Searching frequent itemsets by clustering data

Towards a parallel approach using MapReduce

Maria Malek Hubert Kadima
LARIS-EISTI
Ave du Parc, 95011 Cergy-Pontoise, FRANCE
maria.malek@eisti.fr, hubert.kadima@eisti.fr

Introduction

- Applying association rules mining leads to find relationships between items in large data bases that contain transactions.
- The problem of frequent itemsets has been introduced by *Agrawal in 1993*.
- **This Apriori algorithm** is based on the *downward closure propriety*:
  - if an itemset is not frequent, any superset of it will not be frequent.
- The Apriori algorithm performs a breadth-first search in the search space by generating candidates of length $K + 1$ from frequent k-itemsets.

Frequent itemsets - example

Transactions

100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Results

- If minSupp=2 :
- frequent itemsets :
  - $L_1 = \{1, 2, 3, 5\}$, $L_2 = \{13, 23, 25, 35\}$,$L_3 = \{235\}$.
- Association Rules :
  - R_1 : $2 \longrightarrow 35$ avec $conf = \frac{2}{3}$,
  - R_2 : $3 \longrightarrow 5$ avec $conf = \frac{2}{3}$, etc.

Downward closure propriety

Propriety

Let X_k be a frequent itemset, all frequent itemsets included in X_k are frequent.

1. If ABCD is a frequent itemset,
2. then ABC,ABD, BCD, AB,AC,BC,BD,CD,A,B,C,D are frequent itemsets .

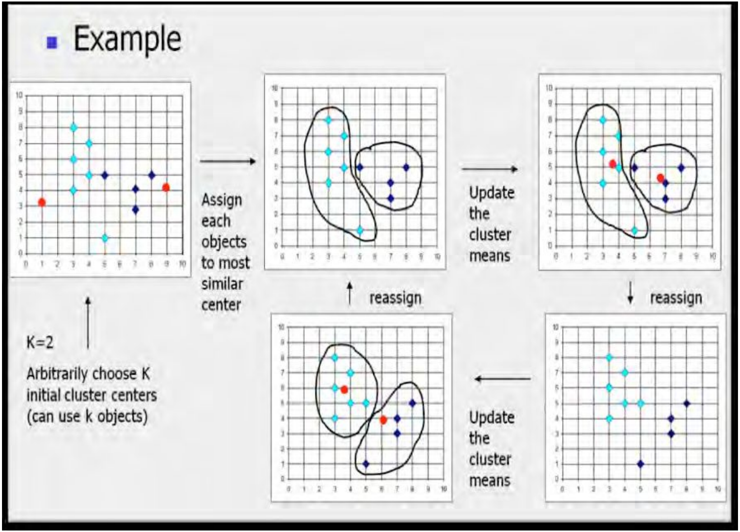
Related work

- **The Apriori algorithm** is based on storing database in the memory.
- When the dataset size is huge, both the memory use and the computational cost still be expensive.
- The idea is to use a given data structure to achieve a condensed representation of the data transactions (like trees or intervals).
- Other algorithms (fasters than Apriori):
  - **FPGrowth**(*J.Han, J. Pei, and Y. Yin -2000*),
  - **Depth first implementation of Apriori** (*W. A. Kosters and W. Pijls -2003*),
  - **A massively parallel FP-Growth** algorithm implemented with the MapReduce framework (*H. Li, Y. Wang, D. Zhang, M. Zhang, and E.Y. Chang-2008*).

Algorithm idea

- The idea is to **start searching from a set of representative examples** instead of testing the 1-itemset,the k-itemset and so on.
- *A clustering algorithm is firstly* applied in order to cluster the transactions into k clusters.
- Each cluster is represented by the most representative example.
- We currently use the k-medoids algorithm in order to cluster the transactions.
- The set of the k representative examples will be used as the starting point for searching frequent itemsets.

K-means algorithm: illustration



Algorithm parameters

Input and outpout

- Input : a set of transactions called D,
- Output :

  A list named *accepted* that contains the retained frequent itemsets.

  A list named *excluded* that contains the retained no-frequent itemsets.

Parameters

K is the initial number of clusters.

minSupp is the threshold used for computing frequent itemsets.

Algorithm structures and definition

Structure

- An intermediate list that we call *candidates* containing the itemsets to test.
- Itemsets will be sorted by their decreasing lengths.

Definition

Global frequent itemsets: Let D be a set of transactions. Let L_i be an i-itemset of length i, L_i is **a global frequent itemset** iff it is frequent in D.

Local frequent itemsets: Let D be a set of transaction segmented on k disjoints clusters. Let L_i be an i-itemset of length i, L_i is **a local frequent itemset** iff it is frequent in the cluster to whose it belongs.

