Partitionnement de données sur grands clusters hétérogènes

Christophe Cérin

christophe.cerin@lipn.univ-paris13.fr

Longue liste de collaborateurs : Michel Koskas, Jean-Christophe Dubacq, Mohamed Jemni, Hazem Fkaier, Jean-Louis Roch...

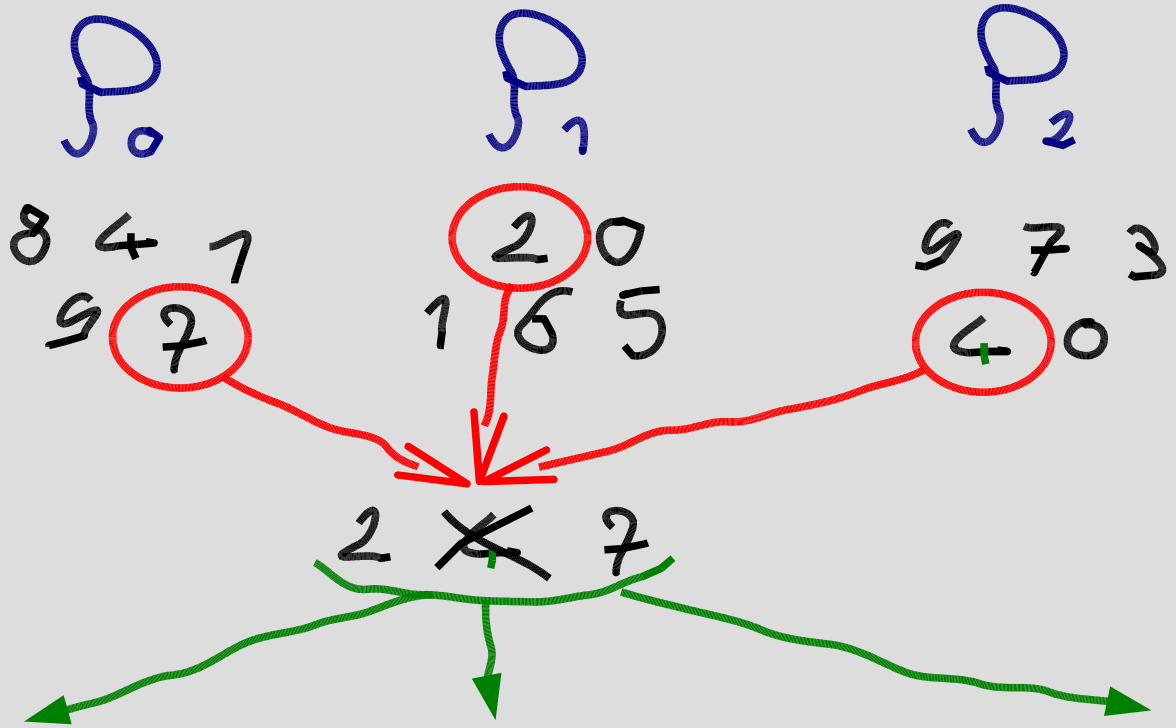
Partitionnement de données sur grands clusters hétérogènes

- Modèle architectural : 1000 PC (dans un centre de calcul ou sur Internet). Initialement les données sont réparties sur les disques locaux (par ex. selon la « puissance » du processeur)
- Statiquement, on veut ajuster/contrôler les volumes de données échangés en les Procs
- Deux applications : le tri et la découverte des épisodes fréquents

Propriétés/Hypothèses pour le méta-schéma de partitionnement

- le coût de passage de N à N' : O(1) c.à.d pas d'overhead liée à la redistribution ;
- un modèle qui s'exprime sur quelques paramètres (Mhz, Bandwidth sont supposés garantis constants) : information statique ;
- général : doit pouvoir s'appliquer pour la plus « grande classe de problèmes possible »
- solutions implémentables (tenir compte aussi de la réalité des machines)

