Rapport :
La dissémination de données.

Sommaire :

Introduction

L'approche pseudo interactive

Avec l'usage croissant des télécommunications sans fils, en particulier des téléphones mobiles, on assiste à une nouvelle donne dans la diffusion d'information : des serveurs sont amenés à fournir de l'information à un parc d'utilisateurs très important alors que le retour d'information est faible (ne serait-ce que parce que le débit d'envoi est bien moins important que le débit reçu). On parle alors d'environnements asymétriques. Dans ce type d'environnement, l'approche interactive atteint ses limites : les temps de diffusion sont trop longs et surtout, le serveur ne peut plus gérer l'afflux de clients trop important. Pour pallier à ces inconvénients, une nouvelle approche a fait jour : la méthode pseudo-interactive. Le serveur diffuse les messages dans un ou plusieurs canaux réservés et le client se connecte à ces canaux et attend que le message qu'il a demandé soit diffusé. Le client a seulement l'illusion de l'interactivité, d'où le nom.

Ce type d'approche, en plus de garantir un temps d'attente moyen constant, a d'autres avantages : dans le cas des systèmes portables, où l'économie d'énergie est importante, un index des prochaines diffusions à venir permet au portable de se mettre en veille, puis de se reconnecter seulement quand le message qui l'intéresse sera diffusé. D'autre part, l'appareil de réception peut gérer un profil client de son utilisateur et télécharger les messages diffusés susceptibles de l'intéresser avant même qu'il ne l'ait demandé.

Mais surtout, cette méthode appliquée à l'Internet est l'une des plus prometteuses pour réduire la saturation du réseau, liée à l'augmentation exponentielle du nombre d'utilisateurs. En effet, seul un petit nombre des informations disponibles est demandé par la plupart des requêtes. Ainsi, pour le serveur cs-www.bu.edu, 0.5% des pages sont demandées par 61% des requêtes, et 10% par 91% des requêtes. Utiliser l'approche pseudo-interactive pour diffuser le petit nombre de messages les plus populaires permettrait à la fois de gagner de la bande passante et de libérer le serveur du traitement d'un grand nombre de requêtes.

Enfin, en ajoutant des coûts de diffusion pour chaque message, la dissémination de données permet de traiter d'autres modèles, comme le problème de la maintenance de machines ou le réapprovisionnement des stocks. Dans la maintenance de machines, on dispose d'un certain nombre de techniciens (les canaux) qui réparent suivant un planning fixé les machines (les messages) dont la probabilité instantanée de défaillance augmente au cours du temps. Dans l'approvisionnement de stocks, on dispose d'un certains nombre de camions (les canaux) qui approvisionnent des stocks (les messages), de manière à minimiser le coût de l'exploitation et l'attente des clients.

Approche du TIPE

Dans ce rapport, nous nous contenterons de l'étude des messages ayant des longueurs uniformes. En effet lorsque les messages sont de longueurs différentes, le problème est nettement plus difficile. En particulier, il n'existe par forcément d'optimal périodique ; des ordonnancements optimaux peuvent présenter des trous (dans lesquels aucun message n'est diffusé), et ce même sans coûts de diffusion. Pour une étude détaillée de ce cas, on pourra consulter [Sch00].

Dans la première partie, nous commençons par établir le coût d'un ordonnancement, ce qui nous permettra de trouver un minorant du coût de tout ordonnancement périodique (il s'agit du minorant d'Ammar et Wong étendu au cas de plusieurs canaux avec coûts de diffusion). Le minorant nous indiquant les fréquences optimales, il nous permettra de décrire plusieurs algorithmes de dissémination de données.

Dans la deuxième partie, nous étudions la performance en pratique des algorithmes donnés ci-dessus. En fait, nous ne nous intéressons pas à la mise en place pratique de la dissémination de données mais nous évaluons l'efficacité intrinsèque de ces algorithmes suivant les différentes distributions de popularité et d'autres contraintes (ainsi les algorithmes produisant

un ordonnancement périodique favorisent la possibilité d'émission d'index des prochains messages diffusés, afin de réaliser des économies d'énergie). C'est pourquoi nous avons écrit un programme qui simule un serveur ayant à traiter les requêtes de clients se loguant en un temps continu. Le serveur a le choix entre l'approche interactive ou l'un des algorithmes issus de l'approche pseudo-interactive. La simulation retourne ensuite le temps d'attente moyen des clients. Le code source de note simulation est donné en appendice.

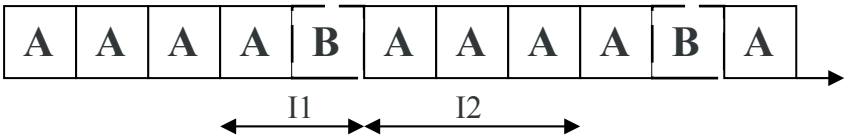
1ère partie : Présentation des résultats théoriques.

1.1-Exemple

Nous commençons par présenter un exemple, afin de clarifier les notions. Nous avons deux messages A et B de même longueur à diffuser sur un canal, sachant que A est quatre fois plus populaire que B. Quel ordonnancement choisir pour minimiser le temps d'attente des clients ? A priori, l'ordonnancement optimal diffuse plus souvent A que B… Examinons deux cas.

Première idée

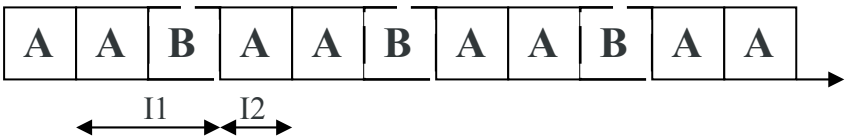
Diffuser A quatre fois plus que B.



Une requête pour B attend $5/2=2.5s$ plus le temps de téléchargement de B soit au total 3.5 secondes. Une requête pour A tombe dans I1 avec une probabilité de $2/5$ et attend $2/2+1=2s$ et tombe dans I2 avec une probabilité de $3/5$ et attend $1/2+1=1.5s$ soit au total $1.7s$. Le temps d'attente moyen est donc de $0.8\times1.7+0.2\times3.5=2.06s$.

Seconde idée

Diffuser A $2=\sqrt{4}$ fois plus que B.



On a de même un temps d'attente pour B de $1+1.5=2.5s$ et un temps d'attente pour A de $1+2/3\times1+1/3\times0.5=1.83s$. Cela nous donne un temps d'attente moyen de $0.8\times1.83+0.2\times2.5=1.97s$. Nous verrons que c'est le temps d'attente optimal : il faut diffuser les messages proportionnellement à la racine carré de leur popularité.

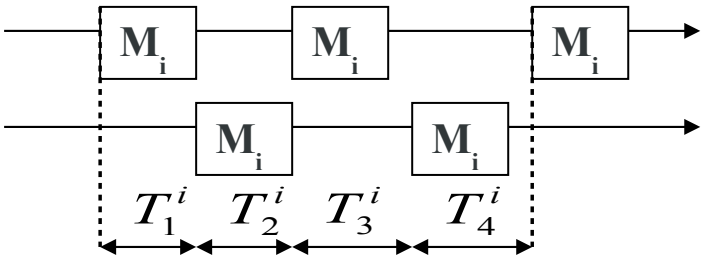
1.2-Calcul du temps d'attente moyen

Données du problème

On a un ensemble de m messages $\{M_1,\ldots M_i,\ldots M_m\}$ de longueurs uniformes à diffuser sur W canaux. Chaque message M_i a une popularité p_i et un coût de diffusion c_i. On définit un ordonnancement par un ensemble de W suites $\{s_1,\ldots s_w\}$ de messages. Si $s_i(n)=M_j$, le message M_j est diffusé au temps n dans le $i^{ème}$ canal. Si $s_i(n)=X$, aucun message n'est diffusé dans le canal i au temps n. L'unité de temps est ici le temps de téléchargement d'un message.

On définit le temps de service TS d'un ordonnancement par le temps d'attente moyen. Si $TS(M_i)$ est le temps d'attente moyen pour une requête du message M_i, alors $TS = \sum_{i=1}^{m} TS(M_i)$. On définit le coût de diffusion C_DIFF par le coût moyen de diffusion des messages. On peut alors définir le coût d'un ordonnancement par COUT=TS+C_DIFF.

On se restreint aux ordonnancements périodiques (en fait, nous montrerons qu'il existe toujours un ordonnancement périodique optimal, donc cette hypothèse ne nuît pas à la généralité). Nous allons introduire quelques notations. Considérons un ordonnancement périodique de période T.



On définit :

- n_i est le nombre de diffusions de M_i durant la période.
- T_1^i est l'intervalle de temps qui sépare le début de la période de la première diffusion de M_i. $T_j^i, 2 \le j \le n_i$ est l'intervalle de temps qui sépare les débuts de la $(j-1)^{ième}$ et la $j^{iéme}$ diffusion de M_i.

Nous allons exprimer le coût d'un ordonnancement en fonction des n_i et des T_j^i :

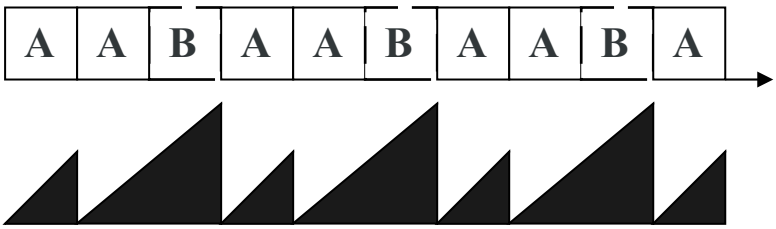
Lemme 1.1 - Soit S un ordonnancement périodique de période T. Alors le coût de S est

$$1 + \frac{1}{T}\sum_{i=1}^{m}\left(p_i \sum_{j=1}^{n_i} \frac{(T_j^i)^2}{2} + n_i c_i \right)$$

Preuve :

Calculons $TS(M_i)$: une requête pour M_i à une probabilité de T_j^i / T de tomber dans l'intervalle T_j^i et attend en moyenne le début du message $T_j^i / 2$ secondes. On obtient donc $TS(M_i) = 1 + \sum_{j=1}^{n_i} (T_j^i)^2 / (2T)$ en ajoutant le temps de téléchargement. De plus, on a directement $C\_DIFF(M_i) = n_i c_i / T$. La formule en découle, puisque $COUT(S) = \sum_{i=1}^{m} p_i [TS(M_i) + C\_DIFF(M_i)]$ ف

On a alors une représentation visuelle simple de $TS(M_i)$ comme aire moyenne de triangles isocèles :



On voit que $TS(A) = 1 + \frac{1}{3}\left(\frac{1^2}{2} + \frac{2^2}{2} \right) = 1.83s$.

1.3-Calcul du minorant

Le résultat précédent va nous permettre de calculer un minorant du coût de tout ordonnancement périodique.

Théorème 1.1 :

Soit $Min1 : \min\left(1+\sum_{i=1}^{m}\left(\frac{p_i\tau_i}{2}+\frac{c_i}{\tau_i}\right)\right)$ $sur\ le\ domaine \begin{cases} \forall i\ \tau_i \geq 1 \\ \sum_{i=1}^{m}\frac{1}{\tau_i} \leq W \end{cases}$

*Min*1 est un minorant de tout ordonnancement périodique et est atteint pour un unique τ^*. Un ordonnancement périodique S a un coût égal à ce minorant ssi il diffuse M_i tous les τ_i^*.

Preuve :

Nous allons d'abord minimiser $TS(M_i)$ (en relaxant la contrainte de non recouvrement des messages). On a $TS(M_i) = 1 + \sum_{j=1}^{n_i}(T_j^i)^2/(2T)$ avec $\sum_{j=1}^{n_i} T_j^i = T$. Comme la fonction x-->x² est strictement convexe, $TS(M_i)$ est minimisé ssi les T_j^i ont même longueur : T/n_i. Posons $\tau_i = T/n_i$. Alors $TS(M_i) \geq 1 + \tau_i/2$. Donc $TS \geq 1 + \sum_{i=1}^{m}\left(\frac{p_i\tau_i}{2}+\frac{c_i}{\tau_i}\right)$ puisque $\frac{c_i n_i}{T} = \frac{c_i}{\tau_i}$.

De plus $n_i \leq T$ donc $\forall i\ \tau_i \geq 1$ et $\sum_{i=1}^{m} n_i \leq WT$ donc $\sum_{i=1}^{m} 1/\tau_i \leq W$, ce qui nous donne le domaine de minimisation. Enfin, la fonction objectif est continue sur un domaine fermé et tend vers l'infini quand $||\tau||$ tend vers l'infini donc elle admet un minimum (les fermés bornés sont compacts en dimension finie). De plus elle est strictement convexe sur le domaine convexe définit par les contraintes donc ce minimum est unique. Le reste en découle alors facilement ڤ

Il ne reste plus qu'à calculer τ^*, c'est l'objet du théorème qui suit.

Théorème 1.2 :

Soit $\tau_i' = \sqrt{\frac{2c_i + \lambda}{p_i}}$ avec λ défini par :

- si $\sum_{i=1}^{m}\sqrt{p_i/(2c_i)} \leq W$ alors λ=0

- sinon λ est l'unique solution positive de $\sum_{i=1}^{m} \sqrt{p_i/(2c_i+\lambda)} = W$

Alors pour tout 1≤i≤m tel que $\tau_i' < 1$, alors $\tau_i^* = 1$. On itère alors les calculs (en enlevant les messages et le nombre de canaux correspondants) jusqu'à ce que pour tout i, $\tau_i' \geq 1$. Alors $\tau_i^* = \tau_i'$.

Preuve :

Cf Appendice B.

On obtient bien une fréquence de diffusion proportionnelle à la racine carrée de la popularité quand les coûts de diffusion sont nuls.

1.4-Présentation des algorithmes

Le minorant nous indique des fréquences de diffusion de $1/(W\tau_i)$. Ceci nous amène à présenter quelques algorithmes. En fait, trouver l'optimal étant NP-dur [Sch00], les algorithmes que nous présentons sont des α-approximations. Il se peut que $\sum_{i=1}^{m} 1/\tau_i^*$ soit plus petit que W. Dans ce cas on rajoute un message fantôme M_0 avec $p_0=c_0=0$ et $1/\tau_0^* = W - \sum_{i=1}^{m} 1/\tau_i^*$.

Algorithme randomisé

On diffuse le message M_i avec une probabilité $s_i = 1/(W\tau_i^*)$. On remarque que l'on a $\sum_{i=0}^{m} s_i = 1$ et qu'avec une probabilité de $(1 - \sum_{i=1}^{m} s_i)$ on ne diffuse rien.

Alors l'espérance de $TS(M_i)$ est de $1 + 1/2 + \sum_{n\geq 1} (1 - Ws_i)^n = 1/2 + \tau_i^*$ donc l'espérance de TS est $1/2 + \sum_{i=1}^{m} p_i\tau_i^*$. De même on a l'espérance de C_DIFF égale à $\sum_{i=1}^{m} c_i s_i W = \sum_{i=1}^{m} c_i/\tau_i^*$. Donc l'espérance du coût de cet algorithme est plus petit que $2*Min_1$. L'algorithme est donc une 2-approximation.

Glouton

L'idée est de diffuser le message dont l'attente cumulée est maximum. Si on note σ_i le temps qui s'est écoulé depuis le début de la dernière diffusion du message M_i, le glouton diffuse le message Mi tel que $p_i\sigma_i\tau_i^* - c_i$ soit maximal.

On peut démontrer que cet algorithme est la dérandomatisation gloutonne de l'algorithme précédent. Il donnera ainsi toujours de meilleurs résultats que celui-ci. C'est donc une 2-approximation [Sch00].

Nombre d'or

L'idée est de construire un ordonnancement périodique qui respecte les proportions $1/W\tau_i^*$ des messages définis par le minorant. Pour cela on va se servir d'une propriété du nombre d'or :

$\varphi = \frac{1+\sqrt{5}}{2}$ est l'irrationnel dont les multiples modulo 1 découpent [0,1] le plus uniformément possible.

