

DG-ADAJ sur Grille 5000

*R. Olejnik, B. Toursel
V. Fiolet, I. Alshabani*

Journées Grid 5000

30 Octobre 2006



Laboratoire d'informatique
de Lille (France)



Université de Mons-Hainault
(Belgique)



Plan

- ✓ **Introduction**
- ✓ **Environnement DG-ADAJ**
- ✓ **Framework CCADAJ**
- ✓ **Contexte du projet DisDaMin**
- ✓ **Algorithmes de Data Mining Distribués**
- ✓ **Déploiement sur grilles**
- ✓ **Conclusions et Perspectives**

Introduction

- **Projet DG-ADAJ :**
 - Plateforme middleware pour Grille
 - Java Computing
 - Observation et Equilibrage de charge
 - Framework de composants
 - Open source
- **Projet DisDaMin (DIStributed Data MINing):**
 - Data Mining : processus d'extraction de connaissances
 - Domaines divers : Banque, assurances, biologie, médecine..
 - Haute Performance (i.e. parallèle et distribué).
 - Classification non supervisée (clustering)
 - Experimentations sur Grid5000

Environnement ADAJ

Introduction à DG-ADAJ

- Contexte
- Objectifs
- Caractéristiques principales de DG-ADAJ
- Architecture & Environnement
- Framework CCADAJ
- Allocation statique optimisée
- Equilibrage de charge dynamique
- Conclusion

DG-ADAJ : Applications Java Distribuées

- Composées d'objets répartis sur les noeuds d'une grille de stations de travail (Desktop Grid).
- Existence de plusieurs tâches (flots de contrôle) dans l'application.
 - Avec des besoins différents en calcul.
 - Avec des communications inter-tâches
- Possibilité d'assigner les tâches à l'une ou l'autre des JVMs installées sur les noeuds.



Autres Projets en Java

- **IBIS**
 - Système pour calculs distribués sur grille
 - Fast RMI, communication de groupe , replication d'objects
- **Proactive**
 - Librairie d'API basé sur MOP(Meta-Object Protocol)
 - Composants et déploiement Fractal
- **MOCCA/H2O**
 - MOCCA : plateforme de composants distribués CCA
 - H2O : plateforme de partage de ressources en Java
 - Distribution et équilibrage des composants sur la plateforme

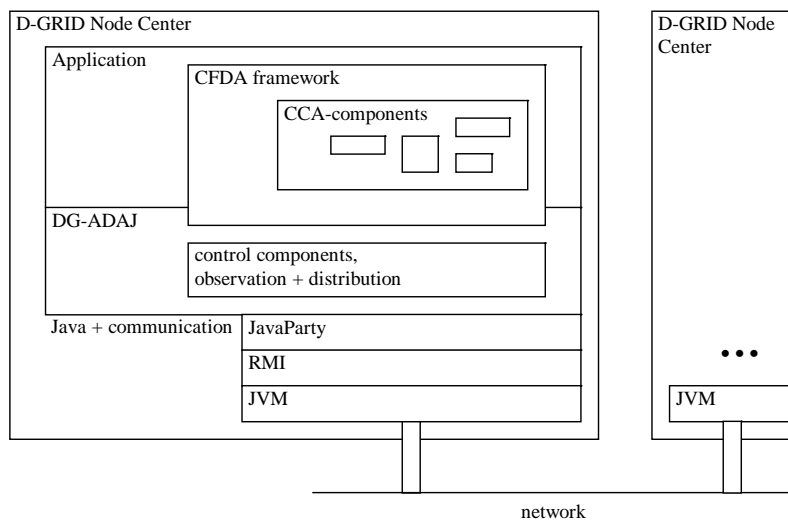
Caractéristiques de DG-ADAJ

- **Image unique de la grille** (Single System Image).
- Mécanismes spéciaux de niveau middleware :
 - Adaptation dynamique et automatique aux variations du calcul et de la disponibilité des noeuds.
 - Placement initial optimisé sur les nœuds.
- Bibliothèque de programmation pour exprimer le parallélisme et la distribution.
- Mise en oeuvre de JavaParty
- Framework de composants pour Grille de Calcul

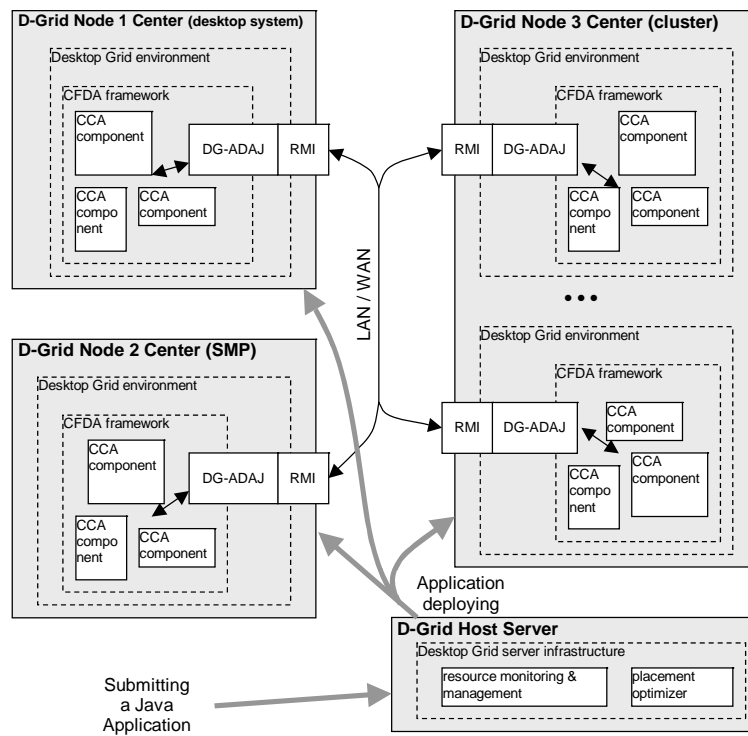
Projet Java Party

- Université de Karlsruhe (Allemagne)
- Plateforme spécifique pour les applications distribuées
- Extention du langage Java : mot-clé remote pour déclarer les classes distribuées
- Utilisation d'un pré-compilateur
- Transparence de l'appel des méthodes pour accéder aux attributs
- Migration des objets non-actifs

