Gestion de ressources Peer to peer pour les Grilles

Georges Da Costa

ISTI (Pisa) /UCY (Cyprus) / Coregrid

Plan

Plan

Contexte

Les Grilles sont difficiles à gérer
Future tendance : Interconnexion de Grilles

Quelques problèmes

- Dynamicité : les clusters vont et viennent
- Administration : qui doit gérer et être responsable
- Grande échelle, réactivité

Que se passe-t-il pour une Grille de 1000 clusters ?

Gestion et découverte de ressources

Grilles : grand nombre de ressources

- Web services
- Stockage
- Vol de cycle des Desktop

Point de vue de l'ordonanceur

- Choisir une place pour exécuter un job
- Différentes ressources : disque, processeur, réseau

$\longrightarrow$ Comment gérer ces ressources physiques

Requêtes

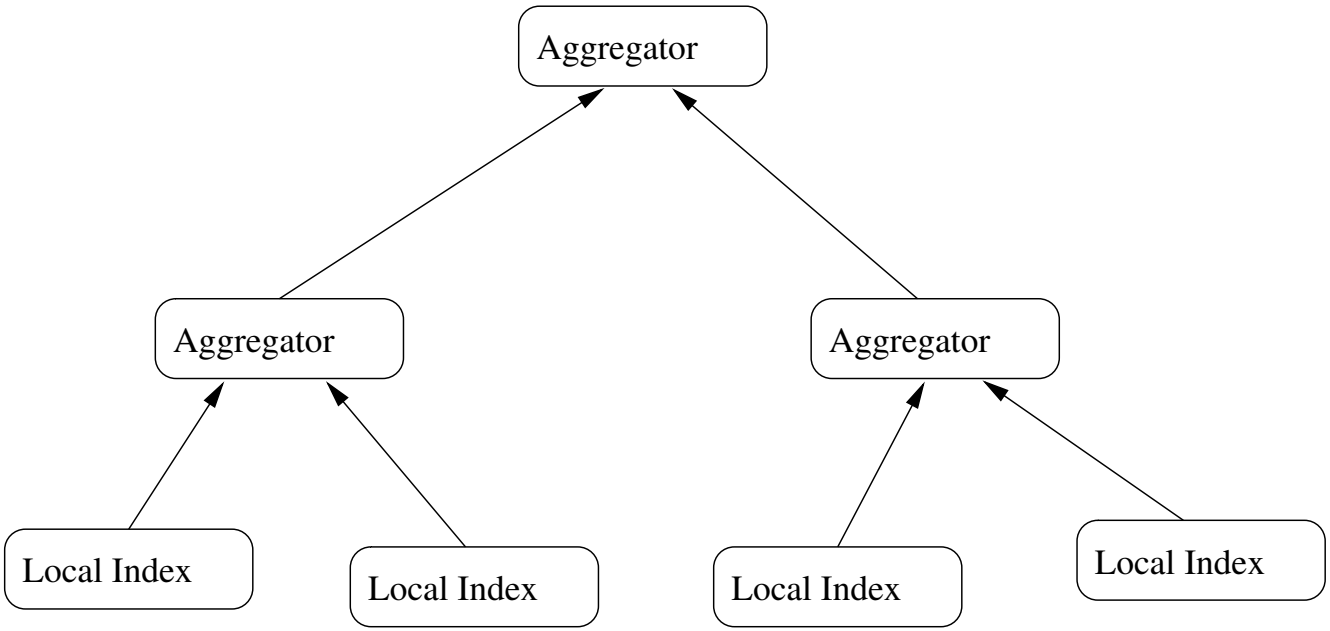
Exemple de requête :32 freecpu and (ethernet or SCI)

Requêtes

Requêtes Get à intervalles Multi-attributs

Les mises à jours sont seulement sur des valeurs (pas sur des intervalles)

Globus



Mise à jour à intervalles réguliers

Limitations de Globus

Système d'*aggregator*

- Lent
- Point d'administration central
- Beaucoup de travail pour les ingénieurs
- Un point particulièrement sensible
- Réponses locales aux requêtes
- Pas d'adaptation aux données
- Basé sur un Timer

