Distributed Machine Learning

Infrequent Pattern Mining

Yi Wang

Itemset Mining

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

- Transaction
- Item
- Itemset
- Support

セカンドライフ ゲーム secondlife	セカンドライフ and ゲーム are Japanese. セカンドライフ means secondlife. ゲーム means game. And secondlife is a game (http://secondlife.com/).	14
映像 映画 ＰＶ ＣＭ 動画コンテンツ 動画投稿 動画 ウェブ	These are all Japaness Kanji and Chinese words means “image”	13
христианство православие orthodox	христианство and православие are Russian. христианство means Christianity. православие means Orthodox. All these three words relate to religious.	8
正妹 taiwan album beauty photo	正妹 is tranditional Chinese in Taiwan. 正妹 means beauty. Lots of people search 正妹 for beauties' photos.	7
whorf piraha chomsky anthropology linguistics	Whorf and Chomsky are all experts in anthropology and linguistics, and they did research in a tribe named Piraha.	6

“Noise” but Interesting Patters from Del.icio.us Tags ▶

Frequent or Infrequent

- Frequent itemset mining
  - Apriori: silly and old loser targeted by all competitors.
  - FP-growth by Jiawei Han, 1663 citations. Hundreds of variants.
- Infrequent itemset mining
  - Long-tail data analysis should focus on infrequency.
  - PFP, 168 citations.
  - No variant..., but implemented in Mahout.

FP-growth

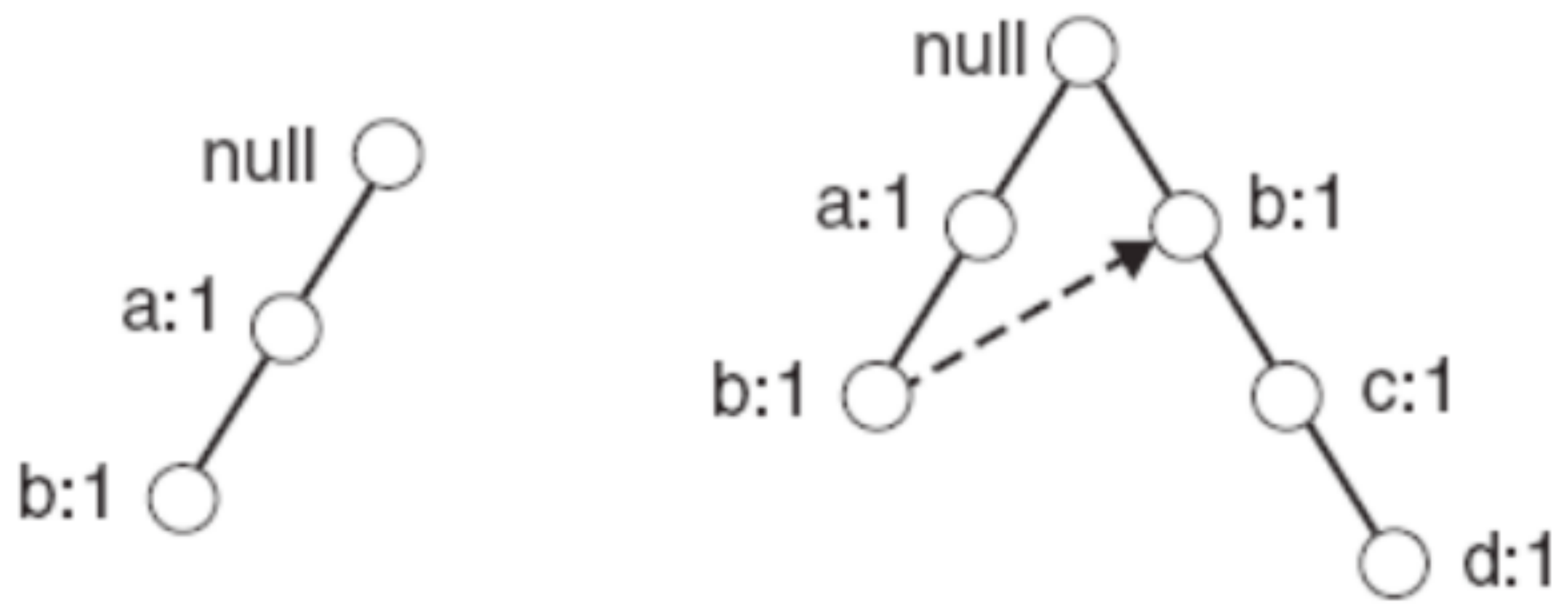
- Jiawei Han's original paper:
  http://web.engr.illinois.edu/~hanj/pdf/dami04_fptree.pdf
- Concise tutorial:
  http://www.florian.verhein.com/teaching/2008-01-09/fp-growth-presentation_v1%20(handout).pdf
- Implementation details:
  http://www.borgelt.net/papers/fpgrowth.pdf

