FDB: A Query Engine for Factorised Relational Databases

Nurzhan Bakibayev, Dan Olteanu, and Jakub Zavodny — Oxford CS

Christian Grant
cgrant@cise.ufl.edu

University of Florida

November 1, 2013
A succinct representation for large relations.
- It is used in Google’s F1 data management system for ML.
- Used for large incomplete/uncertain relations.
- FDB can speed up expected queries.
A succinct representation for large relations.
It is used in Google’s F1 data management system for ML.
Used for large incomplete/uncertain relations.
FDB can speed up expected queries.
This paper discusses basics FDB system.
TL;DR

Given a query: $Q_1 = \text{Order} \bowtie_{\text{item}} \text{Store} \bowtie_{\text{location}} \text{Disp}$
Given a query: \( Q_1 = \text{Order} \bowtie_{\text{item}} \text{Store} \bowtie_{\text{location}} \text{Disp} \)

And a result:
TL;DR

- Given a query: \( Q_1 = \text{Order} \bowtie_{\text{item}} \text{Store} \bowtie_{\text{location}} \text{Disp} \)

- And a result:

- We can represent it as this:
Given a query: $Q_1 = \text{Order} \bowtie_{\text{item}} \text{Store} \bowtie_{\text{location}} \text{Disp}$

And a result:

We can represent it as this:

Or a tree:
Given a query: \( Q_1 = \text{Order} \bowtie_{\text{item}} \text{Store} \bowtie_{\text{location}} \text{Disp} \)

And a result:

We can represent it as this:

Or a tree:

In exponentially -space and +speed for SPJ queries.
Contents

1 Intro
2 F-Representation
3 F-Trees
4 Query Evaluation
   • Normalisation Operator
   • Swap Operator
   • Cartesian Product Operator
   • Selection Operator
   • Projection Operator
5 Query Optimization
6 Cost of an F-Plan
   • Exhaustive Search
   • Greedy Heuristic
6 Experiments
   • Experiment #1
   • Experiment #2
   • Experiment #2
   • Experiment #3
   • Experiment #4
7 Conclusion
8 Thanks
Example Schema

<table>
<thead>
<tr>
<th>Orders</th>
<th>Store</th>
<th>Disp</th>
<th>Produce</th>
<th>Serve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>item</td>
<td>location item</td>
<td>dispatcher</td>
<td>location</td>
</tr>
<tr>
<td></td>
<td>item</td>
<td></td>
<td>location item</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>Milk</td>
<td>Istanbul Milk</td>
<td>Adnan</td>
<td>Guney</td>
</tr>
<tr>
<td>01</td>
<td>Cheese</td>
<td>Istanbul Cheese</td>
<td>Adnan</td>
<td>Guney</td>
</tr>
<tr>
<td>02</td>
<td>Melon</td>
<td>Istanbul Melon</td>
<td>Yasemin</td>
<td>Dikici</td>
</tr>
<tr>
<td>03</td>
<td>Cheese</td>
<td>Izmir Milk</td>
<td>Volkan</td>
<td>Dikici</td>
</tr>
<tr>
<td>03</td>
<td>Melon</td>
<td>Antalya Milk</td>
<td></td>
<td>Dikici</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Antalya Cheese</td>
<td></td>
<td>Antalya</td>
</tr>
</tbody>
</table>

**Figure 1: An example database for a grocery retailer.**
Factorized Representations

- Relational algebra expressions.
- Constructed with singleton relations \( \langle v \rangle \), the union \( \cup \) and product operators \( \times \).
- Exponentially more succinct than relations they encode.
- Fast (constant-delay) enumeration of tuples
Factorized Representations

Definition 1

A factorised representation $E$, or f-representation for short, over a set $S$ of attributes and domain $\mathcal{D}$ is a relational algebra expression of the form

- $\emptyset$, empty relation over $S$;
- $\langle \rangle$, the relation consisting of the nullary tuplem if $S = \emptyset$;
- $\langle A : a \rangle$, the unary relation with a single tuple with value $a$, if $S = \{A\}$ and $a$ is a value in the domain $\mathcal{D}$;
- $(E)$, where $E$ is an f-representation over $S$;
- $E_1 \cup \cdots \cup E_n$, where each $E_i$ is an f-representation over $S$;
- $E_1 \times \cdots \times E_n$, where each $E_i$ is an f-representation over $S_i$ and $S$ is the disjoint union over all $S_i$. 
Factorized Representations

Singleton

- $\langle A : a \rangle$ is a singleton
- $|E|$ the size is the number of singletons
- A single system may have many different f-representations.
- The tuples of a given f-representation can be enumerated with $O(|S|)$ additional space and delay.
Factorized Representations

Singleton

- \( \langle A : a \rangle \) is a singleton
- \( |E| \) the size is the number of singletons
- A single system may have many different f-representations.
- The tuples of a given f-representation can be enumerated with \( O(|S|) \) additional space and delay.