Algorithm description-1

1. Apply the k-medoids on D (the transactions base) and stock the k representative examples as well as the K clusters.
2. Let $C_1, C_2, .., C_k$ be the k representative examples sorted by their decreasing lengths, in D. The list *candidates* is initialized to $C_1, C_2, .., C_k$.
3. While the list *candidates* $\neq \Phi$ do
   - Let C_i be the first element of *candidates*:
   - If $C_i \notin$ *accepted* et $C_i \notin$ *excluded* then
     1. If C_i is a **local frequent itemset** then *update-accepted*(C_i), exit.
     2. If C_i is a **global frequent itemset** then *update-accepeted*(C_i), exit.
     3. else, *update-excluded* (C_i), and add frequent itemsets included in C_i to the *candidates* list.

Algorithm description-2

- The algorithm tests firstly if a given example e is a local frequent itemset, if yes the list called *accepted* is updated,
- otherwise the algorithm tests if e is a global frequent itemset, if yes the list *accepted* is updated,
- otherwise the list *excluded* is updated.

update-accepted(C_i): Add to the list *accepted*, the itemset C_i and all the itemsets included in it.

update-excluded(C_i): Add to the list *excluded*, the itemset C_i and all the itemsets that include C_i.

Preliminary results-1

- Data set is composed of a set of navigation logs extracted from the Microsoft site (UCI-machine learning repository)
- The site is composed of 17 pages with some links between each others that we present by the set of the characters : $\{A, B, C, .., P, Q\}$.
- Initial data logs file contained navigations paths of 388445 users.
- By keeping only users who have sufficient paths length the users number is reduced to 36014.
- We call this set of transactions D.

Preliminary results-2

K	C_1		C_2		C_3		C_4		C_5	
	E_1	$\lvert C_1 \rvert$	E_2	$\lvert C_2 \rvert$	E_3	$\lvert C_3 \rvert$	E_4	$\lvert C_4 \rvert$	E_5	$\lvert C_5 \rvert$
k=2	ABFG	55%	ABDLK	45%						
k=3	ABFFG	46%	ABDKL	23%	ABCF	31%				
k=4	AFG	35%	ABDKL	34%	ABCFJ	17%	ABDFG	14%		
k=5	AFGJ	11%	ABDKL	42%	ABCFJ	19%	ABDFG	23%	BDFGN	5%

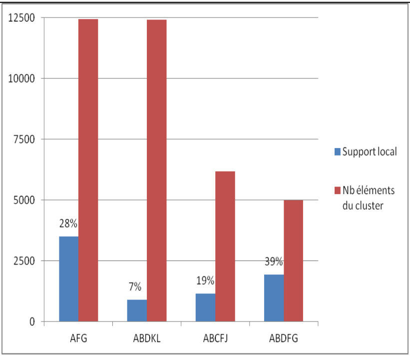
Table: Results of applying k-medoids on the transactions base D, we report for each value of k the representative example Ei of each cluster Ci as well as the cardinality of the cluster.

Preliminary results-2

Itemset	support	found by the novel algorithm
AFGJ	2406	yes
ABCJL	2406	no
ABCFJ	1576	yes
ABCKL	1922	no
ABDFG	2628	yes
ABDGL	1813	no
ABDKL	1735	yes
ABFGJ	1834	no

Table: This tables shows all frequent itemsets whose supports are higher than 1576 and whose length is 4 or 5. Four of the eight itemsets have been found by the novel algorithm.

Preliminary results-3



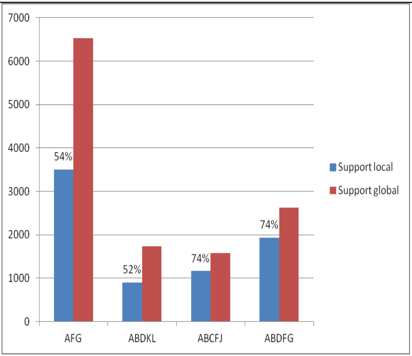


Table: The number of transactions in each cluster when k=4, and for each k the frequency rate of the found representative example. The local and global supports for representative examples when k=4

Preliminary results-4

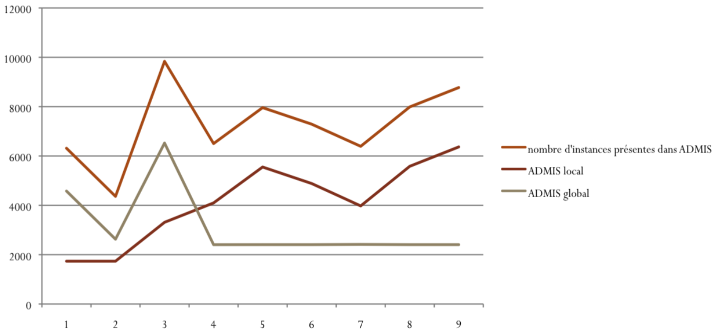


Figure: The red curve shows the number of frequent itemsets found locally, the green one shows the number of frequent itemsets found globally, and the orange one shows all the found frequent itemsets.

