Guillaume GALLANI
FICM 2007
Département *Information & Systèmes*

RAPPORT DE PROJET DE RECHERCHE
FICM 3A

Diffusion peer-to-peer de données sur grille de calcul

Résumé—**Les grilles de calcul se révèlent, de nos jours, indispensables afin de pouvoir mener à bien des calculs extrêmement coûteux en ressources. L'évolution des technologies a permis de rendre ces grilles distribuées, en connectant des sites distants par des réseaux extrêmement performants. De ce fait, il devient capital de pouvoir distribuer efficacement des données sur ces grilles. Cette diffusion se doit d'être aussi rapide que possible, tout en restant sûre et en garantissant l'intégrité des données. Mais surtout, elle doit être adaptée à un passage à grande échelle, un calcul sur une grille pouvant impliquer plusieurs centaines de nœuds.**

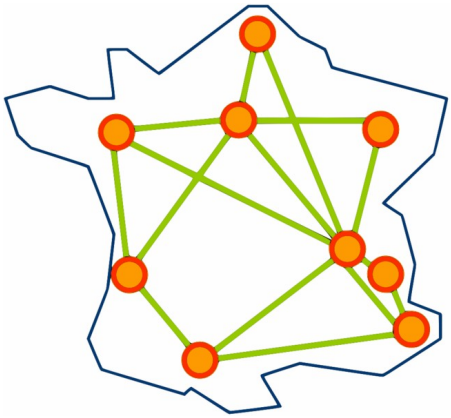
Nous analyserons ici l'influence de l'utilisation du protocole BitTorrent pour réaliser de tels transferts au sein de l'environnement Grid5000. En effet, une telle architecture peer-to-peer devrait permettre de tirer profit du grand nombre potentiel d'utilisateurs dans le cas de propagation de données sur une grille et se révéler plus intéressante que d'autres protocoles. Nous le comparerons ensuite à certains de ces autres protocoles, afin de discuter de son efficacité.

Mots-clés : protocole bittorrent, transferts de données, grille de calcul.

I. Introduction

A. Présentation de Grid5000

Grid5000 est une grille de calcul Française, regroupant 9 sites (Lille, Paris, Nancy, Orsay, Grenoble, Sophia Antipolis, Rennes, Toulouse et Bordeaux) appartenant majoritairement à l'Institut National de Recherche en Informatique et Automatique (INRIA). Une *grille* est un ensemble de *clusters*.



Les sites de Grid5000

Le but de Grid5000 est de fournir une plate-forme adaptable, pouvant être facilement reconfigurée et contrôlée pour réaliser des expériences (« job ») sur des réseaux distribués et/ou parallèles à grande échelle. Ainsi, il est possible de déployer un environnement personnalisé sur l'ensemble des nœuds réservés pour un *job*.

Actuellement, le réseau Grid5000 compte un peu plus de 6000 processeurs répartis sur les différents sites. Ceux-ci sont reliés entre eux par une fibre optique assurant un débit de 10 Gb/s, les nœuds d'un site étant généralement reliés entre eux par une connexion assurant un débit de 1 Gb/s.

B. Protocole bittorrent

BitTorrent est un protocole de transfert de données pair-à-pair créé au début des années 2000. Il est basé sur le constat suivant :

- si un fichier se trouve sur un seul serveur, plus il est demandé, moins il sera accessible (par saturation du serveur).
- cette tendance s'inverse si chaque client ayant téléchargé le fichier devient à son tour serveur.

On découpe donc le fichier à transférer en segments (appelés *pièces*), et l'on distribue le fichier en l'envoyant segment par segment. Une fois qu'un pair a téléchargé une pièce, il devient alors serveur pour cette pièce. De ce fait, chaque pair voulant télécharger le fichier pourra non seulement recevoir des pièces du fichier **par le serveur**, mais également par **n'importe quel autre pair** ayant à disposition une pièce encore non téléchargée.

Avantages :

- on allège la charge du serveur détenant initialement le fichier.
- plus il y a d'utilisateurs, pour le transfert sera efficace (contrairement à un transfert utilisant un protocole classique, tel que FTP par exemple)

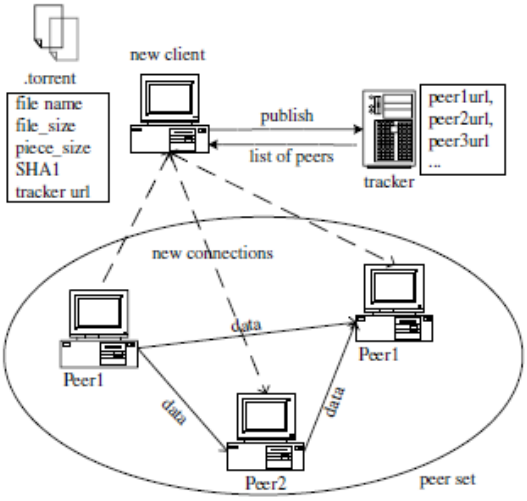
Inconvénients :

- le protocole applique le principe du *tit for tat* (« un prêté pour un rendu »), ce qui implique que plus l'on a partagé (c'est à dire envoyé des données), plus on sera prioritaire pour recevoir des données : cela peut donc rendre le démarrage difficile pour les nouveaux venus sur un transfert.
- Au fil du temps, les fichiers ont tendance à être de moins en moins partagés et donc de moins en moins accessibles. Cet inconvénient ne sera toutefois pas pertinent dans notre cas d'utilisation.

Un client commence par lire les meta-informations d'un fichier au format *.torrent*, contenant les informations nécessaires au transfert :

- l'adresse du *tracker* (cf. plus loin)
- le nombre de pièces et, le cas échéant, leur hash (afin de vérifier leur intégrité)
- la taille des pièces

Le *tracker* est un serveur connaissant en permanence le nombre de pairs et de seeds (les pairs possédant la totalité du fichier) participant au transfert, et qu'interrogent de temps à autre les pairs afin d'obtenir ces informations.



Arrivée d'un nouveau pair

Un pair télécharge en priorité les pièces les plus rares, afin de tenter de maintenir une répartition uniforme et, surtout, d'empêcher qu'une pièce ne soit disponible que chez un petit nombre de clients, voire chez aucun. De plus, un pair change régulièrement les clients auxquels il envoie des données, afin de garantir cette répartition parmi tout l'*essaim* (l'ensemble des pairs).

Dans les dernières versions de BitTorrent, on utilise une *Distributed Hash Table*, qui consiste à héberger les informations du tracker sur plusieurs ordinateurs.

C. Client BitTorrent utilisé

Pour réaliser nos tests, nous utilisons un client basique que nous modifions légèrement. Nous partons du client de démonstration de la librairie LibTorrent[1] et y apportons les altérations suivantes :

– **Écriture de l'avancement du transfert dans un fichier**. Pour pouvoir calculer de façon précise le temps total d'un transfert, il nous faut connaître l'évolution détaillée de celui-ci pour chacun des pairs impliqués. Le client va donc régulièrement (toutes les 100 ms) écrire dans un fichier son avancement dans le transfert.

– **Arrêt automatique du client**. Une fois le transfert terminé chez tous les pairs, il faut pouvoir récupérer automatiquement leurs fichiers d'avancement. Il faut également pouvoir fermer automatiquement le client BitTorrent sur chacun des nœuds, afin de pouvoir relancer d'autres transferts. Une fois le transfert terminé pour un nœud, celui-ci va régulièrement interroger le tracker sur l'avancement général du transfert. Une fois que tous les pairs ont mené à bien le téléchargement, ils ferment automatiquement le client.

Il est important de noter que les performances de ce client ont été comparées à celles d'un autre client, implémenté en Python : *BitTornado*. Les résultats obtenus dans les deux cas de figure étant tout à fait comparables, on peut en conclure que les problèmes rencontrés, que nous détaillerons plus loin, ne sont pas dus à notre implémentation du protocole, mais bien au protocole BitTorrent lui-même.

II. Expérimentation préliminaires

A. Architecture Grid5000

Dans un premier temps, nous avons réalisé quelques expériences destinées à caractériser les limites matérielles de grid5000, plus précisément les vitesses d'écriture des disques durs et de transfert du réseau. En effet, connaître ces limites se révèlera essentiel pour l'analyse de l'efficacité des différentes méthodes testées, car ce sont elles qui détermineront le débit dans certains cas. Les expériences ultérieures sont presque toutes réalisées sur le site de Nancy, possédant deux clusters : *grelon* et *griffon*. On détermine donc au préalable leurs vitesses d'écriture respectives.