Architecture de DG-ADAJ

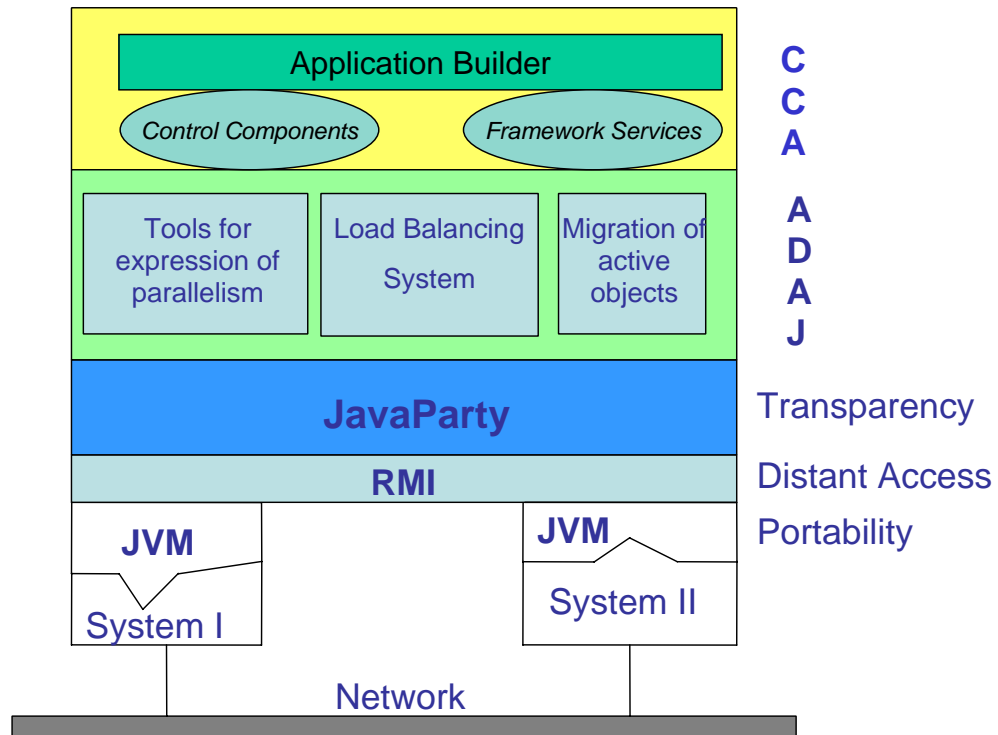


Environnement DG-ADAJ



Framework CCADAJ

CCADAJ Framework



Common Components Architecture: Composants et Ports

- Composants et Ports
 - Les composants interagissent à travers les ports
 - Un composant peut **fournir** un port (service)
 - L'implémentation de l'interface du port
le service que le composant offre
 - Le composant peut **utiliser** un port
 - L'appel de la méthode dans le port
 - ✓ le besoin du composant à utiliser
 - Connecter les composant à travers les ports *provides-uses*

CCA: Frameworks

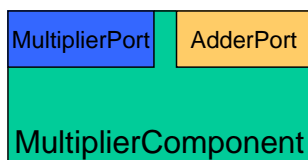
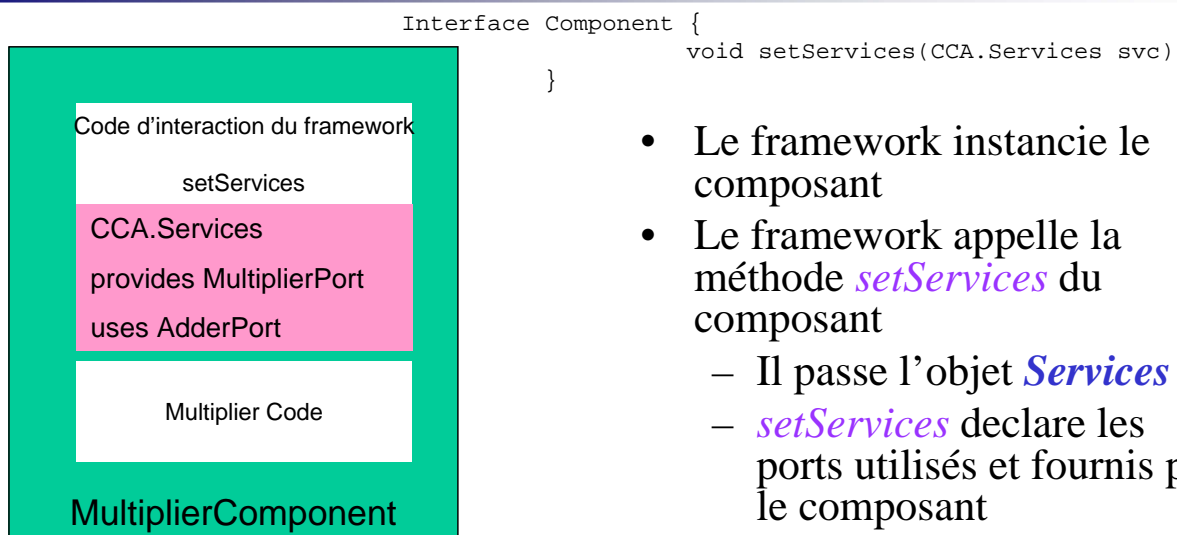
- Framework