Preliminary results-5

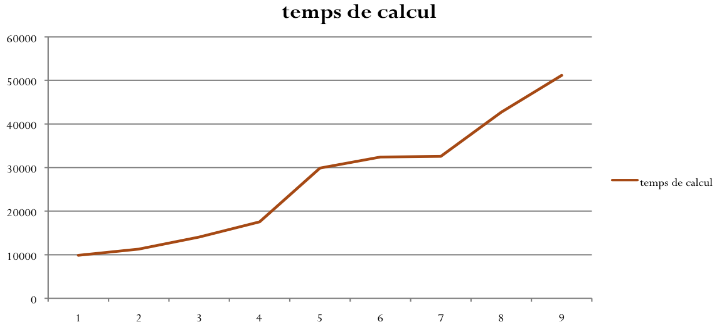


Figure: This figure shows the execution time evolution(micro seconds) in function of K.

MapReduce

MapReduce

A Framework for parallel and distributed computing that has been introduced by Google in order to handle huge data sets using a large number of computers (nodes).

Map Step

- The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes.
- The worker node processes the smaller problem, and passes the answer back to its master node in the form of list of key-values couples.

MapReduce

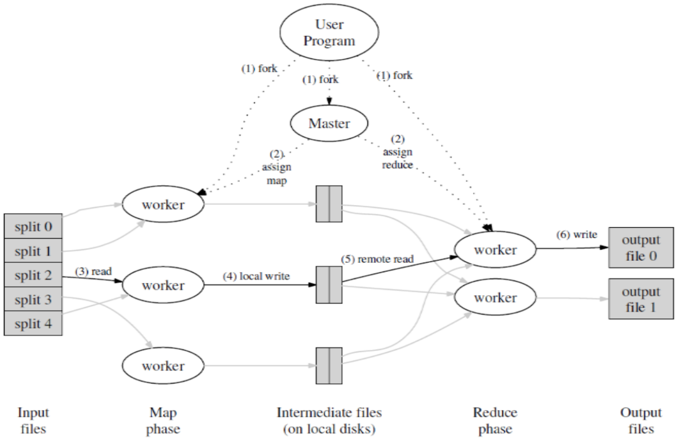
Map Step

- The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes.
- The worker node processes the smaller problem, and passes the answer back to its master node in the form of list of key-values couples.

Reduce Step

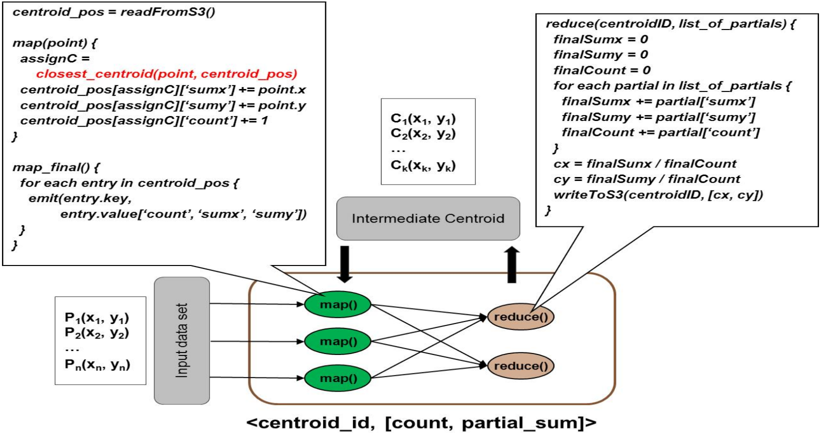
- The master node then collects the answers to all the sub-problems,
- and combines for a given key the intermediates values computed by the different mappers in order to form the output or the answer to the problem.

MapReduce : Schema



K-means implementation: scheme

Source : horkicky.blogspot.com



K-means implementation: description-1

Master Job

- Divide the input data into smaller sub-sets, and distribute them to mapper nodes.
- Randomly choose a list containing k representative examples and sent them to all mappers.
- Launch a MapReduce job for each iteration until the algorithm convergence (until stabilization of representative examples).

K-means implementation: description-2

Master Job

- Launch a MapReduce job for each iteration until the algorithm convergence (until stabilization of representative examples).

MapReduce

1. Mapper function: **compute** *for each example* the closer representative example, and **assign** it to the associated cluster.
2. Reducer function: **collect** *for each representative example* the partial sum of the computed distances from all mappers, and then **re-compute** the k new representative examples list.