Pour ce faire, on crée[1] sur plusieurs nœuds un fichier d'une taille de 1 Go rempli uniquement de zéros. On constate que les valeurs peuvent différer grandement selon les nœuds :

- Pour le cluster *grelon*, les vitesses d'écritures mesurées sur 100 nœuds vont de **30,9** à **9,2 Mo/s**, avec une moyenne de **27 Mo/s**.
- Pour le cluster *griffon*, les valeurs mesurées sur 20 nœuds s'étendent de **47,6** à **Mo/s** pour une moyenne de **35 Mo/s**.

Les nœuds étant connectés entre eux par un réseau à 1Gb/s, on a un débit théorique de **125Mo/s**. On doit donc s'attendre à atteindre au mieux ce débit lors de nos tests.

II. Évaluations de méthodes classiques

A. SCP

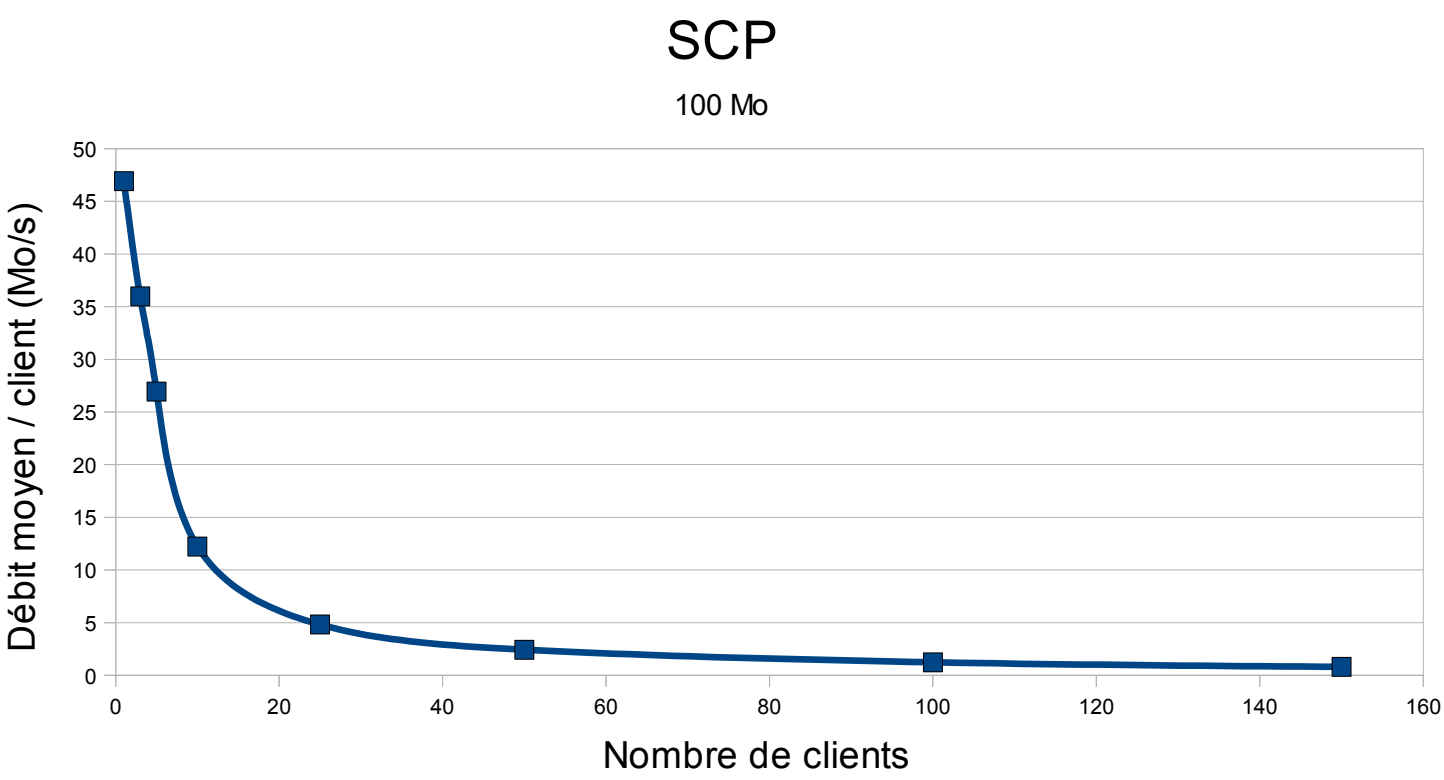
Le *Secure Copy Protocol* permet de copier des fichiers de façon sécurisée via le protocole SSH. Il s'agit d'une méthode de transfert simple, où un seul nœud est utilisé pour envoyer un fichier à tous les autres : les débits devraient donc décroître très rapidement avec l'augmentation du nombre de pairs. On aura ainsi une base de comparaison à partir de laquelle estimer l'efficacité du protocole BitTorrent.

On réalise le transfert simultané d'un fichier de 100 Mo vers un nombre croissant de clients afin de déterminer précisément l'influence de ce paramètre. Le graphique de la page suivante présente l'évolution du débit moyen par client, leur nombre variant de 1 à 150, et les valeurs numériques sont recensées dans le tableau ci-dessous.

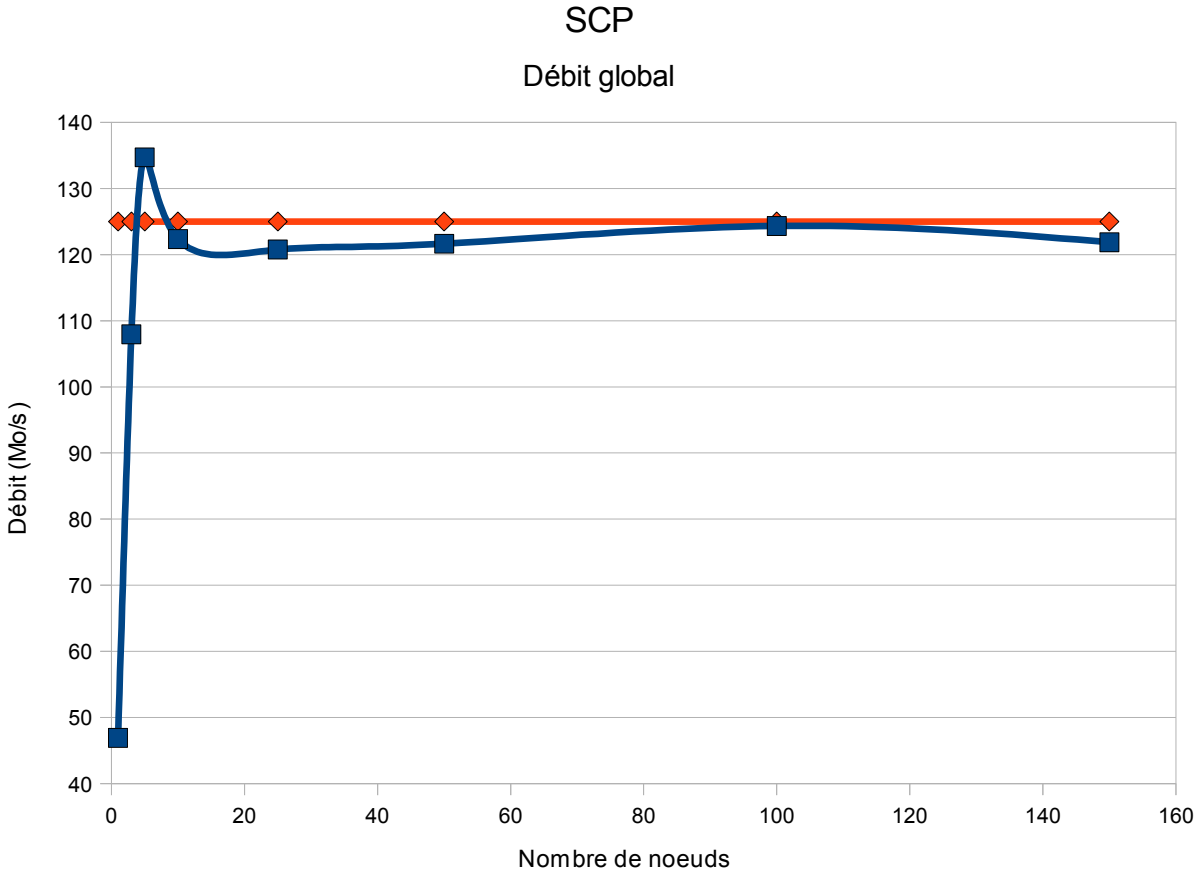
Valeurs des débits (en Mo/s)

Clients	1	3	5	10	25	50	100	150
Débit moyen pour un nœud (Mo/s)	45,11	34,6	25,91	11,76	4,64	2,34	1,2	0,78
Débit global (Mo/s)	45.11	103.8	129.55	117.6	116	117	120	117

1 Commande : dd if=/dev/urandom count=1 bs=1000M of=foo



On constate que le débit individuel moyen s'écroule rapidement : un seul nœud envoie les données, le débit **global** est donc limité par sa bande passante uniquement. On la **sature** (cf figure suivante) à partir de 3 nœuds, valeur à partir de laquelle le débit global reste à peu près égal au *débit maximal théorique* (125 Mo/s, courbe rouge). Pour un faible nombre de nœuds, on n'atteint pas cette valeur, car on est limité par la vitesse d'écriture sur les disques, qui est inférieure.



