

Ordonnancement de workflows dans DIET

Abdelkader Amar

École normale supérieure de Lyon

11 Octobre 2006

Plan

- 1 Introduction
 - Les Workflows
 - Présentation de DIET
 - Motivations et objectifs
- 2 Moteur de workflows
 - Description des workflows
 - Architecture logicielle
 - Ordonnancement
 - Mécanisme de réordonnancement
 - Multi-workflows
- 3 Conclusions

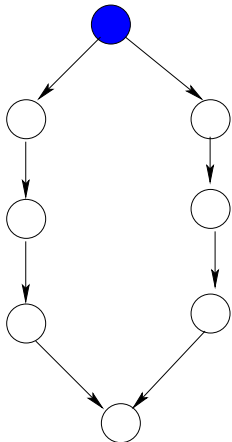
Plan

- 1 Introduction
 - Les Workflows
 - Présentation de DIET
 - Motivations et objectifs
- 2 Moteur de workflows
 - Description des workflows
 - Architecture logicielle
 - Ordonnancement
 - Mécanisme de réordonnancement
 - Multi-workflows
- 3 Conclusions

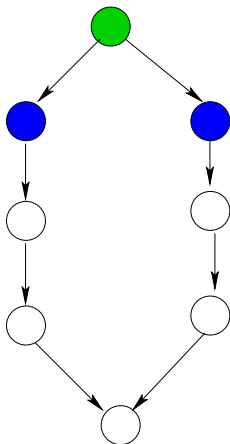
Les workflows

- Ensemble de tâches connectées
- La structure du workflow représente la relation temporelle entre ses tâches
- Généralement représenté par un DAG (*Direct Acyclic Graph*)
 - chaque nœud est une tâche
 - chaque nœud peut avoir un certain nombre de fils ou de parents à condition qu'il n'y ait pas de boucle

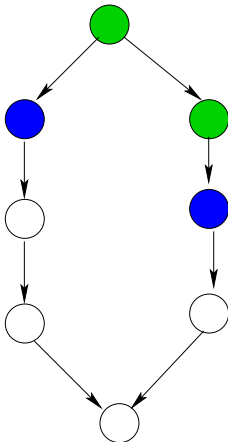
Exécution des workflows



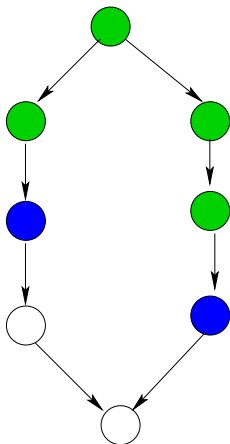
Exécution des workflows



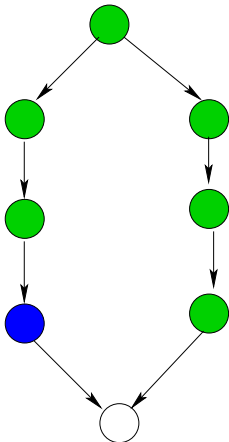
Exécution des workflows



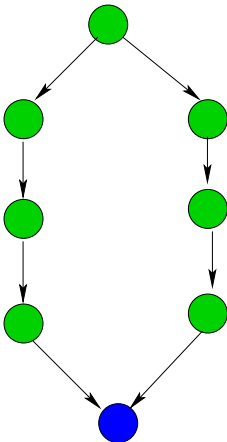
Exécution des workflows



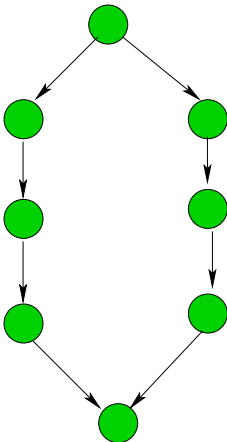
Exécution des workflows



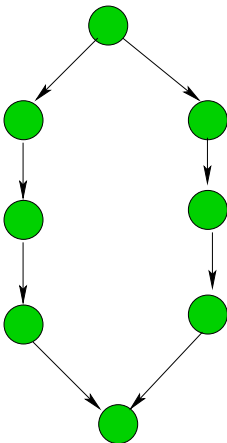
Exécution des workflows



Exécution des workflows



Exécution des workflows



- Deux questions
 - Quel ordre d'exécution entre tâches prêtes?
 - Mapping des tâches sur les ressources?

