{k=i+1}^{j} F_i - F_k} \delta_{i \leq j}$$

δ is the Kronecker symbol, i.e. $\delta_X = 1$ if X is true, 0 otherwise.

Therefore, we express $\boldsymbol{\tau}_n A_n D A_n{}^{-1}$ where D is diagonal with diagonal elements d_s, $s \in [0, n-1]$, and $\boldsymbol{\tau_n} = [1, 0, ..., 0]$ of size n.

Let $v = \boldsymbol{\tau_n} A_n D A_n{}^{-1}$ and v_j the element at index j in line v.

$$\begin{aligned}
v_j &= \sum_{s=0}^{j} \left(\prod_{k=0}^{s-1} \frac{\bar{F_k}}{F_s - F_k} \right) \delta_{0 \leq s} d_s \frac{\prod_{k=s}^{j-1} \bar{F_k}}{\prod_{k=s+1}^{j} F_s - F_k} \delta_{s \leq j} \\
&= \sum_{s=0}^{j} d_s \frac{\prod_{k=0}^{j-1} \bar{F_k}}{\prod_{\substack{k=0 \\ k \neq s}}^{j} F_s - F_k} \\
&= \left(\prod_{k=0}^{j-1} \bar{F_k} \right) \sum_{s=0}^{j} d_s \prod_{\substack{k=0 \\ k \neq s}}^{j} (F_s - F_k)^{-1}
\end{aligned}$$

Since all members of the expression of the number of cache misses in theorem 11 are on this form, the complexity of the algorithm is in $O(c^2)$ for every size c of bidiagonal stochastic matrix for which the pattern is observed, which we suspect is all $\mathbb{N}$. However, we have not been able to prove it.

6.7 Experimental validation

The ratio of cache misses on a single time quantum is representative of the whole run if all quanta have same number of memory accesses, i.e. duration. Figure 6.9 shows the duration of successive time quanta for an execution of the Unix command *ls*. We observe a wide variability on a logarithmic scale. However, the effect of the hypothesis that the time quantum duration is constant is minor compared to the effect of hypotheses on propagation.

The following measurements are taken with Simics, a full system micro-architecture simulator described in [MCE$^+$02]. The simulated platform is a x86 processor with a LRU, fully associative cache, running a Linux operating system. Measurements are taken for different benchmarks running "alone", i.e. only in competition with the default background services of the operating system.

Simics returns the actual number of cache misses in the whole run. In addition, it allows to monitor memory accesses, and it provides access to cache contents at any step in the execution. We use this information to monitor the propagation at every time quantum, and we simulate the same memory accesses on synthetic time quanta with modified propagation and duration. The goal is to replicate the effect of the simplifying hypotheses made by the algorithm of this chapter, in comparison with other hypotheses made by existing or hypothetical algorithms.

The instrumentation of Simics and the simulation of synthetic time quanta required some programming. The code is written in Python using test-driven development. It is available under Artistic License 2 on a public repository [2].

Figures 6.10 to 6.14 show the relative number of cache misses under different assumptions, with regards to the actual number of cache misses.

1. **Warm cache** is the common hypothesis of prediction algorithms that do not account for the effect of context switches. A cache miss occurs for every stack distance greater than the cache capacity.
2. **Full propagation** means that all lines are propagated between two quanta of the same process. By contrast with *warm cache*, a cache miss is counted at every first access to a memory line.
3. **Actual propagation** only differs from the observation by the duration of the time quanta. It is set constant, and equal to the average actual duration.

2. code.google.com/p/mtc-project

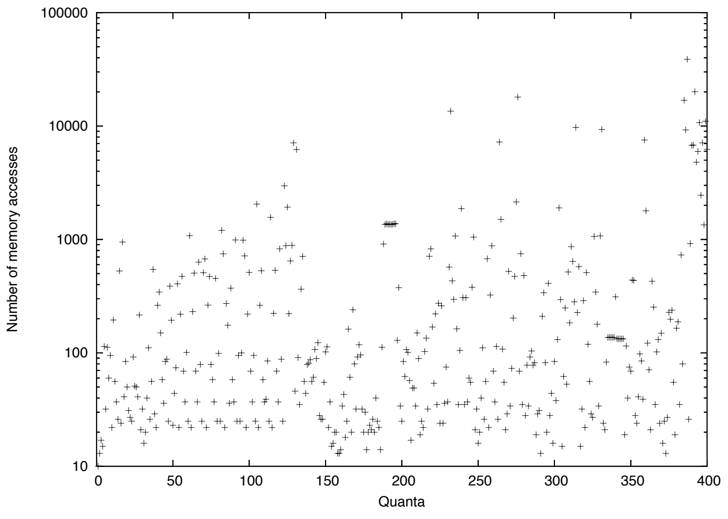


Figure 6.9: Number of memory accesses (= duration) of successive time quanta.