Les systèmes Peer to Peer

Points positifs

- Passage à l'échelle
- Complètement distribués
- Gestion simple

Points négatifs

- Requêtes précise
- Une seule réponse

Plan

Prag

But : Gérer les ressources de manière efficace

Utilisation typique : Ordonnanceur, Utilisateur voulant lancer un job

Efficacité

- Peu de communications
- Faible latence
- Bonne répartition de la charge
- Passage à l'échelle

Qualité

- Nombre de réponses
- Validité des réponses

Modèle de l'environnement

Données réalistes

- Grilles académiques ou de production
- Quelle grille
  - CoreGrid
  - Laboratoire Italien (Egee)
  - Laboratoire Français (Cigri à Grenoble)

Données actuelles

- Requête : nombre de processeurs + données stables
- (en cours) Modélisation du nombre de processeurs libres

Hypothèses globales

Hypothèses

- La majorité des requêtes avec intervalle sont ouvertes
- Il y a différents niveaux de dynamicité (Upgrade/Request ratio)
- Certaines caractéristiques sont fondamentalement changeantes (bande passante, charge)

Requêtes

32+ freecpu and ethernet

Simplification

Les requêtes complexes (and/or) sont coupées et la fusion de leurs résultats est faite sur le noeud d'origine

Deux types de ressources

Dynamiques

- Processeurs libres
- Charge

Stable

- Infrastructure réseau
- Nombre total de processeurs

Répondre aux requêtes

Deux types de contraintes

Ressources dynamiques

Optimiser le coût de la mise à jour *et* des requêtes

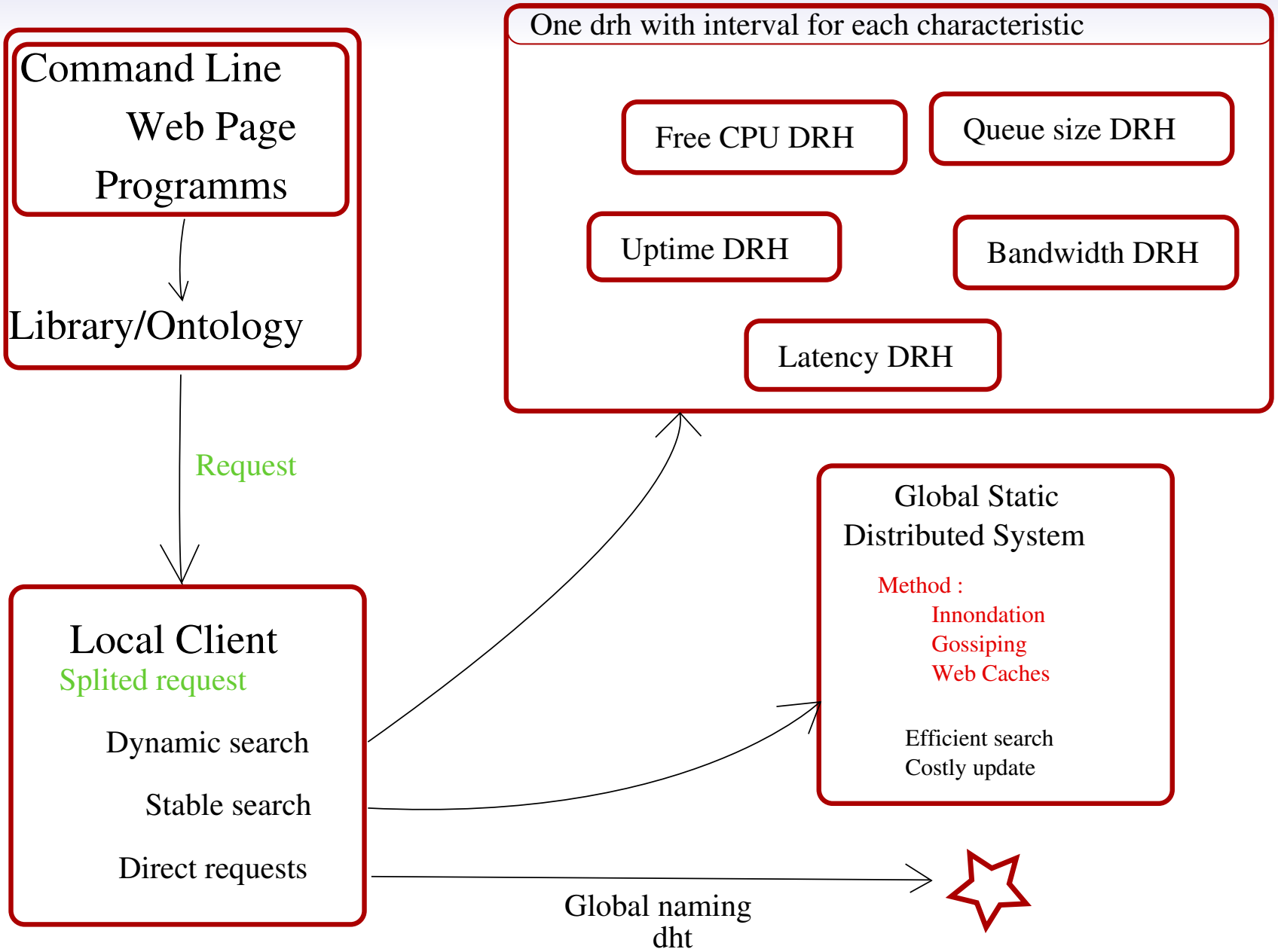
Ressources stables

Optimiser le coût des requêtes

Efficacité : un compromis entre les coûts des mises à jours et des requêtes

Ensuite, un compromis global : Efficacité/Qualité

Structure



Mises à jours liées aux événements

Distributed requests handler

Opérations basiques

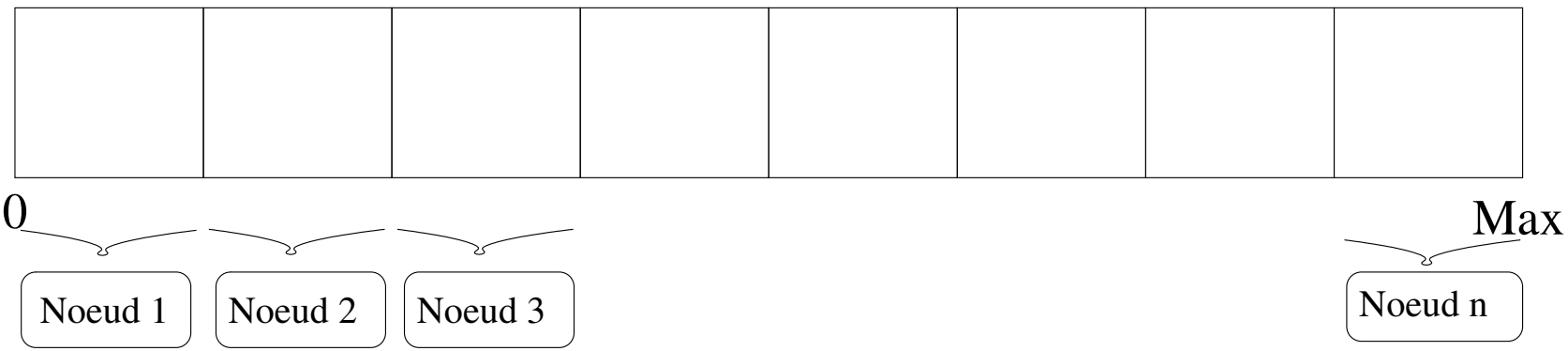
- data list = get($bound_1$, $bound_2$);
- data = get(key);
- update(key, data);

Métriques

- Nombre de messages pour une requête
- Nombre de messages pour une mise à jour
- Nombre de messages pour un départ/arrivée/faute
- Répartition de la charge
- Pourcentage de réponses

Systèmes Peer to Peer actuels avec intervalles

	Get	Mise à jour	Départ/arrivée
Probe	$N^{1/d}$	$N^{1/d}$	N/A
Nikos05	log(n)	log(n)	log(n)
Baton	log(n)	log(n)	log(n)
Ganesan04	log(n)	log(n)	log(n)