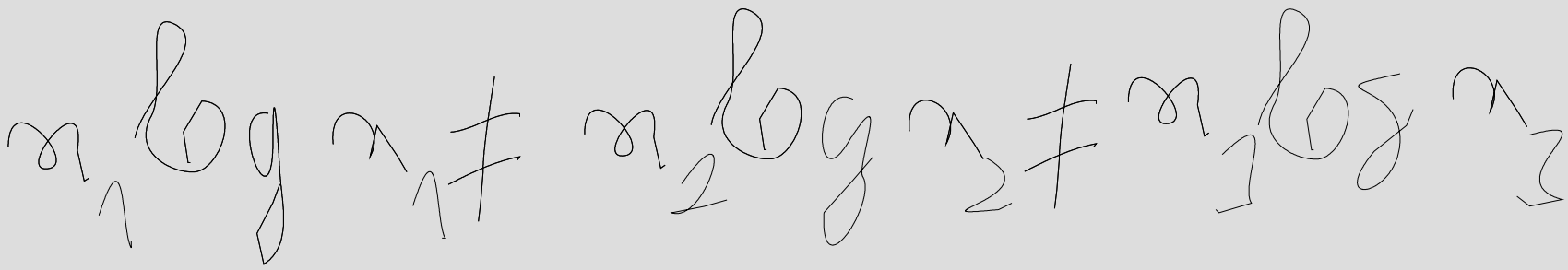
Part 1: il n'y a que les processeurs qui sont hétérogènes



- redistribution / pivots
- tri local ($n \log n$)

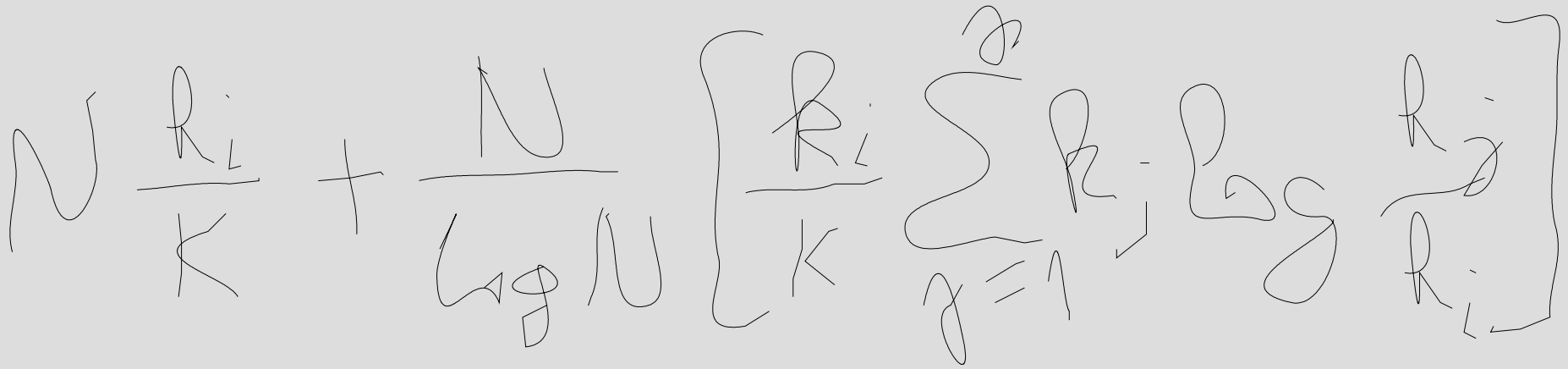
CPU hétérogènes

- On sélectionne le nombre de candidats et de pivots en fonction de la vitesse de chacun (simulation d'une machine virtuelle + utilisation d'un résultat de probabilité) : garantie sur la taille des partitions obtenues AVANT le tri final ;
- Ou est le problème ? (cf cond. initiale)



Résolution du problème

- On part de l'égalité précédente ;
- On fait un développement à l'ordre 1 du log
- On fait quelques hypothèses pour simplifier les calculs
- On obtient un terme (qui se *calcule* en temps constant) :



Avantages / inconvénients

- Coût : O(1)
- La technique ne s'applique que si la fonction de coût qui suit le partitionnement admet un DL ;
- NEW (avril 2005) : on sait aussi faire au même coût (O(1)) si la fonction admet un inverse => nouvelle technique ;

Détails sur la technique 2

$$T = \frac{\tilde{f}(n_1)}{k_1} = \frac{\tilde{f}(n_2)}{k_2} = \cdots = \frac{\tilde{f}(n_p)}{k_p} \tag{7}$$

and equation

$$n_1 + n_2 + ... + n_p = N \tag{8}$$

Let us recall that monotonous increasing functions can have an inverse function. Therefore, for all i, we have $\tilde{f}(n_i) = Tk_i$, and thus:

$$n_i = \tilde{f}^{-1}(Tk_i) \tag{9}$$

Therefore, we can rewrite (8) as:

$$\sum_{i=1}^{p} \tilde{f}^{-1}(Tk_i) = N \tag{10}$$

Fonctions multiplicatives

$$N = \sum_{i=1}^{p} \tilde{f}^{-1}(Tk_i) = \sum_{i=1}^{p} \tilde{f}^{-1}(T)\tilde{f}^{-1}(k_i) = \tilde{f}^{-1}(T) \sum_{i=1}^{p} \tilde{f}^{-1}(k_i) \tag{11}$$