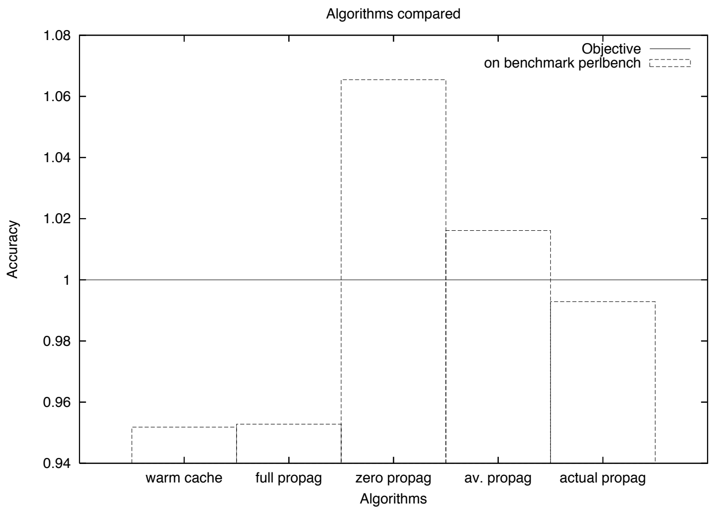


Figure 6.10: perlbench

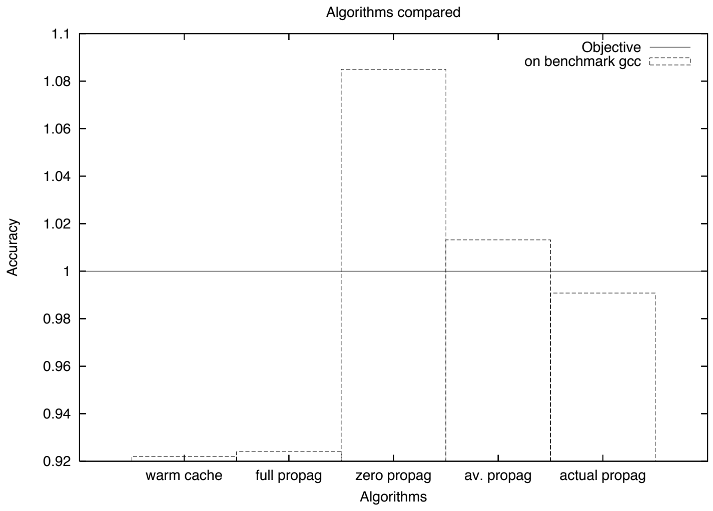


Figure 6.11: gcc

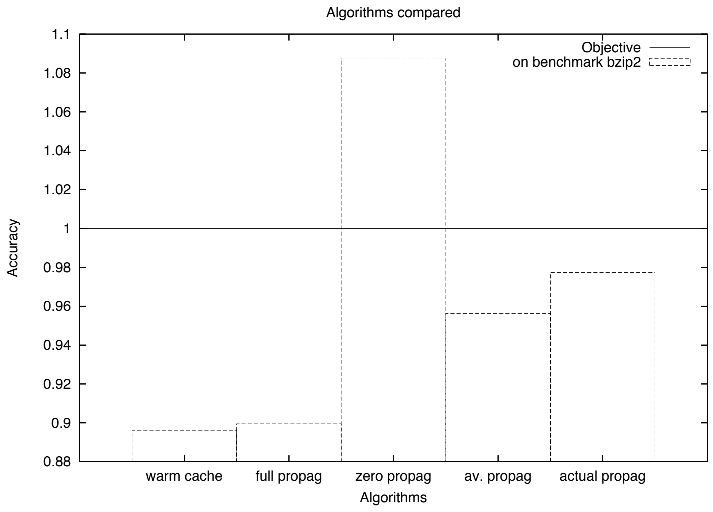


Figure 6.12: bzip2

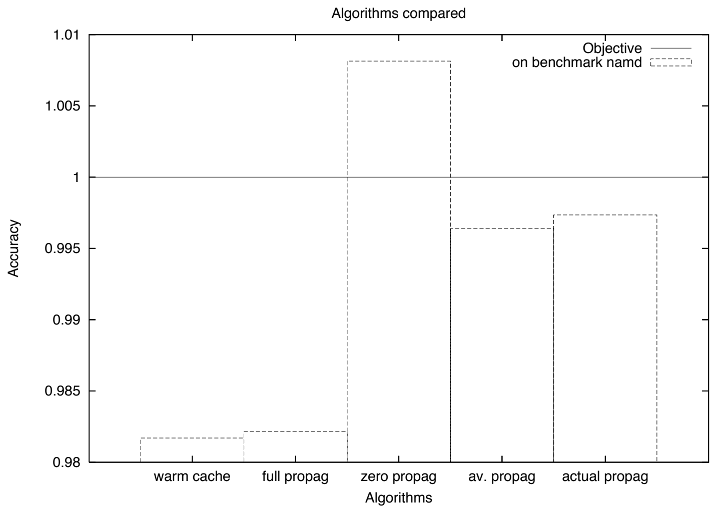


Figure 6.13: namd

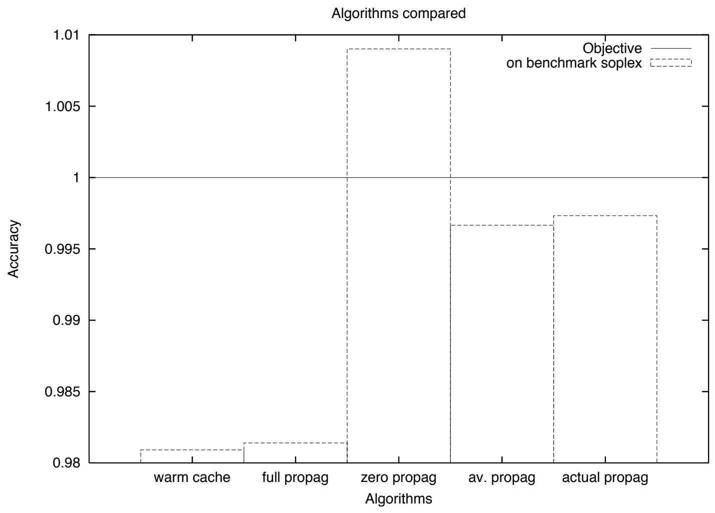


Figure 6.14: soplex

4. **Average propagation** also sets quantum duration as the average of the actual value. In addition, it differs from the observation by the propagation, which is set constant and equal to the average actual propagation.
5. **Zero propagation** is the result of the calculation proposed by theorem 11. It supposes that the time quantum duration is constant and the propagation is zero, i.e. it counts a cache miss for every blind access.

Figures 6.10 to 6.14 confirm that the expression of theorem 11 gives a higher bound of the cache miss ratio, and that the monotasking assumption gives a lower bound. In addition, we observe the distance to the actual value is qualitatively the same for the higher and the lower bounds. The lower bound, however, remains much faster to compute, as shown in chapter 5.

6.8 Conclusion

Since the advent of multitasking, context switches have been known to generate cache misses. However, prior to this work, there has been no mean other than simulation to predict their impact from the observation of memory access patterns. The apparent complexity of the analysis justifies this blank with regards to the coverage of the monotasking case, and sustains the opinion that caches are unpredictable.

Still, the process by which cache warms up at the beginning of a quantum exhibits convenient properties for analysis. Prior algorithms consider that an access of stack distance greater than the cache capacity is a miss. This is true but it only gives a lower bound of the cache miss ratio. In fact, an access of stack distance greater than the number of lines cached since the beginning of the quantum is a possible miss, unless the requested line was written in a previous quantum and not erased by other processes inbetween. Considering that no line is kept between quanta gives a higher bound of the cache miss ratio. We observe in exeriments that higher and lower bounds are equally distant to the actual value.

Conclusion

Synthesis

Architectural patterns appear to largely determine performance outcomes in an environment where multiple objectives collide. The recent adoption of Late Binding motivates a re-thinking of grid architectures for performance. With Late Binding, users bypass grid services developed for interoperability between the autonomous institutions involved. Users with common applications and goals find it beneficial to jointly allocate their workload, instead of relying on a third party service. However, the underlying grid middleware is not primarily designed for, and unaware of the actual allocation taking place.

We learnt the lesson and propose Symmetric Mapping as the pattern for next generation production grids. Symmetric Mapping separates the concerns of grid participants straight from the architectural design.

We imported the notion of Multiple Administrative Domains (MAD) systems from the area of fault tolerance. Although being a MAD *is* the definition of a grid among other distributed computing systems, and is the source of difficulties in allocating grid resources, the MAD problem was not formally addressed before in the study of grid systems. Indeed, Models based on queuing theory traditionally used in grid design do not capture the diverging objectives of autonomous participants. Therefore, we had to create a new model that allows to express multiple constraints and dynamic allocations, and we user it to express the MAD problem for grids and solved it with a formal definition of Symmetric Mapping.

The existence of a solution is subject to the ability to divide the overall allocation into smaller pieces, *containers*, that isolate resource users from resource providers and determine the perceived value on both sides. Under this hypothesis, the different objectives can be solved independently.

In order to simulate the benefits of Symmetric Mapping, we identified objectives of resource users and resource providers. Users are interested in minimum makespan or minimum sum of weighted flows, while providers are concerned with energy consumption and obstruction to maintenance and internal use. Each autonomous participant has specific knowledge and priorities relative to the mechanisms that determine her perceived value of the allocation. Simulations suggest that Symmetric Mapping is beneficial even if participants can only approximately optimize their perceived value.