Problème : Coût d'une requête : log(n)+X

Problème : Répartition de la charge déséquilibrée

Une grande partie des requêtes sont ouvertes

Simulation

Hypothèses

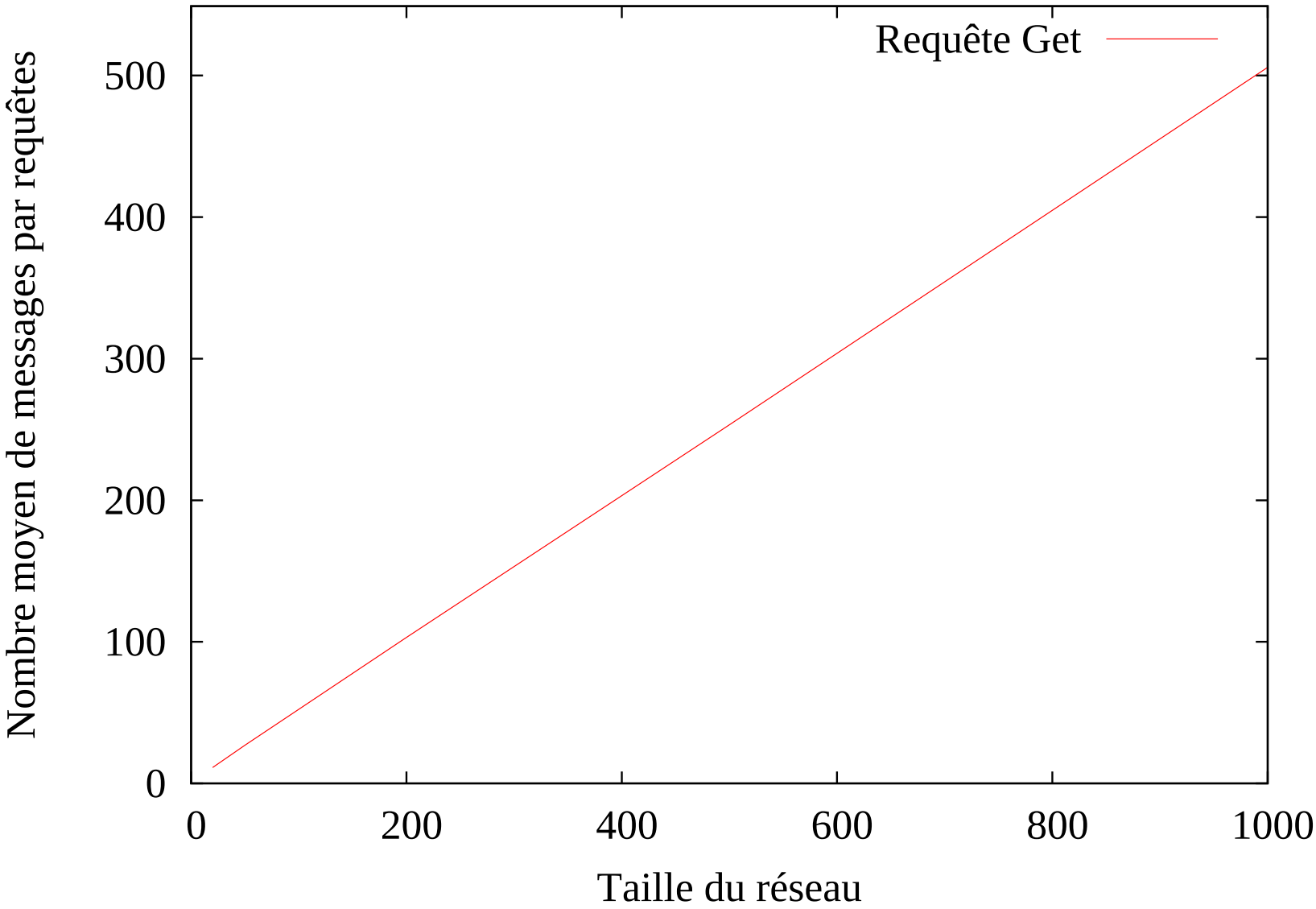
- Intervalles ouverts
- Requêtes uniformes
- Temps entre requêtes long devant leur traitement