FP-tree

- Data structure that represents input transactions in a compact way.
- When we care about only frequent itemsets, FP-tree is much smaller than input transactions.
- FP-tree is a prefix tree where nodes corresponding to items, and
- linked lists that threading nodes containing the same item.

Construct FP-tree

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

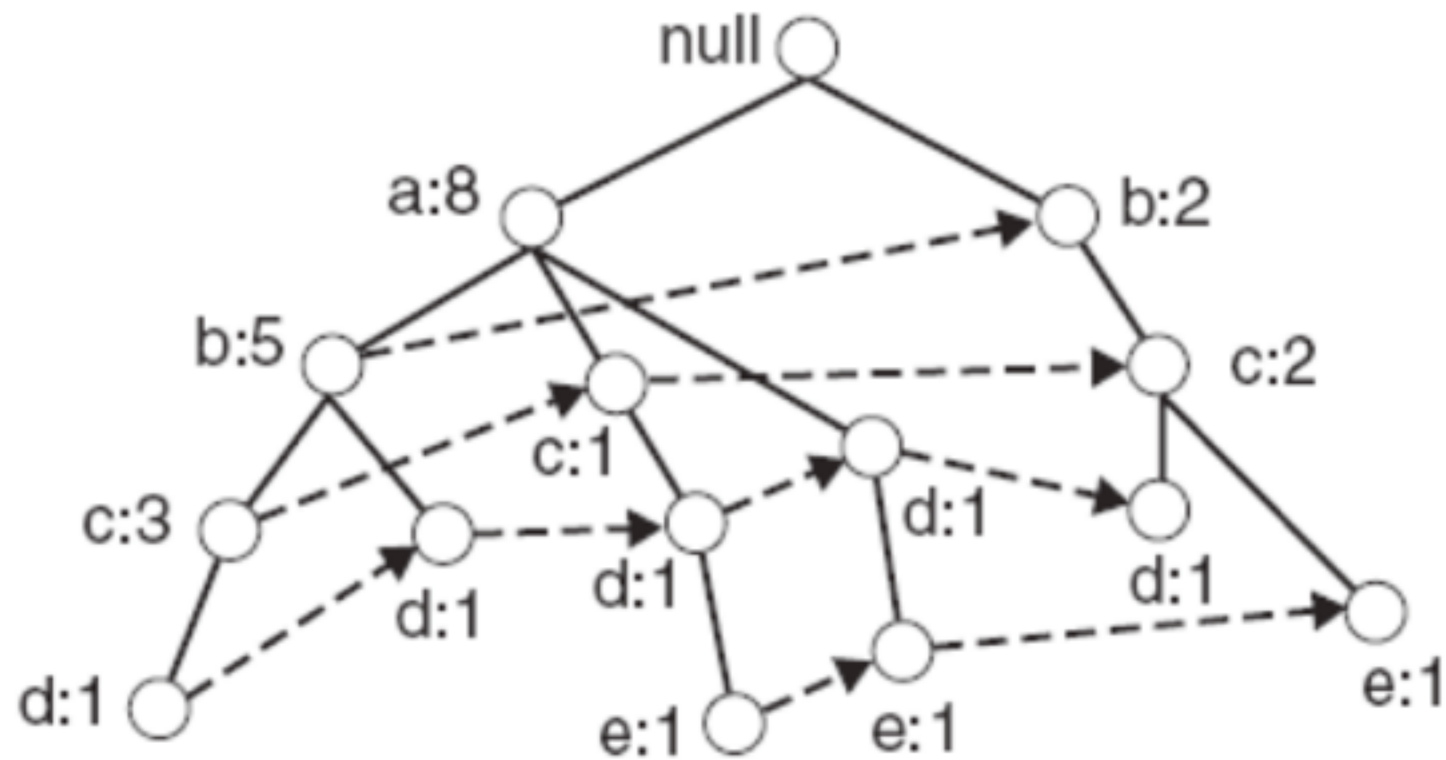


(i) After reading TID=1 (ii) After reading TID=2

- Insert each transaction into a prefix tree.
- Link nodes corresponding to the same item.