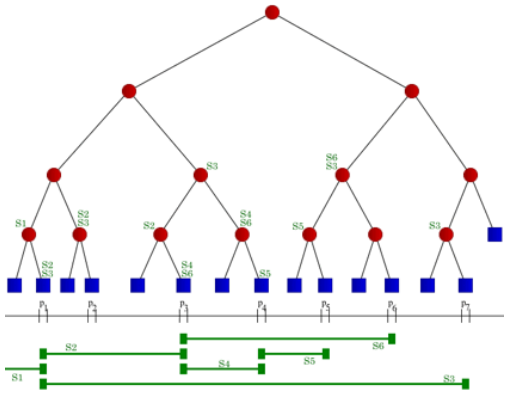
B. Arbres de diffusion

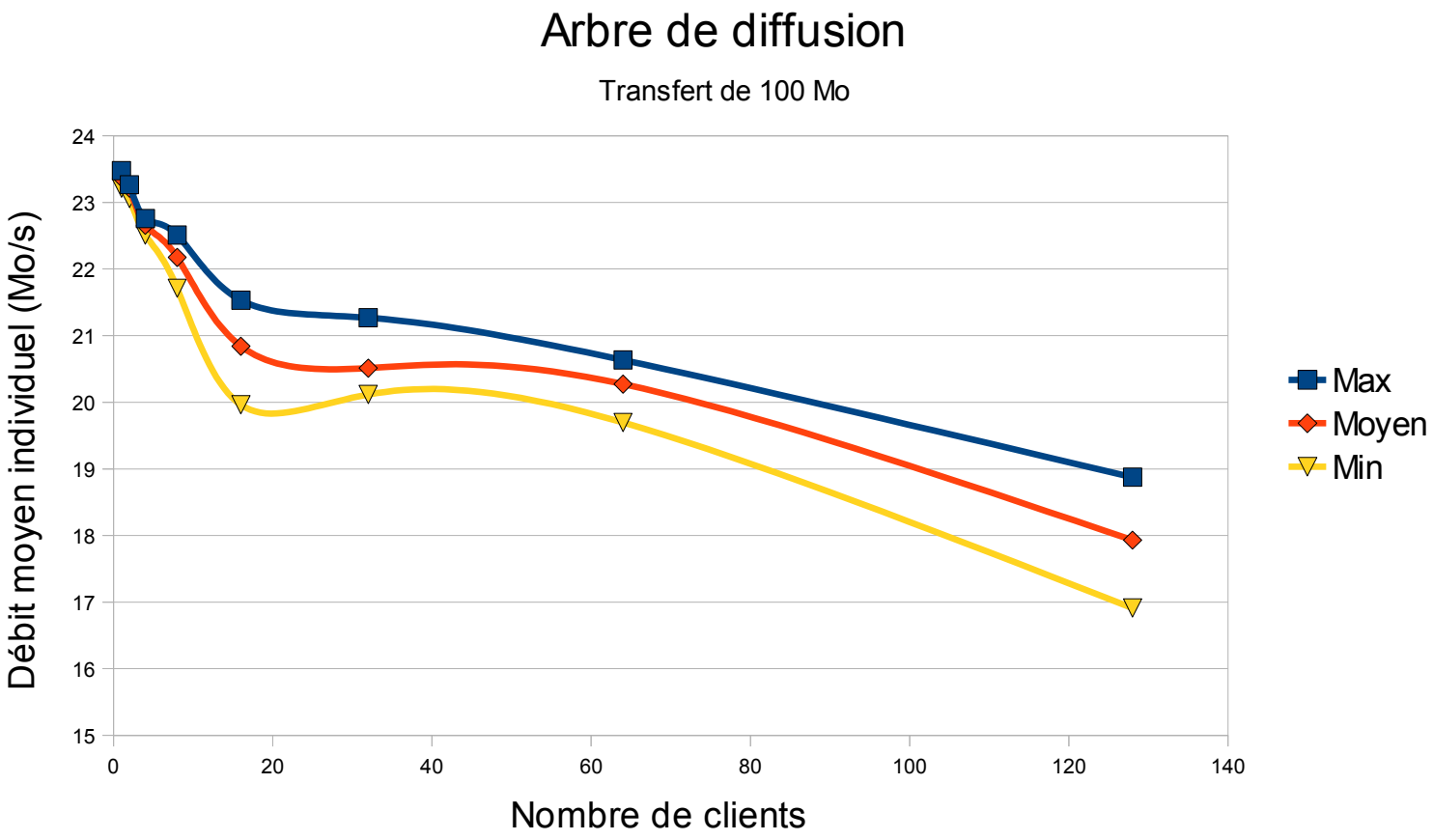
Une autre méthode classique de diffusion de données, très efficace si l'on connaît précisément la topologie du réseau, est l'utilisation d'arbres de diffusion.

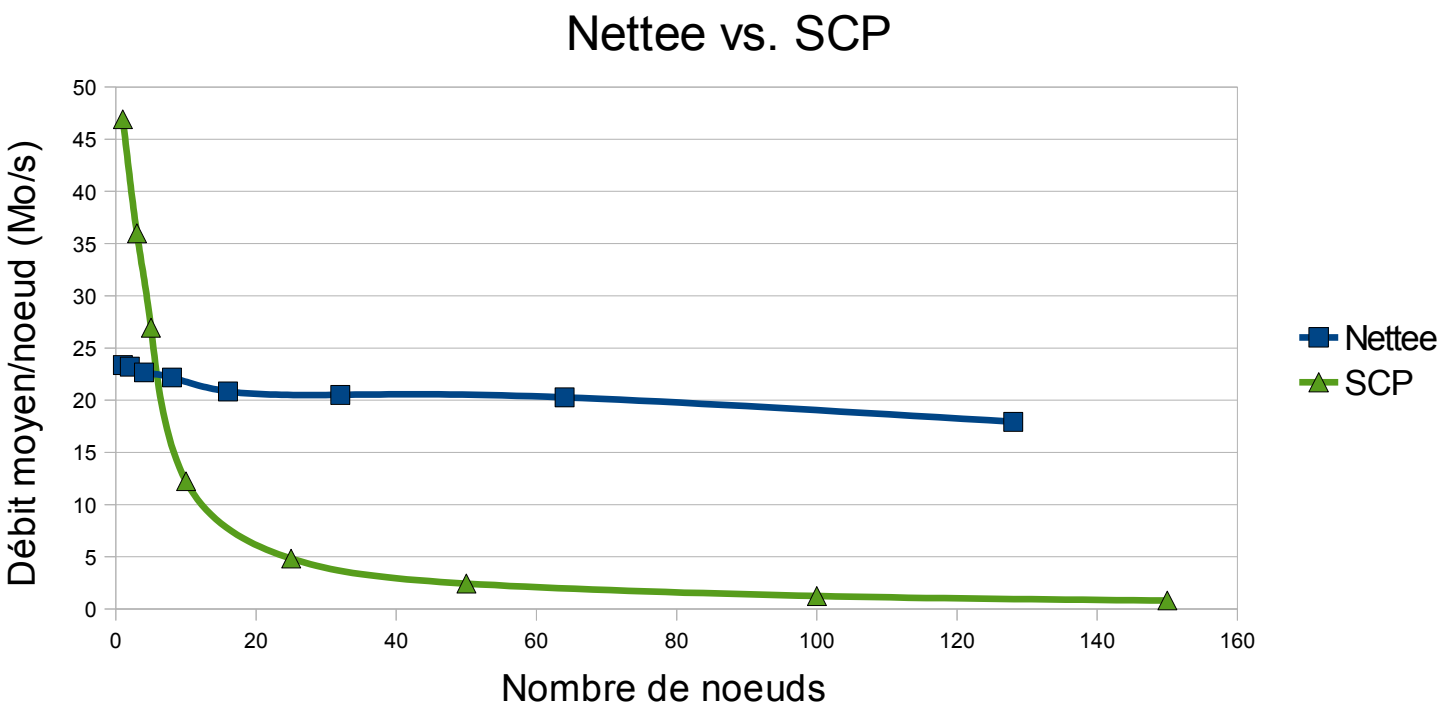
Nous utilisons, pour réaliser l'expérience, *nettee*. Ce programme permet de créer une **chaîne de diffusion**, où chaque nœud se connecte à un unique successeur. Dès lors qu'un nœud reçoit des données de son prédécesseur, il les relaye vers son successeur. Cela permet d'utiliser la bande passante de **tous les nœuds** impliqués dans le transfert, et non pas uniquement celle du premier nœud de la chaîne.