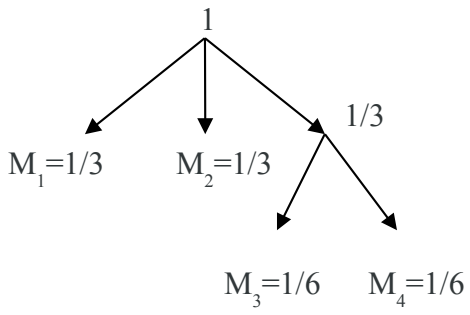
Soit T la longueur de l'ordonnancement périodique à construire, $n_i = \lfloor T/\tau_i^* \rfloor$ le nombre de diffusion de M_i pendant cet intervalle. On attribue à M_i un ensemble $X_i = \{(n_0 + n_{i-1} + j)\varphi \bmod 1, 1 \leq j \leq n_i\}$ de n_i position puis on trie $X_1 \cup \ldots \cup X_m$ (dans l'ordre croissant). On diffuse alors dans le $n^{ième}$ créneau le message M_i tel que la $n^{ième}$ position appartienne à X_i.

L'algorithme est une 9/8 approximation (lorsqu'il n'y a pas de coût de diffusion).

Méthode par arbres :

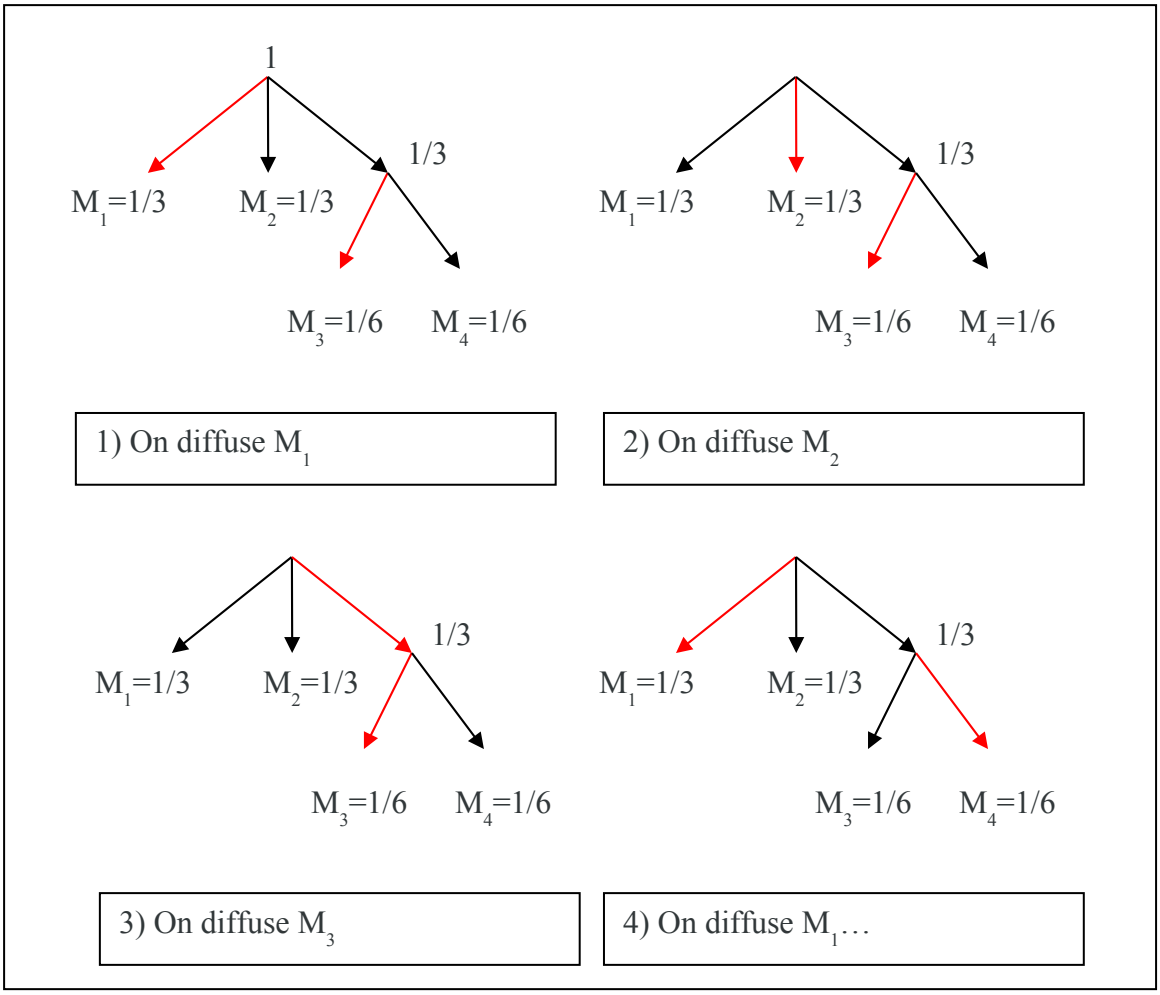
Une autre manière de respecter **exactement** les fréquences en $1/W\tau_i$* des messages est de passer par des arbres. A chaque nœud on associe une probabilité de diffusion *p* (la racine étant affectée d'une probabilité 1). Si il y a n branches partant de ce nœud, on associe une probabilité à chacun des nœuds issus de ces branches de *p/n*.

Ex : pour une distribution de fréquence $M_1=1/3$ $M_2=1/3$ $M_3=1/6$ $M_4=1/6$, on peut associer l'arbre :



Il est alors facile de diffuser les messages en respectant les proportions de fréquence en coloriant les branches. Au départ, pour chaque nœud, on colorie la branche la plus à gauche. Puis on parcourt l'arbre en suivant les branches coloriées et à chaque fois que l'on parcourt une telle branche on gomme la couleur et on colore la branche adjacente. Arrivée au bout on diffuse le message que l'on a atteint.

Ex : sur l'arbre précédent :



On obtient ainsi comme ordre de diffusion : $M_1M_2M_3M_1M_2M_4M_1M_2M_3$…

Mais il n'est pas toujours possible de représenter une distribution de fréquence par un arbre. L'idée est de se ramener à la fréquence en $1/2^n$ immédiatement inférieure. On peut alors se obtenir un arbre modélisant cette distribution de fréquence (cf appendice B).

1.5-Un algorithme dynamique

Tout l'intérêt de la dissémination de données vient du fait que les algorithmes ne dépendent pas des utilisateurs connectés mais seulement de la distribution des popularités des messages. Toutefois, d'un point de vue pratique, il faut obtenir cette distribution de popularité et donc continuer à traiter les connexions clients (en faisant une enquête d'opinion, en recueillant un échantillon…). De plus, l'inconvénient majeur d'une telle approche est que la distribution de popularité peut changer au cours du temps. Par exemple, le matin, où les gens travaillent, les pages Web les plus demandées seront celles traitant de la bourse, de l'économie… Tandis que le soir, ce seront plutôt les pages de loisirs , comme le cinéma, ou le résultat des matchs de football , qui seront privilégiées.

On conçoit alors l'intérêt d'un algorithme dynamique, c'est-à-dire d'un algorithme qui s'adapte à ce changement de popularité. C'est le cas de l'algorithme usuel qui met les requêtes des utilisateurs en file d'attente, et c'est bien son seul avantage face aux algorithmes de distribution de données précédents (cf la simulation dans la partie suivante). Est-il possible de trouver un algorithme qui combine les avantages des deux méthodes ?

La réponse est positive. Nous partons de l'algorithme glouton. L'idée du glouton est de diffuser le message dont l'attente en arrière cumulée est maximale. Si on pose avec les notations habituelles : p_i la probabilité de diffusion du message M_i, ci son coût de diffusion, σ_i le temps qui s'est écoulé depuis la dernière diffusion de ce message et τ_i* l'intervalle de diffusion optimum du message Mi, alors le glouton diffuse le message tel que $p_i\sigma_i\tau_i$*-c_i soit maximum. L'idée est d'obtenir un algorithme qui aboutisse au même résultat sans passer par les p_i ni par les τ_i* (qui dépendent des p_i).

En fait, lorsqu'il n'y a pas de coûts de diffusion, τ_i* est proportionnel à $1/\sqrt{p_i}$, donc le glouton diffuse le message tel que $\sqrt{p_i}\sigma_i$ soit maximal. Or supposons qu'en moyenne N utilisateurs se connectent par unité de temps. A chaque créneau, N*p_i demandent le message p_i. Si la dernière diffusion de M_i remonte à σ_i, alors le temps d'attente cumulée de l'ensemble des clients pour le message M_i est de (en n'oubliant pas que les connexions peuvent arriver à n'importe quel instant du créneau):

$$N * p_i \left[(\sigma_i - 1/2) + (\sigma_i - 1/2 - 1) + (\sigma_i - 1/2 - 2) + ...\right] = N * p_i * \left[\frac{\sigma_i(\sigma_i + 1)}{2} - \frac{\sigma_i}{2}\right] = \frac{N * p_i * \sigma_i^2}{2}$$

On voit donc que diffuser le message tel que la somme des temps d'attentes des clients est maximale revient à diffuser le message tel que $\sqrt{p_i}\sigma_i$ est maximal.

Ainsi, notre glouton dynamique fait ce qu'il y a de plus naturel, il diffuse le message qui a « le maximum d'attente cumulée ».

2ème partie : Résultats de la simulation.

2.1-Utilité d'une simulation

Nous avons donc plusieurs algorithmes à notre disposition. Il reste à déterminer lequel utiliser dans quelle situation. Par ailleurs, lorsqu'on a le choix d'autres stratégies (on songe ici surtout à Internet), il nous faut savoir quand opter pour la dissémination de données se révèlera payant. Pour cela, nous avons besoin d'une simulation qui, en fonction du nombre de messages, de leur popularité, du nombre de clients à servir, du nombre de canaux à disposition, nous donne le temps d'attente moyen par clients. C'est cela que réalise notre programme, dont le code source est donné en annexe.

La principale difficulté de ce programme ne réside pas en la programmation des algorithmes (pour ceux-ci le point délicat est le calcul des τ_i*, c'est-à-dire de λ, mais pour cela nous utilisons la méthode des tangentes de Newton), mais bien en la simulation proprement dite. En effet, la méthode retenue consiste à considérer que les clients se loguent en un temps continu pendant un certain intervalle de temps tandis que le serveur diffuse des messages sur les canaux à disposition, le programme renvoyant le temps d'attente moyen par clients. Pour obtenir des résultats cohérents, il importe que la simulation se déroule sur un intervalle de temps assez long, ce qui implique d'optimiser au maximum cette partie du programme. On trouvera dans le code source du programme une explication plus détaillée des difficultés rencontrées et des choix effectués.

Nous aimerions insister sur l'utilité d'une telle simulation. En effet, l'idée de la dissémination de données remonte à une vingtaine d'années (à 1983 exactement) et fut utilisée à l'époque dans des systèmes comme le télétexte. Il semble maintenant que la dissémination de données soit parvenue à une certaine maturation, en particulier lorsque les messages sont de longueur uniforme (surtout depuis la présentation dans [Sch00] d'un schéma

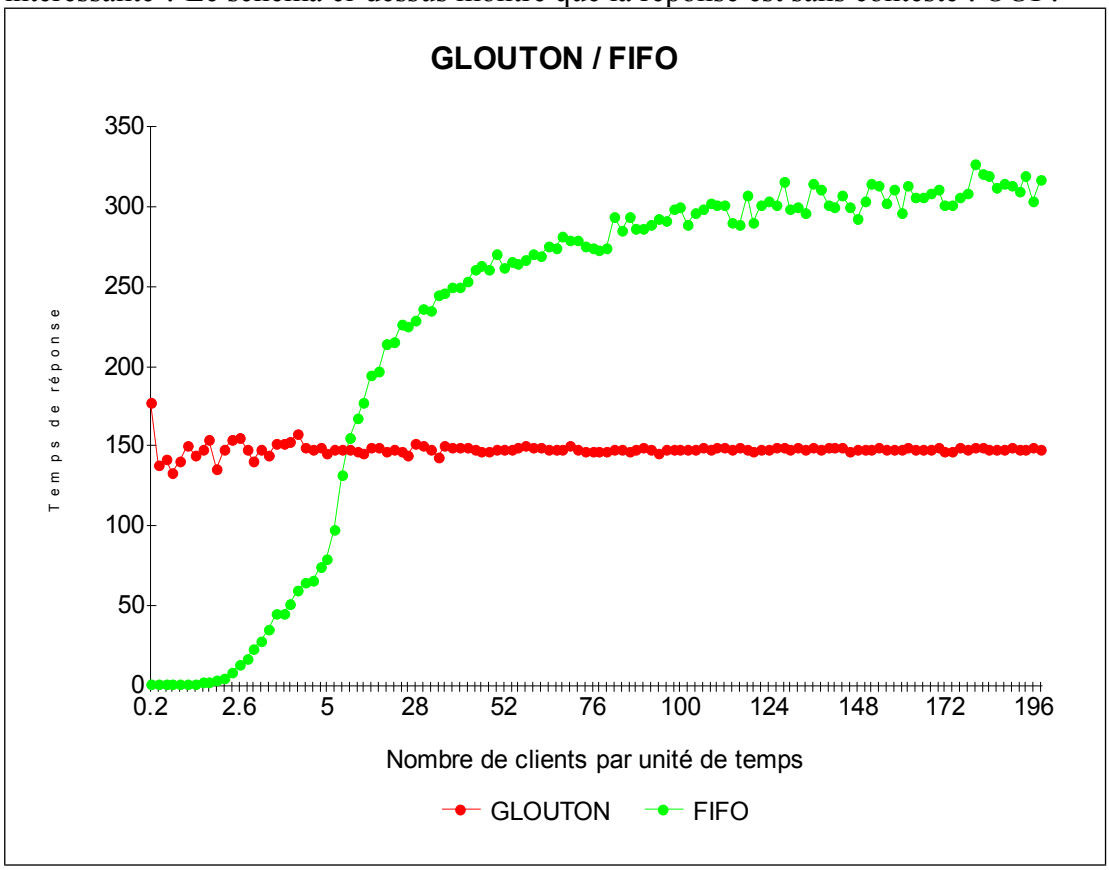
d'approximation uniforme pour ce cas). Des applications de cette méthode sur Internet se sont développées, comme dans Pointcast (qui ne la réalise en fait qu'en partie). On pourra consulter [BC96] et [FZ97] pour une étude détaillée des difficultés de l'implémentation de cette méthode à Internet.

Les résultats de la simulation que nous présentons ont un double but : montrer l'intérêt que peut avoir cette approche, et, dans un deuxième temps, savoir quels algorithmes il vaut mieux utiliser dans quel cas. La partie qui suit donne les résultats les plus significatifs que nous avons obtenus (et qui recoupent les résultats d'autres simulations, par exemple celles de [FZ97]). D'autres résultats, moins important pour notre sujet, seront présentés en annexe.

2.2-Principales données recueillies par la simulation

Comparaison entre l'approche pseudo-interactive et l'approche usuelle

C'est bien sûr la principale interrogation : est-ce que l'approche pseudo-interactive est intéressante ? Le schéma ci-dessus montre que la réponse est sans conteste : OUI !