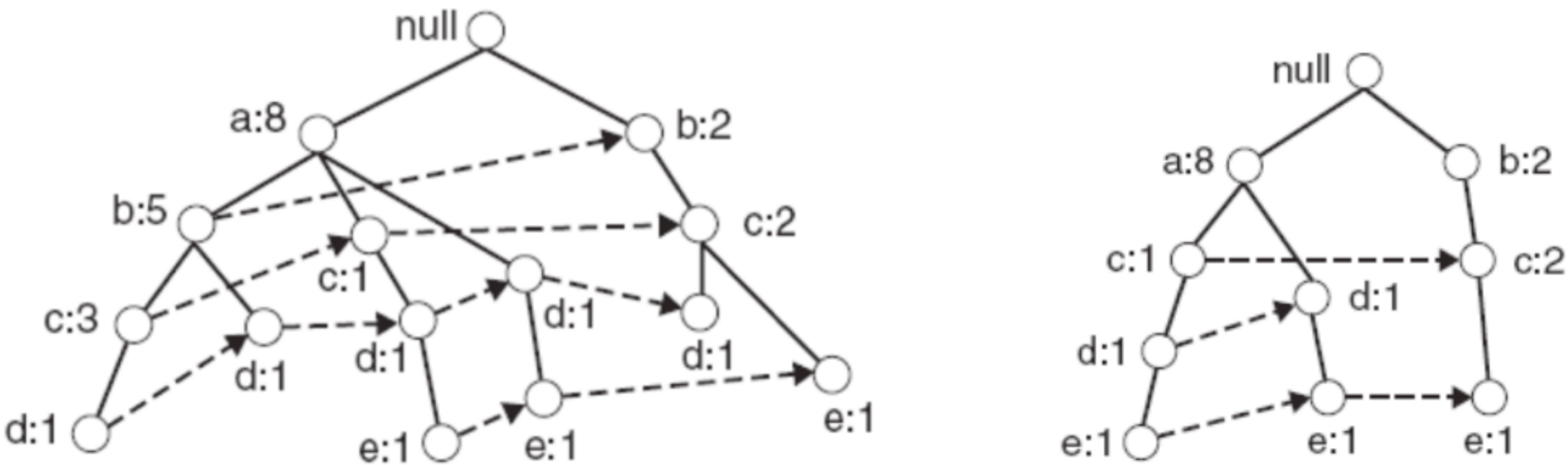
Construct FP-tree

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



(iv) After reading TID=10

Recursive Listing

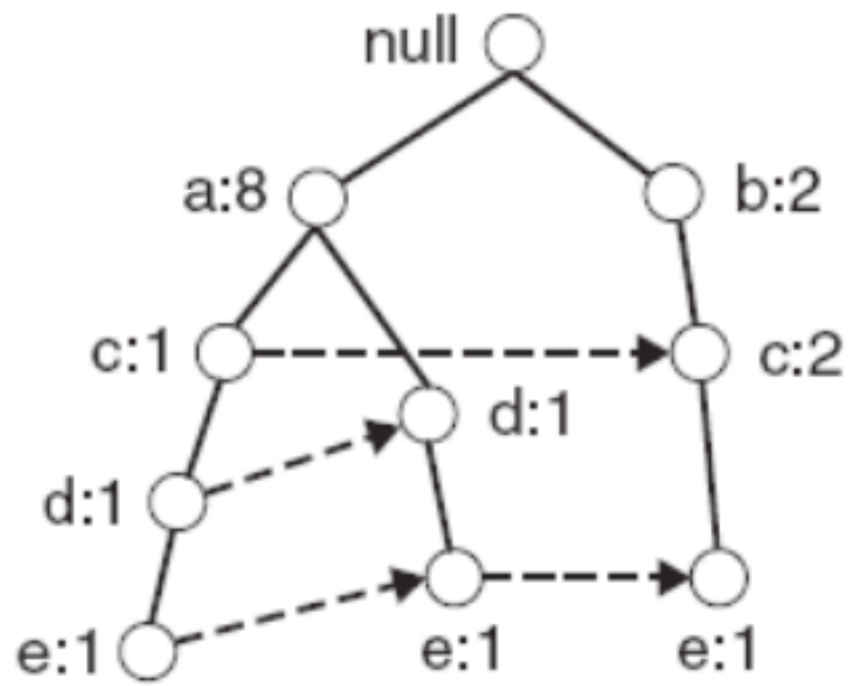


(iv) After reading TID=10