- Un moyen de manipuler les composants

- ✓ Les composer pour en créer des applications
- ✓ Échanges entre les ports sans exposer les détails de l'implémentation du composant
- ✓ Fournir un minimum de services standards
- ✓ Le framework lui-même est un composant et ses services sont des ports
 - Pour les connexions entre les autres framework

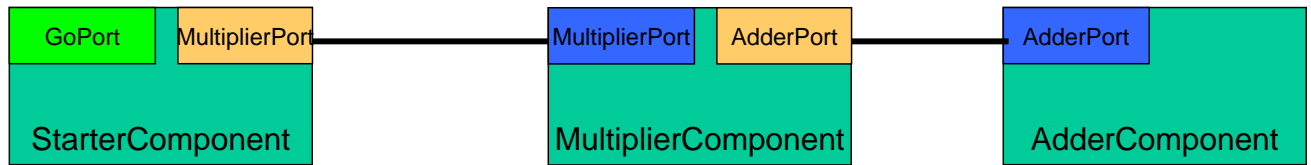
- Un composant implémente l'interface *Component*

- ✓ C'est le code d'interaction avec le framework de l'intérieur du composant



- Le framework instancie le composant
- Le framework appelle la méthode *setServices* du composant
 - Il passe l'objet *Services*
 - *setServices* declare les ports utilisés et fournis par le composant
- Le composant ne sais rien du monde extérieur
 - L'objet *Services* assure la communication avec le framework
- Le framework maintenant sais comment le composant se connecte à d'autres composants

CCA: Interactions entre les composants

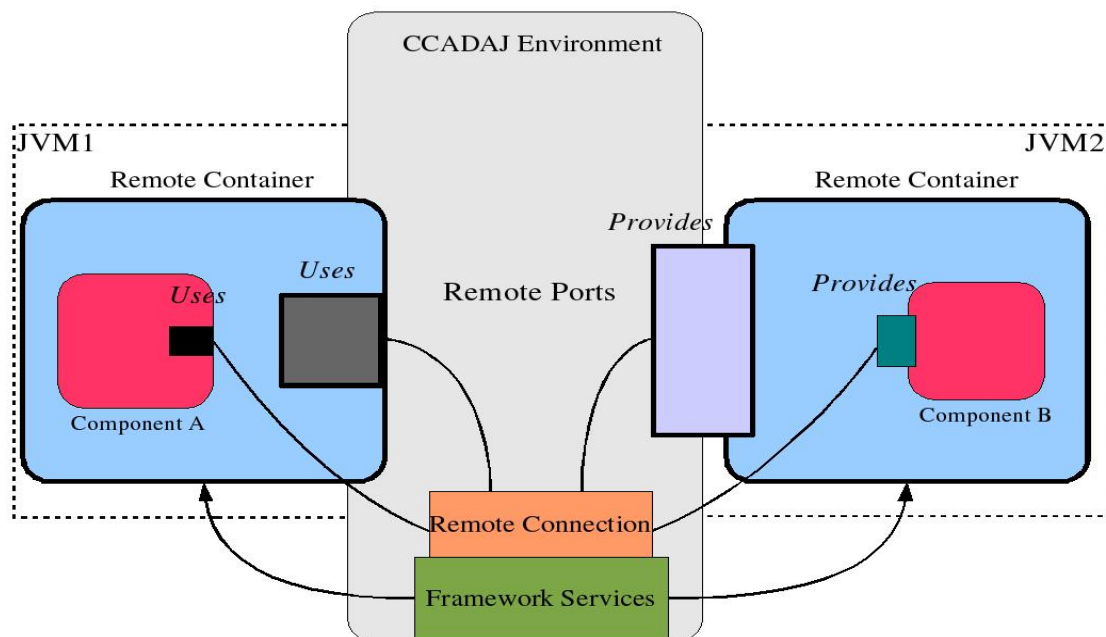


- GoPort
 - Un port *provides* spécial pour « exécuter » le composant
 - Implémente la méthode *go()* qui commence l'exécution du composant
 - Le framework recherche les port **go** et les utilise
- UsesPort (multiplierPort, AdderPort)
 - Appelle *getPort* pour obtenir le port de l'objet Services
 - Appelle la méthode dur le port
 - Ex: `x=multiplierPort.getProduct(y,z);`
- Connection ports uses/provides
 - Pas de uses/uses ou provides/provides
- Les ports sont connecté par types
 - Les types de port doit correspondre
 - Les noms de pors sont unique dans un composant
- Le framework met de l'information sur le composant fournisseur dans l'objet Service du composant utilisateur

connect **StarterComponent** *MultiplierPort* **MultiplierComponent** *MultiplierPort*