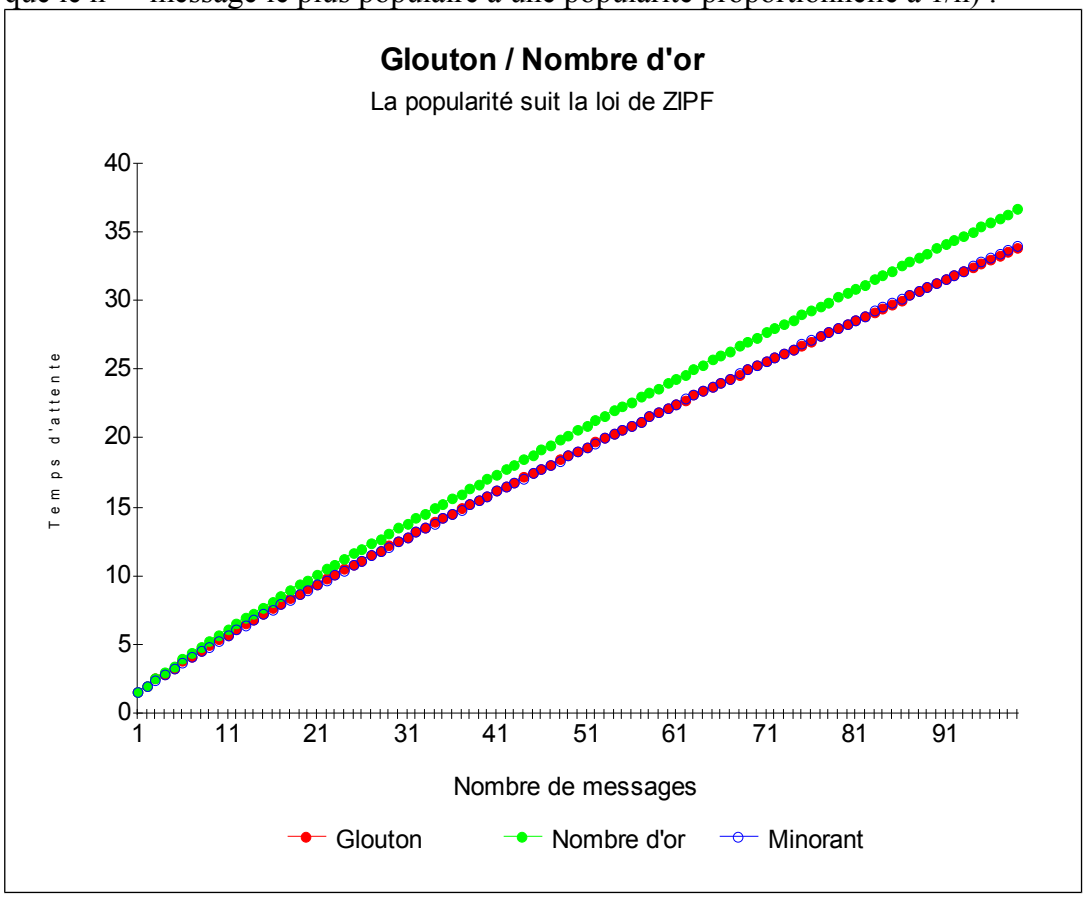
La simulation s'est effectuée sur un ensemble de 1000 messages à diffuser sur un canal. On remarque que le temps d'attente moyen pour l'approche pseudo-interactive reste constant quelque soit le nombre de visiteurs tandis que pour l'approche interactive habituelle, si le temps d'attente est quasiment nul lorsqu'il y a peu de clients connectés au site, il croit très vite avec le nombre de clients pour se stabiliser à une valeur d'attente bien supérieure à celle du glouton. (La forme en S s'expliquant qu'à la fin, tous les messages sont dans la liste d'attente, ce qui fait que l'algorithme par FIFO diffuse les messages cycliquement et donc tend vers un temps d'attente de 1000/2=500).

On voit ici tout l'intérêt de l'approche pseudo-interactive : puisque le temps d'attente ne varie pas avec le nombre de clients, on pourra utiliser cette méthode pour diffuser les

messages les plus populaires, en utilisant la méthode par FIFO usuelle seulement pour les messages les moins demandés.

Comparaison entre le glouton et l'algorithme du nombre d'or

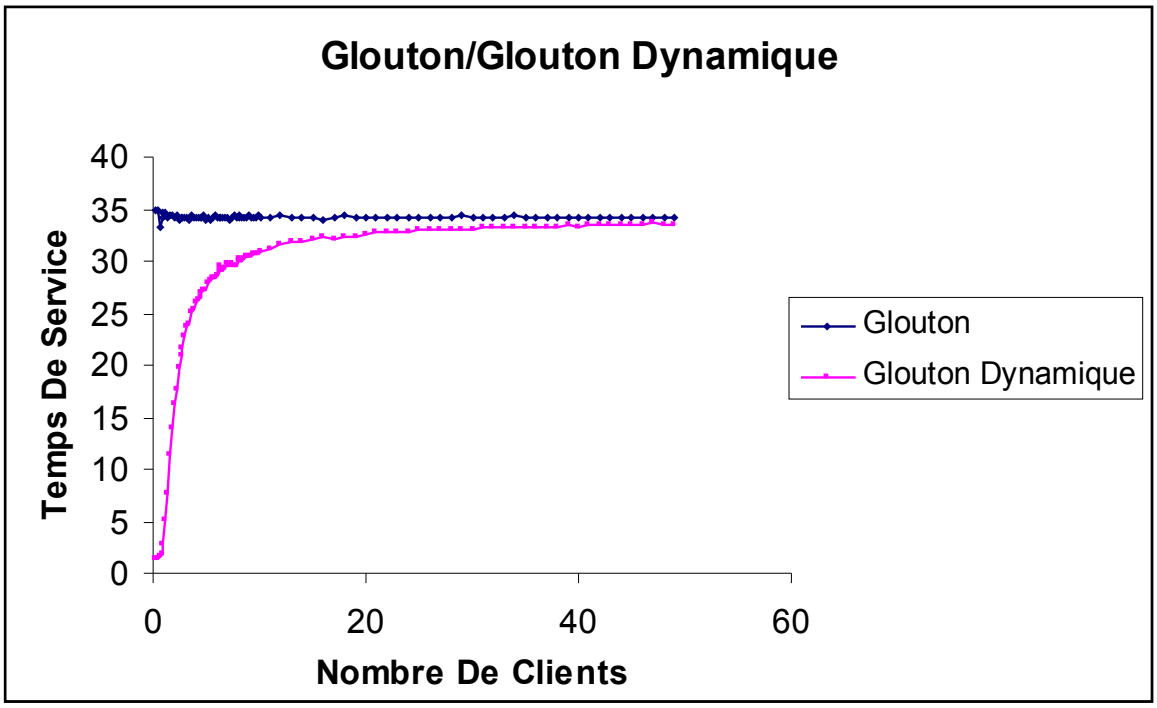
On a vu l'utilité de la dissémination de données par rapport à une approche interactive. Reste à savoir quel algorithme de dissémination de données utiliser. On sait que le glouton donne toujours de meilleurs résultats que l'algorithme randomisé. Reste à comparer le glouton avec celui du nombre d'or. Or l'algorithme du nombre d'or est une 9/8-approximation tandis que le glouton n'est qu'une 2-approximation. On a donc une meilleure borne pour l'algorithme du nombre d'or, mais cela ne veut pas dire qu'il donnera de meilleurs résultats en pratique. Voici le résultat de la simulation lorsque la popularité des messages suit la loi de ZIPF (c'est-à-dire que le $n^{ième}$ message le plus populaire a une popularité proportionnelle à 1/n) :



La simulation s'est effectuée sur 1 canal, la période de l'algorithme du nombre d'or étant de 1000*le nombre de messages. On constate que lorsque la popularité suit la loi de ZIPF, le glouton donne de meilleurs résultats, très proches de l'optimal. Or il se trouve que cette loi modélise bien le comportement des internautes [BC96]. Donc pour une application de la dissémination de données à Internet, on pourra préférer le glouton.

Comparaison entre le glouton et le glouton dynamique :

Nous avons construit le glouton dynamique dans l'espoir qu'il réunisse à la fois les avantages de la méthode habituelle par files d'attentes (à savoir un temps de réponse très faible quand le nombre de clients est peu important) et ceux de la dissémination de données (par exemple obtenir un temps d'attente aussi bon que celui du glouton lorsque le nombre de clients est important). Voici les résultats de notre simulation (on avait 100 messages à diffuser sur un canal).



On voit donc que le glouton dynamique remplit bien toutes nos attentes. Il donne même toujours de meilleurs résultats que le glouton, même quand le nombre de clients est important. Toutefois, le glouton dynamique n'est pas un algorithme de dissémination de données et ne possède pas les deux autres avantages de la dissémination de données, à savoir :

- le glouton dynamique oblige le serveur à traiter les requêtes des clients, utilisant des ressources.
- pour la même raison, on ne peut savoir à l'avance quel message le serveur va diffuser, on ne peut donc pas réaliser un index de diffusion en vue de faire des économies d'énergies.

On aura donc le choix entre la dissémination de données si on privilégie la libération des ressources du serveur ou le glouton dynamique si on privilégie la rapidité de service.

Remarque :
Nous n'avons donné ici que les résultats les plus significatifs. Le lecteur intéressé pourra consulter l'appendice C qui regroupe d'autres résultats de notre simulation.

2.3- Synthèse

Rappelons les avantages de la dissémination de données. D'abord, le serveur diffuse les informations sans avoir à traiter les requêtes utilisateurs, ce qui permet de libérer des ressources. Ensuite, le principe même de la dissémination de données est de diffuser un certain nombre de messages populaires sur un petit nombre de canaux, ce qui permet de libérer de la bande passante. Enfin, et surtout, le temps de service ne dépend que de l'ordonnancement choisi et pas du nombre de clients. Ceci permet de garantir un temps de service constant quel que soit la situation, et en particulier cela permet d'éviter les si frustrants « Denial of Service »…

La simulation montre que le temps de service est tout à fait satisfaisant et donne de (bien) meilleurs résultats que la méthode habituelle dès que le serveur a à traiter un certains nombre de requêtes. On pourrait donc envisager un système hybride, utilisant les algorithmes de

dissémination de données pour les messages les plus populaires, et la méthode habituelle pour les autres.

Conclusion

Nous espérons qu'après cette brève introduction à la dissémination de données, son utilité ne fait plus aucun doute. Nous avons montré, tout au long de la deuxième partie, que la dissémination de données pouvait résoudre un grand nombre de problèmes, de la réduction du temps d'attente sur Internet, à l'économie d'énergie pour les mobiles. Nous avons détaillé à chaque fois les avantages et défauts des algorithmes à notre disposition pour chaque cas, afin de permettre aux centres de diffusion d'informations de choisir l'algorithme utilisé en toute connaissance de cause. Enfin, rappelons que la dissémination de données et la seule méthode envisageable lorsque les communications se déroulent dans un environnement très asymétrique.

Notre présentation ne traite que des messages de longueurs uniformes, alors que c'est évidemment rarement le cas en pratique. Mais du point de vue théorique, les connaissances sont encore limitées dans le cas non uniforme, ce qui explique que nous n'ayons pas voulu développer ce sujet du point de vue expérimental (il est toujours plus satisfaisant, pour la conception d'algorithmes et le test de leur efficacité, de pouvoir s'appuyer sur la théorie, afin de ne pas tomber dans l'empirisme). Toujours dans le contexte non-uniforme, la mise en application en pratique de ce système, en particulier à Internet, reste délicate. C'est pourquoi la dissémination de donnée y est encore peu répandue.

Malgré cela, par son grand potentiel qui n'est pas encore totalement exploré, ses nombreuses utilisations possibles, nous pouvons sans craintes affirmer que la dissémination de données sera au cœur du futur de la diffusion d'information.

Appendices :

Description des appendices:

Dans ce rapport, nous n'avons présenté que les résultats les plus directement en rapport avec la dissémination de données, en omettant certains aspects intéressants. Nous remédions (partiellement) à cela dans ces appendices, en particulier nous donnons la preuve complète des théorèmes que nous avons énoncés.

En particulier :

- Dans l'appendice A, nous prouvons que l'hypothèse que nous avons fait dès le début est justifiée : il existe toujours un ordonnancement périodique optimal. Au passage, la preuve nous fournira un algorithme (non polynomial) donnant l'ordonnancement optimal.
- Dans l'appendice B, nous donnons la preuve de certains théorèmes que nous avons utilisé (dont la preuve sortait légèrement du sujet, c'est pourquoi nous ne la donnons qu'en appendice)
- Dans l'appendice C, nous donnons d'autres résultats de notre programme (toujours par souci de résumer, nous n'avions donné que les résultats les plus significatifs)
- Dans l'appendice D, nous donnons le code source complet et commenté de notre programme
- Enfin, l'appendice E consiste en un bref historique des résultats obtenus sur la dissémination de données et donne notre bibliographie.

Appendice A : Existence d'un ordonnancement périodique optimal.

Le minorant de la première partie, qui est à la base de tous nos algorithmes, utilise le fait qu'il existe toujours un ordonnancement périodique optimal. C'est ce que nous montrons dans cet appendice, après avoir modélisé les ordonnancements par des graphes, et réduit le nombre d'états possibles.

A.1-Construction d'un graphe d'états

Nous allons construire un isomorphisme entre les distributions de messages et des graphes d'états. Nous montrerons ensuite comment l'étude de ces graphes permet de prouver qu'il existe toujours un ordonnancement optimal périodique.

Afin de ne pas alourdir les notations, nous considérons seulement le cas d'un canal. L'extension aux cas de plusieurs canaux se fait sans difficulté. Les sommets du graphe sont des états $\vec{\sigma} = (\sigma_1, ...\sigma_i, ...\sigma_m) \in (N^*)^m$. Être au sommet $\vec{\sigma}$ à l'instant T signifie que la dernière diffusion du message i a commencé à la date T-σ_i (c'est donc l'attente en arrière du message i). Les arrêtes sont de deux types : soit on diffuse le message M_i à l'instant T et on passe alors de l'état (σ_1,…σ_i,…σ_m) à l'état (σ_1+1,…1,…σ_m+1). Sinon on ne diffuse rien (c'est-à-dire le message vide que l'on nomme X) et on passe alors à l'état (σ_1+1,…σ_i+1,…σ_m+1).

On définit le coût d'une arrête par le coût d'attente en arrière. Or le temps d'attente en arrière moyen pour le message M_i est de σ_i-1/2. Ainsi, le coût d'une arrête X sera de $\sum_{i=1}^{m} p_i(\sigma_i + 1 - 1/2) = 1/2 + \sum_{i=1}^{m} p_i\sigma_i$ et le coût d'une arrête M_i de $c_i + 1/2 + \sum_{j\neq i} p_j\sigma_j$. Le coût d'un chemin est alors la limite supérieure du coût moyen de chaque arrête.

Il est évident qu'à tout chemin du graphe, on peut associer un ordonnancement des messages M_i. De plus, par définition, le coût de ce chemin est égal à l'attente en arrière moyenne de l'ordonnancement correspondant. Or on peut montrer que l'attente en arrière est équivalente au temps de service (que l'on a défini dans la première partie pour les ordonnancements périodiques, mais que l'on peut facilement étendre à tout ordonnancement. On pourra

consulter [Sch00] pour l'extension du temps de service à tout ordonnancement et la preuve de l'équivalence avec le temps en arrière). Il suffit donc de montrer que le graphe admet un cycle de coût minimum pour montrer qu'il existe un ordonnancement périodique optimal.

A.2-Borne du nombre d'états

Le problème est que le nombre d'états du graphe que l'on considère est à priori infini. Nous allons montrer en fait que nous pouvons nous contenter de considérer un nombre fini d'états.

Théorème A.1 :

Soit S un ordonnancement des messages $\{M_1, \dots M_m\}$. Alors il existe un ordonnancement S' tel que $COUT(S') \leq COUT(S)$ et l'intervalle entre deux diffusions du message M_i dans S' soit borné par $(2m+c_i)/p_i$.

Preuve :

Soit i tel que l'intervalle de temps entre deux diffusions de M_i soit maximal. Soit 2K la longueur de cet intervalle et T son début. Considérons alors l'ordonnancement S' qui diffuse le message M_i à l'instant T+K et décale tous les autres messages.

Pour tout $j \neq i$, soit Aj l'intervalle de temps entre T+K et la première diffusion de Mj après T+K dans S. On a $Aj \leq 2K$. Le décalage des messages j fait donc augmenter le temps de service de $\sum_{j\neq i} p_j\left((A_j+1)^2 - A_j^2\right) = \sum_{j\neq i} p_j(2A_j+1) \leq 4K+1$ (à un facteur de proportionnalité près). Par contre, diffuser le message Mi à l'instant T+K fait diminuer le temps de service de $pi(2K)^2 - 2pi*K^2 = 2piK^2$ (au même facteur de proportionnalité près). Ainsi le coût de S' est plus petit que le coût de S si K est assez grand, plus précisément si $2pi*K^2-4K-1-ci>0$ ق

A.3-Existence d'un ordonnancement périodique optimal

Nous allons enfin pouvoir en déduire le

Théorème A.2 :

Pour toute distribution de messages $\{M_1, \dots M_m\}$ de longueurs uniformes, il existe un ordonnancement périodique optimal.

Preuve :

Soit G l'ensemble le graphe de l'ensemble des états possibles pour les messages $\{M_1, \dots M_m\}$. D'après le théorème A.1, pour trouver l'ordonnancement optimal, il nous suffit de considérer le graphe fini G/K, où les σ_i appartiennent à $[|1, K_i|]$. Soit C un chemin de ce graphe. Il se décompose en un nombre fini de cycles, dont certains sont parcourus une infinité de fois. Le coût de C est alors la combinaison barycentrique du coût de ces cycles. Il existe donc au moins un cycle dont le coût soit plus petit ou égal à celui de C.