(a) Paths containing node e

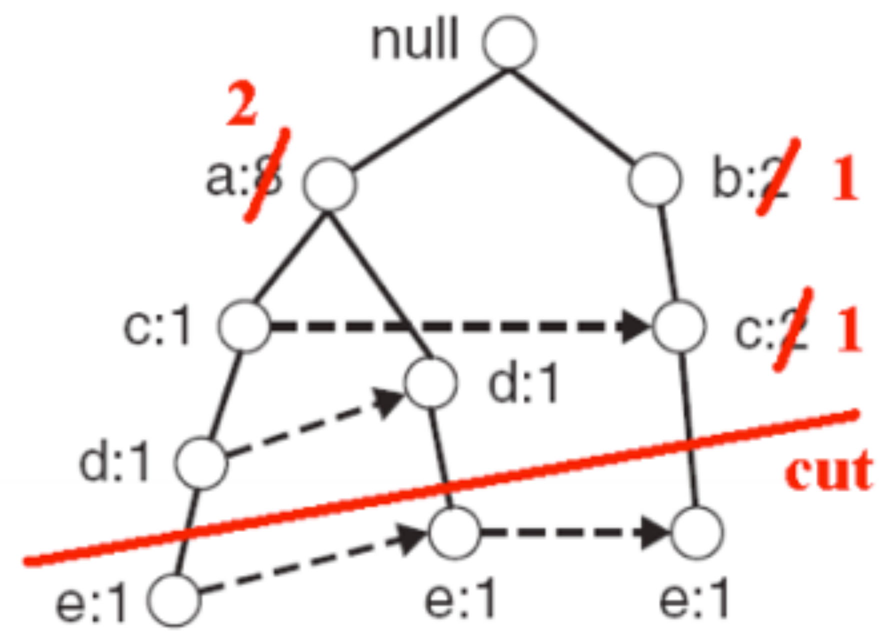
- Frequent itemsets containing e must be in the sub-tree containing e.
- Frequent itemsets containing be must be in the sub-sub-tree.
- This recursive procedure lists all itemsets.