Ordonnancement de workflow : approche aléatoire

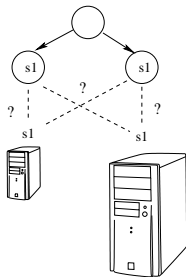
- Seules les dépendances de données permettent de définir l'ordre d'exécution, les tâches prêtes sont exécutées dans un ordre aléatoire
- Exemple : Condor DAGMan
- Inconvénient:
 - aucune prise en compte de la structure du workflow ni des ressources disponibles
- Amélioration: prendre en compte à un instant donné les caractéristiques des tâches prêtes
 - exemple: exécuter la tâche avec le temps de complétion (exécution+transfert de données) le plus petit (le plus grand)

Ordonnancement de workflow: approche statique (1/3)

- Cas 1:
 - ressources homogènes
 - nombre de ressources $>$ nombres de tâches prêtes
- ⇒ l'approche aléatoire est «optimale»

Ordonnancement de workflow: approche statique (1/3)

- Cas 1:
 - ressources homogènes
 - nombre de ressources $>$ nombres de tâches prêtes
⇒ l'approche aléatoire est «optimale»
- Cas 2:
 - ressources hétérogènes

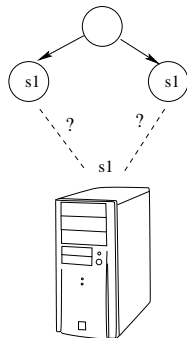


Ordonnancement de workflow: approche statique (1/3)

- Cas 1:
 - ressources homogènes
 - nombre de ressources $>$ nombres de tâches prêtes

⇒ l'approche aléatoire est «optimale»

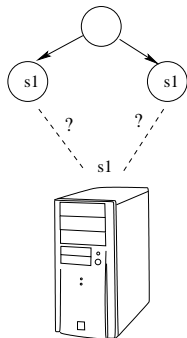
- Cas 2:
 - ressources hétérogènes
 - nombre de ressources $<$ nombres de tâches prêtes



Ordonnancement de workflow: approche statique (1/3)

- Cas 1:
 - ressources homogènes
 - nombre de ressources $>$ nombres de tâches prêtes
 - ⇒ l'approche aléatoire est «optimale»

- Cas 2:
 - ressources hétérogènes
 - nombre de ressources $<$ nombres de tâches prêtes
- ⇒ Affectation d'une priorité à chaque tâche



Ordonnancement de workflow: approche statique (2/3)

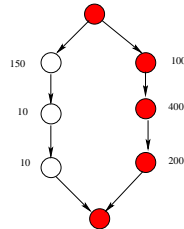
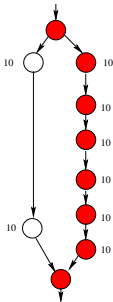
- Plusieurs heuristiques existantes (HEFT, CPA, CPR, etc.).
- Prise en compte de la structure du workflow, des ressources disponibles et éventuellement des coûts de communication
- Obtient généralement de meilleurs résultats par rapport à l'approche aléatoire

Ordonnancement de workflow: approche statique (2/3)

- Plusieurs heuristiques existantes (HEFT, CPA, CPR, etc.).
- Prise en compte de la structure du workflow, des ressources disponibles et éventuellement des coûts de communication
- Obtient généralement de meilleurs résultats par rapport à l'approche aléatoire
- Inconvénients:
 - hypothèse 1 : le client est seul
 - hypothèse 2 : les ressources de la plateforme et leurs caractéristiques sont statiques

Ordonnancement de workflow: approche statique (3/3)

- Algorithmes avec deux passes
 - première passe: affectation d'une priorité aux nœuds
 - deuxième passe: mapping sur les ressources⇒ List Scheduling
- Généralement l'objectif est de minimiser le «chemin critique»



Ordonnancement des workflows

Pour ordonnancer un workflow, il faut disposer :

- d'un algorithme/une heuristique
- des mesures/prédictions de performance
 - ⇒ Mapping + Scheduling
- un environnement de grid computing
 - ⇒ DIET
- + un moteur de workflow

Architecture de DIET

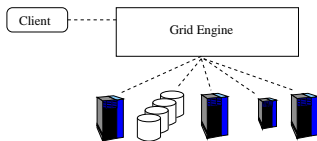
DIET en quelques points

- middleware + outils pour la grille

Architecture de DIET

DIET en quelques points

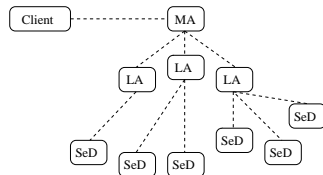
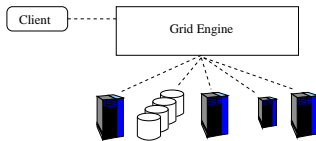
- middleware + outils pour la grille
- approche agent