Du fait de l'équivalence entre les ordonnancements et les chemins du graphe d'état, on a montré que pour tout ordonnancement, il existait un ordonnancement périodique de coût inférieur ou égal. Il existe donc un ordonnancement périodique optimal ق

On remarque au passage que cette preuve donne un algorithme (non polynomial) permettant de trouver un tel ordonnancement périodique optimal : il suffit de trouver dans le graphe d'états fini précédent un cycle de point moyen minimum.

Appendice B :

B.1-Résolution du problème de minimisation:

Soit $Min1 : \min\left(1 + \sum_{i=1}^{m}\left(\frac{p_i \tau_i}{2} + \frac{c_i}{\tau_i}\right)\right)$ $sur\ le\ domaine \begin{cases} \forall i \tau_i \geq 1 \\ \sum_{i=1}^{m} \frac{1}{\tau_i} \leq W \end{cases}$

Nous avons vu que *Min*1 est un minorant de tout ordonnancement périodique et qu'il est atteint pour un unique τ^*. De plus, un ordonnancement périodique S a un coût égal à ce minorant ssi il diffuse M_i tous les τ_i^*.

Ce minorant s'est avéré être très puissant, en particulier tous nos algorithmes de dissémination sont fondés sur les τ_i^*. On conçoit l'importance de pouvoir les calculer. C'est l'objet du théorème suivant :

Théorème B.1 :

Soit $\tau_i' = \sqrt{\frac{2c_i + \lambda}{p_i}}$ avec λ défini par :

- si $\sum_{i=1}^{m} \sqrt{p_i/(2c_i)} \leq W$ alors λ=0
- sinon λ est l'unique solution positive de $\sum_{i=1}^{m} \sqrt{p_i/(2c_i + \lambda)} = W$

Alors pour tout 1≤i≤m tel que $\tau_i' < 1$, alors $\tau_i^* = 1$. On itère alors les calculs (en enlevant les messages et le nombre de canaux correspondants) jusqu'à ce que pour tout i, $\tau_i' \geq 1$. Alors $\tau_i^* = \tau_i'$.

Preuve :

La preuve est assez longue et technique, elle se décompose en deux étapes.

Lemme B.1 :

Soit $Min2 : \min\left(\sum_{i=1}^{m}\left(p_i \tau_i + \frac{2c_i}{\tau_i}\right)\right)$ $sur\ le\ domaine \begin{cases} \forall i \tau_i \geq 0 \\ \sum_{i=1}^{m} \frac{1}{\tau_i} \leq W \end{cases}$

*Min*2 est atteint pour un unique τ' tel que

$\tau_i' = \sqrt{\frac{2c_i + \lambda}{p_i}}$ avec λ défini par :

- si $\sum_{i=1}^{m} \sqrt{p_i/(2c_i)} \leq W$ alors λ=0
- sinon λ est l'unique solution positive de $\sum_{i=1}^{m} \sqrt{p_i/(2c_i + \lambda)} = W$

Preuve :

Soit f(τ)= $\sum_{i=1}^{m}(p_i\tau_i + 2c_i/\tau_i)$ la fonction objectif et g(τ)= $W - \sum_{i=1}^{m} 1/\tau_i$. On remarque que f est continue et que lim $_{\|\tau\|\to\infty}$ f(τ)= ∞. On peut donc se restreindre à un domaine de minimisation fermé borné, d'où l'existence d'un minimum. De plus f étant strictement convexe, ce minimum est unique (s'il y en avait deux, en considérant leur milieu on obtiendrait une contradiction).

Les fonctions $\tau_i \to p_i\tau_i + 2c_i/\tau_i$ sont convexes et atteignent un minimum en $\tau_i^{\#} = \sqrt{2c_i/p_i}$. Si ce minimum vérifie les contraintes, c'est-à-dire si $\sum_{i=1}^{m} 1/\tau_i^{\#} \le W$ alors on a évidemment $\tau_i' = \tau_i^{\#} = \sqrt{2c_i/p_i}$ et λ=0. Sinon, il est facile de voir que f atteint son minimum en τ' ssi g(τ')=0 (car sinon on pourrait diminuer un des τ_i ce qui diminuerait également f).

Le théorème des extrémas liés nous donne $\overrightarrow{grad}\, f(\tau') = \lambda \overrightarrow{grad}\, g(\tau')$ soit $\tau_i' = \sqrt{(2c_i+\lambda)/p_i}$ (τ_i' étant positif) et comme g(τ')=0, on a $\sum_{i=1}^{m} \sqrt{p_i/(2c_i+\lambda)} = W$.

On remarque que Min2 ressemble beaucoup à Min1, seul le domaine de minimisation a changé. Il semble donc probable de pouvoir obtenir τ* à partir de τ'. C'est l'objet du lemme suivant.

Lemme B.2 : Si il existe i_0 tel que τ_{i0}' <1, alors τ_{i0}^{*} =1.

Preuve :

On peut supposer i_0=1. On sait d'après le lemme B.2 qu'il existe un réel λ≥0 tel que $\tau_i' = \sqrt{\frac{2c_i+\lambda}{p_i}}$. Maintenant supposons par l'absurde que τ_1*≠1. Alors τ_1*>1. Donc forcément, g(τ*)=0 (avec les notations du lemme précédent), car sinon on pourrait diminuer τ_1* et le rapprocher vers le minimum de la fonction $\tau \to p_1\tau + 2c_1/\tau$ (car τ_1' est plus grand que ce minimum, cf lemme B.2). Comme on a de plus g(τ')>0, il existe alors i_1 tel que $\tau_{i1}' < \tau_{i1}^{*}$. On peut supposer i_1=2.

Considérons alors le problème de minimisation suivant :

$$Min3 : \min\left(\sum_{i=1}^{m}\left(p_i\tau_i + \frac{2c_i+\lambda}{\tau_i}\right)\right) \text{ sur le domaine } \begin{cases} \forall i\ \tau_i \ge 1 \\ \sum_{i=1}^{m} \frac{1}{\tau_i} \le W \end{cases}$$

On montre comme précédemment que ce minimum est bien définit et est atteint en un unique point τ''. Chacune des fonctions $\tau_i \to p_i\tau_i + (2c_i+\lambda)/\tau_i$ est convexe et atteint un minimum en τ_i'. Il faut donc approcher le plus possible τ' en respectant les contraintes, donc τ_i''=max (1, τ_i'). On remarque que g(τ'')≥ g(τ')≥ 0 donc les contraintes sont bien respectées. Or τ*≠ τ'' (τ_1*>1= τ_1''> τ_1' et 1≤ τ_2*< τ_2'= τ_2''). D'où Min3 n'est pas atteint en τ* et toujours par convexité, en rapprochant τ* vers τ'' on diminue la valeur de la fonction objectif associée à Min3. Ainsi, si l'on prend δ_1<0 et δ_2>0 tels que τ_1*+ $\delta_1 \ge \tau_1$'', τ_2*+ $\delta_2 \le \tau_2$'' et $\delta_1/(\tau_1^*)^2 + \delta_2/(\tau_2^*)^2 = 0$ et tels que τ+δ vérifie les contraintes de minimisation de Min3, alors la fonction objectif associée à Min3 diminue strictement de τ* à τ*+δ.

Or cette variation est égale à $p_1\delta_1 + p_2\delta_2 - \left(\frac{2c_1+\lambda}{\tau_1^{*2}}\delta_1 + \frac{2c_2+\lambda}{\tau_2^{*2}}\delta_2\right)$ (au premier ordre en δ) ce qui se simplifie en $p_1\delta_1 + p_2\delta_2 - \left(\frac{2c_1}{\tau_1^{*2}}\delta_1 + \frac{2c_2}{\tau_2^{*2}}\delta_2\right)$. La fonction objectif associée à Min1 diminue donc strictement de τ* à τ*+δ ce qui contredit la définition de τ* et nous donne l'absurdité voulue.

En combinant ces deux lemmes, on obtient alors trivialement le théorème B.1 : pour calculer τ* on calcule τ'. Puis, s'il existe un i tel que $\tau_i' < 1$, alors $\tau_i^* = 1$. On enlève le message i et on diminue de 1 le nombre de canaux. Lorsque pour tout i, $\tau_i' \geq 1$ alors $\tau^* = \tau'$.

Il reste donc à calculer τ', c'est-à-dire à calculer λ. Or soit λ=0, sinon λ est l'unique solution positive de $\sum_{i=1}^{m} \sqrt{p_i/(2c_i+\lambda)} = W$. Cette dernière fonction étant convexe et décroissante en λ, on pourra calculer λ en utilisant l'algorithme de Newton de manière extrêmement rapide.

B.2 - Construction d'un arbre à partir d'une distribution de popularité:

Soit m messages M1…Mm dont les fréquences idéales de diffusion sont 1/W τ_i^*. On a vu que modéliser cette distribution de fréquence par un arbre permet de diffuser les messages en respectant parfaitement les intervalles de τ_i^* entre chaque diffusion. Comme une telle modélisation n'est pas toujours possible, on se ramène à la fréquence en $1/2^k$ immédiatement inférieure.

Ainsi on se ramène à une distribution de fréquences de $1/2^{n-i}$ et on ajoute des messages « vides » de fréquence $1/2^n$ de façons à ce que la somme des fréquences soit de un. Le lemme suivant nous montre que l'on peut associer un arbre à cette distribution :

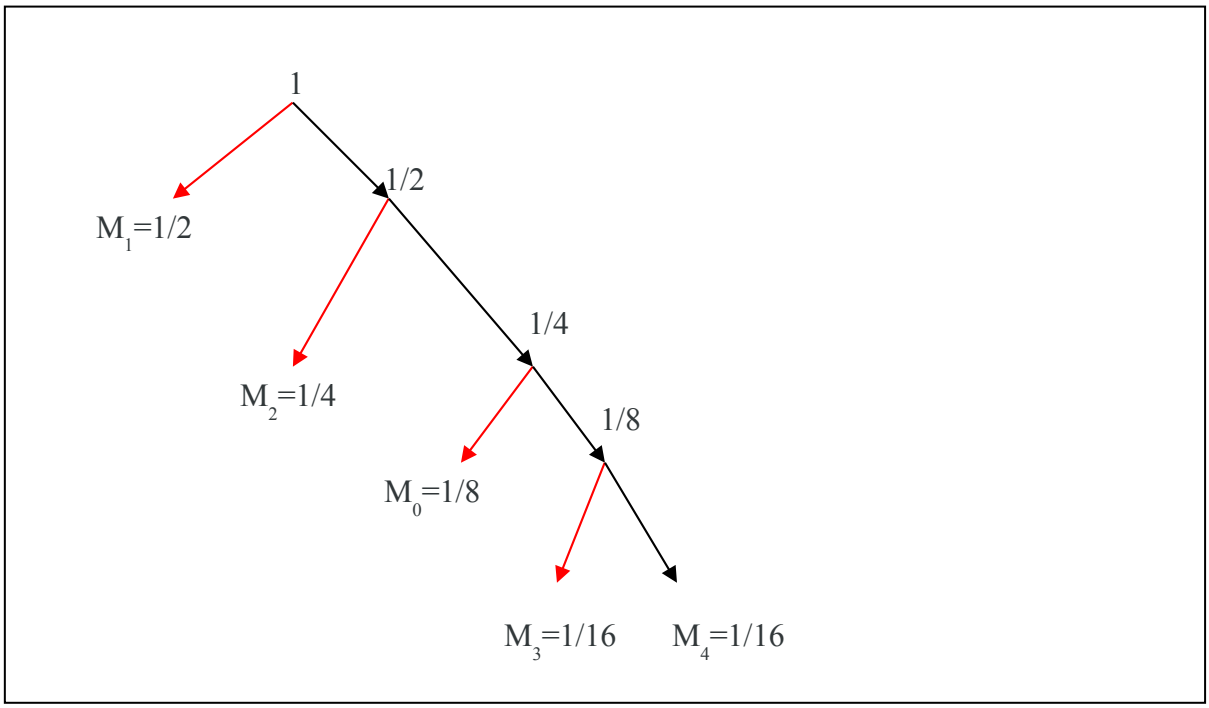
Lemme B.3 – Soit $n_1 \leq n_2 \leq \ldots \leq n_m$ tels que $\sum_{i=1}^{m} 2^{n-i} = 2^N$. Alors il existe $i_0 < m$ tel que $\sum_{i=1}^{i0} 2^{n-i} = 2^{N-1}$

Preuve :

Soit i_0 le plus petit i tel que $S = \sum_{i=1}^{i_0} 2^{n-i}$ soit plus grand ou égal à 2^{N-1}. On a $S + \sum_{i>i0} 2^{n-i} \equiv 2^N \equiv 0 \left[2^{i0}\right]$. Soit $S \equiv 0 \left[2^{i0}\right]$. Donc les i_0-1 derniers chiffres de S (dans son écriture binaire) sont nuls. De plus comme S est plus grand que 2^{N-1}, il y a eu propagation de la retenue quand on a ajouté 2^{i0} et donc les chiffres binaires de S du $2^{\text{ième}}$ au $i_0^{\text{ième}}$ dernier sont nuls. Finalement, tous les chiffres binaires de S sont nuls sauf le premier, donc $S=2^{N-1}$.

Ce lemme nous montre donc qu'en sommant les fréquences par ordre croissant, on finit par tomber sur ½. Il suffit alors de couper le tas en deux et de recommencer, afin d'obtenir un arbre binaire. Comme on diminue au plus les fréquences par deux, on augmente au pire l'intervalle entre deux diffusions par deux, et donc le temps de service par quatre (le temps de service étant une fonction quadratique des intervalles de diffusion). Cet algorithme est donc une 4-approximation.

Ex d'application : Pour une distribution de fréquence de M_1=1/2, M_2=1/3, M_3=1/12, M_4=1/12, on se ramène à une distribution M_1=1/2, M_2=1/4, M_3=1/16, M_4=1/16 et on ajoute un message vide M_0 de fréquence 1/8. On obtient alors l'arbre suivant :

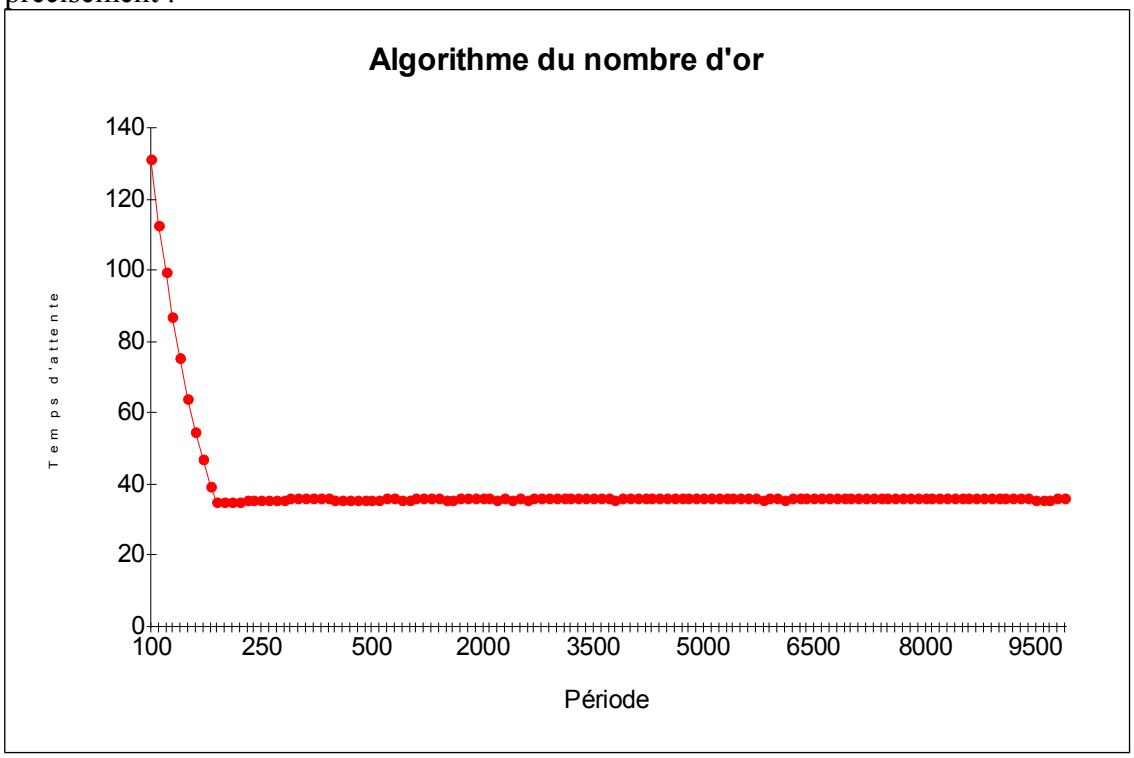


Appendice C :
Autres résultats de la simulation.