Conditional FP-tree



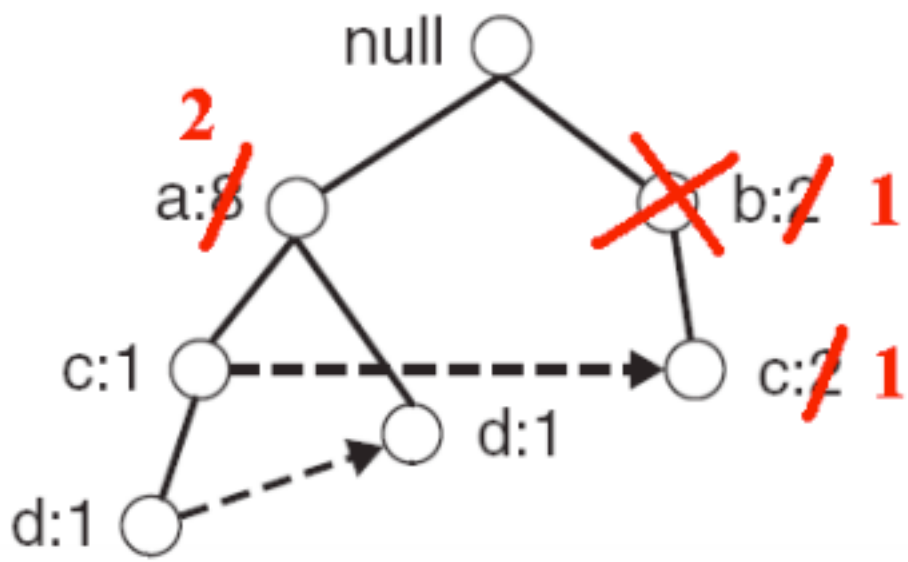
(a) Paths containing node e

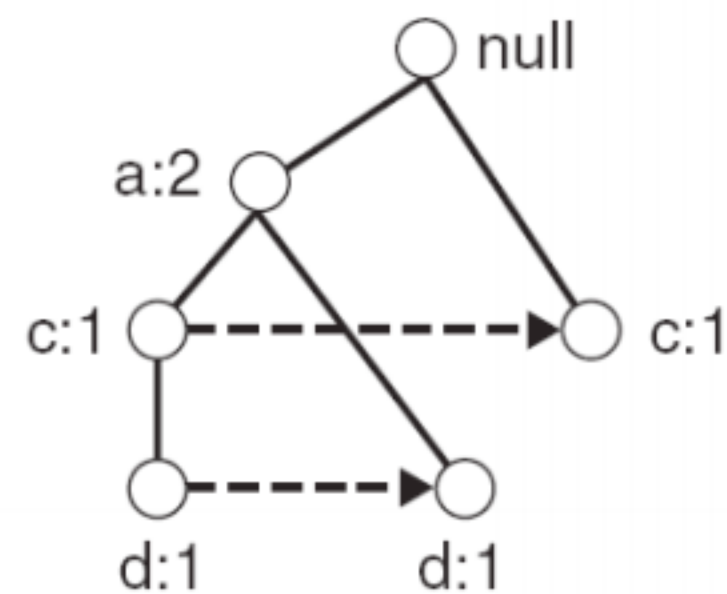
TID	Items
~~1~~	~~{a,b}~~
~~2~~	~~{b,c,d}~~
3	{a,c,d,~~e~~}
4	{a,d,~~e~~}
~~5~~	~~{a,b,c}~~
~~6~~	~~{a,b,c,d}~~
~~7~~	~~{a}~~
~~8~~	~~{a,b,c}~~
~~9~~	~~{a,b,d}~~
10	{b,c,~~e~~}



- Before the recursive procedure goes on, we define Conditional FP-tree.
- The conditional FP-tree for e counts only transactions containing e.

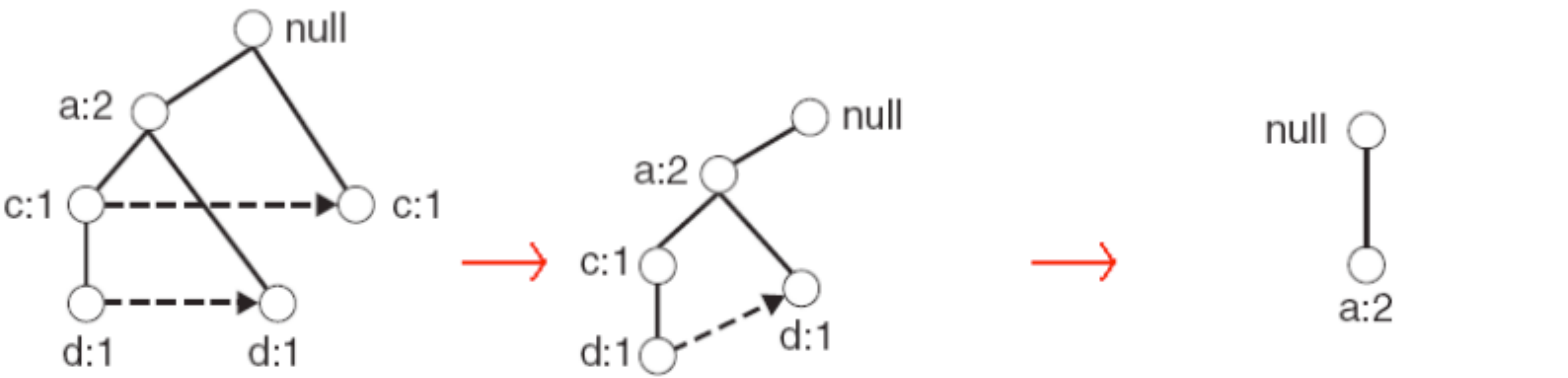
Conditional FP-tree





- Suppose minimum support = 2, "be" with support 1 is infrequent.
- Remove node b, but not c and d, which also have support 1. (why?)

