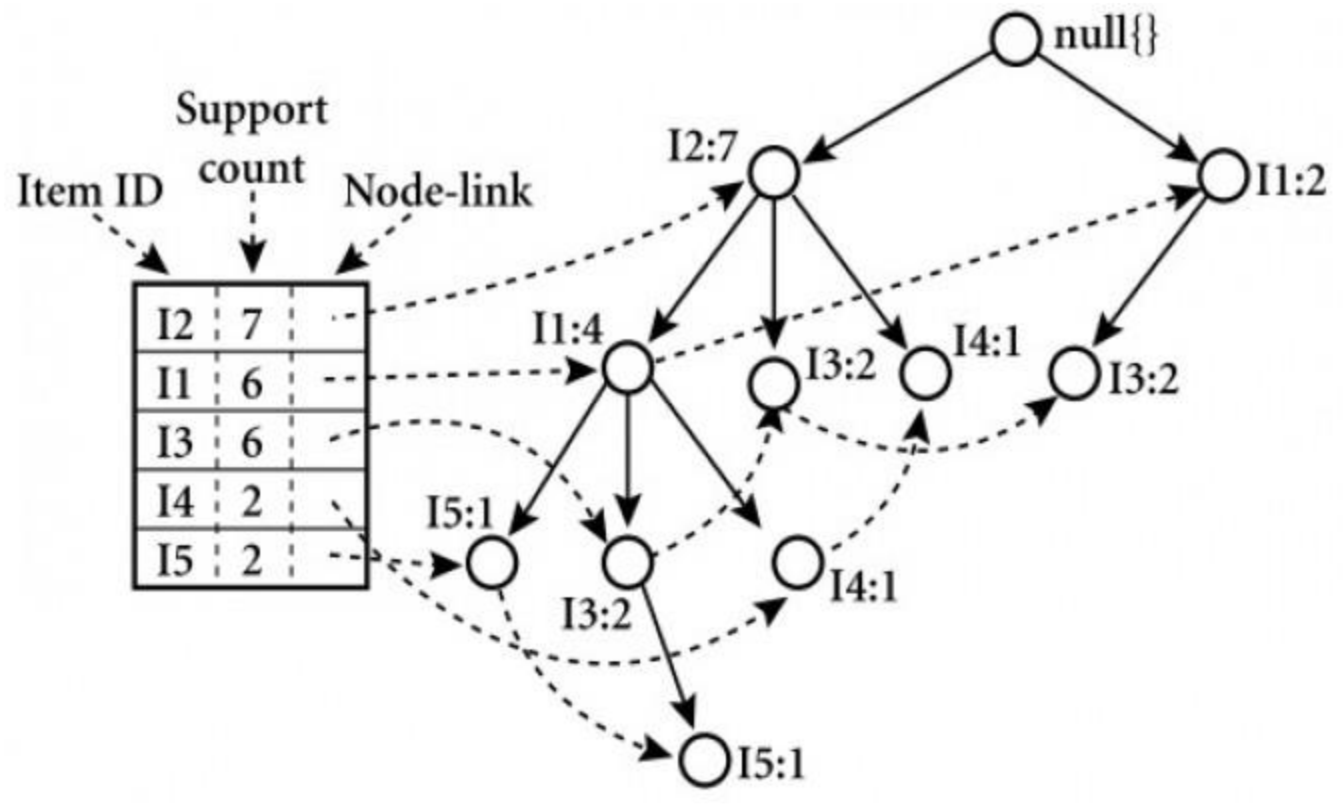
Apriori vs FP-Growth for Frequent Item Set Mining

Comparison between Apriori and FP-Growth algorithms



Frequent Item Set (FIS) mining is an essential part of many Machine Learning algorithms. What this technique is intended to do is to extract the most frequent and largest item sets within a big list of transactions containing several items each. For example:

Let *T* be a list of n transactions [t_1, t_2, ..., t_n]. Each transaction t_i contains a list of k_t items [a_{i1}, a_{2i}, ..., a_{ik}].

So we have T = [t_1 = [a_{11}, a_{12}, ... a_{1k}], t_2 = [a_{21}, a_{22}, ... a_{2k}], ..., t_k = [a_{t1}, a_{t2}, ... a_{tk}]].