Virtual machines are appropriate candidates to implement containers. We proposed a framework that configures and deploys virtual machines, and manages their life-cycle based on declarative descriptions. Providers achieve their objectives by configuring these pools to be backed with appropriate physical resources.

We proposed a framework that reacts to changes in nodes availability, elects new nodes and re-deploys discontinued services, so that users can implement their own permanent resource management services on the transient containers that they obtain on a grid.

The first part of our study suggests that it is possible to map tasks directly and dynamically on heterogeneous servers of a grid. However, the literature does not provide means to predict the performance of a mapping. Cache misses are central to processor performance and vary with user loads in a manner reputedly unpredictable. To start up, we focused on the performance of LRU caches.

The analysis of memory access patterns is widely used for performance prediction under monotasking. A metric, stack distance, captures the locality of memory accesses for matching with processor caches. Fitting stack distance with known probability distributions improves the computational complexity of existing methods. It allows for predictions with constant complexity, which is valuable for online matching. Simulations suggest that the computational improvement does not affect accuracy.

The monotasking hypothesis gives a lower bound on the cache miss ratio. With multitasking, which is the rule on mainstream operating systems, additional misses are caused by context switches. A higher bound of the cache misses overhead from context switches results from a new stochastic analysis of how the cache warms up, i.e. fills up with useful data. Simulations suggest that the higher bound is as close as the lower bound to the actual cache miss ratio. The resulting segment covers the exact cache miss ratio in a finite time quantum, potentially representative of performance of the whole execution. This result fills a blank in performance prediction and helps better inform grid users for a careful placement of their tasks.

Future work

We suspect that the model we introduced for the problem of multiple administrative domains (MAD) in chapter 2 can be applied to other problems and to resource allocation in other environments as it allows to reason formally on dynamic and fine grained allocation under constraints.

The MAD formalism may also apply to resource allocation outside of production grids. We have exhibited one solution based on specific containment properties, and other solutions may exist.

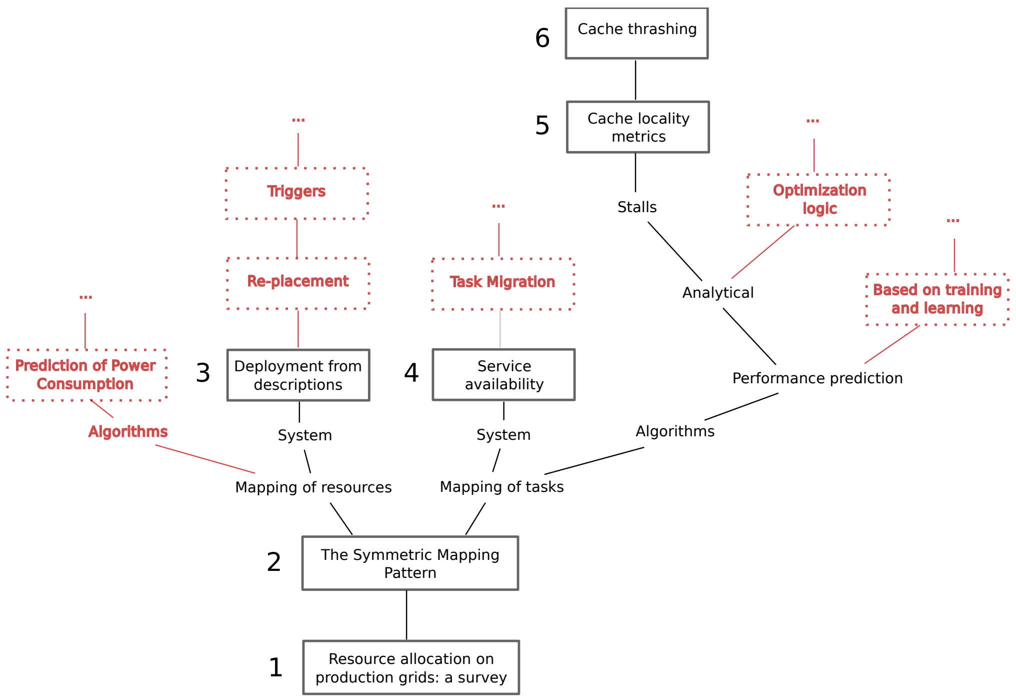


Figure 6.15: Future work.

In addition, this work only opens up the wide question of performance on computing grids. Further research is needed to complete the study of efficient and non obstructive grid resource allocation. Along to the completed investigations, figure 6.15 represents in dashed rectangles some remaining problems.

Among the two sides of resource allocation that we define, we spare more efforts on the user side, i.e. on task placement. Processing speed is an obvious objective we addressed, but energy savings, for instance, are equally important. The algorithms with which providers select physical resources to support virtual environments deserve investigation. Energy-aware scheduling can be explored in view of applications to grids. This would motivate a study on the means and conditions for seamless server pre-emption, for instance based on resource monitoring and virtual machine migration.

In the implementation detailed in chapter 3, isolation with virtual machines provides to some extent the desired containment properties. Instead of resource isolation, we suspect that mechanisms based on incentives and reputation could guarantee the same properties with a low cost on performance.

For dynamic scheduling, the potential distance between execution environments creates additional difficulties. Task migration involves data transfers. Strategies are needed to reduce their cost. Mechanisms that trigger task replacements are not covered in this work.

In chapters 5 and 6 our approach is analytical. We propose to help scheduling by predicting quickly and accurately the affinity between tasks and processors, based on a prior analysis of both. Instead, the use of learning techniques based on recurrent observations is a valid alternative.

Our contributions to performance prediction are limited to the analysis of cache misses, i.e. failures to fetch data from cache, in isolation from processor optimizations. Cache misses are determinant for overall processor performance. However, cache misses are largely mitigated by processor parallelism and optimizations such as prefetching and branch prediction. A model that accurately predicts overall performance remains to be found.

To conclude, this thesis uncovers the potential significance of further research in fine-grained resource analysis and management for grid performance. This is a by-product of our contributions: we point out the real need and current status; we propose a new perspective and its model; and we provide solutions to essential parts of the problem.

Publications

- Chapter 1 is based on a technical report published by Telecom ParisTech [GDJ08].
- Chapter 2 is based on a paper published in the proceedings of SMTPS'09 [GD09] and a technical report published by Telecom ParisTech [GD10].
- Parts of chapter 3 are based on a paper published in the proceedings of VHPC'06 [GPJ^{+}07].
- Chapter 5 is based on a paper published in the proceedings of PDPTA'08 [GJ08].
- Chapter 6 is based on a technical report published by Telecom ParisTech [Gre10].

Bibliography

[AAC$^+$05] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for co-operative services. *SIGOPS Oper. Syst. Rev.*, 39(5):45–58, 2005.

[ABD$^+$07] Sergio Andreozzi, Stephen Burke, Flavia Donno, Laurence Field, Steve Fisher, Jens Jensen, Balazs Konya, Maarten Litmaath, Marco Mambelli, Jennifer M. Schopf, Matt Viljoen, Antony Wilson, and Riccardo Zappi. Glue schema specification version 1.3. glueschema.forge.cnaf.infn.it/Spec/V13, 16 Jan 2007.