### Factorized Representation

(\(01\) \( \times \) Milk \( \times \) Istanbul \( \times \) Adnan) \( \cup \)
(\(01\) \( \times \) Milk \( \times \) Istanbul \( \times \) Yasemin) \( \cup \)
(\(01\) \( \times \) Milk \( \times \) Izmir \( \times \) Adnan) \( \cup \)
(\(01\) \( \times \) Milk \( \times \) Antalya \( \times \) Volkan) \( \cup \ldots \)

### Compact Factorized Representation

(\(\text{Milk} \times (01) \times ((\text{Istanbul}) \times ((\text{Adnan}) \cup \text{Yasemin})) \cup \text{Izmir} \times \text{Adnan} \cup \text{Antalya} \times \text{Volkan}) \cup \text{Cheese} \times ((01) \cup (03)) \times ((\text{Istanbul}) \times ((\text{Adnan}) \cup \text{Yasemin}) \cup \text{Antalya} \times \text{Volkan}) \cup \text{Melon} \times ((02) \cup (03)) \times (\text{Istanbul} \times ((\text{Adnan}) \cup \text{Yasemin}) \cup \ldots) \)
F-Trees

Definition

An *f-tree* for short, over a schema $S$ of attributes is an **unordered rooted forest** with each node labelled by a non-empty subset of $S$ such that each attribute of $S$ labels exactly one node.

<table>
<thead>
<tr>
<th>Q1 F-Tree</th>
<th>Q1 Alternate F-Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="F-Tree Diagram" /></td>
<td><img src="image2" alt="Alternate F-Tree Diagram" /></td>
</tr>
</tbody>
</table>
An *f-tree* for short, over a schema $S$ of attributes is an **unordered rooted forest** with each node labelled by a non-empty subset of $S$ such that each attribute of $S$ labels exactly one node.

- Tree created in $O(|S| \cdot |R| \cdot \log |R|)$.
- We can go back to $R$ in $O(|S| \cdot |R|)$. 
Given a query \( Q = \pi_\mathcal{P} \sigma_\phi (R_1 \times \cdots \times R_n) \)

- Given a query \( Q \), an f-tree \( T \) is an f-tree of \( Q \) if and only if it satisfies the *path constraint*.
- all dependent attributes can only label nodes along a same root-to-leaf path.
- Nodes of a join are only dependent if they are in the \( \mathcal{P} \) list or the same \( R_i \).
F-Trees
Size Bounds

Let $s\mathcal{T}$ be the maximal root-to-leaf path edge cover $\mathcal{T}$.

- For and database $\mathbf{D}$ and f-tree $\mathcal{T}$ the result size is at most $O(|\mathcal{P}| \cdot |\mathbf{D}|^{s(\mathcal{T})})$
Let $s\mathcal{T}$ be the maximal root-to-leaf path edge cover $\mathcal{T}$.

- For and database $\mathcal{D}$ and f-tree $\mathcal{T}$ the result size is at most $O(|\mathcal{P}| \cdot |\mathcal{D}|^{s(\mathcal{T})})$
- If we let $s(Q)$ be the minimal $s(\mathcal{T})$ over all f-trees.
Let $sT$ be the maximal root-to-leaf path edge cover $T$.

- For and database $D$ and f-tree $T$ the result size is at most $O(|P| \cdot |D|^{s(T)})$
- If we let $s(Q)$ be the minimal $s(T)$ over all f-trees.
- $|P| \cdot |D|^{s(Q)} \ll Q(D)$ — This can mean asymptotically smaller, exponential size and time savings
F-plans

- Any select-project-join query can be evaluated by a sequential composition of operators called an *f-plan*.
- f-trees uniquely identify f-representations → operators are in tree form.
- The time complexity of each f-plan operator is $O(|T|^2 N \log N)$, where $N$ is the sum of input and output f-representations and $T$ is the input f-tree.
Normalisation Operator

push-up operator $\psi_B$

normalization $\eta(T)$

- A tree is normalized if the push up operator can not be applied.
- The push-up operator is applied bottom up.

1 All other operators expect normalized inputs
Swap Operator $\mathcal{X}_{A,B}$

Exchanges $B$ with its parent $A$ and promotes children.
Cartesian Product Operator

\[ T' = T_\infty \times T_\in \]
Merge Selection Operator

$\mu_{A,B}$

If $A$ and $B$ are siblings
Merge Selection Operator

\( \mu_{A,B} \)

If \( A \) and \( B \) are siblings

\[
\mu(\ldots, \item, \ldots) = \ldots
\]
Absorb Selection Operator

$\alpha_{A,B}$

If $A$ is an ancestor of $B$

Example:
Selection with Constant Operator

\( \sigma_{A \theta c} \)

Filters out all the values in \( \langle A : a \rangle \) where \( a \n \theta c \)
Removes a node from the tree. Only permitted if $A$ is a leaf node or with other attributes.
Query optimization has two objectives:

- minimizing the *cost* of computing a factorised query result;
- minimizing the *size* of this output representation.
Optimize the f-tree then the f-representation.