The Frequent Item Set for T is FIS_T, where ∀s ∈ FIS_T, s is the most frequent and largest set of items, which:

- Is not contained within another set in FIS_T.
- Appears at least m times (we call this m number as the minimum support threshold).

An example of this FIS technique, a customer Mario, with a minimum support threshold m of 50 % and we want to mine the FIS for Mario.

T_{Mario} = [[beer, wine, rum], [beer, rum, vodka], [beer, vodka], [beer, wine, rum]].

$|T_{Mario}| = 4$

m = 2 (50 % of 4)

FIS_{Mario} = { {beer, wine, rum}, {beer vodka} }

As you can see, {beer, wine rum} and {beer vodka} appear in 2 of the transactions.

You may be asking why isn't {beer, rum} inside FIS_{Mario}, if it appears on 3 of the transactions. The answer is because {beer, rum} is already contained inside {beer, wine, rum}, which is larger.

This is not an actual concrete implementation of the algorithm, we will explain you 2 of many implementations: Apriori and FP-Growth.

Apriori Algorithm

The Apriori algorithm is based on the fact that if a subset S appears k times, any other subsetS' that contains S will appear **k times or less**. So, if S doesn't pass the minimum support threshold, **neither does S'**. There is **no need to calculate S'**, it is discarded **a priori**.

Now we're going to show you an example of this algorithm.

Let's suppose we have a client Mario with transactions [[beer, wine, rum], [beer, rum, vodka], [beer, vodka], [beer, wine, rum]], and a minimum support threshold m of 50 % (2 transactions).

First step: Count the singletons & apply threshold

The singletons for Mario are:
beer: 4,
wine: 2,
rum: 3,
vodka: 2

All of the single items appear m or more times, so none of them are discarded.

Second step: Generate pairs, count them & apply theshold

The pairs created were: { {beer, wine}, {beer, rum}, {beer, vodka}, {wine, rum}, {wine, vodka}, {rum, vodka} }.
Now we proceed to count them and applying the threshold.

{beer, wine}: 2

{beer, rum}: 3

{beer, vodka}: 2

{wine, rum}: 2

{wine, vodka}: 0

{rum, vodka}: 1

{wine, vodka} and {rum, vodka} have not passed the threshold, so they are discarded and any other subcombination both of them can generate.

The remaining pairs are put into a temporal associations set.

Assocs = {{beer, wine}, {beer, rum}, {beer, vodka}, {wine, rum}}

Step N: Generate triplets, quadruplets, etc., count them, apply threshold and remove containing itemsets.

We generate triplets from our pairs.

Triplets = { {beer, wine, rum}, {beer, wine, vodka}, {beer, rum, vodka}, {wine, rum, vodka} }.

Now we count them:

{beer, wine, rum}: 2

{beer, wine, vodka}: 0

{beer, rum, vodka}: 1

{wine, rum, vodka}: 0

Only {beer, wine, rum} has passed the threshold, so now we proceed to add it to Assocs, but first, we have to remove the subsets that {beer, wine, rum} contains.

Before adding our remaining triplet Assocs looked like this: { {beer, wine}, {beer, rum}, {beer, vodka}, {wine, rum} }.

When we add the triplet and remove the subsets that are inside it {beer, wine}, {beer, rum} and {wine, rum} are the ones that should go.

Assocs now look like { {beer, wine, rum}, {beer, vodka} }, and this is our **final result**.

If we had more than 1 triplet after applying the threshold, we should proceed to generating the quadruplets, counting them, applying the threshold, adding them to Assocs and removing the subsets that each quadruplet contains.

Disadvantages of Apriori

- The candidate generation could be extremely slow (pairs, triplets, etc.).
- The candidate generation could generate duplicates depending on the implementation.
- The counting method iterates through all of the transactions each time.
- Constant items make the algorithm a lot heavier.
- Huge memory consumption

Advantages of Apriori

The Apriori Algorithm calculates more sets of frequent items.

FP-Growth

FP-Growth is an improvement of apriori designed to eliminate some of the heavy bottlenecks in apriori. The algorithm was planned with the bennefits of mapReduce taken into account, so it works well with any distributed system focused on mapReduce. FP-Growth simplifies all the problems present in apriori by using a structure called an FP-Tree. In an FP-Tree each node represents an item and it's current count, and each branch represents a different association.