Pour que chaque nœud connaisse son successeur, il faut donc connaître exactement **tous les nœuds** impliqués dans le transfert. Si cela peut s'avérer compliqué, voire impossible, dans un contexte de transfert peer-to-peer « classique », ça n'est plus vrai pour Grid5000. En effet, pour réaliser un *job* sur les nœuds de la grille, il faut les **réserver** : on connaît donc précisément les nœuds qui seront impliqués dans le transfert. Une chaîne de diffusion est donc facilement construite entre eux. Elle devrait, selon toute vraisemblance, donner de meilleurs résultats que ceux obtenus plus haut.

Structure de type arbre

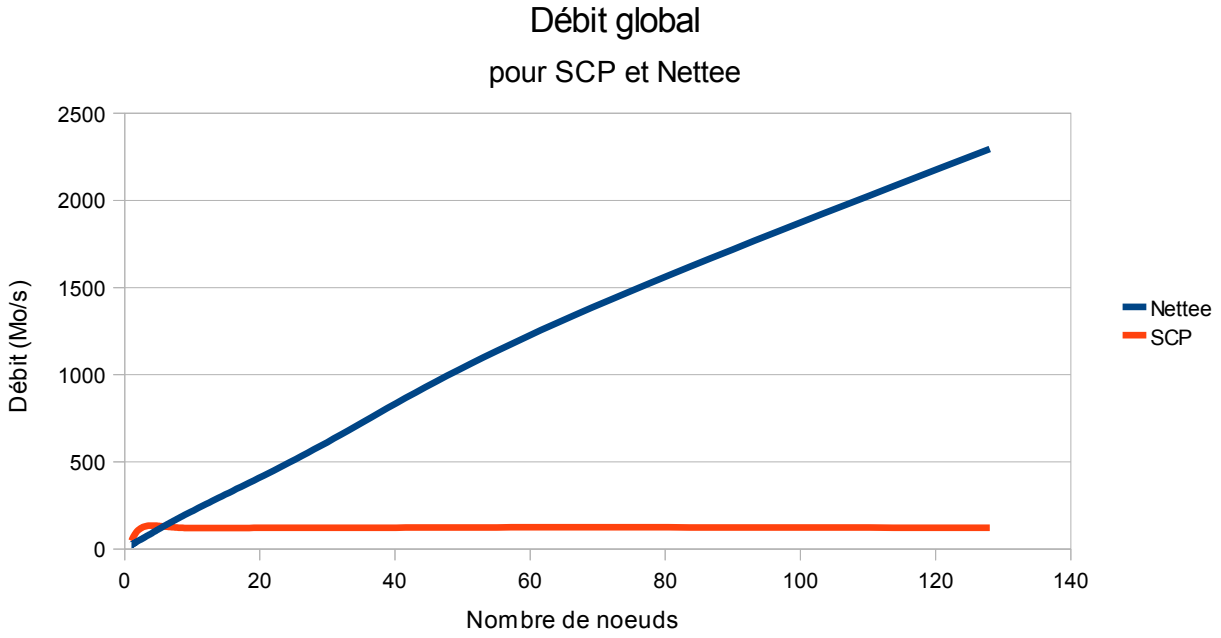






On constate que le débit moyen de chaque nœud diminue bien moins qu'avec SCP lorsque le nombre de clients augmente : en moyenne, on passe de **23,39** à **17,98 Mo/s** (soit une diminution de l'ordre de **20%**), lorsque leur nombre passe de 1 à 128. Pour SCP, on passait de **45,1** à **0,78 Mo/s** (diminution de plus de **98%** !) en passant de 1 à 150 nœuds. Cela est dû au fait que l'on n'est plus limité par la bande passante d'un seul nœud : le débit global peut donc continuer d'augmenter avec le nombre de nœuds impliqués dans le transfert (cf. figure 2). Cette évolution semble d'ailleurs **linéaire**. Pour un **faible nombre** de nœuds, cependant, on a des performances inférieures à celles de SCP.

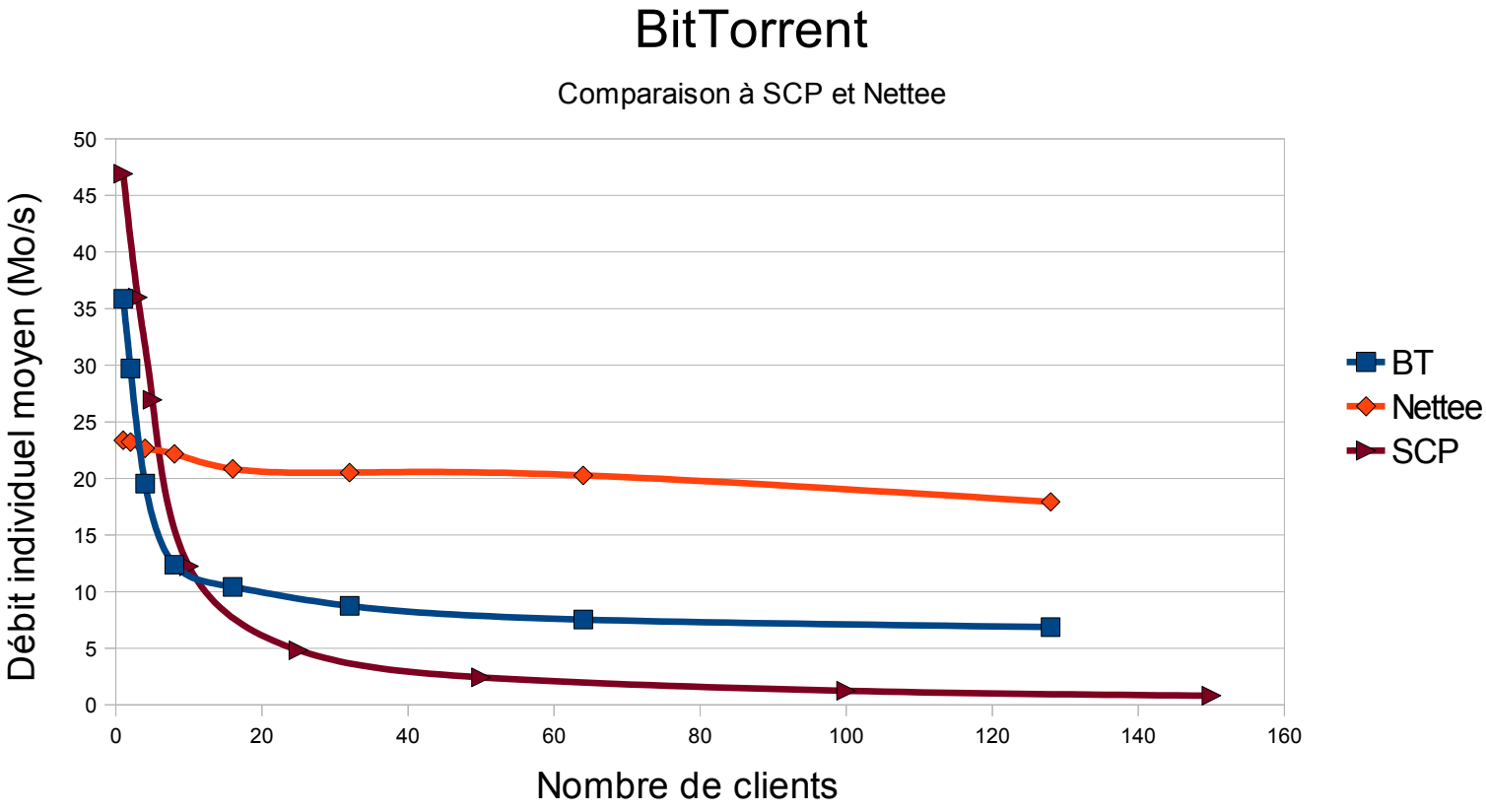
L'augmentation significative du débit global n'étant due qu'au fait que chaque pair émet des données, et non plus un seul, décuplant ainsi la bande passante disponible, on devrait avoir des résultats comparables avec BitTorrent.