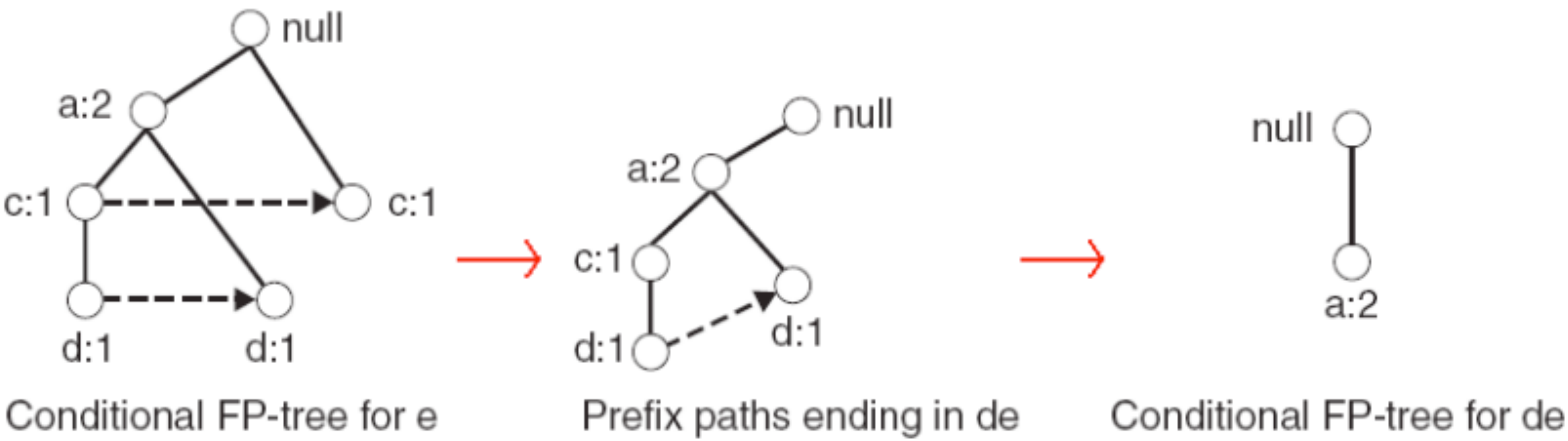
Conditional FP-tree



Conditional FP-tree for e   Prefix paths ending in de   Conditional FP-tree for de

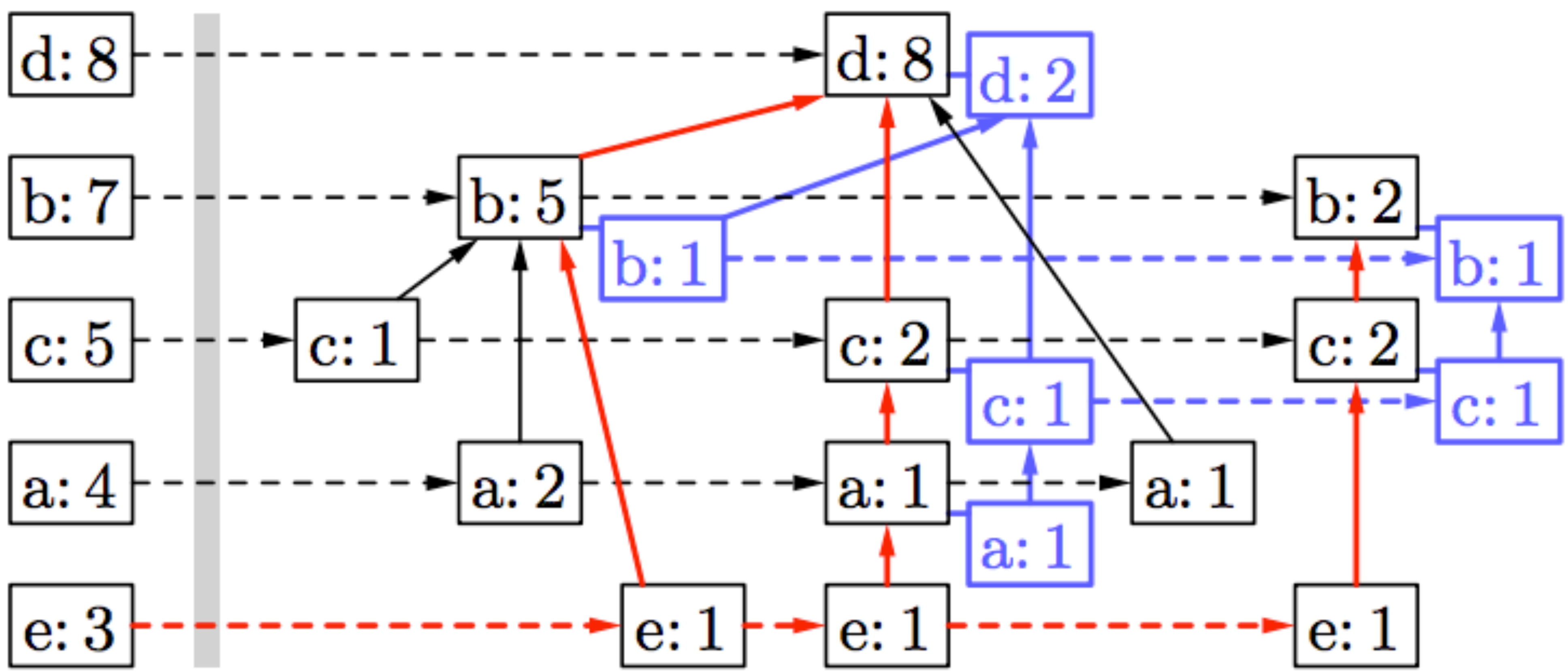
- Because the total support of all c's and d's are large enough.
- Recursively construct conditional FP-trees lists all frequent itemsets.