K-medoids implementation: adaptation

Master Job

- Launch a MapReduce job for each iteration until the algorithm convergence (until stabilization of representative examples).

MapReduce

1. Mapper function: **compute** *for each example* the closer representative example, and **assign** it to the associated cluster.
2. Reducer function: **collect** *for each couple of (cluster, example)* the partial sum of the computed distances from all mappers, and then **re-compute** the k new representative examples list.

Apriori implementation on MapReduce -1

Map

void map(void* map_data)

- **for each** transaction in map_data
  - **for** (i = 0; i < candidates size; i++)
    - match = false ; itemset = candidates[i]
    - match = itemset_exists(transaction, itemset)
    - **if** (match == true) **emit intermediate(itemset, one)**

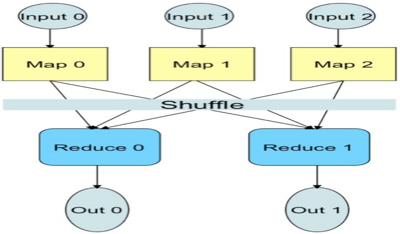
Reduce

- void reduce(void* key, void** vals, int vals_length)
  - count = 0
  - **for** (i = 0; i < vals length; i++) count+ = *vals[i]
  - **if** (count ≥ support_level * num transactions)/100.0 **emit(key, count)**

Apriori implementation on MapReduce-2

Mise jour de la liste des candidats

- void update_frequent_candidates(void * reduce_data_out)
  - j = 0
  - length = reduce_data_out → length
  - **for** (j = 0; $i < length$; j++) temp_candidates[j++] = reduce_data_out → key
  - candidates = temp_candidates



Parallel implementation of our algorithm

- Apply the above parallel version of the the k-medoids algorithm in order to obtain the the data set segmented into k clusters.
- Initialize the list of candidates to the k representative exemples.
- Re-distribute the k obtained clusters on k mappers,
- **Repeat**
  1. Send the list of candidates to the k mappers.
  2. Each **mapper** computes the **local support** of each candidate.
  3. The **reducer** collects all local supports for each candidate and compute for some candidates **the global supports** if necessary.
  4. The **master** updates *accepted, excluded and candidates* lists.
- **Until** the list of candidates is empty.

Conclusion

- A new algorithm for searching frequent itemsets in large data bases.
- The idea is to *start searching from a set of representative examples* instead of testing the 1-itemset, and so on.
- The k-medoids clustering algorithm is firstly applied in order to cluster the transactions into k clusters.
- Each cluster is represented by the most representative example.
- Experimental results show that beginning the search of frequent itemsets from these representative examples **leads to find a significant number of them locally** .

Perspectives

- Update the algorithm in order to find *all frequent itemsets*.
- We have proposed parallel version of this algorithm based on the MapReduce Framework:
- We are now implementing this parallel version and comparing performances with other parallel implementation like the massively parallel FP-Growth algorithm.

Bibliographie I

Rakesh Agrawal and Ramakrishnan Srikant.
Fast algorithms for mining association rules in large databases.
In *VLDB*, pages 487–499, 1994.

Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun.
Map-reduce for machine learning on multicore.
In *NIPS*, pages 281–288, 2006.

Jeffrey Dean and Sanjay Ghemawat.
Mapreduce: Simplified data processing on large clusters.
In *OSDI*, pages 137–150, 2004.

Jiawei Han, Jian Pei, and Yiwen Yin.
Mining frequent patterns without candidate generation.
SIGMOD Rec., 29(2):1–12, May 2000.

Wei Jiang, Vignesh T. Ravi, and Gagan Agrawal.
A map-reduce system with an alternate api for multi-core environments.
In *CCGRID*, pages 84–93, 2010.

L. Kaufman and P.J. Rousseeuw.
Clustering by means of medoids.
In Y. Dodge, editor, *Statistical Data Analysis Based on the Norm and Related Methods*, pages 405,416. North-Holland, 1987.

Bibliographie II

Walter A. Kosters and Wim Pijls.
Apriori, a depth first implementation.
In *Proc. of the Workshop on Frequent Itemset Mining Implementations*, 2003.

Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang.
Pfp: parallel fp-growth for query recommendation.
In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 107–114, New York, NY, USA, 2008. ACM.

Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman.
Mahout in Action.
Manning Publications, 1 edition, January 2011.

Mingjun Song and Sanguthevar Rajasekaran.
A transaction mapping algorithm for frequent itemsets mining.
IEEE Transactions on Knowledge and Data Engineering, 18:472–481, 2006.

Weizhong Zhao, Huifang Ma, and Qing He.
Parallel *k*-means clustering based on mapreduce.
In *CloudCom*, pages 674–679, 2009.