Architecture de DIET

DIET en quelques points

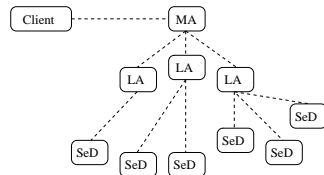
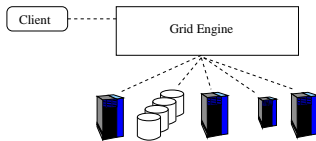
- middleware + outils pour la grille
- approche agent
 - broker = agents hiérarchiques



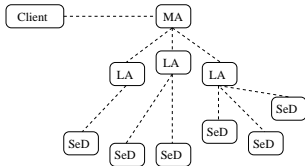
Architecture de DIET

DIET en quelques points

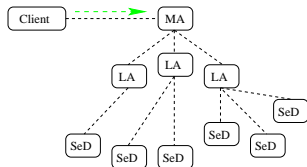
- middleware + outils pour la grille
- approche agent
 - broker = agents hiérarchiques
- basé sur RPC



DIET: fonctionnement

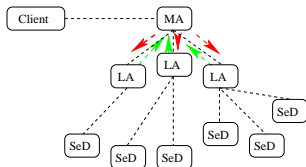


DIET: fonctionnement



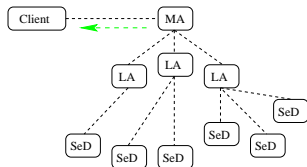
- 1 le client demande un service au Master Agent

DIET: fonctionnement



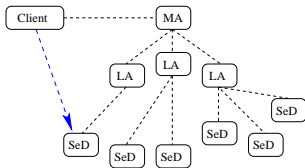
- 1 le client demande un service au Master Agent
- 2 le Master Agent interroge sa hiérarchie

DIET: fonctionnement



- 1 le client demande un service au Master Agent
- 2 le Master Agent interroge sa hiérarchie
- 3 le Master Agent renvoie une liste de SeD au client

DIET: fonctionnement



- 1 le client demande un service au Master Agent
- 2 le Master Agent interroge sa hiérarchie
- 3 le Master Agent renvoie une liste de SeD au client
- 4 le client exécute le service en invoquant directement le SeD

Objectifs

- Avoir un environnement de gestion de workflows
 - construction et exécution
 - utilisation d'algorithmes d'ordonnancement avancé
 - prise en charge des variations du système
 - gestion de multi-workflows

Plan

- 1 Introduction
 - Les Workflows
 - Présentation de DIET
 - Motivations et objectifs
- 2 Moteur de workflows
 - Description des workflows
 - Architecture logicielle
 - Ordonnancement
 - Mécanisme de réordonnancement
 - Multi-workflows
- 3 Conclusions

Langages de workflows

- Pas de standard (XML, scripts)
 - BPEL pas adapté
- Exemples:
 - Condor DAGMan: script
 - Pegasus: DAX (xml)
 - Taverna: XScuffl (xml)
 -
- La plupart des moteurs de workflows utilisent des DAG ou des réseaux de pétri.
- Le workflow est décrit à deux niveaux:
 - abstrait: description uniquement de l'application
 - concret: description de l'application et des informations nécessaires à l'exécution

Description du workflow dans DIET

- Format XML
- Profile diet: problème (id), paramètres (in, out et inout)
- Description des nœuds et des dépendances de données

```

<dag>
  <node id="node_id" path="path_name" [mul="value"]>
    <arg name = "arg_id" type="data_type" value="arg value"
      [base_type="base type" ] [nb_rows="nb_rows" ]
      [nb_cols="nb_cols" ] [matrix_order="matrix_order" ] />
    <in name = "port_id" type="data_type" [sink="source id" ]
      [base_type="base type" ] [nb_rows="nb_rows" ]
      [nb_cols="nb_cols" ] [matrix_order="matrix_order" ]
      [mul="value" ] />
    <out name = "port_id" type="data_type" [sink="sink id" ]
      [base_type="base type" ] [nb_rows="nb_rows" ]
      [nb_cols="nb_cols" ] [matrix_order="matrix_order" ]
      [mul="value" ] />
  </node>
  ...
</dag>

```

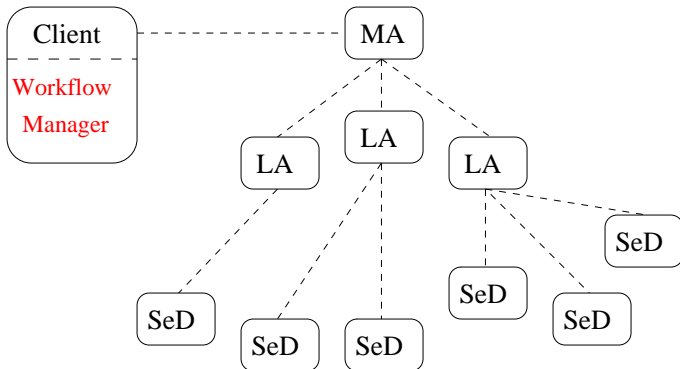
Quelle architecture?