The whole algorithm is divided in 5 simple steps. Here we have a simple example:

Our client is named Mario and here we have his transactions:
T_{Mario}= [[beer, bread, butter, milk] , [beer, milk, butter], [beer, milk, cheese] , [beer, butter, diapers, cheese] , [beer, cheese, bread]]

Step 1:

The first step is we count all the items in all the transactions
T_{Mario}= [beer: 5, bread: 2, butter: 3, milk: 3, cheese: 3, diapers: 1]

Step 2:

Next we apply the threshold we had set previously. For this example let's say we have a threshold of 30% so each item has to appear at least twice.
T_{Mario}= [beer: 5, bread: 2, butter: 3, milk: 3, cheese: 3, ~~diapers: 1~~]

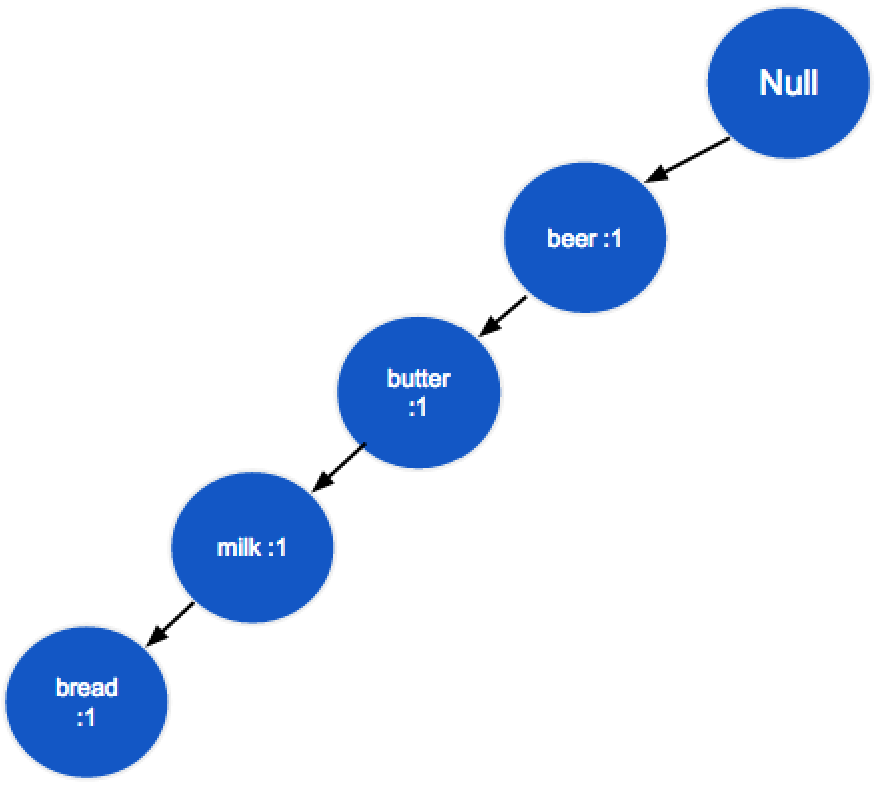
Step 3:

Now we sort the list according to the count of each item.
$T_{MarioSorted}$ = [beer: 5, butter: 3, milk: 3, cheese: 3, bread: 2]

Step 4:

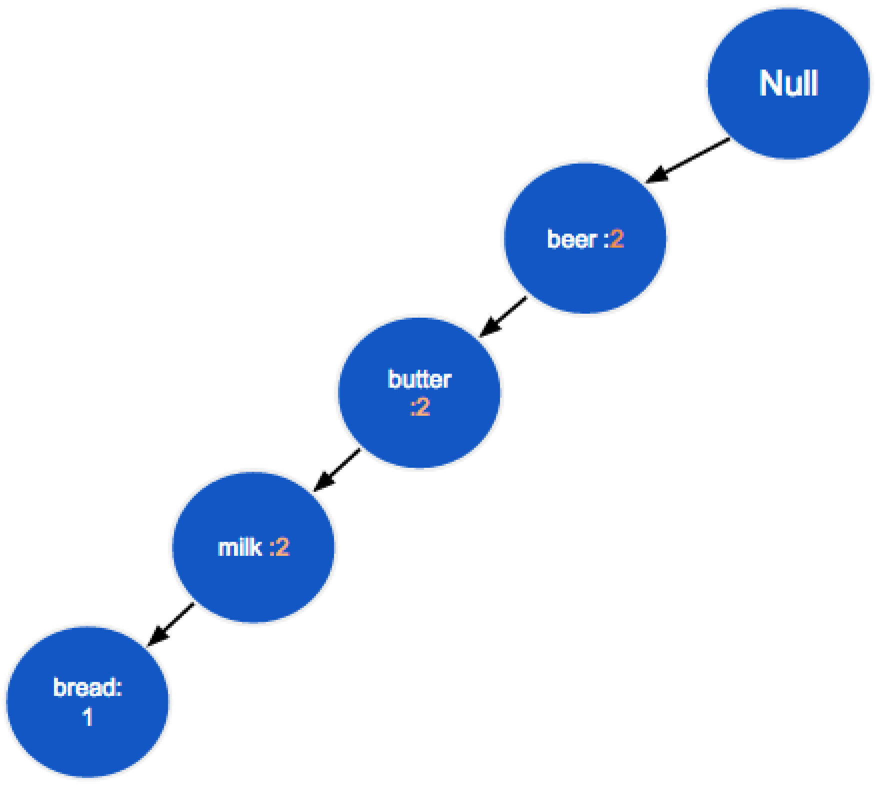
Now we build the tree. We go through each of the transactions and add all the items in the order they appear in our sorted list.

Transaction to add= [beer, bread, butter,

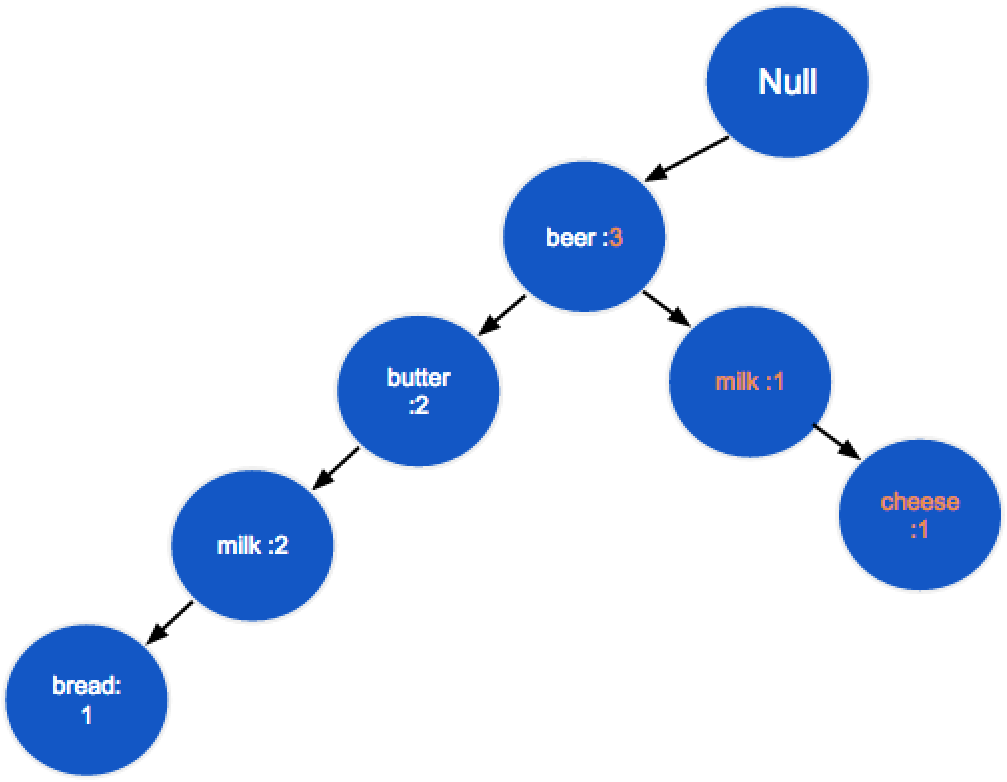


milk]