[ABGL00] K. M. Anstreicher, N. W. Brixius, J.-P. Goux, and J. Linderoth. Solving large quadratic assignment problems on computational grids. Technical report, MetaNEOS project, Iowa City, Iowa 52242, 2000.

[ABK$^+$04] Parvin Asadzadeh, Rajkumar Buyya, Chun Ling Kei, Deepa Nayar, and Srikumar Venugopal. Global grids and software toolkits: A study of four grid middleware technologies. Technical Report GRIDS-TR-2004-5, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, July 1 2004.

[AHH89] A. Agarwal, J. Hennessy, and M. Horowitz. An analytical cache model. *ACM Trans. Comput. Syst.*, 7(2):184–215, 1989.

[All90] Arnold O. Allen. *Probability, statistics, and queueing theory with computer science applications.* Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[AMZ03] Gagan Aggarwal, Rajeev Motwani, and An Zhu. The load rebalancing problem. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 258–265, New York, NY, USA, 2003. ACM Press.

[And03] David P. Anderson. Public computing: Reconnecting people to science. In *Proceedings of the Conference on Shared Knowledge and the Web*, Residencia de Estudiantes, Madrid, Spain, Nov. 17-19 2003.

[And04] P. Andreetto. Practical approaches to grid workload and resource management in the egee project. In *CHEP '04: Proceedings of the Conference on Computing in High Energy and Nuclear Physics*, volume 2, pages 899–902, Interlaken, Switzerland, 09 2004.

[And06] P. Andreetto. Cream: A simple, grid-accessible, job management system for local computational resources. In *Proceedings of CHEP'06*, Mumbay, India, February 2006.

[Ave07] Paul Avery. Open science grid: Building and sustaining general cyberinfrastructure using a collaborative approach. In *Cyberinfrastructure for Collaboration and Innovation*, number CSD5052, June 2007.

[AZV+02] James Annis, Yong Zhao, Jens Voeckler, Michael Wilde, Steve Kent, and Ian Foster. Applying chimera virtual data concepts to cluster finding in the sloan sky survey. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–14, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[BBB+05] I. Bird, K. Bos, N. Brook, D. Duellmann, C. Eck, I. Fisk, D. Foster, B. Gibbard, C. Grandi, F. Grey, J. Harvey, A. Heiss, F. Hemmer, S. Jarp, R. Jones, D. Kelsey, J. Knobloch, M. Lamanna, H. Marten, P. Mato Vila, F. Ould-Saada, B. Panzer-Steindel, L. Perini, L. Robertson, Y. Schutz, U. Schwickerath, J. Shiers, and T. Wenaus. Lcg technical design report. Technical report, CERN, 06 2005.

[BBC+04] Cyril Banino, Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, and Yves Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Transactions on Parallel and Distributed Systems*, PDS-15(4):319–330, April 2004.

[BD01] K. Beyls and E.H. D'Hollander. Reuse distance as a metric for cache behavior. In T. Gonzalez, editor, *Proceedings of the IASTED*

International Conference on Parallel and Distributed Computing and Systems, pages 617–622, Anaheim, California, USA, 8 2001. IASTED.

[BDF$^+$03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[BDM99] Jacek Blazewicz, Maciej Drozdowski, and Mariusz Markiewicz. Divisible task scheduling concept and verification. *Parallel Computing*, 25(1):87–98, January 1999.

[BE99] Mark Brehob and Richard Enbody. An analytical model of locality and caching. Technical report, Michigan State University, Dept of Computer Science and Engineering, 1999.

[Bel05] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[BF07] Cullen Bash and George Forman. Cool job allocation: Measuring the power savings of placing jobs at cooling-efficient locations in the data center. Technical Report HPL-2007-62, HP Labs, Palo Alto, August 2007.

[BGK$^+$03] A. Baranovski, G. Garzoglio, A. Kreymer, L. Lueking, V. Murthi, P. Mhashikar, F. Ratnikov, A. Roy, T. Rockwell, S. Stonjek T. Tannenbaum, I. Terekhov, R. Walker, and F. Wuerthwein. Management of grid jobs and information within samgrid. In *Proceedings of UK e-Science All Hands Conference*, Nottingham, UK, September 2003.

[BH04] E. Berg and E. Hagersten. Statcache: a probabilistic approach to efficient and accurate data locality analysis. In *ISPASS '04: Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 20–27, Washington, DC, USA, 2004. IEEE Computer Society.

[BHK$^+$00] Brett Bode, David M. Halstead, Ricky Kendall, Zhou Lei, and David Jackson. The portable batch scheduler and the maui scheduler on linux clusters. In *Proceedings of the 4th Annual Showcase*

& Conference (LINUX-00), pages 217–224, Berkeley, CA, October 10–14 2000. The USENIX Association.

[BHL+06] Stefano Belforte, Shih-Chieh Hsu, Elliot Lipeles, Matthew Norman, Frank Wu thwein, Donatella Lucchesi, Subir Sarkar, and Igor Sfiligoi. Glidecaf: A late binding approach to the grid. In *Proceedings of CHEP'06*, 2006.

[BMB+08] Xin Bai, Dan C. Marinescu, Ladislau Bölöni, Howard Jay Siegel, Rose A. Daley, and I-Jeng Wang. A macroeconomic model for resource allocation in large-scale distributed systems. *J. Parallel Distrib. Comput.*, 68(2):182–199, 2008.

[BMR+96a] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, Inc., 1996.

[BMR+96b] Grank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: a system of patterns*, volume 1. John Wiley and Sons, 1996.

[BMT09] Vlastimil Babka, Lukáš Marek, and Petr Tuma. When misses differ: Investigating impact of cache misses on observed performance. In *ICPADS '09: Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems*, pages 112–119, Washington, DC, USA, 2009. IEEE Computer Society.

[BPS03] Predrag Buncic, Andreas J. Peters, and Pablo Saiz. The alien system, status and perspectives. In *Proceedings of Computing in High-Energy Physics*, 2003.

[BPSGO04] Predrag Buncic, A. J. Peters, P. Saiz, and J.F. Grosse-Oetringhaus. The architecture of the alien system. In *Proceedings of CHEP'2004*, Interlaken, Switzerland, 09 2004.

[BR04] Ricardo Bianchini and Ram Rajamony. Power and energy management for server systems. *Computer*, 37(11):68–74, 2004.

[BSK05] Bruce Beckles, Sechang Son, and John Kewley. Current methods for negotiating firewalls for the condor system. In *Proceedings of the 4th UK e-Science All Hands Meeting 2005*, Nottingham, UK, September 2005.

[CCF+94] K. Castagnera, D. Cheng, R. Fatoohi, E. Hook, B. Kramer, C. Manning, J. Musch, C. Niggley, W. Saphir, D. Sheppard,

M. Smith, I. Stockdale, S. Welch, R. Williams, and D. Yip. Nas experiences with a prototype cluster of workstations. In *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 410–419, New York, NY, USA, 1994. ACM Press.

[Cer94] Paul E. Ceruzzi. From batch to interactive: The evolution of computing systems, 1957-1969. In *IFIP Congress (2)*, pages 279–284, 1994.