Quelle architecture?

- Généralement le méta-ordonnanceur de workflows se trouve côté client

Quelle architecture?

- Généralement le méta-ordonnanceur de workflows se trouve côté client



Architecture logicielle

Workflow manager dans le client

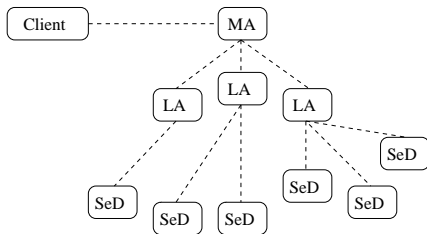
- Inconvénients:
 - aucune coopération entre les différents workflow managers.

Architecture logicielle

Workflow manager dans le client

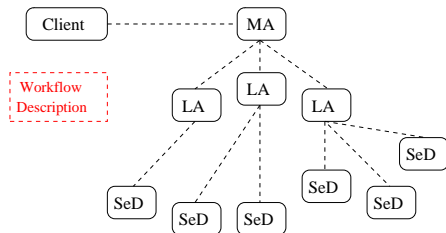
- Inconvénients:
 - aucune coopération entre les différents workflow managers.
- Avantages:
 - plus de souplesse pour tester de nouveaux algorithmes d'ordonnancement.

Architecture 1: fonctionnement

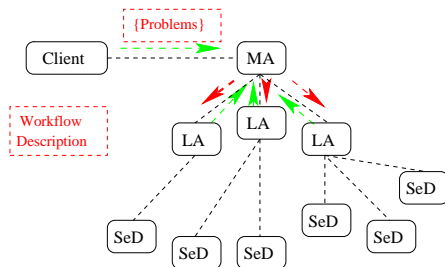


Architecture 1: fonctionnement

- 1 Le client traite la description du workflow

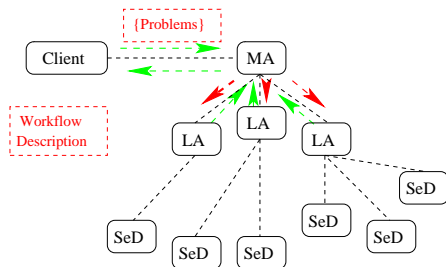


Architecture 1: fonctionnement



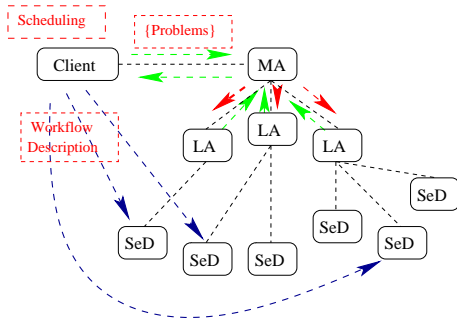
- 1 Le client traite la description du workflow
- 2 Le client envoie la liste des problèmes composants le workflow au Master Agent pour vérifier que le workflow peut être exécuté.

Architecture 1: fonctionnement



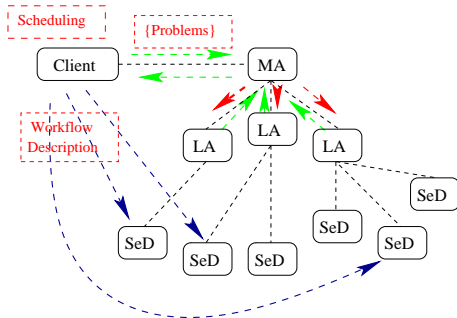
- 1 Le client traite la description du workflow
- 2 Le client envoie la liste des problèmes composants le workflow au Master Agent pour vérifier que le workflow peut être exécuté.
- 3 En cas de succès, le client reçoit la liste des serveurs nécessaires à l'exécution du workflow.