connect **MultiplierComponent** *AdderPort* **AdderComponent** *AdderPort*

Remote Container



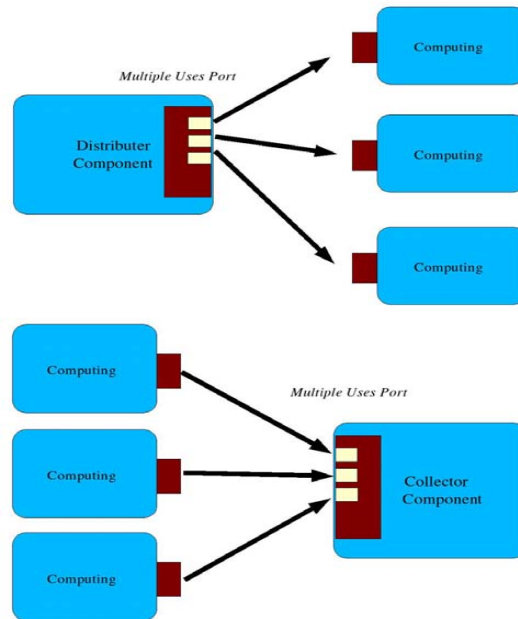
Composants de contrôle

- Aider l'utilisateur à construire une application parallèle à partir des composants
- fonctionnalité pour la programmation parallèle
 - Super-composant
 - Composant Pipeline
 - Composant Parallèle
 - Distributeur
 - Collecteur

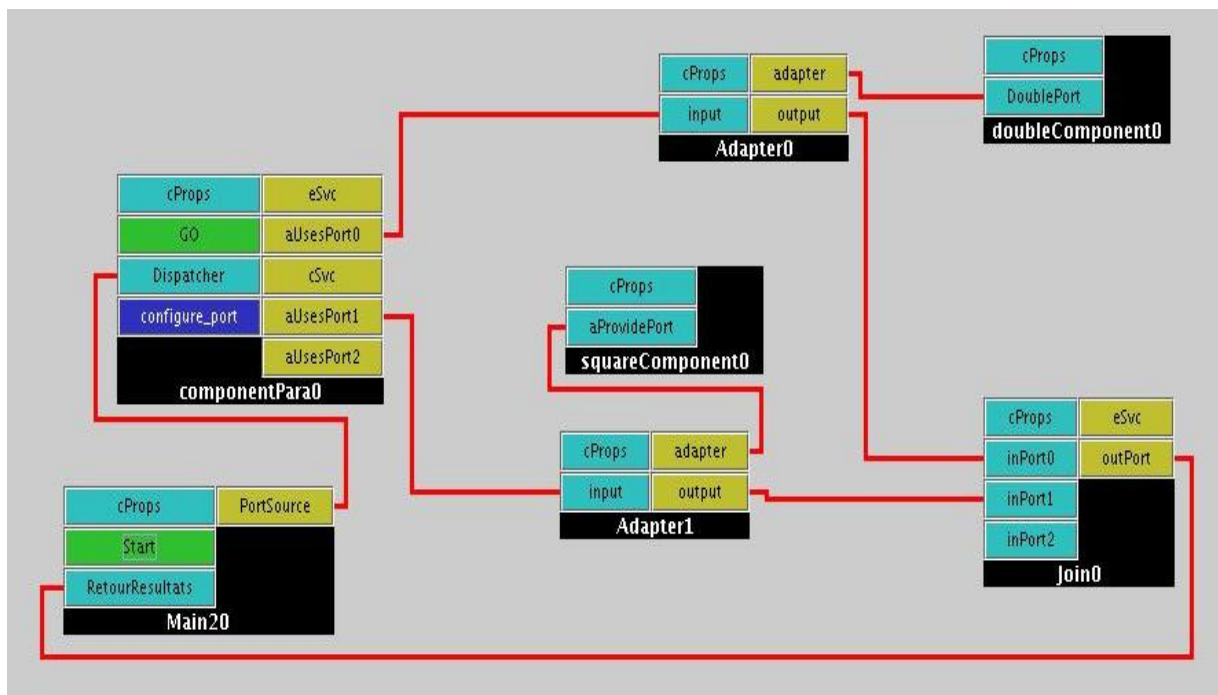
Composant parallèle

- Distributeur
 - Distribuer un tableau de données sur plusieurs composant pour faire du calcul
 - Utiliser un port *multiple* connecté à plusieurs ports de même type d'autres composants
- Collecteur
 - Collecter les données de plusieurs composant de calcul et les mettre dans un tableau.
 - Utiliser un port *multiple*