[CFK99] Karl Czajkowski, Ian T. Foster, and Carl Kesselman. Resource co-allocation in computational grids. In *Proceedings Eighth IEEE International Symposium on High Performance Distributed Computing (8th HPDC'99)*, Redondo Beach, California, USA, March 22 1999. IEEE Computer Society.

[CFK$^+$02] Karl Czajkowski, Ian Foster, Carl Kesselman, Volker Sander, Volker S, and Steven Tuecke. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *In 8th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 153–183, 2002.

[CFK04] Karl Czaijkowski, Ian Foster, and Carl Kesselman. *The Grid 2. Resource and Service Management*, chapter 18, pages 259–283. Morgan Kaufman, 2nd edition, 2004.

[CGKS05] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture. HPCA-11.*, pages 340–351, Feb. 2005.

[CGR$^+$06] L. Cherkasova, D. Gupta, E. Ryabinkin, R. Kurakin, V. Dobretsov, and A. Vahdat. Optimizing grid site manager performance with virtual machines. In *Proc. of the 3rd USENIX Workshop on Real Large Distributed Systems (WORLDS '06)*, Seattle, 11 2006.

[CGV07] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. When virtual is harder than real: Resource allocation challenges in virtual machine based it environments. Technical Report 20070220, HPL, 02 2007.

[Chi04] Andrew A. Chien. *The Grid 2. Computing Elements*, chapter 28, pages 567–591. Morgan Kaufman, 2nd edition, 2004.

[CIL$^+$07] Yuan Chen, Subu Iyer, Xue Liu, Dejan Milojicic, and Akhil Sahai. Sla decomposition: Translating service level objectives to system level thresholds. Technical report, Hewlett-Packard Laboratories, 01 2007.

[CK88] Thomas L. Casavant and Jon G Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEESE*, 14(2), February 1988.

[Cle96] Scott H. Clearwater, editor. *Market-based control: a paradigm for distributed resource allocation.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996.

[CP03] Calin Cascaval and David A. Padua. Estimating cache misses and locality using stack distances. In *ICS*, pages 150–159, 2003.

[CS00] Xinjie Chang and Krishnappa R. Subramanian. A cooperative game theory approach to resource allocation in wireless atm networks. In *NETWORKING '00*, pages 969–978, London, UK, 2000. Springer-Verlag.

[CvR05] Miguel Castro and Robbert van Renesse. *Peer-to-Peer Systems IV: 4th International Workshop, IPTPS 2005, Revised Selected Papers*, volume 3640. Lecture Notes in Computer Science, Ithaca, NY, USA, February 2005.

[Dik01] Jeff Dike. User-mode linux. In *ALS '01: Proceedings of the 5th annual Linux Showcase & Conference*, pages 2–2, Berkeley, CA, USA, 2001. USENIX Association.

[DK90] James Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Stat. Comput.*, 11(5):873–912, 1990.

[DSS$^+$05] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13(3):219–237, 2005.

[EGK$^+$07] M. Ellert, M. Gronager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced resource connector middleware for lightweight computational grids. *Future Gener. Comput. Syst.*, 23(2):219–240, 2007.

[ELvD$^+$96] D. H. J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of condors: load sharing among workstation clusters. *Future Gener. Comput. Syst.*, 12(1):53–65, 1996.

[FACA03] R. Fonseca, V. Almeida, M. Crovella, and B. Abrahao. On the intrinsic locality properties of web reference streams. *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, 1:448–458, 30 March-3 April 2003.

[FB98] Massimo Franceschetti and Jehoshua Bruck. A leader election protocol for fault recovery in asynchronous fully-connected networks. Technical report, 1998.

[Fiu06] Marc E. Fiuczynski. Planetlab: overview, history, and future directions. *Operating Systems Review*, 40(1):6–10, 2006.

[FKNT02] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[Fos02] Ian Foster. What is the grid? a three point checklist. Technical Report 6, Argonne National Labs, 2002.

[Fos05] Ian T. Foster. Service oriented science. *Science*, 308(5723):214–217, 05 2005.

[Fos06] Ian T. Foster. Globus toolkit version 4: Software for service-oriented systems. In *Proceedings of FIP International Conference on Network and Parallel Computing*, volume LNCS 3779, pages 2–13. Springer-Verlag, 2006.

[FTF$^+$02] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steve Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002.

[GAFN94] K. Grimsrud, J. Archibald, R. Frost, and B. Nelson. On the accuracy of memory reference models. In *Proceedings of the 7th international conference on Computer performance evaluation : modelling techniques and tools*, pages 369–388, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[GCC$^+$04] Greg Graham, Richard Cavanaugh, Peter Couvares, Alan De Smet, and Miron Livny. *The Grid 2. Distributed Data Analysis: Federated Computing for High-Energy Physics*, chapter 10, pages 136–145. Morgan Kaufman, 2nd edition, 2004.

[GD07] Xavier Grehant and J. M. Dana. Virtual execution environment supply: a twofold illustration. Technical report, CERN openlab, 05 2007.

[GD09] Xavier Gréhant and Isabelle Demeure. Symmetric mapping: an architectural pattern for resource supply in grids and clouds. In *Proceedings of SMTPS'09*, Rome, May 2009. IEEE.

[GD10] Xavier Gréhant and Isabelle Demeure. Mad resource allocation. Technical report, Telecom ParisTech, 04 2010.

[GDJ08] Xavier Gréhant, Isabelle Demeure, and Sverre Jarp. Towards efficient resource allocation on scientific grids. Technical Report ISSN 0751-1345 ENST D, ENST, Paris, January 2008.

[GGL$^+$03] Patrick Goldsack, Julio Guijarro, A. Lain, G. Mecheneau, P. Murray, and Peter Toft. Smartfrog: Configuration and automatic ignition of distributed applications. Technical report, HP, 2003.

[GHJV95] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, MA, 1995.

[GJ08] Xavier Grehant and Sverre Jarp. Lightweight cache analysis for cache-aware scheduling on heterogeneous clusters. In *Proceedings of PDPTA'08, WorldComp'08*, Las Vegas, 7 2008.

[GLMR07] Tristan Glatard, Diane Lingrand, Johan Montagnat, and Michel Riveill. Impact of the execution context on grid job performances. In *International Workshop on Context-Awareness and Mobility in Grid Computing (WCAMG'07)*, pages 713–718, Rio de Janeiro, May 2007. IEEE.

[GLY00] Jean-Pierre Goux, Jeff Linderoth, and Michael Yoder. Metacomputing and the master-worker paradigm. Technical report, Argonne National Labs, October 17 2000.

[GMP07] Tristan Glatard, Johan Montagnat, and Xavier Pennec. Optimizing jobs timeouts on clusters and production grids. In *In-*

ternational Symposium on Cluster Computing and the Grid (CC-Grid'07), pages 100–107, Rio de Janeiro, May 2007. IEEE.

[GPJ+07] Xavier Grehant, Olivier Pernet, Sverre Jarp, Isabelle Demeure, and Peter Toft. Xen management with smartfrog: on-demand supply of heterogeneous, synchronized execution environments. In *VHPC '07: Proceedings of the Workshop on Virtualization in High-Performance Cluster and Grid Computing*, LNCS. Springer, 08 2007.