Architecture 1: fonctionnement



- 1 Le client traite la description du workflow
- 2 Le client envoie la liste des problèmes composants le workflow au Master Agent pour vérifier que le workflow peut être exécuté.
- 3 En cas de succès, le client reçoit la liste des serveurs nécessaires à l'exécution du workflow.
- 4 Construction de l'ordonnancement puis exécution du workflow

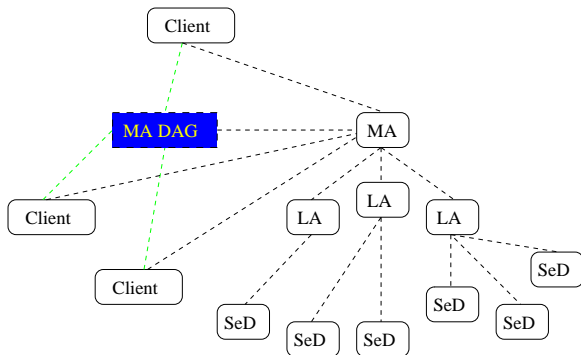
Architecture 1: fonctionnement



- 1 Le client traite la description du workflow
- 2 Le client envoie la liste des problèmes composants le workflow au Master Agent pour vérifier que le workflow peut être exécuté.
- 3 En cas de succès, le client reçoit la liste des serveurs nécessaires à l'exécution du workflow.
- 4 Construction de l'ordonnancement puis exécution du workflow
- 5 Récupération des résultats

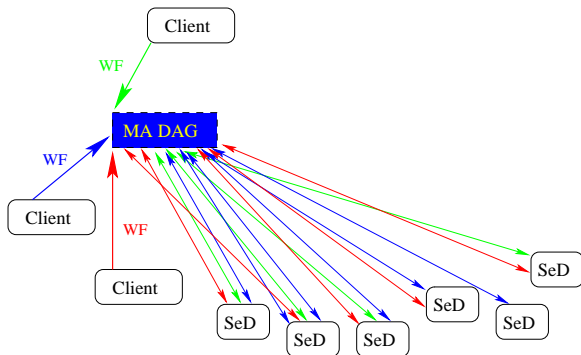
Architecture 2 : Workflow manager externe centralisé : le MA DAG

- Motivation: gérer le multi-workflows/multi-clients



Architecture 2 : Workflow manager externe centralisé : le MA DAG

- Motivation: gérer le multi-workflows/multi-clients
- ⇒ problème : gestion des données!



Architecture 2 : Workflow manager dans le client et le MA DAG

- Diviser le méta-ordonnanceur entre le client et le MA DAG

Architecture 2 : Workflow manager dans le client et le MA DAG

- Diviser le méta-ordonnanceur entre le client et le MA DAG
- La partie client du méta-ordonnanceur :

Architecture 2 : Workflow manager dans le client et le MA DAG

- Diviser le méta-ordonnanceur entre le client et le MA DAG
- La partie client du méta-ordonnanceur :
 - gestion des données

Architecture 2 : Workflow manager dans le client et le MA DAG

- Diviser le méta-ordonnanceur entre le client et le MA DAG
- La partie client du méta-ordonnanceur :
 - gestion des données
- La partie MA DAG du méta-ordonnanceur:

Architecture 2 : Workflow manager dans le client et le MA DAG

- Diviser le méta-ordonnanceur entre le client et le MA DAG
- La partie client du méta-ordonnanceur :
 - gestion des données
- La partie MA DAG du méta-ordonnanceur:
 - création d'un ordonnancement initial des workflows soumis

Architecture 2 : Fonctionnement

Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG

Architecture 2 : Fonctionnement

Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG
- le MA DAG (vérifie auprès du MA) la disponibilité des services

Architecture 2 : Fonctionnement

Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG
- le MA DAG (vérifie auprès du MA) la disponibilité des services
- en cas de succès le MA DAG construit son ordonnancement (ordering et mapping) et renvoie la réponse au client.

Architecture 2 : Fonctionnement

Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG
- le MA DAG (vérifie auprès du MA) la disponibilité des services
- en cas de succès le MA DAG construit son ordonnancement (ordering et mapping) et renvoie la réponse au client.
- le client exécute le workflow

Architecture 2 : Fonctionnement

Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG
- le MA DAG (vérifie auprès du MA) la disponibilité des services
- en cas de succès le MA DAG construit son ordonnancement (ordering et mapping) et renvoie la réponse au client.
- le client exécute le workflow

Mode 2: ordering

Architecture 2 : Fonctionnement

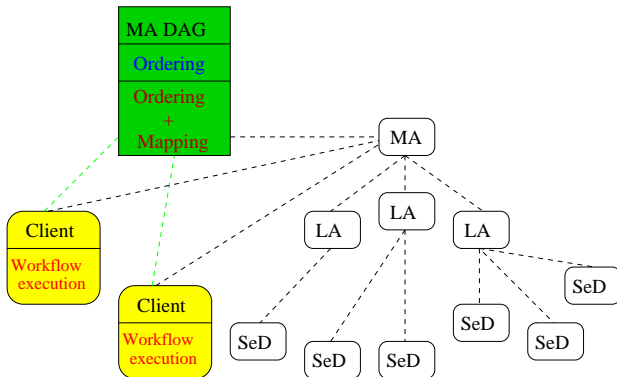
Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG
- le MA DAG (vérifie auprès du MA) la disponibilité des services
- en cas de succès le MA DAG construit son ordonnancement (ordering et mapping) et renvoie la réponse au client.
- le client exécute le workflow

Mode 2: ordering

- la réponse du MA DAG comprend uniquement un ordering, le client repasse par le Master Agent pour trouver le bon SeD.

