Efficient Data-Structures and Parallel Algorithms for Association Rules Discovery

Christophe Cérin
Michel Koskas
Gaël Le Mahec
Jean-Sébastien Gay

cerin@laria.u-picardie.fr
koskas@laria.u-picardie.fr
lemahec@laria.u-picardie.fr
jgay@laria.u-picardie.fr
Outline

Context
  - Large Scale Systems
  - Challenging Applications
  - Association Rule Discovery

Data Mining
  - New Data Structure
  - New Algorithms

Experimental results
  - Our Library - Preliminary results

Conclusion & further work
  - Impl. choices - FT - SQL service
Two types of distributed systems:

1. **P2P**
   - 1–100
   - stable
   - trust
   - identity

2. **Internet Computing**
   - "large scale"
   - 100,000
   - volatile
   - no identity
   - no trust

3. **"grid computing"**
**GRID INITIATIVE IN FRANCE**

ACI Masse de données
1M EUR

70 persons
3 years term project

4 sub-projects

**French National Project: Grid’5000**

Good properties of our approach for mining:

- DB scanned once
- Few and short messages
**INPUT:** list of tickets
(A) champagne 13.45 EUR
(B) appetizer  4.15 EUR
(C) salmon    7.10 EUR

**Ex:** if (AB) occurs 4000 times whereas (ABC) occurs 3000 times, produce: “if AB occurs, then there is a probability of 75% that C occurs too”
**PROBLEM DESCRIPTION**

⇒ Alg. for generating rules:

\[
\text{for all frequent sequences } \beta \text{ do }
\]

\[
\text{for all subsequences } \alpha < \beta \text{ do }
\]

\[
\text{conf} = \frac{fr(\alpha)}{fr(\beta)}
\]

\[
\text{if } \text{conf} > \text{min}_\text{conf} \text{ then }
\]

\[
\text{output } \alpha \Rightarrow \beta
\]

\[
\text{output } \text{conf}
\]

⇒ Main problem: discovering frequent episodes: rare in huge files

* (no enumeration, please!)
**KEY FACT** of parallel / seq. algorithms

- Count the occurrence of each item (cancel if $support < XX \rightarrow 1$-itemsets)
- From 1-item sets, generate 2-itemsets (cancel if $support < XX \rightarrow 2$-itemsets)
- Iterate until producing the last $k$-itemset

**OUR APPROACH:** efficient ADT for counting, candidate ($k$-itemset) generation

\[ \text{Radix Trees} \]
For each item, we keep the line number where it appears.

**Avantages:**
- Complexity of a search linked to the tree height - space - natural parallelism in tree management.
RADIX TREE OPERATIONS (support = intersect)

Union of Radix Tree.

Intersection of Radix Tree.
Problems: load-balancing; maximum number of allowed threads;

Current implementation: constant $k$ (number of maximum threads to be started) in Radix-Tree Class;
Heuristic (# thread > 2): start a thread on a right child - left thread runs until it encounters a leaf.
Goal 1: limit busy waiting.

Goal 2: limit pthread_create calls.

Here: divide by 2 (complete binary tree).
✓ Two informations stored in Radix Trees
✓ We focus on 2-itemset generation
✓ Important note: operations on Radix Trees only!
**MAIN PARALLEL ALGORITHM**

Algorithm executed on each Proc. $0 \leq i \leq p$.

/* Initially, each processor has locally $n/p$ lines of the transaction database where $n$ is the total number of lines and $p$ is the processor number.*/

1- In parallel for each processor:
   Scanning of the local database for construction of 1-itemset tree.

2- In parallel for each processor:
   do
   Broadcast supports.
   /* This part can be de-synchronized */
   /* to perform overlapping (see above) */
   Wait for all supports from others.
   Perform the sum reductions.
   Elimination of unsufficient itemsets support.
   $L_k = \text{rest of } C_k$
   Construction of new candidates sets $C_{k+1}$.
   while ($C_{k+1} \neq \emptyset$)
3- frequent itemsets = $\bigcup L_k$
Bench (1 thread) - SUN bi-Opteron v20z

Time (sec.)

- Lists
- Radix Trees

# items

10 20 30 40 50 60 70 80 90 100 110 120 130 140 150
Two threads > one thread
(http://www.boost.org thread impl.)

Bitset, Hierarchy of bitsets?

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <bitset>
using namespace std;

int main(int nb, char **arg) {
  ifstream fin;
  ofstream fout;
  int i;

  bitset<4294967> aaaa(0);
  bitset<4294967> bbbb(1);
  bitset<4294967> cccc(0);

  for(i=0;i<600;i++) {
    cccc = aaaa | bbbb;
    aaaa[i]=1;
  }
  for(i=0;i<600;i++) {
    cccc = aaaa | bbbb;
    aaaa[2*i]=1;
  }
  for(i=0;i<600;i++) {
    cccc = aaaa | bbbb;
    aaaa[3*i]=1;
  }
  // cout << cccc << "\n";
  return 1;
}
```

Properties

- TIME
- MEMORY SPACE
- EASY TO MANAGE
- DISK SPACE
New approaches for computing candidates

Parallel Alg. + Multithreading of Radix tree operations

MPI code available soon (+ MPI-V → FT MPI developed by F. Cappello in the GridExplorer Initiative)

How to represent Radix Trees (vector of bits?) → efficient library for yet another application in the project:

SQL service
Efficient Data-Structures and Parallel Algorithms for Association Rules Discovery

www.laria.u-picardie.fr/~cerin/zeta.html

cerin@laria.u-picardie.fr
koskas@laria.u-picardie.fr