Composant parallèle



Exemple: Composant parallèle



DG-ADAJ: Optimisation statique

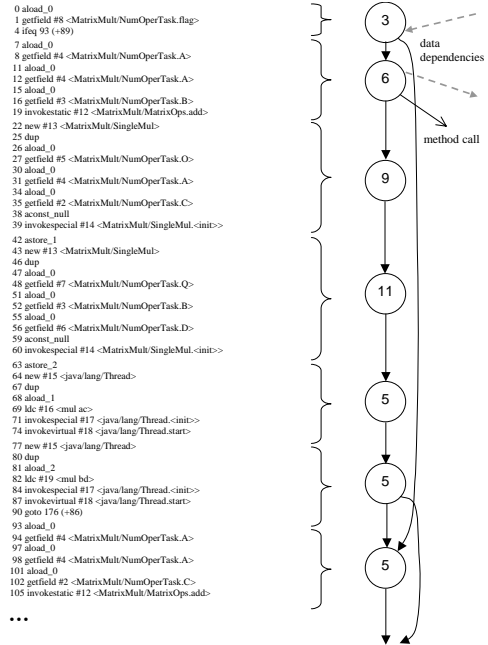
Code :

```

TASK 1
  if flag == true then
    A = A + B
    Res1 = A x C
    Res2 = B x D
  else
    A = A + C
    Res1 = A x B
    Res2 = C x D
  endif

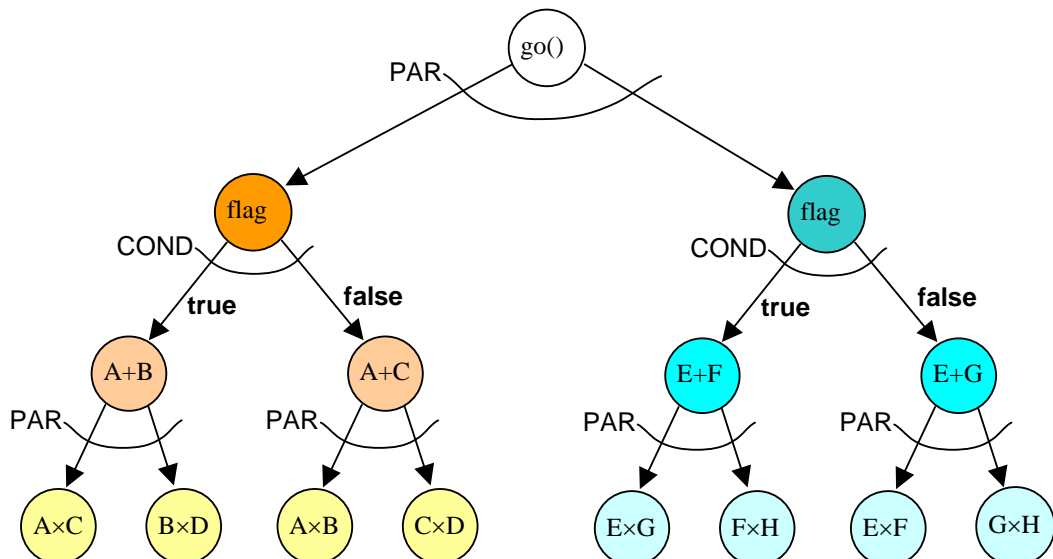
TASK 2
  if flag == true then
    E = E + F
    Res3 = E x G
    Res4 = F x H
  else
    E = E + G
    Res3 = E x F
    Res4 = G x H
  endif
    
```

Control/Data Dependency Graph (CDDG) of a method:



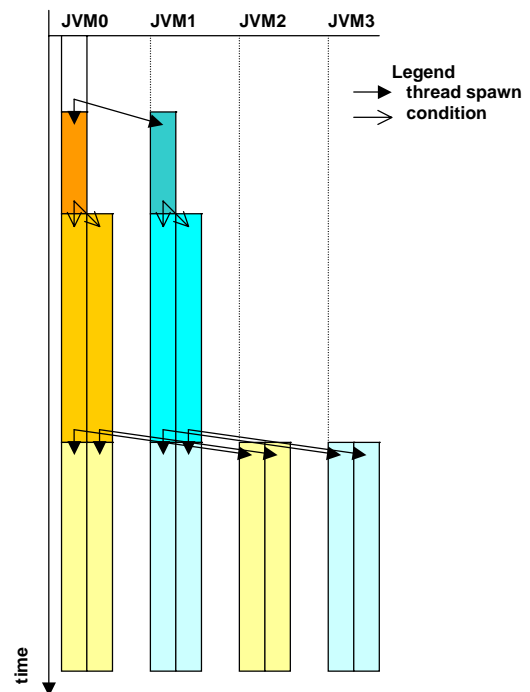
DG-ADAJ: Phase de regroupement

Dans la phase de regroupement (clustering), des macronoeuds sont construits à partir des blocs élémentaires provenant du graphe CDDG.



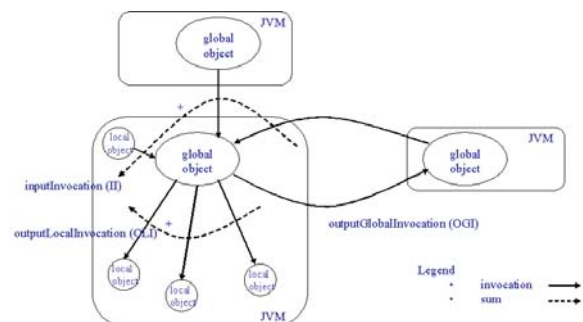
DG-ADAJ: Phase de mappage

- Les **macronoeuds** sont mappés sur les JVMs.
- Les Macronoeuds appartenant à des branches conditionnelles en mutuelle exclusion sont mappés en parallèle sur la même JVM.



DG-ADAJ: Equilibrage dynamique de la charge

- La charge est calculée en fonction de l'intensité des invocations de méthodes.
- Un algorithme en 2-phase est appliqué concurremment avec l'exécution de l'application :
 - Détecte les déséquilibres dans la distribution
 - Corrige ces déséquilibres.
- Les objets sont transférés des machines surchargées vers des machines sous-chargées.



Mécanisme d'observation des relations entre objet

Conclusions DG-ADAJ

- DG-ADAJ est un middleware permettant un équilibrage de charge statique et dynamique, optimisé sur Grille.
- Une analyse statique est utilisée comme une étape préliminaire pour répartir les objets de l'application.
- Un mécanisme d'équilibrage dynamique est basé sur 3 types d'information: une information sur la charge et la performance des noeuds, une information sur les relations dynamiques entre objets et une information déduit de l'analyse statique de code.
- L'optimisation concerne les dépendances statiques et dynamiques entre les objets et leurs méthodes.
- Possibilité de construire des applications distribuées à base de composants.

Travail en cours & Perspectives

- DG-ADAJ en OpenSource
- Aspects tolérance aux fautes
- Système de management des informations
- Expérimentation sur des problèmes réels
- Expérimentation sur Grid'5000
- Projet Européen GOTI (15 partenaires)?