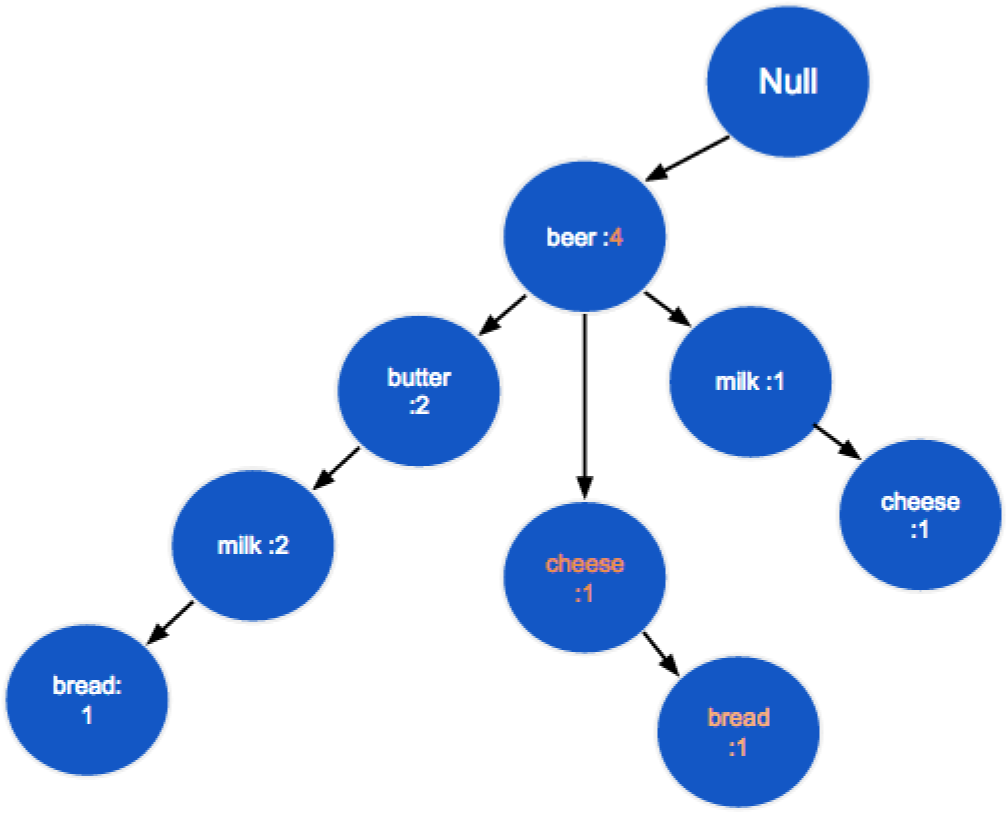
Transaction 2: [beer, milk, butter]



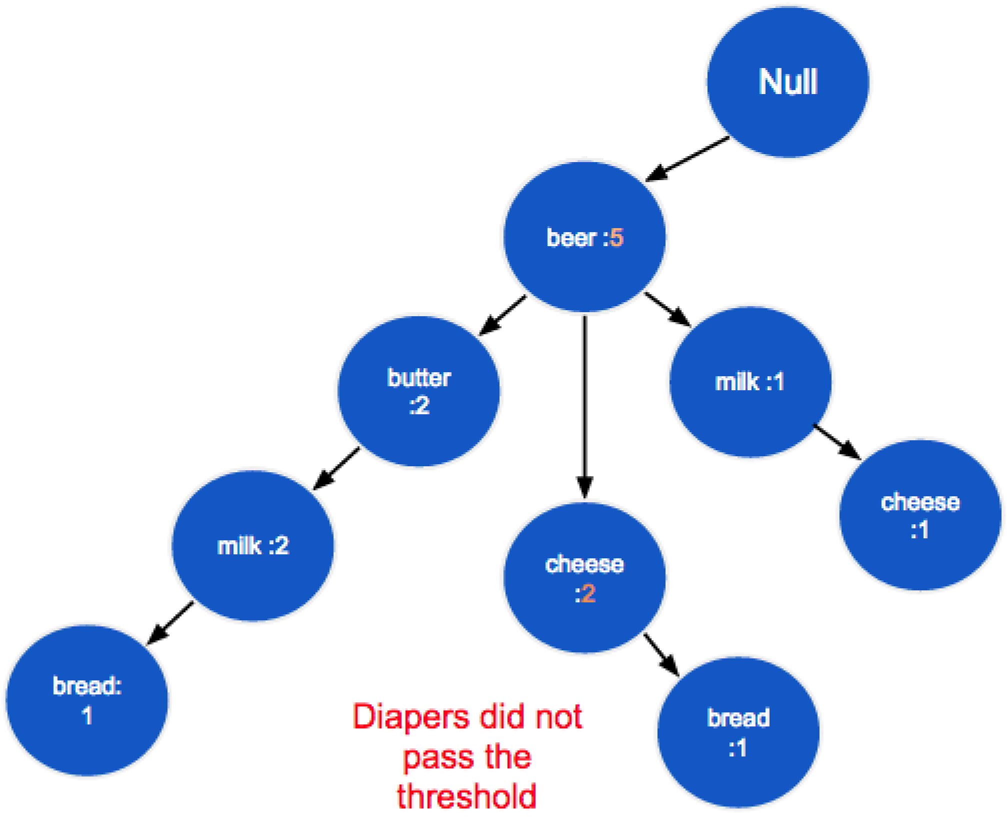
Transaction 3=[beer, milk, cheese]