Architecture 2



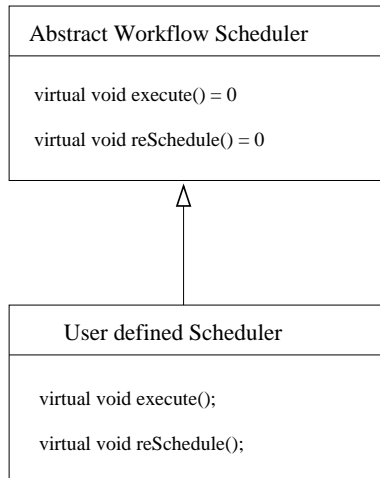
Implémentation

Au lieu d'en choisir une, les architectures retenues peuvent être utilisées conjointement dans la même plateforme.

- Architecture 1: le workflow manager est dans le client
- Architecture 2: utilisation du MA DAG
 - deux modes de fonctionnement: le MA DAG fournit le scheduling complet (ordering et mapping) ou seulement l'ordering

Schedulers dans DIET

- Scheduler de base (par défaut): Scheduler aléatoire
- Flexibilité pour ajouter de nouveaux ordonnanceurs
 - Architecture 1: le client peut fournir son propre scheduler
 - ne nécessite pas la recompilation de DIET
 - Architecture 2: choix du scheduler à la compilation
 - ajouter un nouveau scheduler \Rightarrow la recompilation de DIET



Implémentation dans DIET

- Schedulers avancés
 - Problème de prédiction de performances/temps de communication
 - Prédiction de performances
 - FAST ou CORI (Plugin Scheduler)?

Implémentation dans DIET

- Schedulers avancés
 - Problème de prédiction de performances/temps de communication
 - Prédiction de performances
 - FAST ou CORI (Plugin Scheduler)? ⇒ Plugin Scheduler
 - Temps de communication
 - NWS?
 - ⇒ nécessité de mise en œuvre simple pour l'utilisateur

Réordonnancement

- \forall l'ordonnanceur utilisé:
 - prédiction \Rightarrow risques d'erreur
 - grille de calcul \Rightarrow variations dynamiques

Réordonnancement

- \forall l'ordonnanceur utilisé:
 - prédiction \Rightarrow risques d'erreur
 - grille de calcul \Rightarrow variations dynamiques
 - il y a un risque que l'ordonnancement construit ne soit pas «optimal»

Réordonnancement

- \forall l'ordonnanceur utilisé:
 - prédiction \Rightarrow risques d'erreur
 - grille de calcul \Rightarrow variations dynamiques
 - il y a un risque que l'ordonnancement construit ne soit pas «optimal»
- nécessité de réordonnancer le workflow en cas de problèmes

Quand réordonnancer?

- périodiquement par le MA DAG

Quand réordonnancer?

- périodiquement par le MA DAG
- lorsqu'un nouveau client demande à exécuter un workflow

Quand réordonnancer?

- périodiquement par le MA DAG
- lorsqu'un nouveau client demande à exécuter un workflow
- par le MA lorsque de nouveaux serveurs apparaissent/disparaissent de la plateforme

Quand réordonnancer?

- périodiquement par le MA DAG
- lorsqu'un nouveau client demande à exécuter un workflow
- par le MA lorsque de nouveaux serveurs apparaissent/disparaissent de la plateforme
- quand l'ordonnancement prévu par le client ne correspond pas aux temps réels

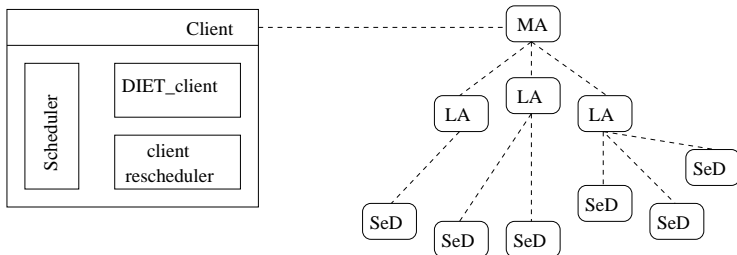
Quand réordonnancer?