Le Projet DisDaMin

Contexte du Projet DisDaMin

Fouille de données distribuées

- Data Mining : processus d'extraction de connaissances
- Domaines divers : Banque, assurances, biologie, médecine..
- Nécessité de calculs hautes performances (i.e. parallèle et distribué).
- Exploitation de grilles de calcul (Grid5000)

La base de données peut être:

- centralisée,
- distribuée par instances (fragmentation horizontale),
- distribuée par attributs (fragmentation verticale): multi-database

Prendre en compte:

- Les spécificités du Data Mining
- Les spécificités de traitement parallèle et distribué: **GRID Computing** (NOW ou grille)

Contexte du Projet DisDaMin

Fouille de données distribuées

Problèmes des algorithmes parallèles existants:

- Adaptés à des ordinateurs avec mémoire partagée et réseaux d'interconnection rapides
- **Coût très élevé** de ces ordinateurs

Projets existants:

- Datamining Grid, Discovery Net, Knowledge Grid, Grid Miner, ...
- Outils et services pour environnement de type grille
- Mécanismes **d'intégration et de déploiement** des **algorithmes classiques** de data mining sur grille de calculs.

But:

- Traitement de **grande quantité de données** pour des problèmes de Data Mining
- Exploitation optimum **des ressources disponibles** (grille, réseau de stations) : CPU, stockage

Paradigme des Traitements Distribués

- Spécificités du support d'exécution
- Pas de mémoire partagée
 - Fragmentation nécessaire de la structure de données
 - Pas de vue globale: vues locales et partielles ⇒ **communications**
- Réseau lent par rapport à la puissance des machines
 - Recouvrir les temps de communication par des instructions CPU
- (A)synchronisme et **pipeline**
 - utiliser les données disponibles dès que possible
 - éviter les temps d'attente inactifs
- Contexte pour assurer l'intér-opérabilité : DG-ADAJ, Java RMI

Problèmes traitées : Règles d'Association

- Identifier des associations de la forme:
 $A \Rightarrow B$
où A et B sont des ensembles d'items (itemsets)
- Problème de référence : analyse du panier de la ménagère
les items sont des articles et l'on recherche ceux qui ont tendance à être achetés ensemble
- Méthode basée sur:
 - **Sur l'identification des itemsets fréquents** ("fortement" présents dans les données)
 - Génération des règles à partir des itemsets fréquents identifiés
- Nécessité de fournir des mesures d'intérêts des résultats

Problème des Règles d'Association Difficultés et avantages d'une vision distribuée

- Spécificités du problème des Règles d' Association
 - Traiter la **base entière** : le traitement suppose **de comparer** chaque partie de la base de données à toutes les autres parties (haute complexité)
 - Méthode basée sur des **critères globaux** (support, fréquence...)
 - Vision locale (partielle) des données en contexte parallèle
 - Nécessité de nouvelles méthodes (heuristiques)
 - Limitation des communications et des synchronisations
- Exploiter autant que possible les capacités de traitement
 - **Traitements locaux** les plus indépendants sur fragments de données parallèles
 - Utilisation du **parallélisme** à plusieurs niveaux du traitement
 - Approche **pipeline** entre les étapes
 - Pas de synchronisations
- Sécurité des données : Format interne pour les communications autorisées

Schéma Général

Pour la recherche de règles d'association

- Phase 1: Préparation des données (sur base de Clustering)

- Discrétisation, filtrage, nettoyage, formatage des données pour le problème.
=> But : Formater les données
- **Distribution des Données** => But : Diminuer la complexité.
 - distribution « intelligente »
 - recherche de **profils** (similarité des données)

- Phase 2: Traitement Distribué (Règles d'association)

- Extraction des itemsets fréquents
(Exécution séquentielle sur chaque fragment de données avec éventuellement des collaborations)
- Validation des Résultats
- Génération des règles

Chaque étape peut être distribuée

Schéma Général

Justification d'une « Distribution Intelligente »

- Traiter ensemble les données les **plus similaires**
- Similarité en terme d'items contenus
- Spécialiser les traitements aux items contenus
=> **Gain de complexité** dû aux items absents
- Critère de distribution:
 - les instances les plus similaires dans un même fragment
 - les instances les moins similaires dans des fragments distincts**= critère de clustering**