Transaction 4=[beer, cheese, bread]

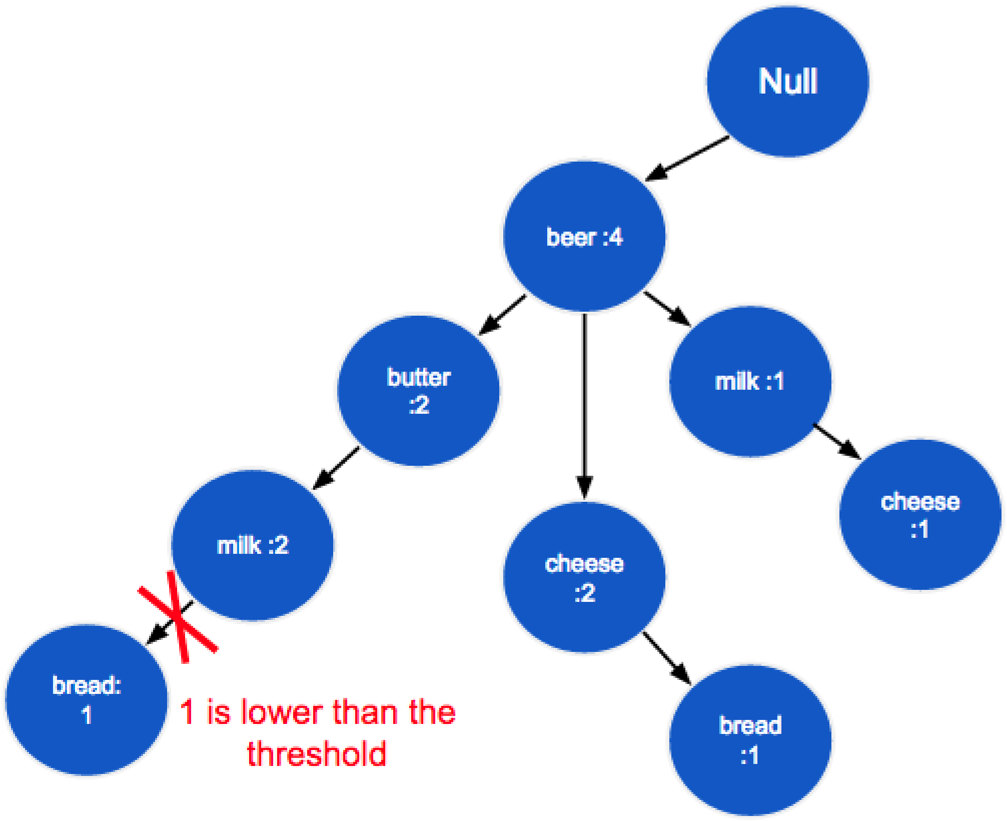


Transaction 5=[beer, cheese, diapers]