C.1- Période optimale pour l'algorithme du nombre d'or :

L'algorithme du nombre d'or construit un ordonnancement périodique de période T, ordonnancement qui respecte les fréquences optimales en $1/W\tau_i^*$. On peut se demander quelle période T il faut choisir pour obtenir les meilleurs résultats (l'algorithme du nombre d'or étant assez proche de l'optimal, cela nous donnera une idée de la période de l'ordonnancement périodique optimal).

A priori, il faut que la période soit au moins plus grande que le nombre de résultats. Plus précisément :

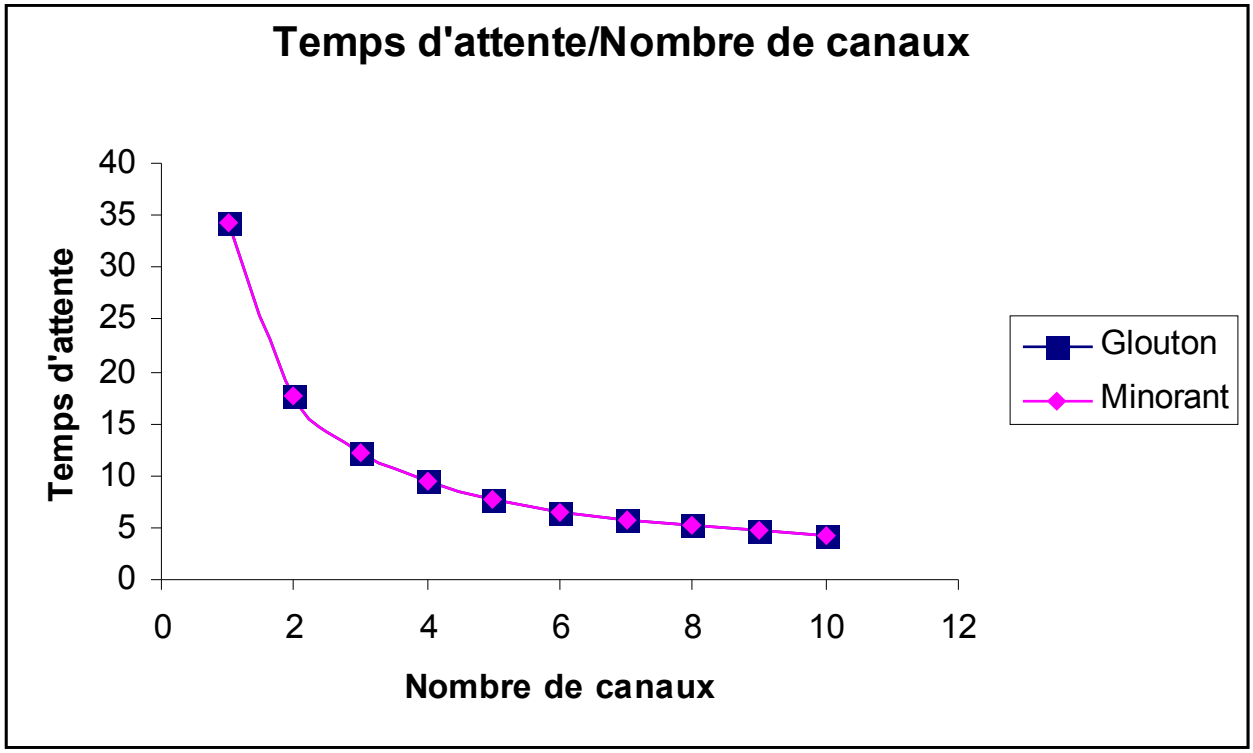


La simulation s'est effectuée sur un ensemble de 100 messages à diffuser sur un canal.
La période optimale est de 400s où le temps d'attente moyen est de 35.52s. On voit donc qu'une période égale à dix fois le nombre de messages est un bon ordre de grandeur de la période optimale.

C.2-Variation du temps d'attente en fonction du nombre de canaux :

Ici on s'intéresse à la variation du temps de service en fonction du nombre de canaux : plus il y a de canaux de diffusion, plus on peut diffuser rapidement des messages. Dans quelle mesure cela influe t'il sur le temps de service ?

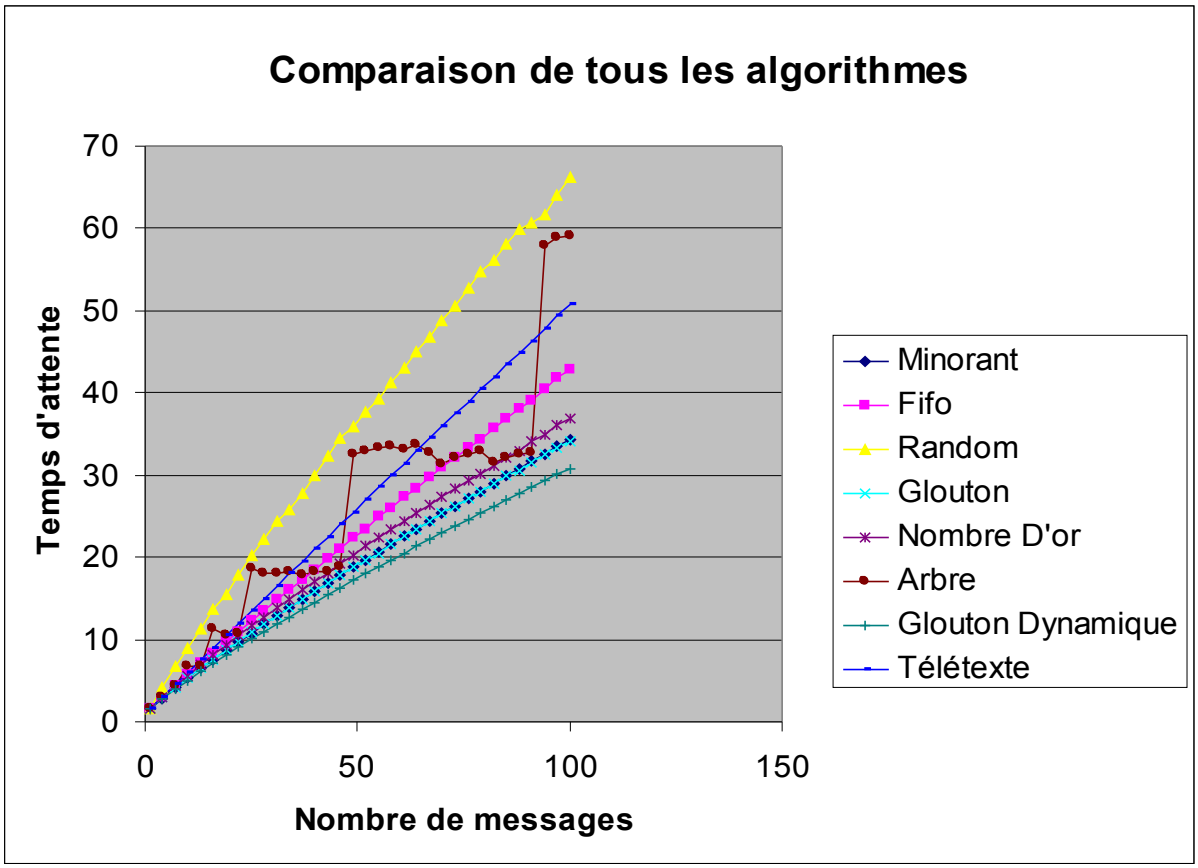
Théoriquement le minorant est inversement proportionnel au nombre de canaux (lorsqu'il n'y a pas de coûts de diffusion, cf 1.3).



On voit que le temps d'attente « réel » suit aussi cette loi en K/W (où W représente le nombre de canal).

L'ajout d'un canal permet de diminuer le coût d'un facteur d'au plus $\frac{W}{W+1}$. Ainsi il vaut mieux utiliser un meilleur algorithme que de chercher à augmenter le nombre de canal pour augmenter la rapidité.

C.3- Comparaison entre tous les algorithmes :



La simulation a été effectuée pour dix clients se connectant par unité de temps. Le nombre de clients n'affecte que les algorithmes qui ne sont pas des algorithmes de dissémination de données, à savoir le Fifo et le glouton dynamique. Théoriquement, quand le nombre de clients tend vers l'infini, le Fifo doit tendre vers l'algorithme du télétexte (il s'agit de l'algorithme utilisé pour le télétexte qui consiste à diffuser les messages cycliquement) et le glouton dynamique vers le glouton.

On constate que la simulation confirme bien l'étude théorique. Ainsi, l'algorithme random est bien deux fois plus long que l'optimal (le minorant), le télétexte a bien un temps de service égal au nombre de message divisé par deux.

Comme on l'a déjà vu, le glouton donne de très bons résultats, très proches du minorant. Quant au glouton dynamique, il est ici plus rapide que le minorant, et donc donnera de meilleurs résultats que n'importe quel algorithme de dissémination. (Expérimentalement on constate bien que le temps de service des algorithmes de dissémination est minoré par le minorant).

Enfin, l'algorithme par arbre a un comportement intéressant : il fonctionne par paliers. Il est peut-être aussi performant que le glouton comme aussi mauvais que le « random ». Ceci est

liée à l'approximation des popularités par des puissances de deux utilisé dans l'algorithme : si les popularités sont des puissances de deux, cet algorithme est optimal.

C.4- Etude plus détaillée des algorithmes de dissémination de données :

La simulation s'est effectuée pour dix messages. La longueur de la simulation a été de cent

Message n°	0	1	2	3	4	5	6	7	8	9
Popularité	0.3414	0.1707	0.1138	0.0854	0.0683	0.0569	0.0488	0.0427	0.0379	0.0341
Temps d'attente optimal pour ce message	3.5105	4.5504	5.3483	6.021	6.6136	7.1494	7.6422	8.1008	8.5315	8.9389

unités de temps. Comme il s'agit d'algorithmes de dissémination de données, le nombre de clients ne joue pas, l'étude a donc été réalisée via le programme « théorie » (voir le code source).

Temps d'attente optimal :

Minorant : 5.30.

Algorithme « random » :

Premiers messages diffusés : 0 2; 2; 3; 4; 7; 3; 9; 2; 1; 0; 8; 2; 9; 7; 0; 5; 4; 1; 7; 0; 2; 7; 7; 2.

Message n°	0	1	2	3	4	5	6	7	8	9
Nombre de messages diffusés	198	133	107	94	98	75	73	74	77	72
Temps d'attente pour ce message	6.139	6.802	9.242	11.167	10.365	12.353	14.5	12.961	11.842	14.607

Temps d'attente moyen : 8.88.

Le random diffuse les messages aléatoirement suivant la probabilité $1/\tau_i$*.

Algorithme « glouton » :

Premiers messages diffusés : 0; 1; 0; 2; 3; 0; 1; 4; 5; 0; 6; 2; 7; 1; 3; 0; 8; 9; 4; 2; 0; 1; 5; 6; 3.

Message n°	0	1	2	3	4	5	6	7	8	9
Nombre de messages diffusés	193	139	111	100	84	80	75	74	73	72
Temps d'attente pour ce message	3.626	4.666	5.557	6.075	7.012	7.266	7.732	7.794	7.875	7.968

Temps d'attente moyen : 5.36.

Le glouton diffuse le message dont l'attente cumulée est maximum.

Algorithme « nombre d'or » :

Premiers messages diffusés : 8; 2; 5; 0; 3; 7; 1; 4; 0; 9; 2; 6; 0; 3; 8; 1; 5; 0; 2; 7; 1; 4; 0; 9; 2.

Message n°	0	1	2	3	4	5	6	7	8	9
Nombre de messages diffusés	200	147	116	95	84	84	74	74	64	63
Temps d'attente pour ce message	3.625	4.732	5.769	6.672	7.236	7.261	8.512	8.305	9.763	9.592

Temps d'attente moyen : 5.65.

L'algorithme du nombre d'or diffuse les messages cycliquement, les messages étant répartis à peu près uniformément sur la période et ont une fréquence proportionnelle à $1/\tau_i$* (la période choisie est de cent unités de temps et est trop longue pour apparaître ici).

Algorithme par « arbre » :

Premiers messages diffusés : 7; 1; 8; 1; 2; 0; 9; 1; 3; 1; 6; 1; 5; 0; 4; 1; 7; 1; 8; 1; 2; 0; 0; 1; 3.

Message n°	0	1	2	3	4	5	6	7	8	9
Nombre de messages diffusés	156	375	63	63	62	62	62	63	63	32
Temps d'attente pour ce message	4.776	2.501	8.96	8.976	8.98	8.976	8.98	8.976	8.964	16.9

Temps d'attente moyen : 6.70.

Cet algorithme modélise la distribution de fréquence par un arbre (cf première partie, on reconnaît bien par les messages diffusés le coloriage rouge et noir). Si le message un est le plus diffusé, cela vient du fait que l'on « bouche les trous » issus de l'approximation en puissances de deux.

Algorithme du « télétexte » :

Premiers messages diffusés : 0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 0; 1; 2; 3; 4.

Message n°	0	1	2	3	4	5	6	7	8	9
Nombre de messages diffusés	101	100	100	100	100	100	100	100	100	100
Temps d'attente pour ce message	6	6.001	5.994	5.989	5.986	5.985	5.986	5.989	5.994	6.001

Temps d'attente moyen : 6.0.

L'algorithme du télétexte diffuse les messages cycliquement. On constate qu'il est très mauvais : ici, il est presque deux fois plus lent que l'optimum. En appliquant le glouton ou l'algorithme du nombre d'or, on pourrait sûrement améliorer les performances du télétexte (il faudrait tout de même connaître la popularité des messages).

Remarques :

On constate que les algorithmes les plus performants sont ceux qui suivent les fréquences indiquées par le minorant. Les algorithmes égalitaires comme celui du télétexte sont mauvais en pratique. Enfin, l'exemple du « random », qui diffuse les messages proportionnellement aux fréquences optimales mais pas assez uniformément (cf les premiers messages diffusés, souvent le même message est diffusé deux fois de suite) montre que l'on doit essayer de garder un intervalle constant entre la diffusion de deux messages.

Appendice D : Code source du programme.

Le programme a été séparé en six modules. Les modules les plus importants sont celui qui calcule le minorant ainsi que les fréquences idéales (cf première partie), celui qui implémente les algorithmes de dissémination de données et enfin celui qui simule un serveur traitant les requêtes de clients. Les autres modules sont un module qui regroupe les données communes aux autres modules, un module permettant de changer dynamiquement les données de la simulation et enfin un module qui permet de lancer le programme et de sauvegarder les résultats de la simulation dans des fichiers.

Chaque module est séparé en un fichier *.mli, qui contient l'interface de ce module (ie : les données et les programmes accessibles à partir d'autres modules) et en un fichier *.ml qui contient le corps des programmes. Ceci permet de faciliter la compilation du programme.