Système de test

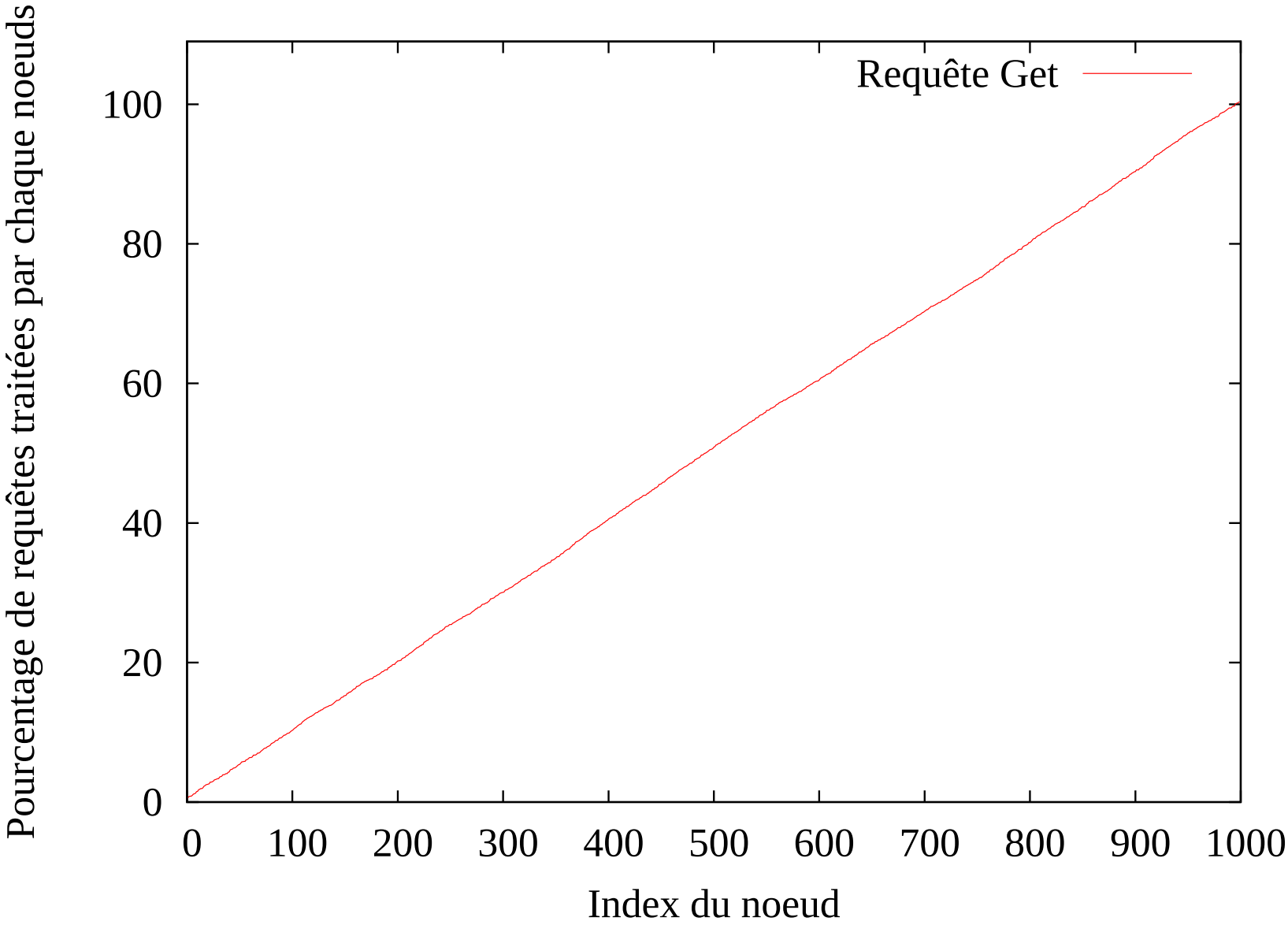
- A la Chord
- Les requêtes d'intervalle sont transmises de proche en proche (traitement séquentiel)