Conditional FP-tree



- Because the total support of all c’s and d’s are large enough.
- Recursively construct conditional FP-trees lists all frequent itemsets.

Project Sub-trees

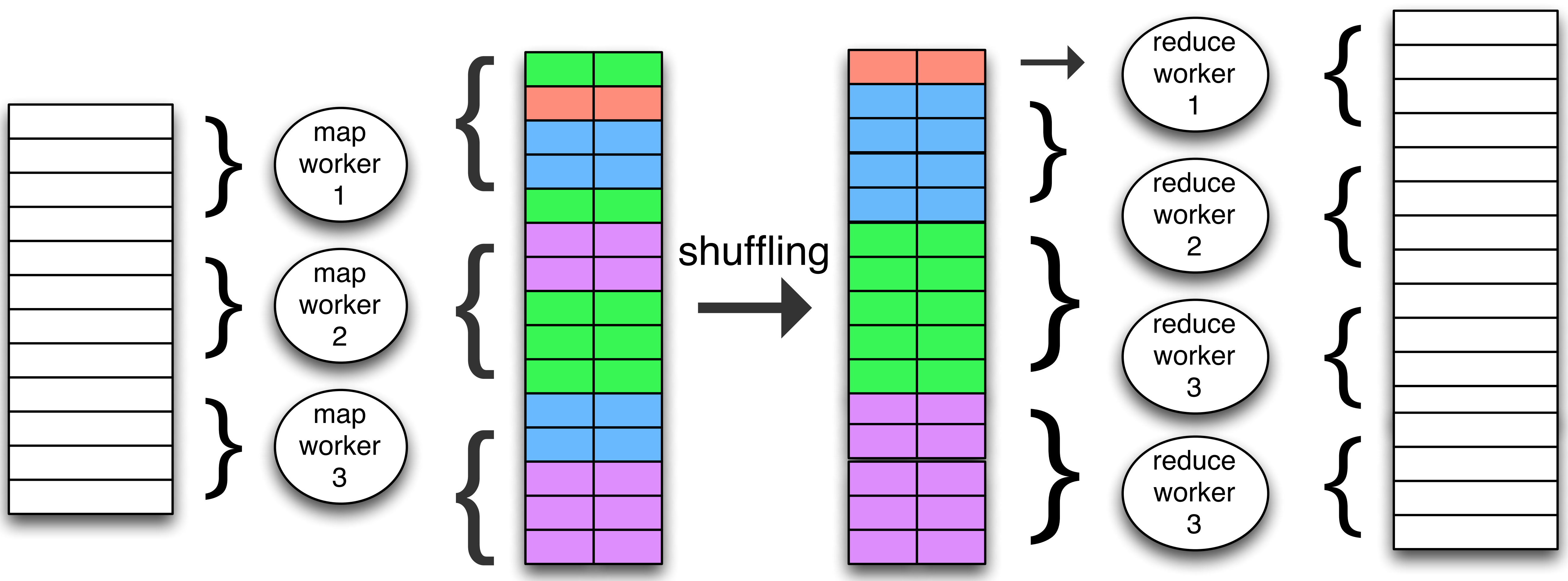


- We cannot really "cut" or "prune" the FP-tree while listing.
- Instead, we create a mirror when we construct conditional FP-trees.
- More memory required than storing the FP-tree.

Distributed Computing

- What if the memory is not enough?
- Solution 1: Increasing minimum support.
- Solution 2: Distributing transactions by items:
  - Shard 1 are transactions containing e; shard 2 are transactions containing b; ...
  - Need an aggregation. E.g., “abc” and “abe”.
  - Distribution is not even, as data is long-tail distributed!
  - How to do fault-recovery?