IV. Évaluation de BitTorrent pour diffusion de données sur une grille

A. Paramètres par défaut

Dans un premier temps, nous allons comparer les résultats obtenus en utilisant notre client BitTorrent à ceux obtenus avec les méthodes classiques décrites plus haut. Dans un second temps, nous étudierons l'influence sur ses performances d'un paramètre inhérent à BitTorrent : la taille d'une pièce. Enfin, nous analyserons l'impact de la topographie du réseau sur ces différents modes de transfert.



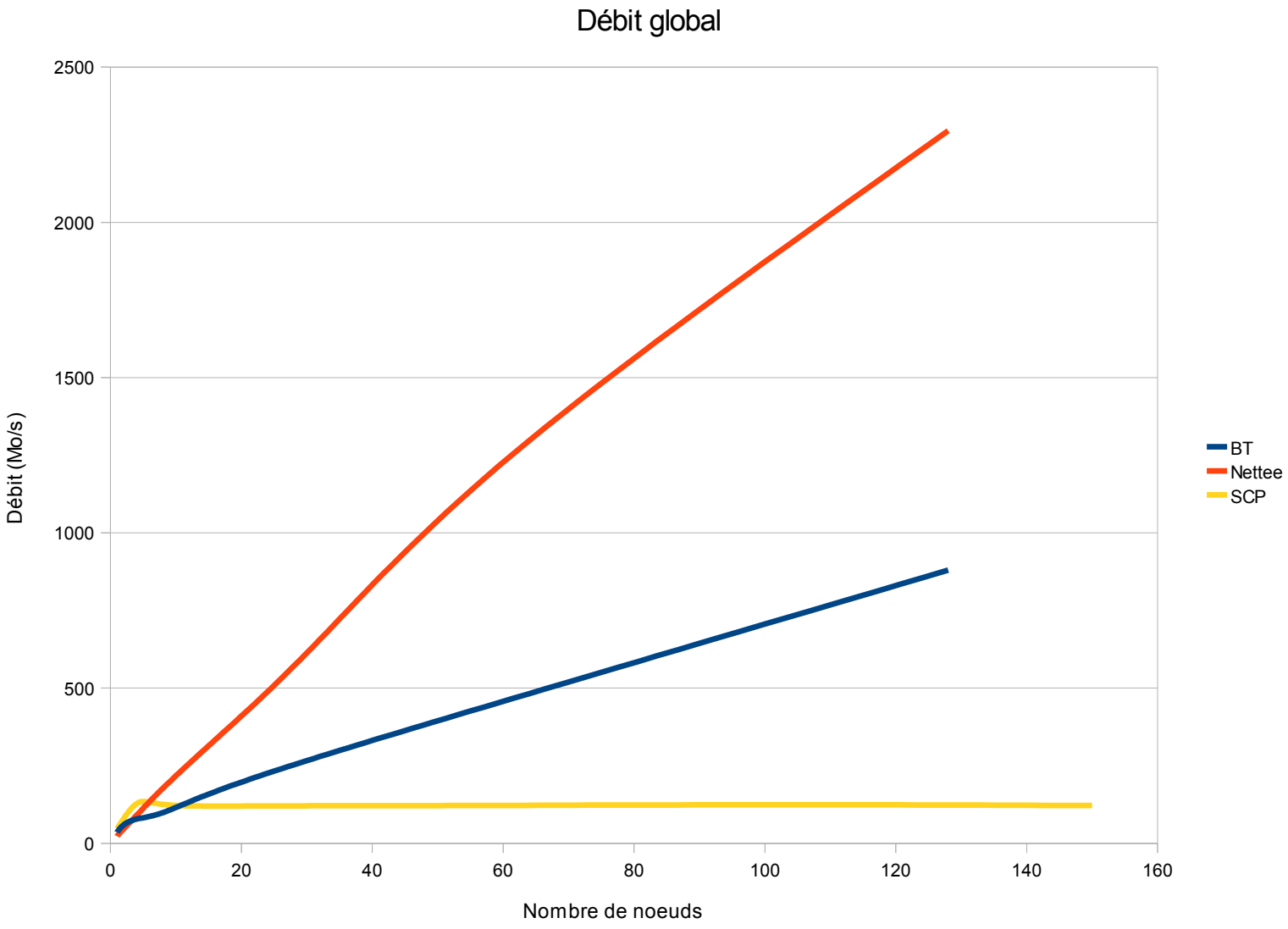
On constate que les performances de BitTorrent se trouvent **entre** celles d'une chaîne de diffusion et de SCP. Si l'on pouvait s'attendre à de meilleures performances qu'avec SCP, du fait de la contribution d'*a priori* tous les nœuds au transfert, on remarque cependant un résultat nettement inférieur à celui obtenu au moyen d'une chaîne de diffusion.

Le tableau ci-dessous récapitule les valeurs numériques pour BitTorrent et la chaîne de diffusion :

Nombre de nœuds	Débit BitTorrent (Mo/s)	Débit chaîne (Mo/s)
1	35.86	23.37
2	29.71	23.21
4	19.55	22.66
8	12.38	22.17
16	10.42	20.84

32	8.74	20.51
64	7.45	20.27
128	6.87	17.93

Si les performances de BitTorrent sont supérieures à celle d'une chaîne pour un petit nombre de nœuds (1-3), elles sont ensuite environ **deux fois** moins bonnes. Comme le montre le graphique ci-dessous, l'évolution du débit global pour BitTorrent est, comme pour une chaîne, **linéaire** et non-limitée, mais cependant plus lente.



B. Influence de la taille des pièces

Par défaut, la taille d'une pièce est de 2^{18} octets, soit environ 131 Ko : pour un fichier de 100 Mo, on en a donc un peu moins de 400. En s'interrogeant sur l'influence de ce paramètre, on prévoit deux choses :