We can then extract the value of T:

$$\tilde{f}^{-1}(T) = \frac{N}{\sum_{i=1}^{p} \tilde{f}^{-1}(k_i)} \tag{12}$$

Combining it with (9) we obtain:

$$n_i = \tilde{f}^{-1}(Tk_i) = \tilde{f}^{-1}(T)\tilde{f}^{-1}(k_i) = \frac{\tilde{f}^{-1}(k_i)}{\sum_{i=1}^{p} \tilde{f}^{-1}(k_i)} N \tag{13}$$

Fonctions polylog

In the case $f(n) = n \ln n$, the inverse function can be computed. It makes use of the Lambert W function $W(x)$, defined as being the inverse function of xe^x. The inverse of $f : n \mapsto n \ln n$ is therefore $g : x \mapsto x/W(x)$.

The function $W(x)$ can be approached by well-known formulas, including the ones given in [15]. A development to the second order of the formula yields:

$$W(x) = \ln x - \ln \ln(x) + o(1) \tag{14}$$

and also

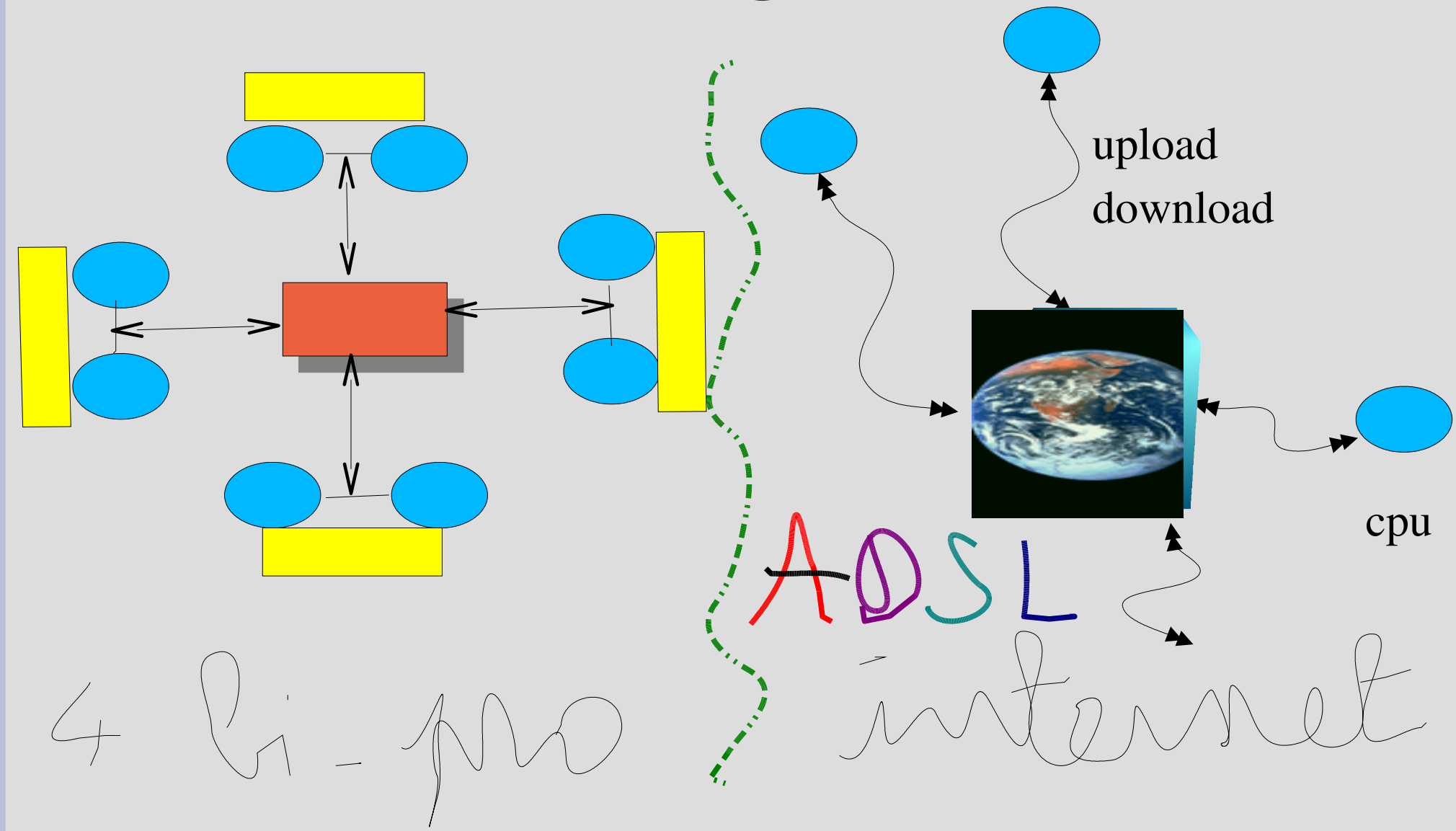
$$\frac{x}{W(x)} = \frac{x}{\ln(x)} \frac{1}{1 - (\ln \ln(x)/\ln(x)) + o(1)} = \frac{x}{\ln(x)} \left(1 + \frac{\ln \ln(x)}{\ln(x)} + o\left(\left(\frac{\ln \ln(x)}{\ln(x)}\right)^2\right)\right) \tag{15}$$