D.1- Calcul du minorant :

```
(*****************Fichiers Minorant.mli et Minorant.ml:************)

value minorant : int -> float vect -> float vect -> float vect
and calc_min : int -> float vect -> float vect -> float
and calc_min_message : int -> float vect -> float vect -> int -> float;;

(*********************************************Calcul du
minorant*****************************)

let Abs x=
  if x<.0. then -.x else x;;

(*Calcul de lambda dans le minorant, très rapide->ne pas hésiter à prendre
une précision élevèe*)
(*Newton fonction, dérivée, approximation allouée, point de départ*)
let rec Newton f f' epsilon x=
   if Abs(f(x))<=.epsilon then x
   else Newton f f' epsilon (x-.f(x)/.f'(x));;
```

```
(*Calcule le minorant en relaxant les contraintes Taui>=1*)
(*minorant_relaxe nombre de canaux,table des popularités, table des coûts*)
let minorant_relaxe w p c=
  let m=vect_length p in
  let sigma=ref 0. in
  for i=0 to m-1 do
    sigma:=!sigma+.sqrt(p.(i)/.(2.*.c.(i)))
  done;
  (*Calcul de lambda, si jamais sigma n'est pas dans la frontière*)
  let lambda=if !sigma<=float_of_int(w) then 0.
             else ((*Comme la fonction objectif est convexe décroissante,
pour que Newton converge, il faut lui entrer
                     une valeur telle que f x soit < 0*)
                   let essai=ref (float_of_int m /. (2.*.float_of_int
(w*w))) in
                   let f=(fun x-> let r=ref 0. in for i=0 to m-1 do r:=!
r+.sqrt(p.(i)/.(2.*.c.(i)+.x)) done;
                                  float_of_int w -. !r) in
                   while f !essai >. 0. do essai:=!essai/.100. done;
             (*Appel de Newton avec une précision de 10^-12*)
             Newton f  (fun x-> let r=ref 0. in for i=0 to m-1 do
                                                  r:=!r+.sqrt(p.(i))*.
(2.*.c.(i)+.x)**(-1.5)/.2.
                                               done;
                                !r)
                          (10.**(-12.)) !essai) in
  (*Calcul des Tau*)
  let Tau=make_vect m 0. in
  for i=0 to m-1 do
    Tau.(i)<-sqrt((2.*.c.(i)+.lambda)/.p.(i))
  done;
  Tau;;


(*Le type est utilisé par l'algo iter_minorant*)
type message={Popu:float;Cout:float};;

(*Lorsque minorant_relaxé renvoit un Taui<1, on vire le message et un canal
et on itère*)
(*iter_minorant nb_canaux, table des messages*)
let iter_minorant w m_vect=
  let m=vect_length m_vect in
  let Vire=make_vect m 0 in
  let Sol=make_vect m 0. in
  let nb_vire=ref 0 in
  let virer i=
    incr nb_vire;
    for j=i to m-1 do
      Vire.(j)<-Vire.(j)+1
    done in
  let b=ref false in
  while not !b do
    b:=true;
    (*On construit le tableau de message en enlevant ceux qui ont été
virés*)
    let P=make_vect (m- !nb_vire) 0. and C=make_vect (m- !nb_vire) 0. in
    for i=0 to m- !nb_vire-1 do
      P.(i)<-m_vect.(i+Vire.(i)).Popu;
      C.(i)<-m_vect.(i+Vire.(i)).Cout;
    done;
    let Tau=minorant_relaxe (w- !nb_vire) P C in
```

```
    for i=0 to m- !nb_vire-1 do
      if Tau.(i)<1. then (Sol.(i+Vire.(i))<-1.;virer (i+Vire.(i));b:=false)
    done;
    if !b then (
      (*On entre les derniers Taui* - ceux qui ne sont pas égaux à 1*)
      for i=0 to m- !nb_vire-1 do
        Sol.(i+Vire.(i))<-Tau.(i)
      done
    )
  done;
  Sol;;


(*Calcule les Taui, avec ou sans coût*)
(*minorant nb_canaux, popu, couts*)
let minorant w p c=
  let m=vect_length p in
  let vide={Popu=0.;Cout=0.} in
  let vect=make_vect m vide in
  for i=0 to m-1 do
    vect.(i)<-{Popu=p.(i);Cout=c.(i)}
  done;
  iter_minorant w vect;;

let minor_sans_cout w p=
  let m=vect_length p in
  minorant w p (make_vect m 0.);;

(*Calcul le minorant, avec ou sans coût de diffusion*)
let calc_min w p c=
  let m=vect_length p in
  let Tau=minorant w p c in
  let min=ref 1. in
  for i=0 to m-1 do
    min:=!min+.p.(i)*.Tau.(i)/.2.+.c.(i)/.Tau.(i)
  done;
  !min;;

let calc_min_sans_cout w p=
  let m=vect_length p in
  let Tau=minor_sans_cout w p in
  let min=ref 1. in
  for i=0 to m-1 do
    min:=!min+.p.(i)*.Tau.(i)/.2.
  done;
  !min;;

(*Calcule le temps d'attente optimal pour un message donné*)
let calc_min_message w p c i=
  let m=vect_length p in
  let Tau=minorant w p c in
  1.+.Tau.(i)/.2.+.c.(i)/.Tau.(i);;
```

D.2- Données communes aux modules:

```
(*****************Fichiers Variables.mli et Variables.ml:*************)
(*Types*)
(*file_simul permet de calculer le temps moyen d'attente.
  Premier sert pour les FIFOS, indique le premier message demandé*)
```

```
type file_simul= {mutable Nb_clients:int;mutable Attente_cum:float;mutable
Premier:int};;
exception Trouve of int;;

value nb_canaux : int ref
and nb_messages : int ref
and popularite : float vect ref
and popu_cum : float vect ref
and duree : float vect ref
and couts : float vect ref
and cumul : float vect -> float vect
and normalise : float vect -> float vect
and zipf : int -> float vect
and unif : int -> float -> float vect
and init_popularite : unit -> unit
and init_duree : unit -> unit
and init_couts : unit -> unit
and arrondir : float -> float
and puissance : int -> int -> int;;

(**************Variable contient les types, algo ou constantes utilisées
par plusieurs des modules*********)

(*Serveur*)
let nb_canaux= ref 1;;
let nb_messages= ref 1;;


(*Calcul des coûts et des popularités, dans un tableau*)
(*Messages de même longueur, distribution suivant la loi de Zipf*)
let popularite=ref (make_vect !nb_messages 0.);; (*la longueur de
popularite peut changer*)
let popu_cum=ref (make_vect !nb_messages 0.);;   (*ce qui explique pourquoi
on prend une référence*)
let duree=ref (make_vect !nb_messages 1.);;
let couts=ref (make_vect !nb_messages 0.);;


(*Construction des tableaux*)
(*Construction d'une tableau de somme cumulée à partir d'un tableau de
float*)
let cumul p=
  let n=vect_length p in
  let p_cum=make_vect n 0. in
  p_cum.(0)<- p.(0);
  for i=1 to n-1 do
    p_cum.(i)<- p_cum.(i-1)+. p.(i);
  done;
  p_cum;;

(*Normalisation*)
let normalise p=
  let n=vect_length p in
  let r=ref 0. in
  for i=0 to n-1 do
    r:=!r+. p.(i)
  done;
  for i=0 to n-1 do
    p.(i)<-p.(i)/. !r
  done;
  p;;
```

```
(*Crée un tableau suivant la loi de Zipf:le nieme message a une popularite
proportionelle à 1/n*)
let zipf n=
  let l=make_vect n 0. in
  for i=0 to n-1 do
    l.(i)<-1./.(float_of_int(i+1))
  done;
  l;;

(*unif crée un tableau de n éléments qui différent au plus de epsilon,sert
pour avoir des popularités uniformes*)
let unif n epsilon=
  let l=make_vect n 0. in
  let r=ref 0. in
  for i=0 to n-1 do
    l.(i)<-1./.(float_of_int n)+.random__float epsilon -. epsilon;
    if l.(i)<.0. then l.(i)<-0.;
    r:=!r+.l.(i)
  done;
  l;;

let init_popularite()=
  popularite:=normalise(zipf !nb_messages);
  popu_cum:=cumul !popularite;;
init_popularite();;

let init_duree()=
  duree:=make_vect !nb_messages 1.;;

let init_couts()=
  couts:=make_vect !nb_messages 0.;;

(*Sous-prog utilisés*)
let arrondir x=floor(100.*.x)/.100.;;
let rec puissance x y=
  let r=if y>0 then puissance x (y/2) else 1 in
  if y mod 2 = 1 then x*r*r
  else r*r;;
```

D.3- Les algorithmes de dissémination de données:

```
(***********Fichiers Algorithmes.mli et Algorithme.ml:***************)

#open "Variables";;

(*Pour différencier les types, on est obligé de noter différement les
labels message*)
type message_dissemine={Message1:int;Place:float};; (*type utilisé par
l'algo de tri*)
type message_arbre={Message2:int;Freq:int};; (*type utilisé pour construire
l'arbre*)
type arbre=
  |Arbre of branche * branche
  |Feuille of message_arbre
and branche={mutable Couleur:bool;Branche:arbre};;

value ordonancement : message_dissemine vect ref
and T_nbor : float ref;;
```

```
value determine_message0 : file_simul vect -> int
and Tau : float vect ref
and init_Tau : unit -> unit
and init_proba : unit -> float vect
and determine_message1 : unit -> int
and determine_message2 : float vect -> int
and init_ordonancement : float -> unit
and determine_message3 : int -> int
and init_arbre : unit -> arbre
and determine_message4 : unit -> int
and determine_message5 : file_simul vect -> int -> int
and determine_message6 : unit -> int;;

#open "Variables";;
#open "Minorant";;

(*Algorithmes de dissemination de données*)
(*Variables utilisées:à besoin de Minorant.ml et de Variables.ml*)

(************************ ALGO 0*******************)
(*Message à diffuser:méthode habituelle par FIFO*)
(*Coût linéaire en le nombre de messages*)
let determine_message0 v=
  let r=ref v.(0).Premier and indice=ref 0 in
  for i=1 to !nb_messages-1 do
    if v.(i).Premier< !r then (r:=v.(i).Premier;indice:=i)
  done;
  if !r=max_int then -1 else !indice;;

(*Calcul des proportions idéales:sert à 1 2 3 et 4**)
let Tau=ref (minorant !nb_canaux !popularite !couts);;
let init_Tau()=
  Tau:=minorant !nb_canaux !popularite !couts;;



(*Note:pour les algos qui suivent, qui implémentent les coûts, il se peut
qu'aucun message ne soit diffusé
  à l'instant T. L'algo renvoit alors le nombre -1*)
(************************ ALGO 1*******************)
(*Probabiliste: message diffusé proportionnellement à sqrt p_i*)

(*Calcule les probablités de diffusion et les probas cumulées de
diffusion*)
let proba=ref (make_vect !nb_messages 0.);;
let proba_cum=ref (make_vect !nb_messages 0.);;
let init_proba()=
  proba:=make_vect !nb_messages 0.;
  proba_cum:=make_vect !nb_messages 0.;
  !proba.(0)<-1./.(float_of_int !nb_canaux *. !Tau.(0));
  !proba_cum.(0)<- !proba.(0);
  for i=1 to !nb_messages-1 do
    !proba.(i)<-1./.(float_of_int !nb_canaux *. !Tau.(i));
    !proba_cum.(i)<- !proba_cum.(i-1)+. !proba.(i);
  done;
  !proba;;
init_proba();;

(*Diffuse aléatoirement un message suivant les probas plus haut*)
(*Coût linéaire en le nombre de messages*)
```

```
let determine_message1()=
  let r=random__float 1. in
  try (
    for i=0 to !nb_messages-1 do
      if !proba_cum.(i)>.r then raise (Trouve i)
    done;
    -1)
  with Trouve i->i;;



(************************ ALGO 2*******************)
(*Dérandomatisation gloutonne de 1*****************)

(*état est le tableau des sigmai*)
(*Coût linéaire en le nombre de messages*)
let determine_message2 etat=
  let min=ref 0. and indice=ref (-1) in
  for i=0 to !nb_messages-1 do
    let r= !couts.(i)-. !popularite.(i)*.etat.(i)*. !Tau.(i) in
    if r<=. !min then (min:=r;indice:=i)
  done;
  !indice;;



(****************Tri Fusion**********************)
(*Sert pour 3 et 4. ppe x y renvoie le plus petit des éléments*)
let rec fusion ppe= fun
  |[] l2 -> l2
  |l1 [] -> l1
  |(p::q as l1) (u::v as l2)-> if ppe p u then p::(fusion ppe q l2) else
u::(fusion ppe l1 v);;
let rec partage = fun
  |[] -> ([],[])
  |[p]-> ([p],[])
  |(p::q::r) -> let (l1,l2)=partage r in (p::l1,q::l2);;
let rec tri_fusion ppe = fun
  |[]->[]
  |[p]->[p]
  |l->let (l1,l2)=partage l in fusion ppe (tri_fusion ppe l1) (tri_fusion
ppe l2);;



(************************ ALGO 3*******************)
(*Nombre d'or*************************************)
let T_nbor=ref 1000.;; (*période des messages*)

let theta=(1.+.sqrt(5.))/.2.;; (*nb d'or*)
let dec x=x-.floor(x);; (*x mod 1*)

(*tri avec tri_fusion*)
let inf1 x y=x.Place<. y.Place;;

(*Construit l'ordonnancement périodique*)
(*coût T log T où T est la période*)
let construct_period T=
  (*nonT est la fréquence de diffusion du message vide*)
  let nonT=
    let r=ref 0. in
```

```
    for i=0 to !nb_messages-1 do
      r:=!r+.1./. !Tau.(i)
    done;
    int_of_float(T*.(float_of_int !nb_canaux -. !r)) in
  (*list_mess contient la liste des messages qu'il faudra trier, chaque
message de la liste ayant une valeur
    de type freq_cum*nb d'or mod 1*)
  let list_mess=ref [] in
  for j=1 to nonT do
    list_mess:={Message1=(-1);Place=dec(float_of_int  j *. theta)}:: !
list_mess
  done;
  let freq_cum=ref nonT in
  for i=0 to !nb_messages-1 do
    (*ni est la fréquence de diffusion du message i*)
    let ni=int_of_float(T/. !Tau.(i)) in
    for j=1 to ni do
      list_mess:={Message1=i;Place=dec(float_of_int (!freq_cum+j) *.
theta)}:: !list_mess
    done;
    freq_cum:=!freq_cum+ni
  done;
  tri_fusion inf1 !list_mess;;

print_string "\nCalcul en cours de l'ordonancement périodique...\n";;
let ordonancement=ref (vect_of_list (construct_period 1.));;
let init_ordonancement n=
  ordonancement:=vect_of_list (construct_period n);;

(*Donne le n_ième message à diffuser*)
(*Coût:1*)
let determine_message3 n=
  let m=vect_length !ordonancement in
  !ordonancement.(n mod m).Message1;;



(************************ ALGO 4*******************)
(******************Utilisation d'arbres************)
let message_arbre_vide()={Message2= -1;Freq=0};;
let inf2=(fun x y->x.Freq<y.Freq);;

(*Parcours donne la feuille indiquée par les couleurs tout en les
schwitchant*)
let rec parcours arb=match arb with
  |Feuille i->i
  |Arbre (g,d)->
    if g.Couleur then (g.Couleur<-false;d.Couleur<-true;parcours g.Branche)
    else (d.Couleur<-false;g.Couleur<-true;parcours d.Branche);;

(*Prend en argument une liste de puissances de 2 (dans Freq) triée par
ordre croissant et dont la somme est une puissance de
  deux et renvoit deux listes de puissances de 2 dont les sommes sont
égales*)
let partage l=
  let sum=list_it (fun x y->x.Freq+y) l 0 in
  constr l [] 0
where rec constr a b s=
  if s=sum/2 then (a,rev b)
  else match a with
```

```
    |[]->failwith "Entrez des puissances de deux par ordre croissant dont
la somme est une puissance de deux!"
    |p::q -> constr q (p::b) (s+p.Freq);;

(*Construit l'arbre à partir d'une liste de puissances de 2 ... *)
let rec construct_arb = fun
  |[p]->Feuille p
  | l ->let (l1,l2)=partage l in Arbre ({Couleur=true;Branche=construct_arb
l1},

{Couleur=false;Branche=construct_arb l2});;

(*approche x par la puissance de 2 immédiatement supérieure*)
let approx x=
  let r=puissance 2 (int_of_float (log x /. log 2.)) in
  if r>=int_of_float (ceil x) then r else 2*r;;

(*Transforme un tableau en liste de puissances de 2 ... *)
let construct_message t trou=
  let n=vect_length t in
  let min=ref t.(0) in
  for i=1 to n-1 do
    if t.(i)<. !min then min:=t.(i)
  done;
  for i=0 to n-1 do
    t.(i)<-t.(i)/. !min
  done;
  let l=ref [] and puiss_cum=ref 0 in
  for i=0 to n-1 do
    let r=approx t.(i) in
    l:= {Message2=i;Freq=r} :: !l;
    puiss_cum:=!puiss_cum+r
  done;
  (*on rajoute les messages vides, si les coûts de diffusion sont non nuls
    Sinon, on voit qu'il vaut mieux rajouter de vrais messages.
    L'algo comblera les trous par le message trou sauf si trou=-2 auquel
cas il
    comble les trous en allant du premier message au dernier - celà
fonctionne mieux si la distribution est de Zipf*)
  let rest=ref ((approx (float_of_int !puiss_cum)) - !puiss_cum) in
  let compteur=ref 1 and num=ref 0 in
  while !rest>0 do
    if !rest mod 2=1 then (if trou= -2 then (l:={Message2= !num;Freq= !
compteur}:: !l;incr num)
                                        else l:={Message2= trou;Freq= !
compteur}:: !l);
    compteur:=!compteur*2;rest:=!rest/2;
  done;
  tri_fusion inf2 !l;;

let arb=ref (Feuille (message_arbre_vide()));;
let init_arbre()=
  let n=vect_length !Tau in
  let freq=make_vect n 0. in
  for i=0 to n-1 do
    freq.(i)<-1./.(float_of_int !nb_canaux *. !Tau.(i));
  done;
  (*Si les couts de diff ne sont pas nuls, on comble les trous par le
message vide*)
  if !couts=make_vect !nb_messages 0. then arb:=construct_arb
(construct_message freq (-2))
```

```
                                        else arb:=construct_arb
(construct_message freq (-1));
 !arb;;

let determine_message4()=(parcours !arb).Message2;;



(************************ ALGO 5******************)
(****************GLOUTON Dynamique******************)

let determine_message5 tabl t= (*tabl représente le tableau des files
d'attentes, cf simul, t représente le temps*)
  let max=ref 0. and indice=ref (-1) in
  for i=0 to !nb_messages-1 do
    let r=sqrt(float_of_int(tabl.(i).Nb_clients*(t+1)) -. tabl.
(i).Attente_cum) -. !couts.(i) in
    if r>=. !max then (max:=r;indice:=i)
  done;
  !indice;;

(********************** ALGO 6 ********************)
(* Algorithme très mauvais utilisé pour le télétexte*)

(*Cet algo consiste à diffuser cycliquement les messages!
  D'où un temps d'attente horrible égal à nombre de messages/2*)

let message_actuel=ref (-1);;
let determine_message6()=
  incr message_actuel;
  !message_actuel mod !nb_messages;;
```