[Gra02] *Globus Resource Allocation Manager (GRAM) 1.6 documentation*, 2002.

[Gre07] Derek. Greer. Interactive application architecture patterns. Aspiring Craftsman, 2007.

[Gre10] Xavier Grehant. Stochastic analysis of cache thrashing. Technical report, Telecom ParisTech, 03 2010.

[GS06] Fei Guo and Yan Solihin. An analytical model for cache replacement policy performance. In *SIGMETRICS '06/Performance '06: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 228–239, New York, NY, USA, 2006. ACM.

[GSM+07] Tristan Glatard, Gergely Sipos, Johan Montagnat, Zoltán Farkas, and Péter Kacsuk. *Workflow Level Parametric Study Support by MOTEUR and the P-GRADE Portal*, chapter 18, pages 279–299. Springer-Verlag, 2007.

[GST70] J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Syst. J.*, 9(2):78–117, 1970.

[Gur03] Yuri Gurevich. *Abstract State Machines: An overview of the Project*, chapter 2, pages 6–13. Lecture Notes in Computer Science (LNCS). Springer Berlin / Heidelberg, Microsoft Research, One Microsoft Way, Redmond, WA 98052, 2003.

[Har03] Aaron Harwood. *Networks and Parallel Processing Complexity*. Melbourne School of Engineering, Department of Computer Science and Software Engineering, 2003.

[HHTE07] Rahman Hassan, Antony Harris, Nigel Topham, and Aris Efthymiou. Synthetic trace-driven simulation of cache memory. In *AINAW '07: Proceedings of the 21st International Conference* *on Advanced Information Networking and Applications Workshops*, pages 764–771, Washington, DC, USA, 2007. IEEE Computer Society.

[HP06] John Hennessy and David Patterson. *Computer Architecture - A Quantitative Approach.* Morgan Kaufmann, 1990,1996,2003,2006.

[HS89] M. D. Hill and A. J. Smith. Evaluating associativity in cpu caches. *IEEE Trans. Comput.*, 38(12):1612–1630, 1989.

[Hum06] Michael Humphrey. Altair's PBS - altair's PBS professional update. In *SC*, page 28. ACM Press, 2006.

[HWZ05] Bernardo A. Huberman, Fang Wu, and Li Zhang. Ensuring trust in one time exchanges: solving the qos problem. *Netnomics*, 7(1):27–37, 2005.

[ICG$^+$06] David Irwin, Jeffrey Chase, LAura Grit, Aydan Yumerefendi, David Becker, and Kenneth G. Yocum. Sharing networked resources with brokered leases. In *In Proceedings of the USENIX Technical Conference*, 06 2006.

[IGF05] Saeed Iqbal, Rinku Gupta, and Yung-Chin Fang. Job scheduling in hpc clusters. Technical report, Dell Power Solutions, 2005.

[JB07] Janet L. Wiener Jennifer Burge, Partha Ranganathan. Cost-aware scheduling for heterogeneous enterprise machines (cash'em). In *Proceedings of USENIX'07*, Santa Clara, CA, June 2007.

[JIPH07] Andhi Janapsatya, Aleksandar Ignjatovic, Sri Parameswaran, and Joerg Henkel. Instruction trace compression for rapid instruction cache simulation. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 803–808, San Jose, CA, USA, 2007. EDA Consortium.

[J.T06] J.T.Moscicki. Efficient job handling in the grid: short deadline, interactivity, fault tolerance and parallelism. In *EGEE User Forum*, Geneva, Switzerland, March 2006. CERN.

[JZ09] Qiu Jing and Zhou Zheng. Distributed resource allocation based on game theory in multi-cell ofdma systems. *International Journal of Wireless Information Networks*, 16:44–50, 2009.

[KDF04] Katarzyna Keahey, Karl Doering, and Ian Foster. From sandbox to playground: Dynamic virtual environments in the grid. In *GRID*

'04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), pages 34–42, Washington, DC, USA, 2004. IEEE Computer Society.

[KFFZ05] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Scientific Programming*, 13(4):265–275, 2005.

[KL01] Alexey Kalinov and Alexey Lastovetsky. Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers. *Journal of Parallel and Distributed Computing*, 61(4):520 – 535, 2001.

[KMP+04] Evangelos Kotsovinos, Tim Moreton, Ian Pratt, Russ Ross, Keir Fraser, Steven Hand, and Tim Harris. Global-scale service deployment in the xenoserver platform. In *Proceedings of the First Workshop on Real, Large Distributed Systems (WORLDS '04)*, San Francisco, December 2004.

[KS04] Tejas S. Karkhanis and James E. Smith. A first-order superscalar processor model. *SIGARCH Comput. Archit. News*, 32(2):338, 2004.

[KSS+07] Jong-Kook Kim, Sameer Shivle, Howard Jay Siegel, Anthony A. Maciejewski, Tracy D. Braun, Myron Schneider, Sonja Tideman, Ramakrishna Chitta, Raheleh B. Dilmaghani, Rohit Joshi, Aditya Kaul, Ashish Sharma, Siddhartha Sripada, Praveen Vangari, and Siva Sankar Yellampalli. Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. *J. Parallel Distrib. Comput.*, 67(2):154–169, 2007.

[Lai05] Kevin Lai. Markets are dead, long live markets. *SIGecom Exch.*, 5(4):1–10, 2005.

[Lan77] Gérard Le Lann. Distributed systems - towards a formal approach. In *IFIP Congress*, pages 155–160, 1977.

[LB06] Colin Low and Andrew Byde. Market-based approaches to utility computing. Technical report, HPL, 02 2006.

[LGS+08] Fang Liu, Fei Guo, Yan Solihin, Seongbeom Kim, and Abdulaziz Eker. Characterizing and modeling the behavior of context switch misses. In *PACT '08: Proceedings of the 17th international con-*

ference on Parallel architectures and compilation techniques, pages 91–101, New York, NY, USA, 2008. ACM.

[LHA+04] E. Laure, F. Hemmer, A. Aimar, M. Barroso, P. Buncic, A. Di Meglio, L. Guy, P. Kunszt, S. Beco, F. Pacini, F. Prelz, M. Sgaravatto, A. Edlund, O. Mulmo, D. Groep, S.M. Fisher, and M. Livny. Middleware for the next generation grid infrastructure. In *Proceedings of Computing in High Energy Physics*, Interlaken, Switzerland, 09 2004.

[LHC+04] L.Maigne, D. Hill, P. Calvat, V. Breton, R. Reuillon, D.Lazaro, Y. Legr, and D. Donnarieix. Parallelization of monte carlo simulations and submission to a grid environment. *Parallel Processing Letters journal*, 14(2):177–196, June 2004.

[LHC+06] Hurng-Chun Lee, Li-Yung Ho, Hsin-Yen Chen, Ying-Ta Wu, and Simon C. Lin. Efficient handling of large scale in-silico screening using diane. In *Poster in EGEE'06 Conference, Enabling Grids for E-Science*, Geneva, Switzerland, 2006.