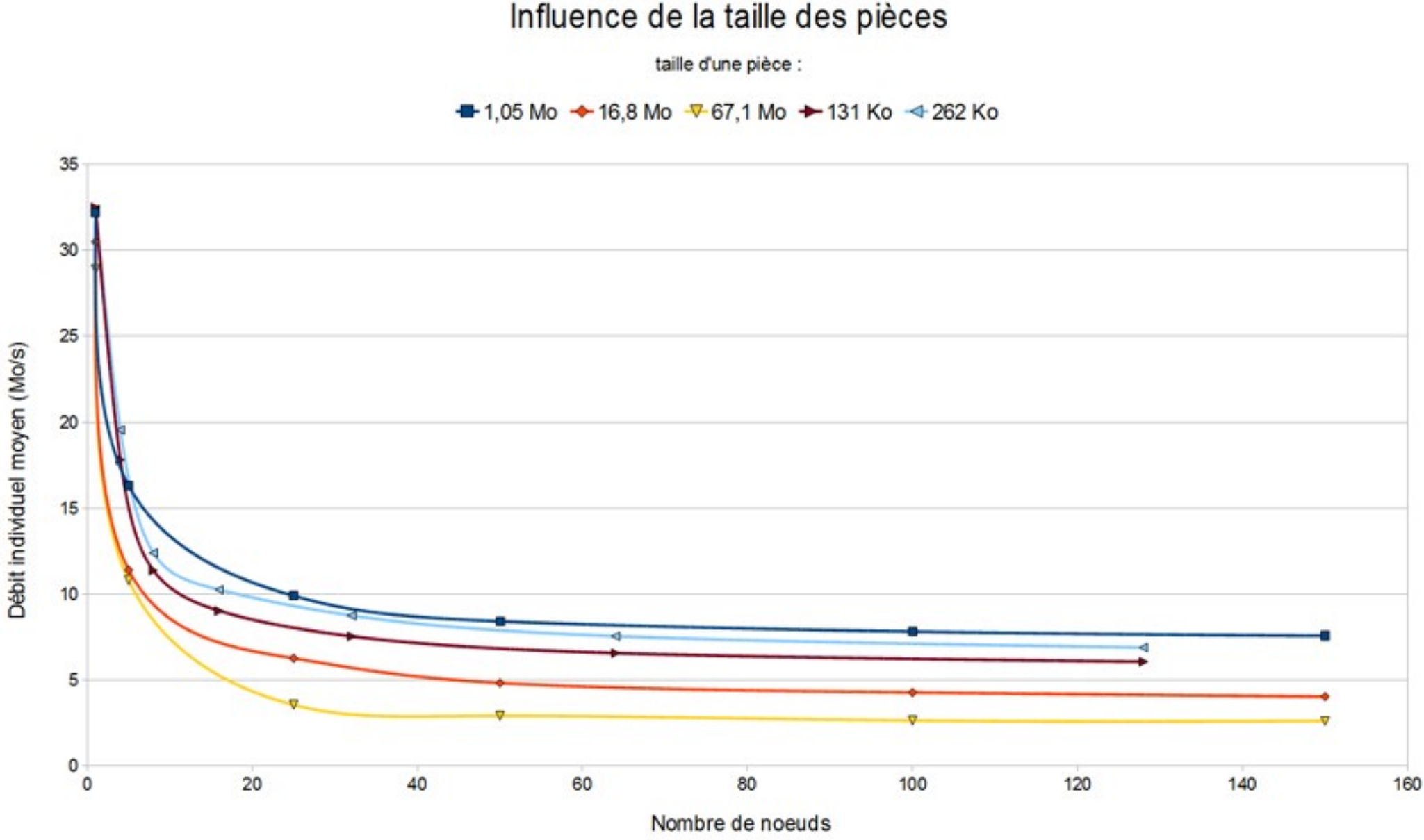
– **Si l'on a un grand nombre de pièces**, il faudra interroger régulièrement le tracker afin de les localiser, ce qui fera perdre du temps au client. De plus, les récupérer toutes se révèlera complexe.

– **Si l'on a peu de pièces**, les pairs perdront moins de temps à les localiser. En revanche, il sera plus long de les télécharger : en conséquence, moins de pairs seront à même de les mettra à disposition des autres.

Il faut donc déterminer la taille de pièces offrant le meilleur **compromis** entre ces deux cas de figure : on réalise donc des transferts pour différentes tailles, à la fois supérieurs et inférieures à celle par défaut.

La courbe page suivante présente l'évolution du débit individuel moyen des nœuds. Les meilleures performances sont obtenues avec des pièces de 220 octets (environ 1 Mo) : comme on l'avait présenti, les tailles supérieures (trop importantes) et inférieures (trop faibles) se révèlent moins efficaces.

Cependant, ces performances restent toujours nettement inférieures à celles d'une chaîne de diffusion : nous analyserons les raisons de cet écart dans notre conclusion.

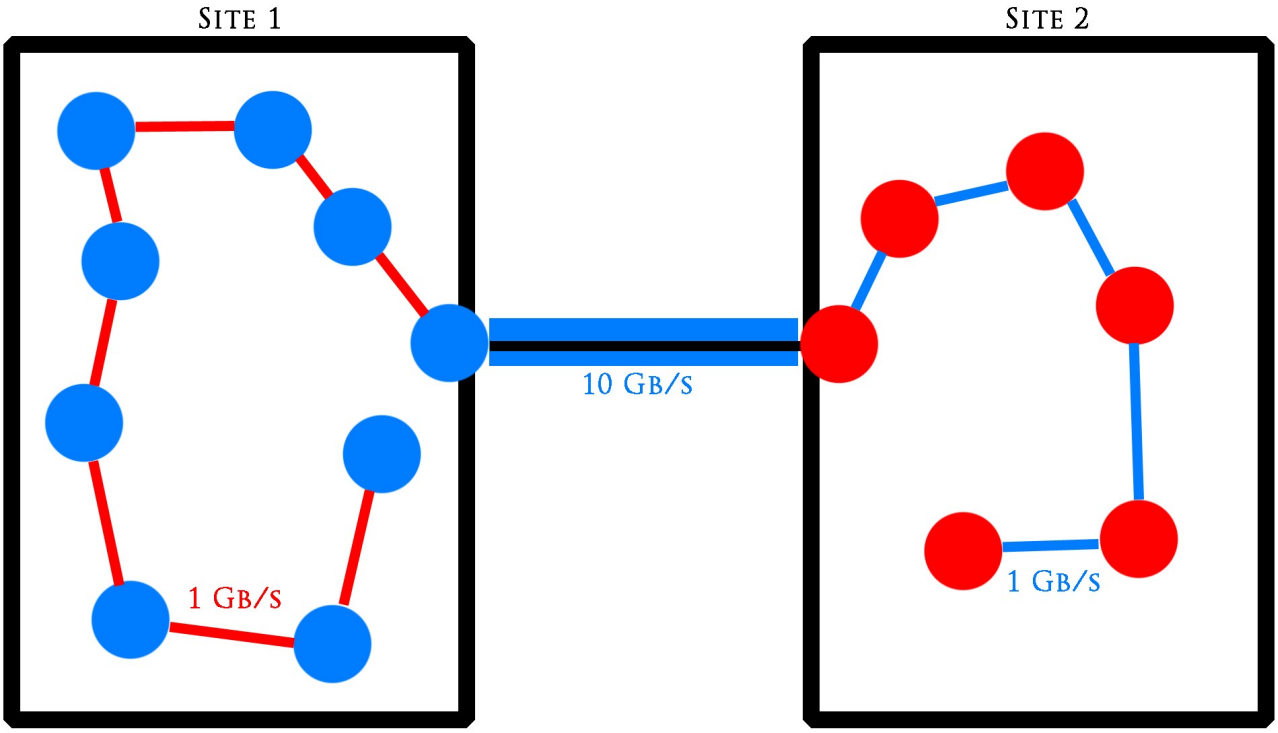

Influence de la taille des pièces
taille d'une pièce :
1,05 Mo
16,8 Mo
67,1 Mo
131 Ko
262 Ko
Débit individuel moyen (Mo/s)
0
5
10
15
20
25
30
35
0
20
40
60
80
100
120
140
160
Nombre de noeuds

C. Influence de topologie du réseau

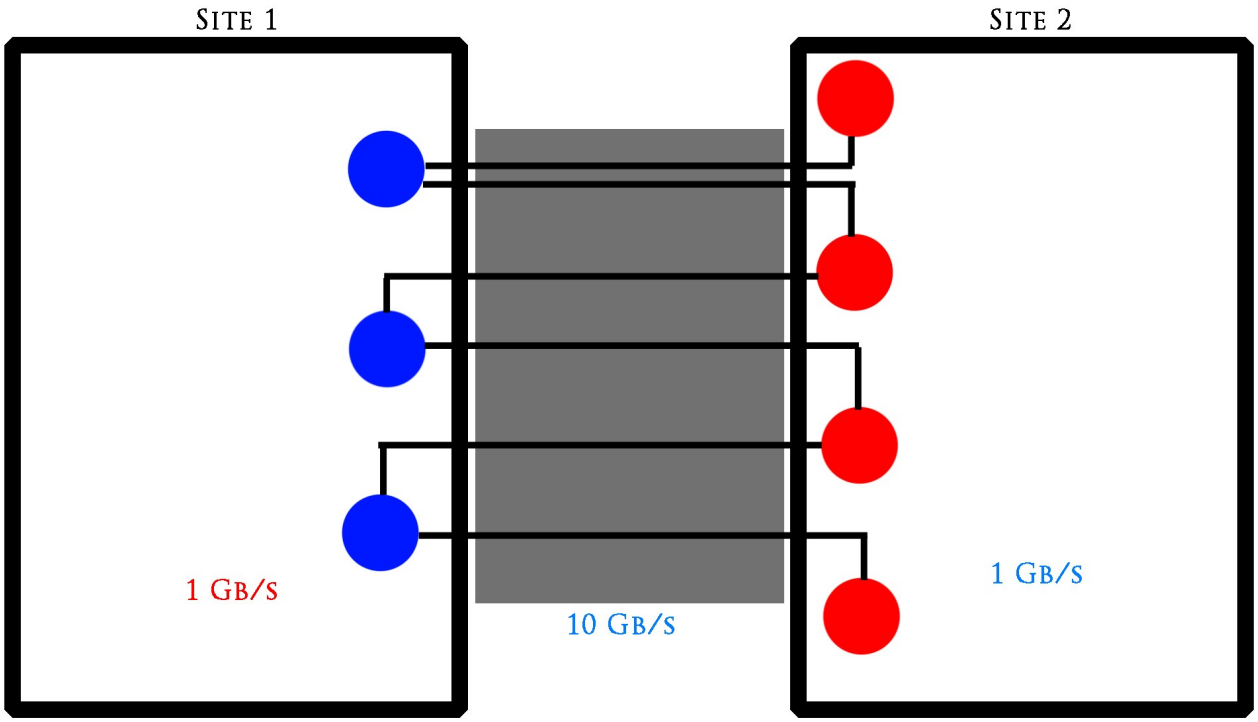
Lorsque nous avons réalisé nos expériences avec une chaîne de diffusion, nous nous étions placés dans le cas le plus favorable. En effet, tous les nœuds impliqués appartenaient au **même site**, et la chaîne pouvait se faire **trivialement** sans la présence de goulots d'étranglement : chaque nœud étant relié à un autre du même site par une connexion à 1 Gb/s, la chaîne pouvait fournir un débit élevé.