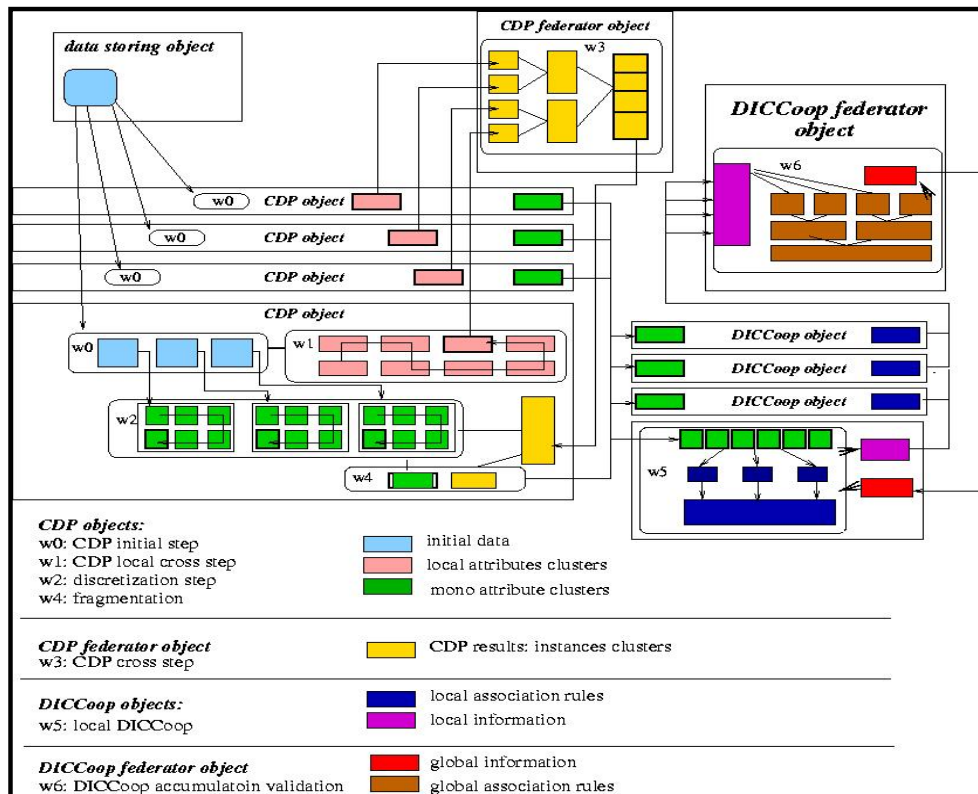
Déploiement des algorithmes DisDaMin sur grille

- Il faut assurer:
 - Le Pipeline
 - Le recouvrement de communications par du calcul
- Entre la phase 1 et la phase 2, cela a amené à proposer le clustering progressif distribué: **CDP**
 - pré-traitement d'un attribut pendant que les résultats précédents sont « agglomérés ».
- Utilisation de l'asynchronisme
 - utiliser les résultats du pré-traitement dès qu'ils sont disponibles
 - éviter les barrières de synchronisation
- Echanges d'information durant l'étape de traitement: algorithme **DICCOOP**
 - respectant les caractéristiques d'asynchronisme

Déploiement sur GRID'5000

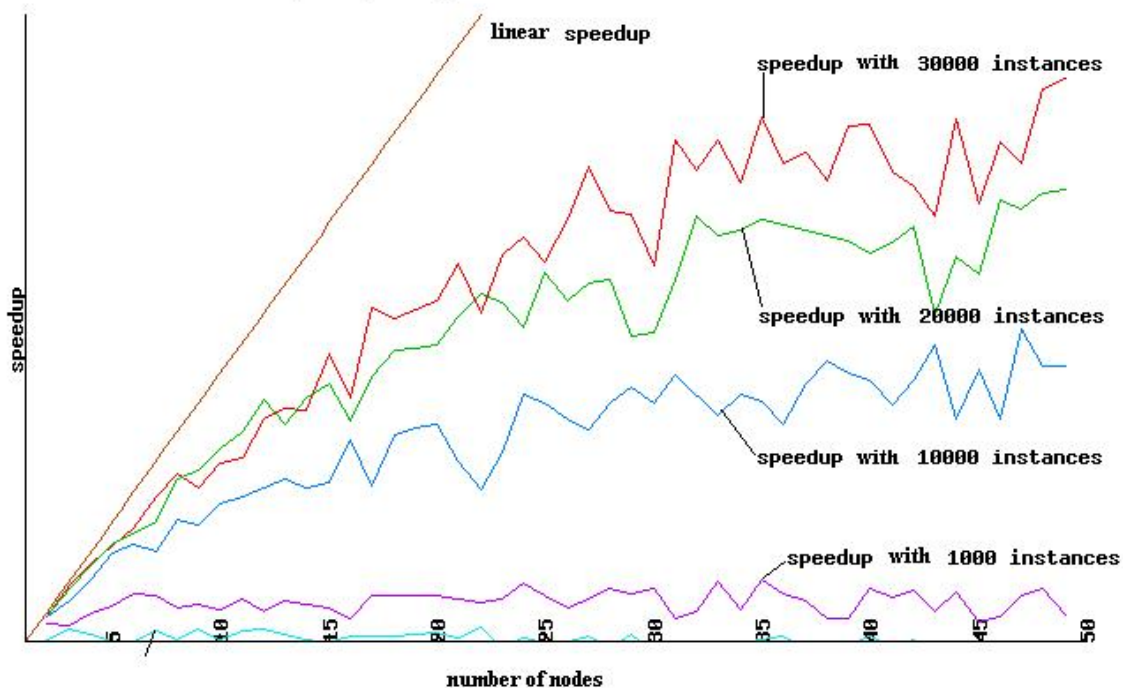
- Des expériences sont actuellement menées sur l'infrastructure GRID'5000.
- Hypothèse: la grille est constituée de **sous-grappes**.
- La distribution des données sur la grille peut être réalisée:
 - en envoyant les fragments de données aux nœuds proxy
le proxy sera alors en charge de redistribuer les données sur la sous-grappe dont il a la charge
 - en envoyant directement les données aux nœuds de traitement
 - en réalisant les envois de données attributs par attribut permettant ainsi le traitement des attributs sur les nœuds au fur et à mesure des réceptions avec recouvrement des communications.
- Le choix de distribution des attributs peut être:
 - basée sur des considérations physiques
 - basée sur une estimation de complexité des données