[Lit07] Maarten Litmaath. glite job submission chain v.1.2. litmaath.home.cern.ch/ litmaath/UI-WMS-CE-WN, June 2007.

[LMW99] Yau-Tsun Steven Li, Sharad Malik, and Andrew Wolfe. Performance estimation of embedded software with instruction cache modeling. *ACM Trans. Des. Autom. Electron. Syst.*, 4(3):257–279, 1999.

[LSV06] Arnaud Legrand, Alan Su, and Frederic Vivien. Minimizing the stretch when scheduling flows of biological requests. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 103–112, New York, NY, USA, 2006. ACM Press.

[LZ09] Yu Liu and Wei Zhang. Exploiting stack distance to estimate worst-case data cache performance. In *SAC*, pages 1979–1983, 2009.

[Mac04] Marlon Machado. Enable existing applications for grid: Batch anywhere, independent concurrent batch, and parallel batch. Technical report, IBM, June 2004.

[MAS+99] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59:107–131, 1999.

[MCE$^+$02] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, Feb 2002.

[MDP$^+$00] Dejan S. Milojičić, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. *ACM Computing Surveys*, 32(3):241–299, 2000.

[MHS$^+$04] J.T. Moscicki, H.C.Lee, S.Guatelli, S.C. Lin, and M.G.Pia. Biomedical applications on the grid: Efficient management of parallel jobs. In *NSS*, Rome, Italy, October 2004. IEEE.

[Mil00] V. Milutinovic. Caching in distributed systems. *Concurrency, IEEE [see also IEEE Parallel & Distributed Technology]*, 8(3):14–15, Jul-Sep 2000.

[Miu06] Kenichi Miura. Overview of japanese science grid project naregi. *Progress in Informatic*, 3(1349-8614):67–75, 2006.

[MKK$^+$05] Anirban Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling strategies for mapping application workflows onto the grid. In *HPDC-14. Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*, pages 125–134, July 2005.

[MMC04] Gabriel Marin and John Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models. *SIGMETRICS Perform. Eval. Rev.*, 32(1):2–13, 2004.

[Mos03] J.T. Moscicki. Diane - distributed analysis environment for grid-enabled simulation and analysis of physics data. In *NSS*, Portland, Oregon, USA, October 2003. IEEE.

[MST$^+$05] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Willy Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 13–23, New York, NY, USA, 2005. ACM.

[MWV00] Navneet Malpani, Jennifer L. Welch, and Nitin Vaidya. Leader election algorithms for mobile ad hoc networks. In *DIALM '00: Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 96–103, New York, NY, USA, 2000. ACM.

[Nil07] P. Nilsson. Experience from a pilot based system for atlas. In *Proceedings of Computing in High Energy Physics (CHEP'07)*, Victoria, Canada, September 2007.

[NLJ$^+$05] A. Nishandar, D. Levine, S. Jain, G. Garzoglio, and I. Terekhov. Extending the cluster-grid interface using batch system abstraction and idealization. In *Proceedings of Cluster Computing and Grid 2005 (CCGrid05)*, Cardiff, UK, May 2005.

[NS06] Zsolt Nemeth and Vaidy Sunderam. Virtualization in grids: A semantical approach. In Jos C. Cunha and Omer F. Rana, editors, *Grid Computing: Software environments and Tools*, chapter 1, pages 1–18. Springer Verlag, January 2006.

[NYI$^+$05] Hidemoto Nakada, Motohiro Yamada, Yasuyoshi Itou, Satoshi Matsuoka, Jaime Frey, and Yasumasa Nakano. Design and implementation of condor-unicore bridge. In *HPCASIA '05: Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, page 307, Washington, DC, USA, 2005. IEEE Computer Society.

[PACL05] Heidi Pan, Krste Asanović, Robert Cohn, and Chi-Keung Luk. Controlling program execution through binary instrumentation. *SIGARCH Comput. Archit. News*, 33(5):45–50, 2005.

[Pen02] Rob Pennington. Terascale clusters and the teragrid. In *Proceedings for High Performance Computing Asia*, pages 407–413, Dec 16-19 2002.

[PG95] V Phalke and B Gopinath. An interreference gap model for temporal locality in program behavior. In *in Proceedings of the 1995 ACM SIGMETRICS Conference*, pages 291–300, 1995.

[PLL00] Roberto De Prisco, Butler Lampson, and Nancy Lynch. Revisiting the algorithm. *Theoretical Computer Science*, 243(1-2):35 – 91, 2000.

[PMP$^+$04] Joshua J. Pieper, Alain Mellan, JoAnn M. Paul, Donald E. Thomas, and Faraydon Karim. High level cache simulation for heterogeneous multiprocessors. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 287–292, New York, NY, USA, 2004. ACM.

[PRV08] Jean-François Pineau, Yves Robert, and Frédéric Vivien. The impact of heterogeneity on master-slave scheduling. *Parallel Comput.*, 34(3):158–176, 2008.

[PSP06] Stuart Paterson, P. Soler, and C. Parkes. *LHCb Distributed Data Analysis on the Computing Grid*. PhD thesis, University of Glasgow, September 2006.

[PW07] Sanjay Padhi and Rodney Walker. Cronus: A condor glide-in based atlas production executor. In *Proceedings of CHEP'07*, September 2007.

[PZU$^+$07] Pradeep Padala, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, Kenneth Salem, and Kang G. Shin. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the EuroSys 2007*, pages 289–302. ACM Press, 03 2007.

[Qlu06] Qlusters Inc., 1841 Page Mill Road, G2, Palo Alto, CA 94304. *openQRM Technical Overview: Open Source - Data Center Management Software*, 11 2006.

[Ram00] Rajesh Raman. *Matchmaking frameworks for distributed resource management*. PhD thesis, University of Wisconsin at Madison, 2000. Supervisor-Miron Livny.

[Rau77] B. R Rau. Properties and applications of the least-recently-used stack model. Technical report, Stanford University, Stanford, CA, USA, 1977.

[RCAM06] Jerry Rolia, Ludmila Cherkasova, Martin Arlitt, and Vijay Machiraju. Supporting application qos in shared resource pools. Technical Report 20060118, HPL, 01 2006.

[Reb05] David Rebatto. Egee batch local ascii helper (blahp). In *HEPiX Meeting*, Karlsruhe, Germany, May 2005.

[RIG$^+$06] Lavanya Ramakrishnan, David Irwin, Laura Grit, Aydan Yumerefendi, Adriana Iamnitchi, and Jeff Chase. Toward a doctrine of containment: Grid hosting with adaptive resource control. In *SC '06: Supercomputing, 2006. Proceedings of the ACM/IEEE SC 2006 Conference on Supercomputing*, number 0-7695-2700-0, pages 20–20, New York, NY, USA, 2006. Renaissance Computing Institute, IEEE Computer Society, ACM Press.

[RLS98] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, Chicago, IL, July 28-31 1998.

[RMX05] Paul Ruth, Phil McGachey, and Dongyan Xu. "viocluster: Virtualization for dynamic computational domains". In *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'05)*, Boston, MA, September 2005.

[Rob06] L Robertson. Status of the lcg project. Technical Report CERN-RRB-2006-112, CERN, Geneva, Oct 2006.