10000 requêtes
1000 noeuds

Coût des requêtes dans un système usuel



Répartition de la charge dans un système usuel



Premières expérimentations

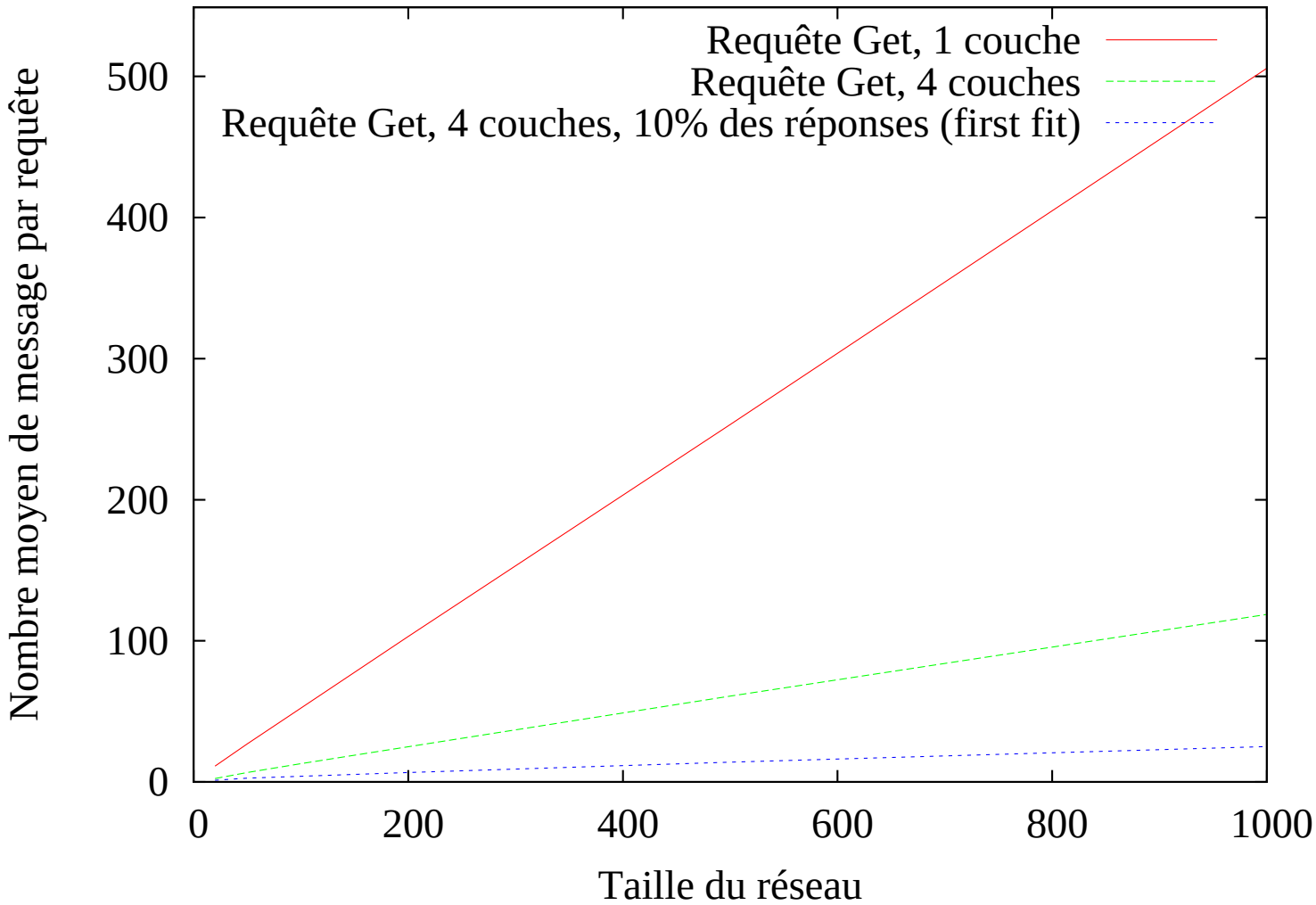
Que se passe t'il si il y a plusieurs DHT ?

Nouveau système de test

- 4 DHT identiques
- Mettre à jour : 4 mises à jours
- Requête Get : Une requête sur une DHT aléatoirement choisie