Déploiement sur GRID'5000



Déploiement sur GRID'5000

Compare speed-up with different instance numbers



Conclusions

- Les résultats confirment l'intérêt
 - > du schéma global de recherche de règles d'association (test par clustering centralisé)
 - > de l'algorithme CDP en tant que méthode de clustering
- L'utilisation des spécificités distribuées ont amenées à proposer de nouvelles méthodes
 - > utilisation d'une approche par clustering pour une distribution optimale
 - ⇒ **Clustering Distribué Progressif**
 - > Collaboration dans les traitements
 - ⇒ **Algorithme DICCoop**
- Beaucoup d'informations disponibles
 - > du pré-traitement
 - ⇒ utilisées dans la phase de distribution (clustering distribué progressif)
 - > du pré-traitement et de la phase de distribution
 - ⇒ à utiliser dans la recherche de règles

Perspectives

- Utilisation du Clustering Distribué Progressif
 - > dans le schéma général pour la distribution
- Insertion des considérations distribuées autant que possible
 - > pipeline / recouvrement des communications
 - > asynchronisme
 - > ...
- Validation de la phase de traitement collaboratif
 - > travail sur la granularité de distribution
- Affinement de la phase de validation
 - > inclusion dans la phase de traitement
 - > à partir des traitements collaboratifs



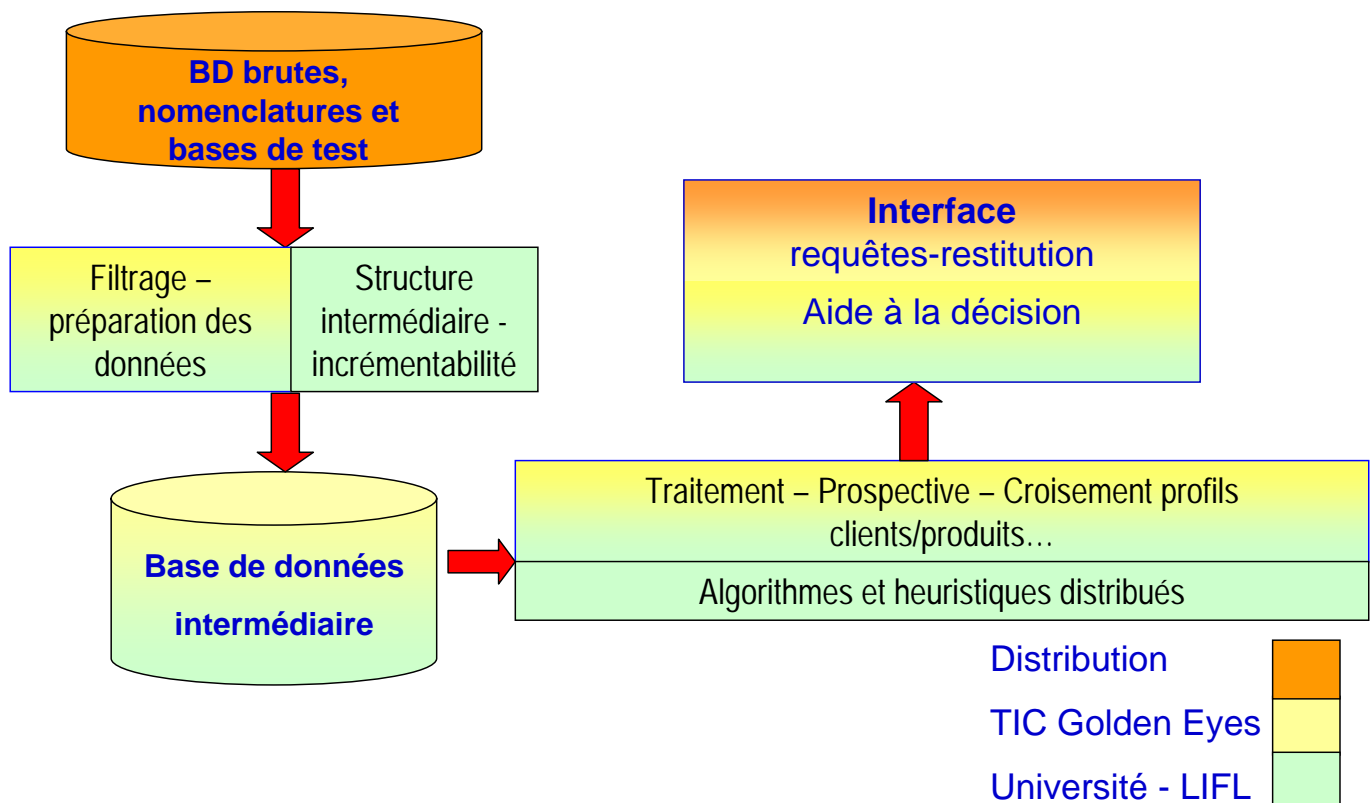
Classement **A**nalyse et **R**echerche des **O**bjets **L**iés :

Développement d'outils d'extraction de connaissances analysant le comportement client

Exploration exhaustive de l'ensemble des transactions effectuées et croisement avec des informations événementielles

Analyse de grandes masses de données impliquant la nécessité de nouvelles approches (heuristiques et traitements répartis).

Intégration dynamique des données en vue de leur exploitation en temps réel.



Perspectives

- Traitement de bases de données de très grande taille
- Vérification des gains apportées par un déploiement optimisé
 - › gain par le pipeline et le recouvrement des communications
 - › anticipations possibles pour empêcher les temps d'inactivité
- Gestion de la tolérance aux pannes et de l'ajout ou du retrait d'un noeud de traitement
- Etudier la possibilité de distribution de manière verticale et horizontale pour la phase 1
 - › cas où un attribut ne tient pas en mémoire
- Dédution de règles
 - › à partir des itemsets fréquents identifiés

QUESTIONS?