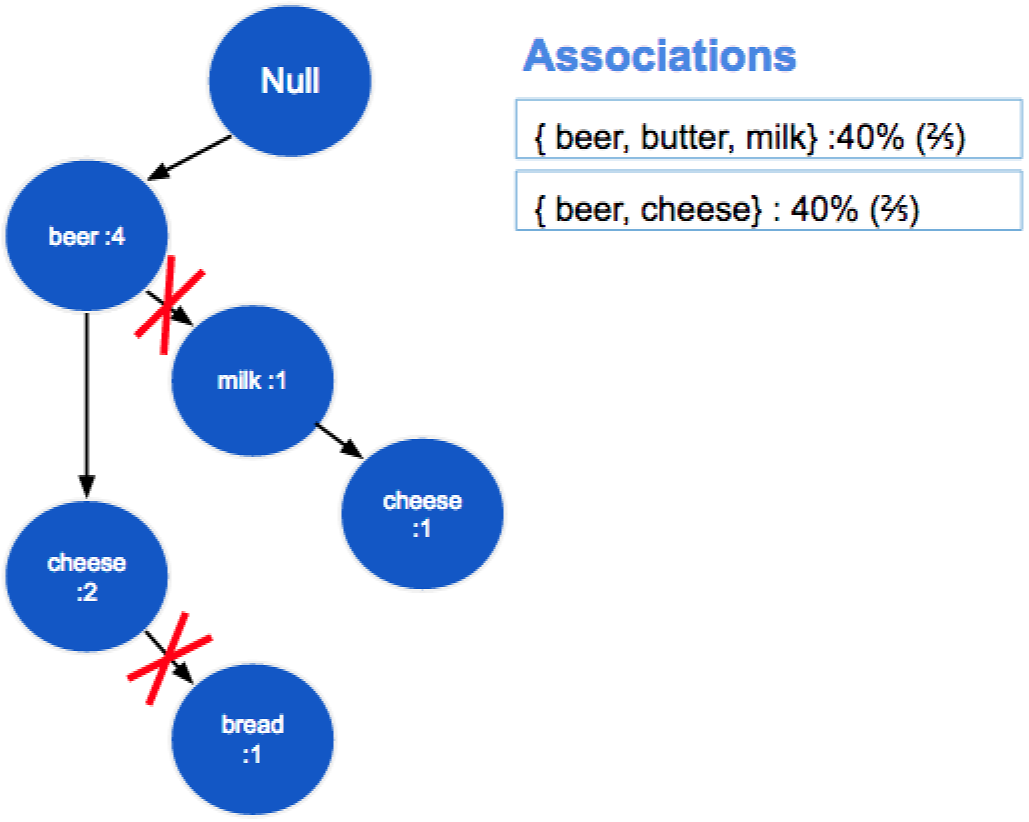
Step 5:

In order to get the associations now we go through every branch of the tree and only include in the association all the nodes whose count passed the threshold.



Associations

{ beer, butter, milk} :40%



FP-Growth Biggest Advantages

The biggest advantage found in FP-Growth is the fact that the algorithm only needs to read the file twice, as opossed to apriori who reads it once for every iteration.

Another huge advantage is that it removes the need to calculate the pairs to be counted, which is very processing heavy, because it uses the FP-Treee. This makes it O(n) which is much faster than apriori.

The FP-Growth algorithm stores in memory a compact version of the database.

FP-Growth Bottlenecks

The biggest problem is the interdependency of data. The interdependency problem is that for the parallelization of the algorithm some that still needs to be shared, which creates a bottleneck in the shared memory.

Apriori vs FP-Growth

Algorithm	Technique	Runtime	Memory usage	Parallelizability
Apriori	Generate singletons, pairs, triplets, etc.	Candidate generation is extremely slow. Runtime increases exponentially depending on the number of different items.	Saves singletons, pairs, triplets, etc.	Candidate generation is very parallelizable
FP-Growth	Insert sorted items by frequency into a pattern tree	Runtime increases linearly, depending on the number of transactions and items	Stores a compact version of the database.	Data are very inter dependent, each node needs the root.

We ran some tests using Spark for both the Apriori and FP-Growth algorithms. Each file has multiple customers with multiple transactions. We parallelized the algorithms per client and our results were:

File	Apriori	FP-Growth
Simple Market Basket test file	3.66 s	3.03 s
"Real" test file (1 Mb)	8.87 s	3.25 s
"Real" test file (20 Mb)	34 m	5.07 s
Whole "real" test file (86 Mb)	4+ hours (Never finished, crashed)	8.82 s

Conclusions

FP-Growth beats Apriori by far. It has less memory usage and less runtime. The differences are huge. FP-Growth is more scalable because of its linear running time.

Don't think twice if you have to make a decision between these algorithms. Use FP-Growth.