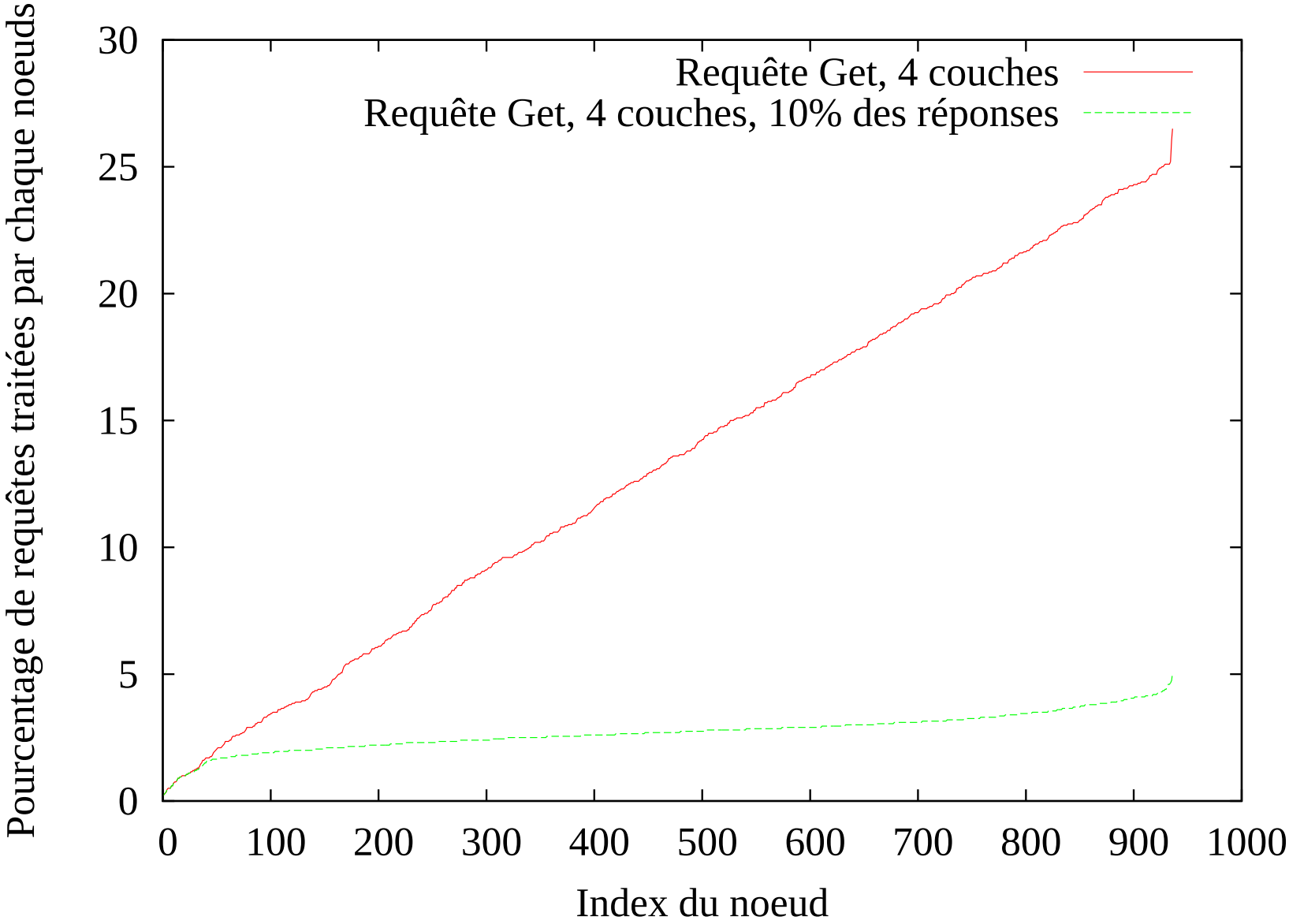
Mises à jours plus coûteuses mais requêtes Get moins

Coût des requêtes



Une petite latence implique des données plus à jours

Répartition de la charge



Unification

Comment choisir le nombre de couches ?

Évaluer à priori le nombre de couches en fonction du ratio :
Get/Mises à jours

Problèmes

- Le ratio peut varier
- Difficile de le connaître à priori

$\Longrightarrow$ Faire évoluer ce nombre

- 1 Dht $\longrightarrow$ cas usuel
- n Dht $\longrightarrow$ duplication totale

(n est le nombre de noeuds du système)

Unification

Opérations de base

- Fusion (une DHT en devient deux)
- Séparation (deux DHT en deviennent une)