This approximation leads us to the following second-order approximation, that can be used to numerically compute the value of T:

Theorem 2 *Initial values of n_i can be computed by*

$$\sum_{i=1}^{p} \frac{Tk_i + Tk_i \ln \ln(Tk_i)}{(\ln(Tk_i))^2} = N \text{ and } n_i = \frac{Tk_i + Tk_i \ln \ln(Tk_i)}{(\ln(Tk_i))^2}$$

... et maintenant le réseaux est hétérogène



Le cas « clusters/clusters »

- Dans la modélisation : attention au mélange des genres ;
- Idée : découpler les phases de communication et de calcul (barrière)
- B = comunication cost in the cluster
- k.B = communication cost outside the cluster
- Cost model: max{max{n_i,j/B},max{n_i,j/k.B}}
- On fait l'hypothèse (m log m)/k = cte (cf prec)

Exemple numérique

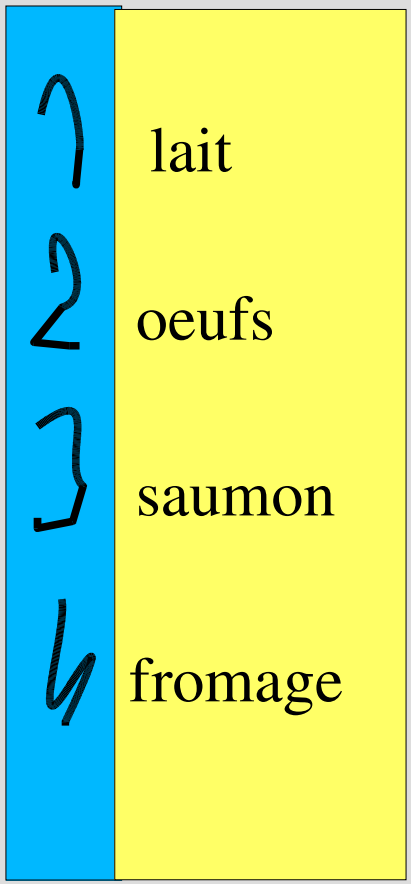
Le cas Internet

- On exprime le temps de transfert vers M_i en fonction du débit d'upload et de download, de ce que l'on avait (d_i) => A_i,j = _i
- On exprime le temps de tri en fonction de A_i,j et A_i,i ... qui est egal a ß_i
- On exprime les contraintes (_i, ß_i ne dépendent pas de i....)
- P^2 +p(p+1) inconnues, 4p-1 contraintes... système qui se résoud.