- périodiquement par le MA DAG
- lorsqu'un nouveau client demande à exécuter un workflow
- par le MA lorsque de nouveaux serveurs apparaissent/disparaissent de la plateforme
- **quand l'ordonnancement prévu par le client ne correspond pas aux temps réels**

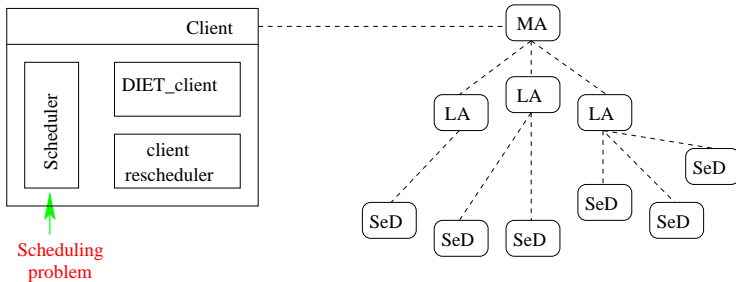
Quand réordonnancer?

- périodiquement par le MA DAG
- lorsqu'un nouveau client demande à exécuter un workflow
- par le MA lorsque de nouveaux serveurs apparaissent/disparaissent de la plateforme
- quand l'ordonnancement prévu par le client ne correspond pas aux temps réels
 - delta et le nombre de nœuds

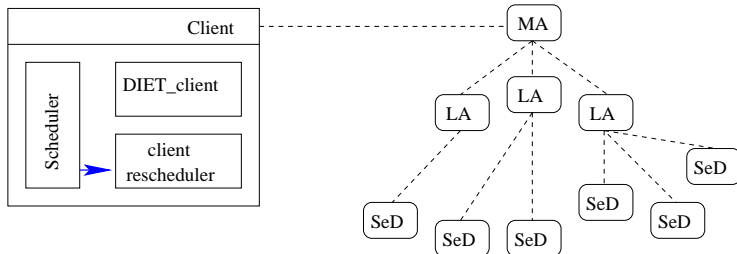
Implémentation



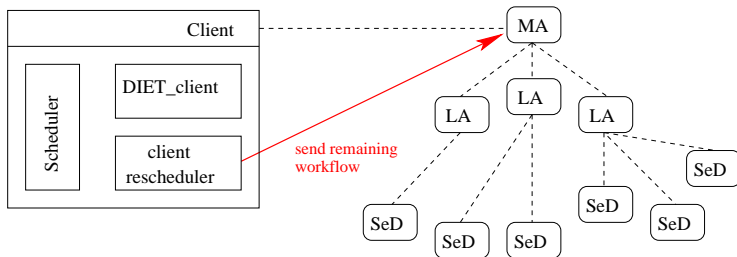
Implémentation



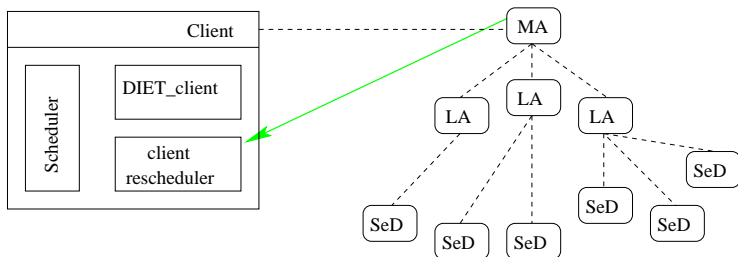
Implémentation



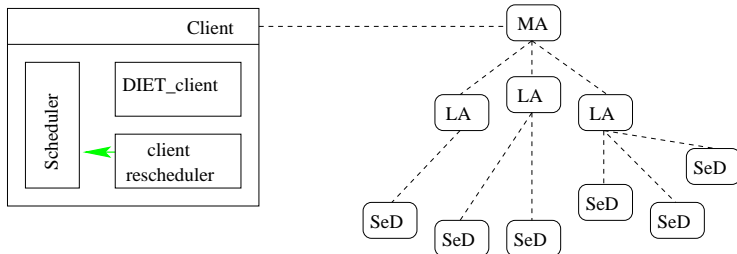
Implémentation



Implémentation



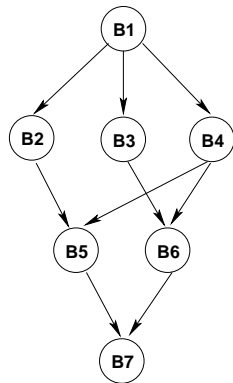
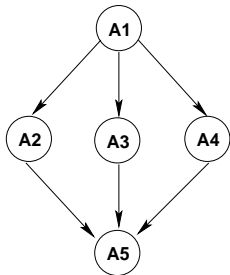
Implémentation



