APRIORI DATA MINING IN THE CLOUD

Case study: d60 Raptor smartAdvisor

Jan Neerbek
Alexandra Institute
Agenda

- d60: A cloud/data mining case
- Cloud
- Data Mining
- Market Basket Analysis
- Large data sets
- Our solution
The Alexandra Institute is a non-profit company that works with application-oriented IT research.

Focus is pervasive computing, and we activate the business potential of our members and customers through research-based user-driven innovation.
The case: d60

- Danish company
- A similar products recommendation engine
- d60 was outgrowing their servers (late 2010)
- They saw a potential in moving to Azure
The setup

- Webshops
- Internet
- Log shopping patterns
- Product Recommendations
- Do data mining
The cloud potential

- Elasticity
- No upfront server cost
- Cheaper licenses
- Faster calculations
Challenges

- No SQL Server Analysis Services (SSAS)
- Small compute nodes
- Partitioned database (50GB)
- SQL server ingress/outgress access is slow
The cloud
The cloud and services
Data layer service

- Application specific (schema/layout)
- SQL, table or other
- Easy a bottleneck
- Can be difficult to scale
Messaging service
Task Queues

- Standard data structure
- Build-in ordering (FIFO)
- Can be scaled
- Good for asynchronous messages
DATA MINING
Data mining is the use of automated data analysis techniques to uncover relationships among data items.

Market basket analysis is a data mining technique that discovers co-occurrence relationships among activities performed by specific individuals.

[about.com/wikipedia.org]
Market basket analysis

Customer 1
- Avocado
- Milk
- Butter
- Potatoes

Customer 2
- Milk
- Diapers
- Avocado
- Beer

Customer 3
- Beef
- Lemons
- Beer
- Chips

Customer 4
- Cereal
- Beer
- Beef
- Diapers
Market basket analysis

Itemset (Diapers, Beer) occur 50%

Frequency threshold parameter
Find as many frequent itemsets as possible
Market basket analysis

Popular effective algorithm: FP-growth 😊
Based on data structure FP-tree
Requires all data in near-memory 😞
Most research in distributed models has been for cluster setups 😞
Building the FP-tree (extends the prefix-tree structure)

Customer1
- Avocado
- Milk
- Butter
- Potatoes
Building the FP-tree

Customer2
- Milk
- Diapers
- Avocado
- Beer

Avocado

Butter

Milk

Potatoes
Building the FP-tree

Customer2
Milk
Diapers
Avocado
Beer

Avocado
Butter
Beer
Milk
Diapers
Potatoes

Milk
Building the FP-tree

Avocado
  /    
Butter  Beer
    /  
Milk   Diapers
  / 
Potatoes Milk
Building the FP-tree

Avocado
  ├── Butter
  │    └── Milk
  │         └── Potatoes
  └── Beer
    ├── Diapers
    ├── Milk
    └── Chips
        ├── Cereal
        └── Diapers
    └── Beer
        └── Lemon
FP-growth

Grows the frequent itemsets, recursively

```java
FP-growth(FP-tree tree)
{
    ...
    for-each (item in tree)
    {
        count = CountOccur(tree, item);
        if (IsFrequent(count))
        {
            OutputSet(item);
            sub = tree.GetTree(tree, item);
            FP-growth(sub);
        }
    }
}
```
FP-growth algorithm
Divide and Conquer

Traverse tree

- Avocado
  - Butter
  - Milk
  - Potatoes
- Beef
  - Beer
  - Diapers
- Beer
  - Chips
  - Cereal
- Cereal
  - Milk
  - Lemon
  - Diapers
FP-growth algorithm
Divide and Conquer

Generate sub-trees

- Avocado
  - Butter
  - Milk
  - Potatoes

- Beef
  - Beer
  - Milk
  - Cereal
  - Chips
  - Lemon
  - Diapers
FP-growth algorithm
Divide and Conquer

Call recursively
The FP-tree does not fit in local memory; what to do?

- Emulate Distributed Shared Memory
Distributed Shared Memory?

- To add nodes is to add memory
- Works best in tightly coupled setups, with low-latency, high-speed networks
The FP-tree does not fit in local memory; what to do?

- Emulate Distributed Shared Memory
- Optimize your data structures
- Buy more RAM
- Get a good idea
Get a good idea

- Database scans are serial and can be distributed
- The list of items used in the recursive calls uniquely determines what part of data we are looking at
Get a good idea
Get a good idea

These are *postfix paths*
BRINGING IT ALL TOGETHER
Buckets

- Use postfix paths for messaging
- Working with *buckets*
FP-growth revisited

```java
FP-growth(FP-tree tree) {
    foreach (item in tree) {
        count = CountOccur(tree, item);
        if (IsFrequent(count)) {
            OutputSet(item);
            sub = tree.GetTree(tree, item);
            FP-growth(sub);
        }
    }
}
```
Communication

Diagram showing a data layer connecting four nodes.
Revised Communication

![Diagram]

- Node
- Node
- Node
- Node
- MQ
- Data layer
Running FP-growth

1. Distribute buckets
2. Count items (with postfix size=n)
3. Collect counts (per postfix)
4. Call recursive
5. Standard FP-growth
Running FP-growth

Distribute buckets

Count items (with postfix size=n)

Collect counts (per postfix)
Call recursive

Standard FP-growth
Collecting what we have learned

- Message-driven work, using message-queue
- Peer-to-peer for intermediate results
- Distribute data for scalability (buckets)
- Small messages (list of items)
- Allow us to distribute FP-growth
Advantages

- Configurable work sizes
- Good distribution of work
- Robust against computer failure
- Fast!
So what about performance?
Thank you!

Questions?