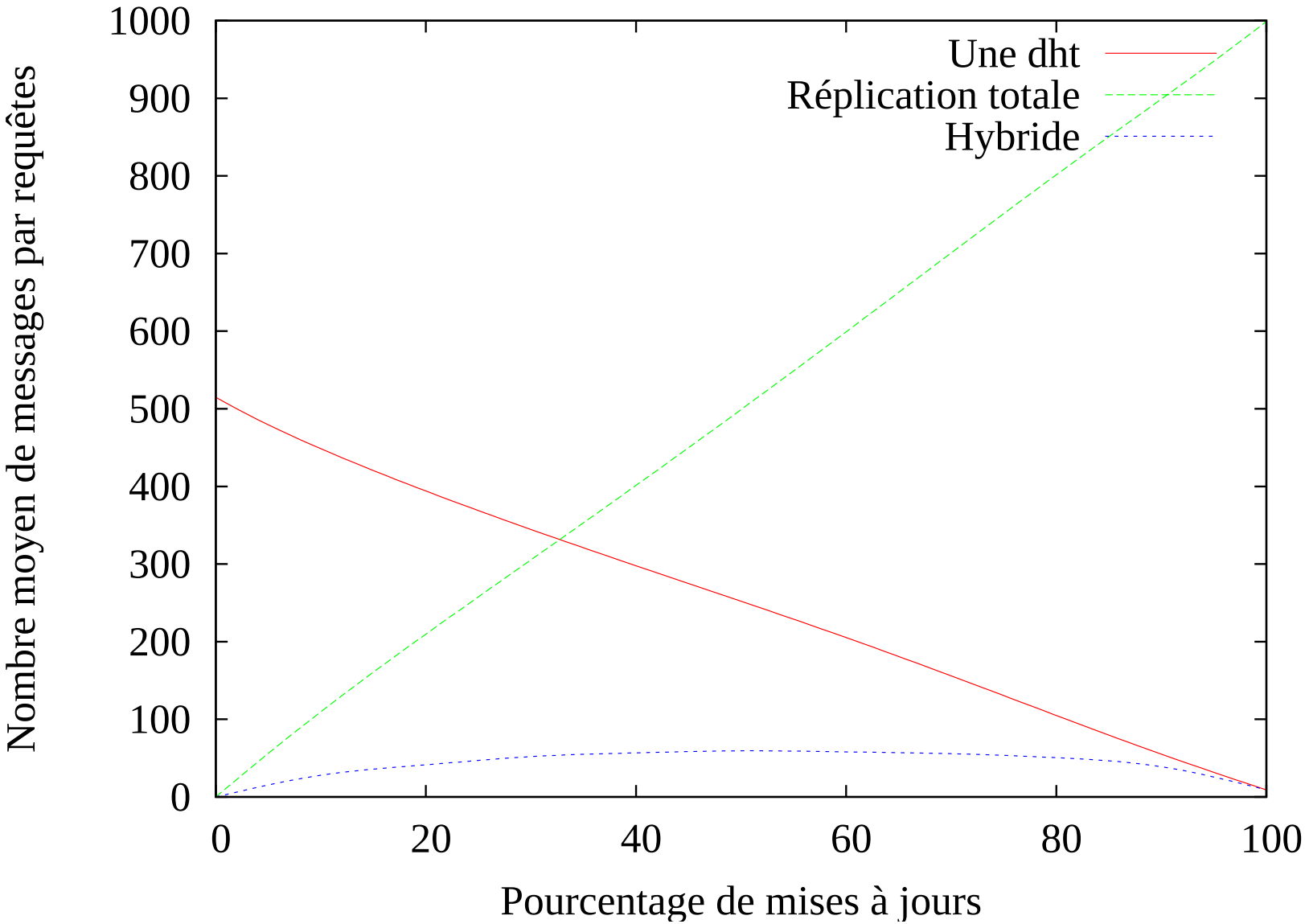
Le noeud responsable de la valeur 0 décide

Les autres noeuds lui envoient leurs valeurs

Algorithme de recuit simulé global

(stabilisation très longue)

Unification



Plan

Conclusion

Conclusion

- Globus est améliorable pour les grandes Grilles
- Nous proposons une architecture *auto-** pour gérer les ressources des grandes Grilles

Perspectives

- Modéliser les ressources
- Simulation de différents systèmes Peer to Peer gérant les intervalles
- Évaluer l'opération p DHT $\longrightarrow p+1$ DHT
- Définir et évaluer le compromis Qualité/Efficacité
- Intégrer les Multi-Attributs au niveau des requêtes

Questions ?