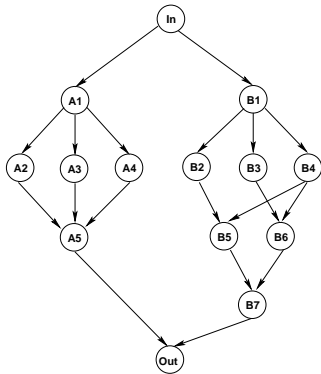
Multi-workflows

- Première approche:
 - En mode statique: fusion des DAGs en un seul
 - ⇒ Comment ordonnancer le DAG résultat?
 - En mode *online*: Traitement des DAGs dans leur ordre d'arrivée
- Pas beaucoup de travaux
 - ⇒ Henan Zhao & Al
 - Prise en compte de l'équité entre DAGs

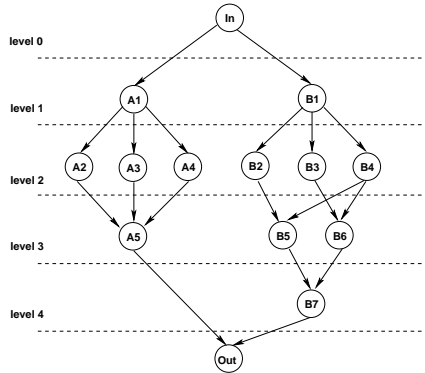
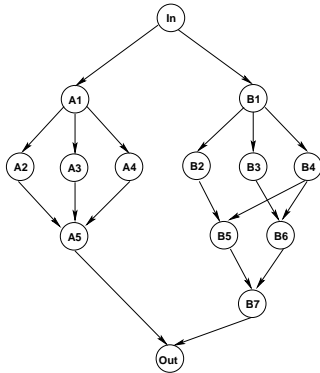
Multi-workflows: Fusion des DAGs (1/2)



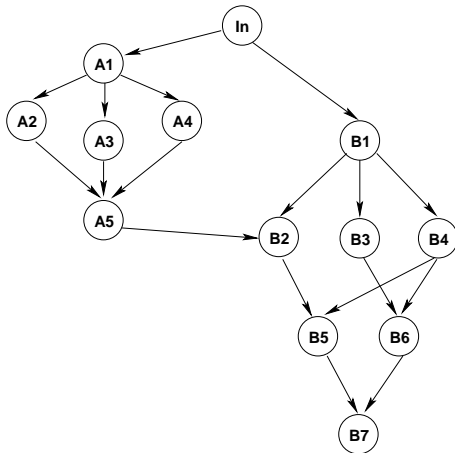
Multi-workflows: Fusion des DAGs (1/2)



Multi-workflows: Fusion des DAGs (1/2)



Multi-workflows: Fusion des DAGs (2/2)



Multi-workflows: Fairness on finish time

- Idée générale: privilégier le DAG qui est le plus ralenti dans un ordonnancement multi-workflow par rapport à son ordonnancement seul
- Algorithme simplifié:
 - 1 ordonnancer tous les DAGs séparément.
 - 2 trier les DAGs selon leur *slowdown* (initialisé à zéro).
 - 3 tant qu'il y a des tâches
 - prendre la première tâche (t) prête du premier DAG et l'ordonnancer
 - $slowdown(dag) = finish_time_own(t) / finish_time_multi(t)$

Multi-workflows: Fairness on finish time

Adaptation à DIET

- Contrainte: Connaissance au préalable de tous les DAGs à ordonnancer
- ⇒ difficilement applicable au cas réel
- Pour passer au mode *online*:
 - Mécanismes de monitoring
 - A chaque nouvelle soumission: réordonnancer tous les DAGs sans prendre en considération les tâches déjà exécutées
 - Mécanismes de réordonnancement

Plan

- 1 Introduction
 - Les Workflows
 - Présentation de DIET
 - Motivations et objectifs
- 2 Moteur de workflows
 - Description des workflows
 - Architecture logicielle
 - Ordonnancement
 - Mécanisme de réordonnancement
 - Multi-workflows
- 3 Conclusions

Conclusions

- Design et implémentation d'une architecture pour la gestion des workflows dans DIET
- Flexibilité dans la mise en place d'algorithmes de gestion de workflow (niveau client et MA Dag)
 - Ordonnancement aléatoire, Round Robbin et HEFT
- Mise en place de mécanismes de réordonnancement
- Gestion multi-workflows
 - Fusion de DAG (C1) et Fairness On Finish Time