D.4- Pour changer dynamiquement la popularité des messages :

```
(**************Fichiers ChangeDyna.mli et ChangeDyna.ml:*****************)

value reload : float -> unit
and change_popu : float vect -> float vect
and change_couts : float vect -> float vect ref;;

(***Cette partie permet de modifier la popularité des messages et le coût
sans avoir à tout réinitialiser***)

#open "Variables";;
#open "Algorithmes";;

(*Pour remettre les variables à jour en cas de changement de nb_messages ou
nb_canaux*)
let reload T=
  init_popularite(); (*les popularités suivent la loi de Zipf*)
  init_couts(); (*les coûts sont nuls*)
  init_Tau(); (*issu du minorant*)
  init_proba(); (*pour l'algo random*)
  init_ordonancement T; (*pour l'algo du nombre d'or*)
  init_arbre() (*pour l'algo se servant des arbres*);
  print_string "Variables initialisées avec succès!";;

(*Pour pouvoir tester les algos sur d'autres distributions que Zipf*)
(*Pour pouvoir étudier les algos dynamiques*)
let change_popu l=
```

```
  nb_messages:=vect_length l;
  popularite:=normalise l;
  popu_cum:=cumul !popularite;
  (*Réinitialisation des algos qui dépendent évidemment tous de la
popularité!*)
  init_couts(); (*on réinitialise les coûts à 0 parceque le nombre des
messages peut avoir changer*)
  init_Tau();
  init_proba();
  init_ordonancement (!T_nbor);
  init_arbre();
  !popularite;;

let change_couts c =
  if vect_length c <> !nb_messages then failwith "Erreur : il faut entrer
un nombre de coûts égal au nombre de messages!";
  couts:=c;
  init_Tau();
  init_proba();
  init_ordonancement (float_of_int(vect_length !ordonancement));
  init_arbre();
  couts;;
```

D.5- Simulation d'un serveur:

```
(*************Fichiers Simulation.mli et Simulation.ml:*****************)

type popu_dyna={Popu: float vect;Duree: int};;

(*Constantes permettant d'indiquer quel algo on veut utiliser*)
value FIFO : int
and RAND : int
and GLOUTON : int
and NBOR : int
and ARBRE : int
and DYNA : int
and TELETEXT : int;;

value simul : int -> int -> int -> float
and simul2 : int -> int -> float -> int -> float
and theorie : int -> int -> float
and theorie2 : int -> int -> int list * int vect * float vect * float
and simul_all : int -> int -> unit
and simul_dyna : popu_dyna vect -> int -> int -> float -> int -> float
and simul_all_dyna : popu_dyna vect -> int -> float -> int -> unit;;

#open "Variables";;
#open "Minorant";;
#open "Algorithmes";;
#open "ChangeDyna";;

(*Simulation du log des clients*)
(*Variables utilisées:à besoin de Minorant.ml et de Variables.ml (et de
launch.ml pour la partie dynamique*)

(************************************************** Simulation
**************************************************)

(*Utilisé pour la simu. Les clients arrivent en un temps continu, d'où le
type de Arrivee.
```

```
  Numéro est utilisé pour l'algo FIFO*)
type client={Arrivee : float;Numero : int};;

(*Sélection d'un message voulu pour un client*)
let rand_mess()=
  let r=random__float 1. in
  try (
    for i=0 to !nb_messages-1 do
      if !popu_cum.(i)>.r then raise (Trouve i)
    done;
    !nb_messages-1)
  with Trouve i->i;;

(*Gestion des files d'attentes*)
let file_simul_vide()={Nb_clients=0;Attente_cum=0.;Premier=max_int};;
(*Ajoute un client à une file d'attente. Coût:1*)
let add f c=
  f.Nb_clients<-f.Nb_clients+1;
  f.Attente_cum<-f.Attente_cum+.c.Arrivee;
  if c.Numero<f.Premier then f.Premier<-c.Numero;;

(*Calcule le nombre de clients dans le tableau des files d'attentes*)
let nombre_de_clients v=
  let r=ref 0 in
  for i=0 to !nb_messages-1 do
    r:=!r+v.(i).Nb_clients
  done;
  !r;;

(*Ajoute l'attente cumulée,f représentant la file d'attente pour le message
diffusé*)
(*Coût:1*)
let add_attente f t=
  float_of_int(f.Nb_clients*(t+2)) -. f.Attente_cum;; (*t+2 en prenant en
compte le temps de téléchargement*)

(*AJoute l'attente cumulée des clients n'ayant pas été servis à la fin de
la simulation*)
(*tps_attente tableau des files d'attentes, temps; Coût:linéaire en le
nombre de clients*)
let tps_attente_restant v t=
  let r=ref 0. in
  for i=0 to !nb_messages-1 do
    r:=!r+.add_attente (v.(i)) t
  done;
  !r;;
(********************** La simulation:
************************************)

let FIFO=0;;
let RAND=1;;
let GLOUTON=2;;
let NBOR=3;;
let ARBRE=4;;
let DYNA=5;;
let TELETEXT=6;;

let imprimer commentaire i=print_string commentaire;print_float (arrondir
i);print_newline();;
let imprimer_int commentaire i=print_string commentaire;print_int
i;print_newline();;
```

```
(*simul algorithme, longueur de la simulation, nombre de clients se loguant
par unité de temps.
  La variation du nombre_de_clients ne fait varier le temps d'attente que
pour FIFO et GLOUTON_DYNA
   ->utilité de la dissémination de données*)
let simul algo long_simu freq_pers=
  let tabl_attente=make_vect !nb_messages (file_simul_vide()) in
  let etat=make_vect !nb_messages 0. in (*etat sert à glouton, représente
la durée depuis la dernière diffusion*)
    (*Initialisation de la table d'attente, l'initiation plus haut ne
créant qu'une seule référence*)
    for z=0 to !nb_messages-1 do
      tabl_attente.(z)<-file_simul_vide()
    done;
  let nb_clients=ref 0 and tmps_attente=ref 0. in (*Pour calculer le temps
d'attente moyen*)
    for i=0 to long_simu do
      (*log des clients*)
      for j=1 to freq_pers do
        add tabl_attente.(rand_mess()) {Arrivee=float_of_int(i)
+.random__float 1.;Numero= !nb_clients};
        incr nb_clients
      done;
      (*Actualisation de l'état*)
      for j=0 to !nb_messages-1 do
        etat.(j)<-etat.(j)+.1.
      done;
      (*Diffusion dans les canaux*)
      for j=0 to !nb_canaux-1 do
        let message=(if algo=1 then determine_message1() else
                     if algo=2 then determine_message2 etat else
                     if algo=3 then determine_message3 (i* !nb_canaux+j)
else
                     if algo=4 then determine_message4() else
                     if algo=5 then determine_message5 tabl_attente i else
                     if algo=6 then determine_message6 () else
                                    determine_message0 tabl_attente) in
        if message<> -1 then (
          etat.(message)<-0.;
          tmps_attente:=!tmps_attente+.add_attente (tabl_attente.(message))
i +. !couts.(message);
          tabl_attente.(message)<-file_simul_vide();
        )
      done;
    done;
    (*Résultats*)
    (*On a deux méthodes de calcul du temps de service moyen, qui
convergent toutes les deux pour une durée de simu
      assez grande, mais la deuxième est beaucoup plus fiable.
    imprimer "Temps d'attente moyen actuel: " (!tmps_attente /.
float_of_int (!nb_clients-nombre_de_clients tabl_attente));*)
    (!tmps_attente+. tps_attente_restant tabl_attente long_simu) /.
float_of_int !nb_clients;;


(*simul2 est exactement comme simul, sauf que le nombre de clients se
loguant dans une unité de temps n'est
  plus entier mais probabiliste d'espérance freq_pers.
  simul2 est un peu plus lent, c'est pourquoi j'ai conservé simul1
-importance de la rapidité pour des
```

```
  simus se déroulant sur un long intervalle de temps
  enfin, simul2 permet également de présenter les résultats par intervalles
de longueurs de temps, ce
  qui est utile pour tester la variation du temps d'attente lorsque le
comportement des clients change*)
let simul2 algo long_simu freq_pers pallier=
  let client_cum=ref 0. in (*utilisé pour calculer le nombre de clients se
loguant en une unité de temps*)
  let tabl_attente=make_vect !nb_messages (file_simul_vide()) in
  (*Initialisation de la table d'attente, l'initiation plus haut ne créant
qu'une seule référence*)
  for z=0 to !nb_messages-1 do
    tabl_attente.(z)<-file_simul_vide()
  done;
  let etat=make_vect !nb_messages 0. in (*etat sert à glouton*)
  let nb_clients=ref 0 and tmps_attente=ref 0. in
    for i=1 to long_simu do
      (*log des clients*)
      client_cum:=!client_cum+.random__float (2.*.freq_pers);
      while !client_cum>.1. do
        add tabl_attente.(rand_mess()) {Arrivee=float_of_int(i)
+.random__float 1.;Numero= !nb_clients};
        incr nb_clients;
        client_cum:=!client_cum-.1.
      done;
      (*Actualisation de l'état*)
      for j=0 to !nb_messages-1 do
        etat.(j)<-etat.(j)+.1.
      done;
      (*Diffusion dans les canaux*)
      for j=0 to !nb_canaux-1 do
        let message=(if algo=1 then determine_message1() else
                     if algo=2 then determine_message2 etat else
                     if algo=3 then determine_message3 (i* !nb_canaux+j)
else
                     if algo=4 then determine_message4() else
                     if algo=5 then determine_message5 tabl_attente i else
                     if algo=6 then determine_message6 () else
                                    determine_message0 tabl_attente) in
        if message<> -1 then (
          etat.(message)<-0.;
          tmps_attente:=!tmps_attente+.add_attente (tabl_attente.(message))
i +. !couts.(message);
          tabl_attente.(message)<-file_simul_vide();
        )
      done;
      (*résultats intermédiaires:*)
      if i mod pallier=0 then (
        imprimer_int "Pallier :" i;
        imprimer_int "Clients restant à servir:" (nombre_de_clients
tabl_attente);
        imprimer "Attente restante moyenne:" ((tps_attente_restant
tabl_attente i)/.float_of_int(nombre_de_clients tabl_attente));
        print_newline());
    done;
    (*Résultats*)
    (*imprimer "Temps d'attente moyen actuel: " (!tmps_attente /.
float_of_int (!nb_clients-nombre_de_clients tabl_attente));*)
    (!tmps_attente+. tps_attente_restant tabl_attente long_simu) /.
float_of_int !nb_clients
```

```
    (*tmps_attente représente l'attente moyenne de service pour un client,
le reste représente l'attente moyenne des
      clients n'ayant pas encore étés servis*);;


(********************** Attente moyenne théorique ************************)
(*Tous les algos de dissémination de donnés n'ont pas besoin de connaître
quels clients sont logués.
  L'algo ci_dessous calcule le temps d'attente en arrière moyen. On peut
montrer que ce temps converge vers le
  temps de service. C'est ce que l'on observe en pratique lorsqu'on compare
théorie et simul pour une longueur de
  simulation d'environ 100*nb_messages
  En fait, theorie calcule le coût du chemin du graphe des états*)

(*théorie algo à utiliser, longueur de la simulation*)
let theorie algo long=
  (*etat.(i) est le temps d'attente en arrière du message i*)
  let etat=make_vect !nb_messages 0. in
  let cout_tot=ref 0. in
    for i=0 to long do
      (*Actualisation de l'état*)
      for j=0 to !nb_messages-1 do
        etat.(j)<-etat.(j)+.1.
      done;
      (*Diffusion dans les canaux*)
      for j=0 to !nb_canaux-1 do
        let message=(if algo=2 then determine_message2 etat else
                     if algo=3 then determine_message3 (i* !nb_canaux+j)
else
                     if algo=4 then determine_message4 () else
                     if algo=6 then determine_message6 () else
                                    determine_message1()) in
        if message<> -1 then (etat.(message)<-0.;cout_tot:=!cout_tot+. !
couts.(message));
      done;
      cout_tot:=!cout_tot+.1.5; (*+1 pour le temps de téléchargement, +1/2
à cause de la continuité du temps*)
      for j=0 to !nb_messages-1 do
        cout_tot:=!cout_tot +. !popularite.(j)*.etat.(j)
      done;
    done;
    !cout_tot/.float_of_int long;;

(*comme simul2, theorie2 est exactement comme theorie sauf qu'il donne des
résultats plus complets
  en particulier il donne la liste des messages diffuser par l'algo et le
temps d'attente pour chaque message,
  ce qui permet d'étudier le comportement égalitariste ou non des
différents algos...*)
let theorie2 algo long=
  (*etat.(i) est le temps d'attente en arrière du message i*)
  let etat=make_vect !nb_messages 0. in
  let cout_tot=ref 0. in
  let messages_diffusés=ref [] in (*contient la liste des messages
diffusés*)
  let nb_message_vect=make_vect !nb_messages 0
  and cout_message=make_vect !nb_messages 0. in (*pour avoir le temps
d'attente de chaque message*)
    for i=0 to long do
      (*Actualisation de l'état*)
```

```
      for j=0 to !nb_messages-1 do
        etat.(j)<-etat.(j)+.1.
      done;
      (*Diffusion dans les canaux*)
      for j=0 to !nb_canaux-1 do
        let message=(if algo=2 then determine_message2 etat else
                     if algo=3 then determine_message3 (i* !nb_canaux+j)
else
                     if algo=4 then determine_message4 () else
                     if algo=6 then determine_message6 () else
                                    determine_message1()) in
        if message<> -1 then (etat.(message)<-0.;cout_tot:=!cout_tot+. !
couts.(message);
                              nb_message_vect.(message)<-nb_message_vect.
(message)+1;
                              messages_diffusés:=message:: !
messages_diffusés);
      done;
      cout_tot:=!cout_tot+.1.5; (*+1 pour le temps de téléchargement, +1/2
à cause de la continuité du temps*)
      for j=0 to !nb_messages-1 do
        cout_tot:=!cout_tot +. !popularite.(j)*.etat.(j);
        cout_message.(j)<-cout_message.(j) +. etat.(j);
      done;
    done;
    for j=0 to !nb_messages-1 do
      cout_message.(j)<-cout_message.(j) /. float_of_int long +. 1.5 (*+1
pour le temps de téléchargement, +1/2 à cause de la continuité du temps*)
    done;
    (rev !messages_diffusés, nb_message_vect, cout_message, !
cout_tot/.float_of_int long);;

let simul_all long freq=
  print_string "\n";
  imprimer "Minorant: " (calc_min !nb_canaux !popularite !couts);
  print_string "FIFO:\n";
    imprimer "  Temps d'attente moyen asymptotique: " (simul FIFO long
freq);
  print_string "RAND:\n";
    imprimer "  Théorie: " (theorie RAND long);
    imprimer "  Temps d'attente moyen asymptotique: " (simul RAND long
freq);
  print_string "GLOUTON:\n";
    imprimer "  Théorie: " (theorie GLOUTON long);
    imprimer "  Temps d'attente moyen asymptotique: " (simul GLOUTON long
freq);
  print_string "NB OR:\n";
    imprimer "  Théorie: " (theorie NBOR long);
    imprimer "  Temps d'attente moyen asymptotique: " (simul NBOR long
freq);
  print_string "ARBRE:\n";
    imprimer "  Théorie: " (theorie ARBRE long);
    imprimer "  Temps d'attente moyen asymptotique: " (simul ARBRE long
freq);
  print_string "GLOUTON DYNAMIQUE:\n";
    imprimer "  Temps d'attente moyen asymptotique: " (simul DYNA long
freq);
  print_string "TELETEXTE:\n";
    imprimer "  Théorie: " (theorie TELETEXT long);
    imprimer "  Temps d'attente moyen asymptotique: " (simul TELETEXT long
freq);;
```

```
(**************************Simulation
dynamique*************************************)
(*simul_dyna permet de simuler des variations de popularité*)

let simul_dyna popu_vect algo longueur freq_pers pallier=

  let sortie1=open_out ("C:\Temp\ " ^(string_of_int algo)^ "Pallier.txt")
in
  let sortie2=open_out ("C:\Temp\ " ^(string_of_int algo)^
"ClientsRestants.txt") in
  let sortie3=open_out ("C:\Temp\ " ^(string_of_int algo)^
"TmpsAttente.txt") in
  let sortie4=open_out ("C:\Temp\ " ^(string_of_int algo)^
"Popularité.txt") in



  let m=vect_length popu_vect in
  nb_messages:=vect_length popu_vect.(0).Popu;
  let duree=ref 0 in
  for i=0 to m-1 do
    duree:=!duree+popu_vect.(i).Duree
  done;
  (*Calcul de la popularité moyenne*)
  let popu_moyen=make_vect !nb_messages 0. in
  for i=0 to !nb_messages-1 do
    popu_moyen.(i)<-
      (let r=ref 0. in
       for j=0 to m-1 do
         r:=!r+.popu_vect.(j).Popu.(i)*.float_of_int(popu_vect.(j).Duree)/.
(float_of_int !duree)
       done;
       !r)
  done;
  change_popu popu_moyen;
  (*tabl_temps_attente et tabl_clients permettent de calculer le temps de
service à chaque changement
    de la distribution des popularités*)
  let tabl_temps_attente=make_vect (m+1) 0. and tabl_clients=make_vect
(m+1) 0 in

  (*Simulation proprement dite, basée sur Simul2
    On ne peut utiliser directement simul2 car celà reviendrait à vider les
files d'attentes à
    chaque changement de popularité*)
  let client_cum=ref 0. in (*utilisé pour calculer le nombre de clients se
loguant en une unité de temps*)
  let tabl_attente=make_vect !nb_messages (file_simul_vide()) in
  (*Initialisation de la table d'attente, l'initiation plus haut ne créant
qu'une seule référence*)
  for z=0 to !nb_messages-1 do
    tabl_attente.(z)<-file_simul_vide()
  done;
  let etat=make_vect !nb_messages 0. in (*etat sert à glouton*)
  let nb_clients=ref 0 and tmps_attente=ref 0. in
  (*popularité ne sert que pour les algos et le calcul du minorant.
Lorsqu'un client se logue, pour savoir quel
  message il demande, on utilise popu_cum. Donc ici on se sert de
popu_moyen pour les algos, et à chaque changement
```

```
  de distribution de popularité on change uniquement popu_cum, ce qui
change uniquement les habitudes des clients
  dans la simulation*)
  let popu_num=ref 0 in let popu_change=ref (popu_vect.(!
popu_num).Duree*longueur/ !duree) in
  for i=1 to longueur do
    (*log des clients*)
    client_cum:=!client_cum+.random__float (2.*.freq_pers);
    while !client_cum>.1. do
      add tabl_attente.(rand_mess()) {Arrivee=float_of_int(i)
+.random__float 1.;Numero= !nb_clients};
      incr nb_clients;
      client_cum:=!client_cum-.1.
    done;
    (*Actualisation de l'état*)
    for j=0 to !nb_messages-1 do
      etat.(j)<-etat.(j)+.1.
    done;
    (*Diffusion dans les canaux*)
    for j=0 to !nb_canaux-1 do
      let message=(if algo=1 then determine_message1 () else
                   if algo=2 then determine_message2 etat else
                   if algo=3 then determine_message3 (i* !nb_canaux+j) else
                   if algo=4 then determine_message4 () else
                   if algo=5 then determine_message5 tabl_attente i else
                   if algo=6 then determine_message6 () else
                                  determine_message0 tabl_attente) in
      if message<> -1 then (
        etat.(message)<-0.;
        tmps_attente:=!tmps_attente+.add_attente (tabl_attente.(message)) i
+. !couts.(message);
        tabl_attente.(message)<-file_simul_vide();
      )
    done;
    (*résultats intermédiaires:*)
    if i mod pallier=0 then (
      imprimer_int "Pallier :" i;
      imprimer_int "Clients restant à servir:" (nombre_de_clients
tabl_attente);
      imprimer "Attente restante moyenne:" ((tps_attente_restant
tabl_attente i)/.float_of_int(nombre_de_clients tabl_attente));
      print_newline();

    output_string sortie1 (string_of_int i);output_string sortie1 "\n";
    output_string sortie2 (string_of_int ((nombre_de_clients
tabl_attente)));output_string sortie2 "\n";
    output_string sortie3 (string_of_float ((tps_attente_restant
tabl_attente i)/.float_of_int(nombre_de_clients
tabl_attente)));output_string sortie3 "\n"
    );

    (*changement de la popularité des messages:*)
    if i= !popu_change then (
      tabl_clients.(!popu_num+1)<- !nb_clients - tabl_clients.(!popu_num)-
(nombre_de_clients tabl_attente);
      tabl_temps_attente.(!popu_num+1)<- !tmps_attente+.
(tps_attente_restant tabl_attente i)-. tabl_temps_attente.(!popu_num);
      imprimer ("  Popularité n°"^string_of_int (!popu_num+1)^" ")
               (tabl_temps_attente.(!popu_num+1)/. (float_of_int
(tabl_clients.(!popu_num+1)+(nombre_de_clients tabl_attente))));
      print_newline();
```

```
      output_string sortie4 (string_of_float (tabl_temps_attente.(!
popu_num+1)/. (float_of_int (tabl_clients.(!popu_num+1)+(nombre_de_clients
tabl_attente)))));output_string sortie4 "\n";

      if i<>longueur then (
        popu_cum:=cumul(normalise popu_vect.(!popu_num).Popu);
        incr popu_num;popu_change:= !popu_change + popu_vect.(!
popu_num).Duree*longueur/ !duree))
  done;
  (*Résultats*)
  output_string sortie4 (string_of_float ((!tmps_attente+.
(tps_attente_restant tabl_attente longueur)) /. (float_of_int !
nb_clients)));output_string sortie4 "\n";
  close_out sortie1;
  close_out sortie2;
  close_out sortie3;
  close_out sortie4;

  (!tmps_attente+. (tps_attente_restant tabl_attente longueur)) /.
(float_of_int !nb_clients);;

let simul_all_dyna popu_vect longueur nb_clients pallier=
  print_string "FIFO:\n";
    imprimer "  Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
FIFO longueur nb_clients pallier);
  print_string "\nRAND:\n";
    imprimer "  Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
RAND longueur nb_clients pallier);
  print_string "\nGLOUTON:\n";
    imprimer "  Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
GLOUTON longueur nb_clients pallier);
  print_string "\nNB OR:\n";
    imprimer "  Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
NBOR longueur nb_clients pallier);
  print_string "\nARBRE:\n";
    imprimer "  Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
ARBRE longueur nb_clients pallier);
  print_string "\nGLOUTON DYNAMIQUE:\n";
    imprimer "  Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
DYNA longueur nb_clients pallier);;
```