Cependant, dans le cas où les nœuds impliqués dans le transfert appartiennent à plusieurs sites différents, l'étape de formation de la chaîne devient primordiale. Les sites ne sont reliés entre eux que par une unique fibre optique, aussi faudra-t-il minimiser les connexions par celle-ci.

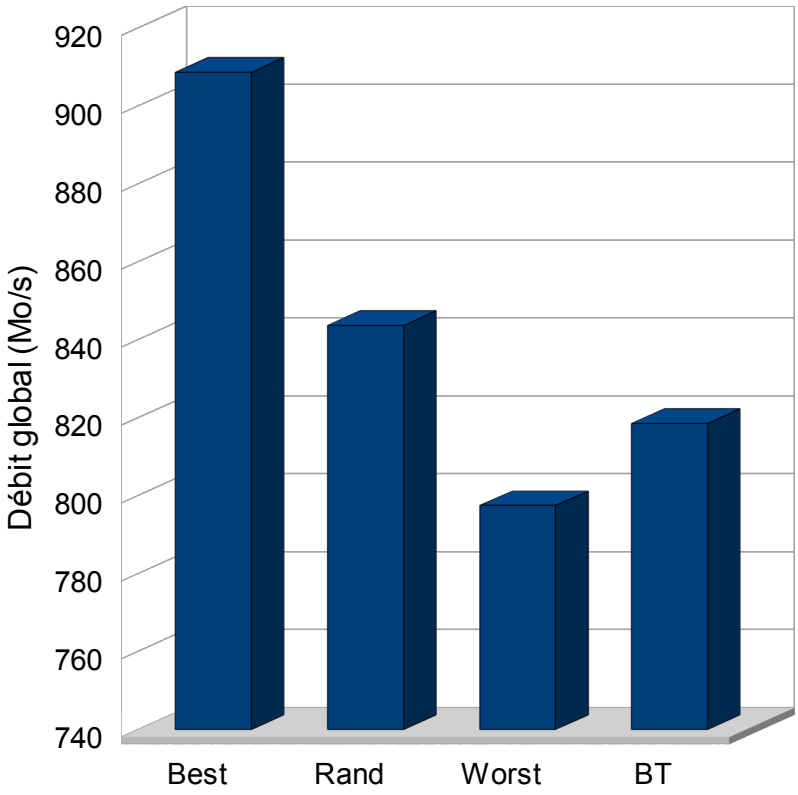
Le cas le plus favorable à Nette est celui où tous les nœuds d'un site sont reliés **entre eux**, et où les deux sites ne sont reliés que par un nœud de chaque (cf. schéma). Dans cette configuration, **un seul** nœud utilise la connexion **entre** les sites, sa bande passante disponible est donc élevé.



Inversement, le cas le plus défavorable à Nette est celui où la chaîne relie chaque nœud à deux nœuds (successeur et prédécesseur) de **l'autre** site. Dans cette configuration, un **grand nombre** de connexions se font par la fibre optique reliant les sites : la bande passante allouée à chaque nœud l'utilisant est donc **d'autant plus faible qu'il y a de nœuds**.



On mesure les débits obtenus avec une chaîne dans **trois** cas de figures (meilleur et pire cas cités précédemment, et cas intermédiaire où le successeur d'un nœud est choisi aléatoirement parmi ceux restants), ainsi qu'avec BitTorrent, pour **deux groupes de 50 nœuds** appartenant respectivement aux sites de Nancy et d'Orsay. Le graphique ci-dessous récapitule les valeurs moyennes des débits obtenus :



Valeurs numériques[2] (débit global en Mo/s) :

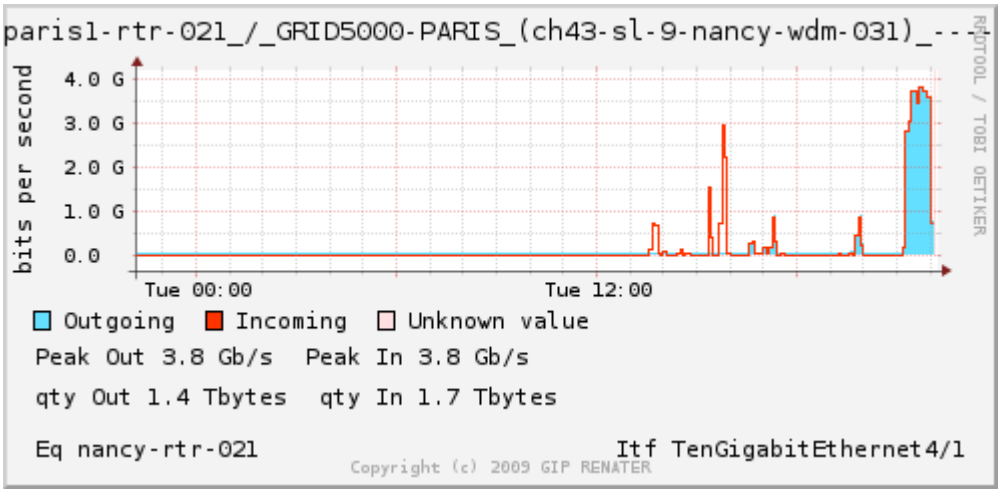
Chaîne - pire cas	Chaîne – aléatoire	Chaîne - meilleur cas	BT	BT – un seul site	Chaîne – un seul site
797.49	843.74	908.64	818.59	~710	~1860

On peut remarquer plusieurs choses :

– une chaîne offre toujours de **meilleures performances** que BitTorrent, mais la différence est bien **moins importante** que lorsque l'on se cantonnait à un seul site. Le pire cas pour une chaîne se révèle moins performant que BitTorrent.

– le débit était meilleur sur un seul site pour la chaîne : en passant à deux, même dans la meilleure configuration possible, on a divisé le débit par deux.

– il a, en revanche, légèrement augmenté pour BitTorrent.



Activité réseau sortant de Nancy

2 Les valeurs des deux dernières cases sont extrapolées des résultats des autres expériences, en supposant la croissance linéaire entre 64 et 128 nœuds

V. Travaux connexes

Actuellement, de nombreuses études existent autour de la diffusion de données sur grille de calcul. Initialement, différents projets se concentraient sur les mouvements de données, leur accès et la gestion des métadonnées sur une grille : GridFTP[6] et GFarm[7] en sont de bons exemples. GridFTP est un protocole destiné à réaliser des transferts sûrs, fiables et offrant de bonnes performances, en passant par la détection d'erreurs, la gestion de sources multiples et les transferts parallèles. GridFarm, quand à lui, est un système de fichiers permettant le traitement en parallèle de données par plusieurs applications.