Part 2: recherche des épisodes fréquents (data mining)

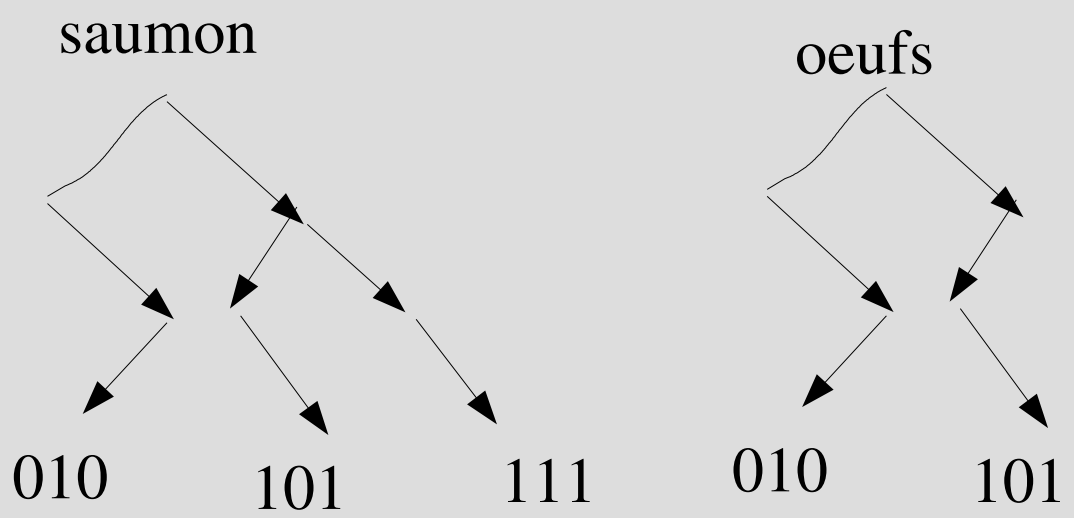
- On veut un algorithme parallèle pour des clusters de clusters et/ou grille hétérogènes en terme de CPU et de réseaux => problématique sous étudiée
- On va montrer comment les idées précédentes peuvent s'adapter à ce cas ;
- Autre propriété de l'algorithme parallèle : les échanges de données ne concernent que peu de « choses » : checkpointing est léger

Principe de l'algorithme (processeurs homogènes)



① la base est distribuée

②



Principe de l'algorithme (processeurs homogènes)

- DEBUT: localement, chaque processeur calcul le support de ces items
- on fait un échange total des supports pour connaître le support global de chaque item
- il garde les items dont les supports > seuil (on a fabriqué les 1-itemsets) ;
- GOTO DEBUT en considérant les 2, puis 3 puis.. itemsets fréquents obtenus par intersection d'arbres de radicaux.

Remarques / précisions

- Les arbres sont de hauteur fixée ; (pb lié au stockage disque)
- En cas de panne : c'est pas des « bouts de base » qui sont échangées mais des entiers => checkpoint est de taille réduite !
- L'algorithme = que de l'intersection
- L'intersection se parallélise naturellement
- MAIS quid de la localité spaciale/temporelle?

Exemple d'exécution (PIV 3Ghz)

0: 502395
9: 114575
13: 86168
14: 199549
15: 120620
19: 146478
21: 78889
22: 278030
23: 145346
24: 80816
29: 79949
30: 187984
31: 304319
39: 98470
Nb 1-itemset=13, rc=2
2-itemset: (0, 14) of size: 170422
2-itemset: (0, 15) of size: 111327
2-itemset: (9, 19) of size: 81877
2-itemset: (0, 21) of size: 77907
2-itemset: (0, 22) of size: 258730
2-itemset: (14, 22) of size: 90510
2-itemset: (15, 22) of size: 85564
2-itemset: (0, 23) of size: 107795
2-itemset: (14, 23) of size: 69099
2-itemset: (9, 29) of size: 79943
2-itemset: (19, 29) of size: 79943
2-itemset: (0, 30) of size: 168395
2-itemset: (22, 30) of size: 94153
2-itemset: (0, 31) of size: 254331
2-itemset: (14, 31) of size: 91325
2-itemset: (22, 31) of size: 127894
2-itemset: (23, 31) of size: 79322
Nb 2-itemset=17, rc=2

3-itemset: (0, 14, 22) of size: 85273
3-itemset: (0, 14, 31) of size: 77368
3-itemset: (0, 15, 22) of size: 81396
3-itemset: (9, 19, 29) of size: 79943
3-itemset: (0, 22, 30) of size: 90057
3-itemset: (0, 22, 31) of size: 122018
Nb 3-itemsets=6, rc=2
Nb 4-itemsets=0, rc=2

Total number of lines (elements) involved in intersections : 89342969

[cerin@taipei DEVEL]$ time ./a.out > toto

real 0m17.970s
user 0m16.471s
sys 0m0.072s

Action Concertée Incitative [ACI]
Globalisation des Ressources Informatiques
et des Données [GRID]

Détails d'implémentation

- Instructions x86-32 utilisées : prefetchT0, movdqu, pand & Co.
- Instructions PowerPC / Altivec utilisées : lvx, vand & Co, stx, dcbtst (Data Cache Block Touch for Store)

http://publibn.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixassem/alangref/dcbtst.htm

GCC 4.1 peut-il faire mieux ?