Products and Selections are always evaluated first (cheapest).

Selection \((A = B)\) can only be done if they are on the same path or siblings.
Cost of an F-Plan

- use asymptotic bounds of $s(\mathcal{T})$.
- $\mathcal{T}_{\text{initial}} = \mathcal{T}_0 \supseteq \mathcal{T}_1 \supseteq \ldots \supseteq \mathcal{T}_k = \mathcal{T}_{\text{final}}$, with an evaluation time of $O(|\mathbf{D}^{s(f)}| \cdot \log |\mathbf{D}|)$, where $s(f) = \max(s(\mathcal{T}_0), \ldots, s(\mathcal{T}_k))$. 
Cost of an F-Plan

- Use asymptotic bounds of $s(T)$.

$T_{\text{initial}} = T_0 \not\rightarrow T_1 \not\rightarrow \ldots \not\rightarrow T_k = T_{\text{final}}$, with an evaluation time of $O(|D|^{s(f)} \cdot \log |D|)$, where $s(f) = \max(s(T_0, \ldots, s(T_k))$.

- Or, use cardinality of estimate from intermediate f-trees using RDBMS style techniques.
Exhaustive Search

- Enumerate plans and the one with the shortest path (Dijkstra) between $T_{init}$ to $T_{final}$ is least cost.
- Search space is $O(n^{4n})$ with $n - p$ attributes in the projection list.
Greedy Heuristic

- Only restructures nodes that appear in selections and projections.
- In projection ordering chooses the least cost at each step.
- Computing the tree take $O(|T|^2)$. 
Experiments

Contents

1 Intro
2 F-Representation
3 F-Trees
4 Query Evaluation
  • Normalisation Operator
  • Swap Operator
  • Cartesian Product Operator
  • Selection Operator
  • Projection Operator
5 Query Optimization
6 Experiments
  • Cost of an F-Plan
  • Exhaustive Search
  • Greedy Heuristic
  • Experiment #1
  • Experiment #2
  • Experiment #2
  • Experiment #3
  • Experiment #4
7 Conclusion
8 Thanks

cgrant (UF)  FDB: Factorised RDBMS  November 1, 2013  28 / 39
Experiments

Implementation

- Compare SQLite and PostgreSQL with FDB and RDB
- for SQLite and PostgreSQL tried to disable expensive writing (try to make it in-memory)
- Implemented in C++
- Optimal f-representation computed with multi-way merge-sort join algorithm.
Experiment Design

- Generate random data for queries
- Repeat a number of times for optimizations time, execution time, representation sizes and f-plan quality
- Tuples generated independently with a uniform or Zipf distribution.
- Queries are equality joins
Experiment #1

- Average time for optimising a query on flat data.
- $A = 40$ Attributes, $R = 1, \ldots, 8$ relations, $K = 1, \ldots, 9$ selections
Experiment #1

- Average time for optimising a query on flat data.
- \(A = 40\) Attributes, \(R = 1, \ldots, 8\) relations, \(K = 1, \ldots, 9\) selections

![Graph showing average cost of an optimal f-tree for random query of \(K\) equalities on \(R\) relations.](image-url)
Experiment #1

- Average time for optimising a query on flat data.
- \( A = 40 \) Attributes, \( R = 1, \ldots, 8 \) relations, \( K = 1, \ldots, 9 \) selections
Experiment #2

Average costs of f-plans and resulting f-trees, computed by full search and greedy query optimisers. Input f-trees have $R = 4$ relations, $A = 10$ attributes.
Experiment #2

Finding an f-plan for random queries L equalities on an input f-tree with R=4 relations, A=10 attributes and K equalities.

(number K of equalities in input f-tree \(K + L < A\))

- \(L = 6\)
- \(L = 5\)
- \(L = 4\)
- \(L = 3\)
- \(L = 2\)
- \(L = 1\)
Experiment #3

Compare performance of FDB, RDB, SQLite and PostgreSQL

Results size

Evaluation Time

3 relations with 3 attributes each Zipf distribution over [1 .. 100]

Factorised/flat result size ratio for selected TPC-H queries.
Experiment #3

Compare performance of FDB, RDB, SQLite and PostgreSQL.

Each tuple size decreases results size by 20

Results size vs Evaluation time
Experiment #3

Compare performance of FDB, RDB, SQLite and PostgreSQL.

In $Q_A$ is a many-many query. This is highly compressible so the factorized result is small.
Experiment #4
Experiment over Factorized data

Results size

Evaluation time
Conclusion

- Factorizing Relational Data prevents the explosion of data size for large joins.
- This paper described techniques and experiments to decrease this size and improving query performance.
- For MLNs and other graph DBs that require many joins this can significantly improve query performance.
Thank you

Questions?