La gestion de ces métadonnées est un élément clé pour les grilles de données[8]. Bitdew[2] est une API implémentable en Java et supportant les protocoles HTTP, FTP et BitTorrent. Elle est axée vers la gestion de données pour des grilles hautement distribuées (les *Desktop Grids*, utilisant les ordinateurs de particuliers les mettant à disposition), qui diffèrent grandement des grilles traditionnelles dans leur gestion de données. De ce fait, le framework se veut très résistant aux erreurs dues notamment à la volatilité des nœuds ou aux interruptions de transfert, tout en essayant de garantir une meilleure utilisation des ressources réseau en local, via l'utilisation de replicas (une copie dans un cache local du fichier). Leurs tests montrent de meilleures performances avec le framework qu'avec FTP, particulièrement lorsque le nombre de clients augmente, et ce grâce à l'utilisation d'une architecture peer-to-peer. D'autres expériences, réalisées avec le framework GridTorrent ([4], [5]), une implémentation pour les grilles de BitTorrent, montrent une fois encore l'efficacité des *replicas* en local. Le framework propose le *Replica Location Service* pour assurer leur gestion, en centralisant les informations sur les replicas, et peut également se connecter aux serveurs GridFTP qui existeraient sur la grille. Les test, même si réalisés sur un petit nombre de nœuds, montrent l'efficacité supérieure de GridTorrent par rapport à GridFTP pour les cas où la bande passante est limitée ou lorsque le nombre de pairs augmente.

Des chercheurs d'une équipe Microsoft ([3], [9]) développent le *network coding*, qui consiste à envoyer, en plus des pièces initiales du fichier, des combinaisons linéaires de ces pièces. Les différentes combinaisons linéaires de pièces peuvent ensuite être décodées – ce qui implique une plus grande utilisation de ressources pour les clients – pour récupérer les pièces originales du fichier. Cela permettrait *a priori* d'accroître les performances du transfert, mais une comparaison avec nos propres expériences n'est pas possible, du fait des topologies très différentes dans les deux cas. Leur papier n'offre pas non plus de comparaison entre des transferts avec et sans *network coding*, ce qui rend difficile l'estimation de son efficacité.

Différentes approches, comme SplitStream[10] ou Coopnet[11], utilisent des arbres de données, aléatoires ou déterministes. Cependant, l'utilisation d'abres de plus court chemin ne permet pas toujours des performances optimales. SPIDER[12] utilise un algorithme permettant l'établissement de plusieurs arbres reliés à la source initiale. De cette façon, chacun de ces arbres peut être optimisé pour chacun des réseaux locaux qui pourraient être impliqués : il est d'ailleurs possible de rentrer la topologie des ces réseaux en paramètre (autrement, elle sera déterminée dynamiquement). Les résultats expérimentaux donnent des performances quatre fois supérieures à celles de BitTorrent, ce qui est comparable à nos expériences, dans la mesure où une chaîne de données nous donnait des résultats deux fois supérieurs.

Enfin, une étude comparative des différentes méthodes existantes de diffusion de données sur grille a été menée par Samer Al-Kiswany et al. [11], et s'appuie notamment sur quelques différences majeures entre un contexte peer-to-peer *classique* (plusieurs milliers d'utilisateurs, des

réseaux très hétérogènes, une faible bande passante et des fichiers de taille moyenne) et celui de la diffusion de données scientifiques sur une grille (au maximum quelques centaines d'utilisateurs, un volume de données bien plus important et des performances réseaux souvent plus élevées) dans le but de répondre à la question suivante : étant donné les caractéristiques des grilles, quels avantages les solutions peer-to-peer peuvent-elles apporter à la diffusion de données sur celles-ci ? Il en ressort que les avantages du peer-to-peer sur d'autres techniques classiques, et notamment les arbres de diffusion, se font ressentir lorsque la bande passante disponible décroît : typiquement, lorsque le nombre d'utilisateurs augmente. De plus, le peer-to-peer permet d'assurer un meilleur équilibre dans la participation de chaque nœud.

VI. Conclusion

On s'attendait à obtenir de meilleurs résultats avec *BitTorrent* qu'avec des méthodes classiques pour la diffusion de données sur une grille de calcul, partant de l'hypothèse que dans un tel contexte, le grand nombre potentiel de nœuds aurait dû tirer fortement profit de ce protocole. Si *BitTorrent* offre effectivement des performances bien supérieures à celles obtenues avec le *Secure Copy Protocol* (pour un nombre de nœuds de l'ordre de la centaine, le débit moyen est pratiquement 10x supérieur), on reste cependant en deçà des résultats d'un transfert via une simple chaîne de diffusion. Selon toute vraisemblance, l'écart se creusera encore dans le cas où l'on utilisera des arbres, pour peu qu'ils soient intelligemment construits.

Cela peut s'expliquer par plusieurs raisons. Tout d'abord, le protocole *BitTorrent* est très verbeux : un transfert en chaîne de diffusion, au contraire, ne l'est que très peu une fois la chaîne établie. Les multiples connexions d'un nœud au *tracker* ou aux autres pairs introduisent donc une latence qu'on ne retrouvera pas avec une chaîne, menant à un écart de performances.

Le protocole *BitTorrent* est également prévu pour être très efficace sur les réseaux de type *WAN*, où la bande passante sera plus limitée que sur des réseaux locaux, et où l'on peut s'attendre à voir apparaître des goulots d'étranglement. Une chaîne de diffusion, au contraire, tirera fortement profit des hautes performances d'un réseau local. De plus, il n'est initialement pas destiné à réaliser un transfert « unique », mais à maintenir un certain nombre de seeds afin d'augmenter le débit pour différents transferts asynchrones.

Dans le contexte de *Grid5000*, les réseaux reliant les différents sites entre eux et les *switches* interconnectant les nœuds offrent d'excellentes performances en terme de débit, limitant la présence de tels goulots à des cas particuliers pouvant *a priori* être évités. Sur une grille plus distribuée, ça ne serait en revanche plus le cas, et une chaîne de diffusion en pâtirait considérablement plus que *BitTorrent*.

Ainsi, s'il apparaît que les transferts en chaîne de diffusion sont plus adaptés que *BitTorrent* pour la diffusion sur une grille du type *Grid5000*, il n'en sera très probablement pas de même pour une grille de topologie différente. De plus, ce genre de transfert ne permet pas que des nœuds rejoignent le transfert après son démarrage, et est bien moins résistant aux problème liés au départ ou à la défaillance d'un nœud de la chaîne.

Enfin, même en se cantonnant à *Grid5000*, on pourrait s'attendre à voire *BitTorrent* prendre le pas sur la chaîne de diffusion dans le cadre d'un transfert mettant en œuvre un nombre de sites différents suffisamment important.

Pour conclure, on peut se poser la question suivante : quelles performances donnerait un protocole basé sur BitTorrent, mais adapté spécifiquement aux grilles ? En effet, dans ce contexte, on n'a a priori pas à craindre que des clients se contentent de recevoir des données sans contribuer à leur tour : le principe du *tit-for-tat* dans la sélection des pairs à qui transmettre des pièces n'est donc plus pertinent. On pourrait donc modifier ces politiques de sélection de façon à pouvoir tirer parti des meilleurs débit en local, comme le ferait un arbre de diffusion, tout en conservant la robustesse de BitTorrent aux goulots d'étranglement.

VII. Bibliographie

[1] Libtorrent : http://libTorrent.rakshasa.no/

[2] Gilles Fedak, Haiwu He, Franck Cappello *“Bitdew : a programmable environment for large-scale data management and distribution ”*

[3] Christos Gkantsidis, John Miller, Pablo Rodriguez *“Anatomy of a P2P content distribution system with Network Coding”*

[4] Antonis Zissimos, Katerina Doka, Antony Chazapis and Nectarios Koziris *“GridTorrent: Optimizing data transfers in the Grid with collaborative sharing”*

[5] Ali Kaplan, Geoffrey C. Fox, Gregor von Laszewski *“GridTorrent Framework: A High-performance Data Transfer and Data Sharing Framework for Scientific Computing”*

[6] *W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. “The Globus Striped GridFTP Framework and Server”*

[7] *O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi “Proc. of the 2nd IEEE/ACM Symposium on Cluster Computing and the Grid (CCGrid’02)”*

[8] *G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman. “Proceedings of SuperComputing”*

[9] Christos Gkantsidis, John Miller, Pablo Rodriguez *“Anatomy of a P2P Content Distribution system with Network Coding”*

[10] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, *“SplitStream: High-Bandwidth Multicast in Cooperative Environments”*

[11] *Samer Al-Kiswany, Matei Ripeanu,* Adriana Iamnitchi, Sudharshan Vazhkudai *“Beyond Music Sharing: An Evaluation of Peer-to-Peer Data Dissemination Techniques in Large Scientific Collaborations”*

[12] *Ganguly, S., Saxena, A., Bhatnagar, S., Banerjee, S.,* et al. *"Fast replication in content distribution overlays"*