- Nouvelle manière de générer le code (pb avec la gestion des paramètres de fonctions ?)
- Nouvelle options de vectorisation :
  -ftree-vectorize => pour utiliser automatiquement les inst. MMX, SSE, Altivec
- Exemple de résultat avec GCC 4.1 : très prometteur, mais...

Vectorisation GCC 4.1 (experimental)

```
#include<stdio.h>
#include <stdlib.h>

main(){

 int i;
 unsigned char a[2048];
 unsigned char b[2048];
 unsigned char c[2048];

 srand(13);

 for(i=0;i<2048;i++) {
  a[i] = (unsigned char) (255.0*rand()/(RAND_MAX+1.0));;
  b[i] = a[i]/2;
 }

 for(i=0;i<2048;i++) {
  c[i] = b[i] & a[i];
 }

 for(i=0;i<2048;i++) {
  printf("%dl",c[i]);
 }

}
```

gcc -S -O4 -ftree-vectorize -msse2 -ftree-loop-linear essai.c

```
.L4:
        movl    %edx, %eax
        incl    %edx
        sall$4, %eax
        cmpl    $129, %edx
        movdqa    -2088(%ebp,%eax), %xmm0
        pand    -16(%edi,%eax), %xmm0
        movdqa    %xmm0, -16(%esi,%eax)
        jne .L4
        movl    $1, %ebx
        .p2align 4,,15
```

Vectorisation GCC 4.1 (experimental)

```
cerin@linux:~> gcc -S -O4 -ftree-vectorize -msse2 -ftree-loop-linear
-fprefetch-loop-arrays -falign-loops -fspeculative-prefetching essai.c
```

pas de prefetch !

Retour à des arguments théoriques...

- Clément (Julien) - Arbres Digitaux et Sources Dynamiques. - Thèse de doctorat de l'université de Caen (sept 2000).
- Tries <> Arbres de radicaux <> Bitsets
- Dans la thèse, le coût de l'intersection est étudié selon différentes lois de probabilité (Poisson..) mais ce qui nous intéresse c'est une loi constante !

Qualités de la bibliothèque

- Efficace (ASM) sur les plate-formes courantes du marché ;
- On veut qu'un item puisse apparaître sur au plus 2^50 = 1125899906842624 lignes
- Il faut une hiérarchie de bitsets – Comment stocker sur disque ? Combien de niveaux dans un fichier => combien de fichiers et de répertoires ? Quel impact sur l'OS et le SF ?

Choix possible...

On découpe les 50 bits en 2 => 25 bits poids faibles + 25 bits poids forts (4Mo pour la racine)

Il y a pour un mot donné d'une colonne donnée un arbre racine et 2^25 arbres-feuilles au pire.

Ces arbres feuilles sont regroupés par 1024 dans des fichiers.

Le nombre total de fichiers est donc au pire 1 + 2^25/1024 = 32769 fichiers.

Retour sur l'hétérogénéité (CPU)

- Initialement la base est découpée en bouts de taille proportionnelle à la vitesse du CPU ;
- On construit les 1-itemsets par des insertions en O(h) : linéaire en fonction du nombre de lignes => pas d'ajustement/rééquilibrage à faire.
- Pour générer les (k+1)-itemsets, on sait que l'on a X intersections à calculer sur des « ensembles » de taille t_i

Retour sur l'hétérogénéité

- Comment répartir les calculs pour finir en même temps ?
- Cela dépend du nombre moyen d'accès disques et d'opérations pand nécessaires pour intersecter deux « arbres » au format précédent ;
- Cette complexité est inconnue (pour l'instant)
- On remarque aussi qu'il y a plus de dynamicité (le X varie à chaque nouvelle génération)... donc il faudrait faire un calcul de rééquilibrage à chaque itération ! (alg. polyphase versus 1 phase pour tri)

Conclusion