D.6- Pour lancer le programme:

```
(*************Fichiers Launch.mli et Launch.ml:********* *************)

value launch : unit -> unit
and result : int -> int -> float -> int -> int -> int -> float
and enregistre : int -> int -> int -> unit;;

(*************Permet de charger les modules et d'initialiser les
constantes*******************)

directory "C:\Documents and Settings\Dams\Mes documents\Tipe\Dissémination
de Données";;
#open "Variables";;
#open "Algorithmes";;
#open "ChangeDyna";;
#open "Simulation";;
```

```
random__init (int_of_float (sys__time()));;
let io s=print_string s;print_newline();
         int_of_string(read_line());;
let io_float s=print_string s;print_newline();
               float_of_string(read_line());;

(*Pour initialiser les variables*)
let launch()=
  nb_canaux:=io "Entrez le nombre de canaux: ";
  nb_messages:=io "Entrez le nombre de messages: ";
  T_nbor:=io_float "Période de l'ordonancement? ";
  reload !T_nbor;
  print_string "\nPour démarrer la simulation: simul_all longueur
nb_de_clients";
  print_string "\nRemarque: Pour avoir des résultats fiables, il est
recommandé que";
  print_string "\nlongueur=10*nb_messages voire 100*nb_messages";
  print_string "\nLe minorant est asymptotique est n'est valabe que pour
les alortithmes de dissémination de données";
  print_string "\nDes algorithmes peuvent donc avoir un temps de service
inférieur au minorant!";;


(* Comme launch sauf qu'on entre directement la valeur des variables en
argument *)
let result nbcanaux nbmessages Tnbor algo longsimu freqsimu=
  nb_canaux:=nbcanaux;
  nb_messages:=nbmessages;
  reload Tnbor;
  simul algo longsimu freqsimu;;



(**Utilisé pour faire des courbes à partir des données de la simulation**)
let enregistre debut fin step=
  let sortie1=open_out "C:\Temp\Simul2\Nombre de clients1.txt" in
  let sortie2=open_out "C:\Temp\Simul2\Glouton1.txt" in
  let pos_GLOBAL=ref debut in
  while !pos_GLOBAL<fin do
    output_string sortie1 (string_of_int !pos_GLOBAL);output_string sortie1
"\n";
    let r=result 1 100 (float_of_int(!pos_GLOBAL*10)) GLOUTON 10000 !
pos_GLOBAL in
    output_string sortie2 (string_of_float (arrondir r));output_string
sortie2 "\n";
    pos_GLOBAL:=!pos_GLOBAL+step;
  done;
close_out sortie1;
close_out sortie2;;

print_string "\nPour lancer: launch()\n";;
```

Appendice E:
Etat de l'art, bibliographie.

Résultats connus

Tout le problème consiste à trouver un bon ordonnancement des messages afin de minimiser le temps d'attente des clients. Or même si on considère que les messages à diffuser sur un canal (avec coût de diffusion) sont de longueur uniforme, trouver l'ordonnancement optimal est NP-difficile. On recherchera donc des algorithmes polynomiaux qui soient des α-approximations[1].

Dans le cas des messages de longueur uniforme, Gecsei démontre en 1983 que si on tire aléatoirement les messages à diffuser, il faut alors les tirer avec une probabilité proportionnelle à la racine carrée de leur popularité. En 1984, Ammar et Wong [AW85] obtiennent, pour leur étude sur le télétexte, un minorant du temps d'attente dans le cadre d'un ordonnancement périodique. Ils démontrent ainsi que l'algorithme de Gecsei est une 2-approximation. Leur minorant leur donnant des fréquences optimales de diffusion, ils l'utilisent dans un algorithme qui construit un ordonnancement périodique en optimisant en priorité les messages les plus populaires. Cette approche sera reprise par Schabanel [Sch00] en 2000 (diffuser optimalement les messages les plus importants), qui aboutira ainsi à un schéma d'approximation polynomial[2].

Ammar et Wong proposent également deux autres approches déterministes : une gloutonne, qui consiste à diffuser le message dont l'attente cumulée est la plus importante, et l'autre basée sur une propriété du nombre d'or, qui leur servira à construire un ordonnancement périodique : les multiples de φ modulo 1 découpent le plus uniformément possible [0 ; 1]. Ces algorithmes seront étendus en 1998 au cas de plusieurs canaux par Bar-Noy, Bhatia, Noar et Schieber, qui démontrent également que le glouton est la dérandomisation gloutonne de l'algorithme de Gecsei et donc une 2-approximation, tandis que l'algorithme basé sur le nombre d'or est une 9/8 approximation. Enfin, en 1995, Anily, Glass et Hassin montrent qu'il existe un ordonnancement optimal qui est périodique en modélisant la diffusion des messages par un graphe.

La mise en place pratique de la dissémination de données pose d'autres problèmes, en particulier l'application à Internet. Ainsi, l'approche usuelle de la méthode pseudo-interactive consiste en l'actualisation du cache des serveurs locaux, ce qui nécessite de prendre en compte la topologie du réseau et l'origine géographique des requêtes (voir [BC96]). Pour un état de l'art des différents protocoles possibles, voir [FZ97].

Bibliographie :

[AW85] : Ammar et Wong, The Design Of Teletext Broadcast Cycles, 1984.

Dans cet article, Ammar et Wong cherchent à optimiser les temps d'attentes du Télétexte. Ils définissent pour la première fois le temps de service d'un ordonnancement cyclique de

[1] La solution donnée par l'algorithme est de coût d'au plus α fois le coût optimal.

[2] Pour tout ε, il construit un algorithme polynomial qui est une (1+ ε)-approximation.

message et trouvent un minorant de ce temps de service. Enfin, ils donnent un algorithme (dont le fonctionnement ressemble à celui du nombre d'or) visant à atteindre ce minorant.

[BC96] : Bestavros et Cunha : Server-initiated Document Dissemination for the WWW, IEEE Data Engineering Bulletin, 1996.

Dans cet article, les auteurs étudient la topologie du réseau. En particulier, ils montrent que les pages les plus demandées ne représentent qu'une faible proportion de l'ensemble des pages du réseau, ce qui justifie pleinement l'intérêt de la dissémination de données appliquées à Internet. Enfin, ils montrent que la structure de popularité des pages Internet vérifie bien la loi de Zipf, ce qui valide les hypothèses que nous avons faites sur les popularités dans notre simulation.

[FZ97] : Franklin et Zdonik, A Framework for Scalable Dissemination-Based Systems, 1997.

Dans cet article, les auteurs comparent plusieurs protocoles de transfert des donnés, en particulier le protocole pull (ie : les données sont échangées ssi le client le demande) et le protocole push (ie : les données susceptibles d'être demandées par le client sont automatiquement téléchargées par le logiciel d'accès à Internet). Ensuite, ils comparent les méthodes possibles de transmission des informations, en particulier Unicast (le serveur envoie l'information à un seul client à la fois) et 1 to N (le serveur peut envoyer l'information à un nombre quelconque de clients en même temps). Rappelons que la dissémination de données repose sur l'idée d'un canal diffusant périodiquement les messages demandés, donc est basée sur l'approche push & 1 to N. Les auteurs étudient ensuite les difficultés pratiques à mettre en place un tel protocole (ie : comment peut-on créer un tel canal de diffusion ?).

[Sch00] : Schabanel, Algorithmes d'approximation pour les télécommunications sans fil : Ordonnancement pour la dissémination de données et Allocation statique de fréquences, thèse présentée en 2000.

Dans cette thèse, Schabanel rappelle d'abord les résultats obtenus sur la dissémination de messages ayant même longueur. En particulier, il montre que les deux définitions du temps de service (temps d'attente en avant ou temps d'attente en arrière) sont équivalents. Dans une deuxième partie, il étudie le cas des messages de longueurs différentes et il obtient une généralisation du minorant qu'Ammar et Wong avaient obtenus dans le cadre des messages de longueur uniforme. Dans une troisième partie, il étudie le cas ou l'on peut découper les messages en plusieurs morceaux de mêmes longueurs. Enfin il donne un schéma d'approximation polynomial pour la dissémination de messages de longueur uniforme[3].

[3] ie :il construit pour tout ε un algorithme polynomial réalisant une 1+ ε approximation.