MapReduce



map worker 1
map worker 2
map worker 3
shuffling
reduce worker 1
reduce worker 2
reduce worker 3
reduce worker 3

PFP: Outline

- Distributed data structure
  - Tree represented by Set.
  - Each branch is a tuple.
- Distributed Algorithm
  - Load Balancing
  - Parallel FP-growth
  - Aggregation

PFP: Frequency Counting

Procedure: Mapper(key, value=T_i)
foreach *item* a_i *in* T_i **do**
    Call $Output(\langle a_i, '1' \rangle)$;
end

Procedure: Reducer(key=a_i, value=$S(a_i)$)
C $\leftarrow$ 0;
foreach *item* '1' *in* T_i **do**
    C $\leftarrow$ C + 1;
end
Call $Output(\langle null, a_i + C \rangle)$;

PFP: Distributing Tasks

Procedure: Mapper(key, value=T_i)
Load G-List;
Generate Hash Table H from G-List;
a[] $\leftarrow$ Split(T_i);
for $j = |T_i| - 1$ *to* 0 **do**
    $HashNum \leftarrow$ getHashNum(H, $a[j]$);
    if $HashNum \neq Null$ **then**
        Delete all pairs which hash value is $HashNum$ in H;
        Call $Output(\langle HashNum, a[0] + a[1] + ... + a[j] \rangle)$;
    end
end

PFP: Distributing Tasks

Map inputs (transactions) key="": value	Sorted transactions (with infrequent items eliminated)	Map outputs (conditional transactions) key: value
f a c d g i m p	f c a m p	p: f c a m m: f c a a: fc c: f
a b c f l m o	f c a b m	m: f c a b b: f c a a: f c c: f
b f h j o	f b	b: f
b c k s p	c b p	p: c b b: c
a f c e l p m n	f c a m p	p: f c a m m: f c a a: f c c: f

PFP: Distributing FP-growth

Procedure: Reducer(key=gid,value=DB_{gid})
Load G-List;
$nowGroup \leftarrow$ G-List$_g id$;
$LocalFPtree \leftarrow$ clear;
foreach T_i in $DB_(gid)$ **do**
    Call $insert - build - fp - tree(LocalFPtree, T_i)$;
end
foreach a_i in $nowGroup$ **do**
    Define and clear a size K max heap : HP;
    Call $TopKFPGrowth(LocalFPtree, a_i, HP)$;
    foreach v_i in HP **do**
        Call $Output(\langle null, v_i + supp(v_i) \rangle)$;
    end
end

PFP: Distributing FP-growth

Map outputs (conditional transactions) key: value
p: f c a m m: f c a a: fc c: f
m: f c a b b: f c a a: f c c: f
b: f
p: c b b: c
p: f c a m m: f c a a: f c c: f

Reduce inputs (conditional databases) key: value		Conditional FP−trees
p:	{ f c a m / f c a m / c b }	{(c:3)} | p
m:	{ f c a / f c a / f c a b }	{ (f:3, c:3, a:3) } | m
b:	{ f c a / f / c }	{} | b
a:	{ f c / f c / f c }	{ (f:3, c:3) } | a
c:	{ f / f / f }	{ (f:3) } | c

PFP: Aggregation

Procedure: Mapper(key, value=$v + \text{supp}(v)$)
foreach *item* a_i *in* v **do**
    Call $Output(\langle a_i, v + \text{supp}(v)\rangle)$;
end
Procedure: Reducer(key=a_i, value=$S(v + \text{supp}(v))$)
Define and clear a size K max heap : HP;
foreach *pattern* v *in* $v + \text{supp}(v)$ **do**
    if $|HP| < K$ **then**
        insert $v + \text{supp}(v)$ into HP;
    else
        if $\text{supp}(HP[0].v) < \text{supp}(v)$ **then**
            delete top element in HP;
            insert $v + \text{supp}(v)$ into HP;
        end
    end
end
Call $Output(\langle null, a_i + C\rangle)$;

PFP: Conclusion

- In Mahout: http://mahout.apache.org/users/misc/parallel-frequent-pattern-mining.html
- What if sub-transaction databases are still too big to fit in a FP-tree?
  - Iterative MapReduce jobs for {conditional-}*FP-trees.