[ROLV06] Robert Ricci, David L. Oppenheimer, Jay Lepreau, and Amin Vahdat. Lessons from resource allocators for large-scale multiuser testbeds. *Operating Systems Review*, 40(1):25–32, 2006.

[Rot95] H.G. Rotithor. On the effective use of a cache memory simulator in a computer architecture course. *Education, IEEE Transactions on*, 38(4):357–360, Nov 1995.

[RRT+08] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No "power" struggles: coordinated multi-level power management for the data center. In *ASPLOS XIII: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 48–59, New York, NY, USA, 2008. ACM.

[SAB+03] P. Saiz, L. Aphecetcheb, P. BunImageiImagea, R. PiskaImaged, J. E. Revsbeche, and V. Imageegod. Alien - alice environment on the grid. *Nucl. Instrum. Meth.*, A502:437–440, 2003.

[SAB+05] Andreas Savva, Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, and Darren Pulsipher. Job submission description language (jsdl) specification. http://forge.gridforum.org/projects/jsdl-wg, November 2005. GFD-R.056.

[SBP03] Pablo Saiz, Predrag Buncic, and Andreas J. Peters. Alien resource brokers. In *Proceedings of CHEP'03*, June 2003.

[SCL06a] Ram Srinivasan, Jeanine Cook, and Olaf Lubeck. Performance modeling using monte carlo simulation. *IEEE Computer Architecture Letters*, 5(1):38–41, 2006.

[SCL06b] Ram Srinivasan, Jeanine Cook, and Olaf Lubeck. Ultra-fast cpu performance prediction: Extending the monte carlo approach. In *SBAC-PAD '06: Proceedings of the 18th International Symposium on Computer Architecture and High Performance Computing*, pages 107–116, Washington, DC, USA, 2006. IEEE Computer Society.

[SDR01] G. Edward Suh, Srinivas Devadas, and Larry Rudolph. Analytical cache models with applications to cache partitioning. In *ICS '01: Proceedings of the 15th international conference on Supercomputing*, pages 1–12, New York, NY, USA, 2001. ACM.

[Sfi07] Igor Sfiligoi. glideinwms - a generic pilot-based workload management system. In *Proceedings of Computing in High Energy Physics (CHEP'07)*, 2007.

[SL03] Sechang Son and Miron Livny. Recovering internet symmetry in distributed computing. In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 542, Washington, DC, USA, 2003. IEEE Computer Society.

[Smi82] Alan Jay Smith. Cache memories. *ACM Comput. Surv.*, 14(3):473–530, 1982.

[Smi98] James E. Smith. A study of branch prediction strategies. In *ISCA '98: 25 years of the international symposia on Computer architecture (selected papers)*, pages 202–215, New York, NY, USA, 1998. ACM.

[SOBS04] Hongzhang Shan, Leonid Oliker, Rupak Biswas, and Warren Smith. Job scheduling in heterogeneous grid environment. In *ADCOM2004: International Conference on Advanced Computing and Communication.*, 2004.

[SSkP$^+$07] Xudong Shi, Feiqi Su, Jih kwon Peir, Ye Xia, and Zhen Yang. Cmp cache performance projection: accessibility vs. capacity. *SIGARCH Comput. Archit. News*, 35(1):13–20, 2007.

[SSR01] Florian Schintke, Jens Simon, and Alexander Reinefeld. A cache simulator for shared memory systems. In *ICCS '01: Proceedings of the International Conference on Computational Science-Part II*, pages 569–578, London, UK, 2001. Springer-Verlag.

[ST03] John S. Seng and Dean M. Tullsen. The effect of compiler optimizations on pentium 4 power consumption. In *INTERACT '03: Proceedings of the Seventh Workshop on Interaction between Compilers and Computer Architectures*, page 51, Washington, DC, USA, 2003. IEEE Computer Society.

[SVL01] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2001. USENIX Association.

[Ter02] I. Terekhov. Meta-computing at d0. *Nuclear Instruments and Methods in Physics Research (ACAT-02)*, 502/2-3(NIMA14225):402–406, June 2002.

[tGC06] the GridPP Collaboration. Gridpp: development of the uk computing grid for particle physics. *Journal of Physics G: Nuclear and Particle Physics*, 32:N1–N20, 2006.

[TGSR04] Andrei Tsaregorodtsev, Vincent Garonne, and Ian Stokes-Rees. Dirac: A scalable lightweight architecture for high throughput computing. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 19–25, Washington, DC, USA, 2004. IEEE Computer Society.

[TGV08] Qinghui Tang, Sandeep Kumar S. Gupta, and Georgios Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Transactions on Parallel and Distributed Systems*, 19(11):1458–1472, 2008.

[TS92] John Turek and Dennis Shasha. The many faces of consensus in distributed systems. *Computer*, 25(6):8–17, 1992.

[TTL02] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., 2002.

[Van08] D. C. Vanderster. *Resource Allocation and Scheduling Strategies using Utility and the Knapsack Problem on Computational Grids.* PhD thesis, University of Victoria, 2008.

[vHCF$^+$03] Eric van Herwijnen, Joel Closier, Markus Frank, Clara Gaspar, Francoise Loverre, Sebastien Ponce, Roberto Graciani Diaz, Domenico Galli, Umberto Marconi, Vincenzo Vagnoni, Nicholas Brook, K. Buckley, A.and Harrison, Michael Schmelling, Ulrik Egede, Andrei Tsaregorotsev, V. Garonne, B. Bogdanchikov, Ivan Korolko, Juan P. Washbrook, A.and Palacios, Sander Klous, Juan J. Saborido, Akram Khan, A. Pickford, A. Soroko, V. Romanovski, G.N. Patrick, Genady Kuznetsov, and Miriam Gandelman. Dirac - distributed infrastructure with remote agent control. In *Proceedings of Computing in High Energy Physics*, 2003.

[WE00] F. Wolf and R. Ernst. Data flow based cache prediction using local simulation. In *HLDVT '00: Proceedings of the IEEE International High-Level Validation and Test Workshop (HLDVT'00)*, page 155, Washington, DC, USA, 2000. IEEE Computer Society.

[Wei98] Jon B. Weissman. Metascheduling: A scheduling model for metacomputing systems. In *Proceedings of High Performance Distributed Computing (HPDC'98)*, pages 348–349, 1998.

[WLrW06] Torre Wenaus, Miron Livny, and rank Wrthwein. Preliminary plans for just-in-time workload management in the osg extensions program. Technical report, US Atlas, October 2006. based on SAP proposal of March 2006.

[XZQ00] Xiao, Zhang, and Qu. Effective load sharing on heterogeneous networks of workstations. In *IPPS: 14th International Parallel Processing Symposium*, pages 431–438, Los Alamitos, May 1–5 2000. IEEE Computer Society Press.

[YMM05] Leo T. Yang, Xiaosong Ma, and Frank Mueller. Cross-platform performance prediction of parallel applications using partial execution. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 40, Washington, DC, USA, 2005. IEEE Computer Society.

[ZZWD93] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a load sharing facility for large, heterogenous distributed computer systems.

Software: Practice And Experience, 23(12):1305–1336, December 1993.