Acorn

Aggressive Caching in Distributed Data Processing Frameworks

Lana Ramjit
UCLA

Matteo Interlandi
Microsoft

Eugene Wu
Columbia

Ravi Netravali
UCLA
Big Data Processing

relational data

distributed computing
Iterative Workloads

- Graph Processing
  - Connected Components
  - PageRank
- Machine Learning
  - Belief Propagation
  - k-means clustering
- Interactive Data Exploration
  - single or multiple users

Overlap between iterations should not be recomputed!
Caching Avoids Recomputing Overlap

Solution: Cache manager should find as many opportunities as possible to reuse old results

- transparently find caching opportunities
  - no user input!

- automatically rewrite incoming queries to use cache
Two Challenges

1. Pipeline introduces obstacles for effective caching

2. User Defined Functions (UDFs)
A Series of Queries

Query 1

```
people.filter{p => p.age > 18}
```

Query 2

```
people.join(pets, "id === owner")
  .filter(people.age > 18)
```
Example: Query 1

```
people.filter(age > 18)
```

**Cache**

- `filter` `{ p => p.age > 18 }
  - `table` people

**Optimization**

- `filter` `{ p => p.age > 18 }
  - `table` people

**Physical Planning**

- `filter` `{ p => p.age > 18 }
  - `FileScan` people
Example: Query 2

```
people.join(pets, "id === owner")
  .filter(people.age > 18)
```

Cache Substitution

- `select *`
  - `filter people.age > 18`
  - `join (owner, id)`
    - `table people`
    - `table pets`

Optimization

- `select *`
  - `join (owner, id)`
    - `filter people.age > 18`
      - `table people`
      - `table pets`

Physical Planning

- `select *`
  - `hashjoin (owner, id)`
    - `filter people.age > 18`
      - `filesan pets`
    - `table pets`
  - `filter people.age > 18`
    - `filesan people`

Cache

- `filter { p => p.age > 18 }`
Missed Opportunity!

Physical Planning

Cache Substitution

select *
filter
people.age > 18
join (owner, id)
table
people

Optimization

select *
join (owner, id)
table
people

Should match the cache, but blackbox UDF prevents match

filter
{ p => p.age > 18 }
table people

doesn’t exactly match tree in the cache!

Physical Planning

select *
hashjoin (owner, id)
table
pets

filter
people.age > 18
filesan pets

filter
people.age > 18
filesan people

filesan people

filesan pets
Two Challenges (recap)

1. Pipeline introduces obstacles for effective caching
   ○ Cache compares unoptimized instead of optimized plans
   ○ unoptimized == uncanonicalized

2. User Defined Functions (UDFs)
   ○ prevent high coverage
   ○ blackboxes to optimizers
So, fix the pipeline?

Current Pipeline

Optimization-first pipeline
Optimization Is Slow

Optimization time per iteration of connected components algorithm
Why Is Optimization Slow?

- General optimizer
  - targeting diverse workloads
  - custom rules

- Immutable data
  - can't "update" underlying data
  - all updates are logged in query plan
  - ➔ very large query plans
So, fix the pipeline?

Current Pipeline

- Cache
- Optimization
- Physical Planning

Optimization-first pipeline (slow!)

- Optimization
- Cache
- Physical Planning

Insight: not all optimizations help caching!

- Partial Optimization
- Cache
- Optimization
- Physical Planning
Partial Optimization

Boolean Simplification
Constant Propagation

ID Reassignment
Filter Pruning
Object Elimination
Custom Rules
...

Canonicalizing Rules (cheap)

Other rules (expensive)
Partial Optimization Scales

```
<table>
<thead>
<tr>
<th>Iteration #</th>
<th>Full Optimization</th>
<th>Partial Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>100</td>
</tr>
</tbody>
</table>
```

Keep the canonicalizing benefits of optimization without the price.
Partial Optimization Uses Cache More

- TPC-DS benchmark
- iterative style queries
- scaled to 100GB on a 16 machine cluster
- **Performance improved by 2.2X**
Two Challenges

1. Pipeline introduces obstacles for effective caching
   ○ Cache compares unoptimized instead of optimized plans
   ○ unoptimized == uncanonicalized

2. UDFs
   ○ prevent high coverage
   ○ blackboxes to optimizers
UDFs

UDFs are blackboxes that hide caching opportunities

select *
where age > 18
from table people

select *
{ p => p.age > 18 }
from table people
# UDF Translation

<table>
<thead>
<tr>
<th>Program Synthesis</th>
<th>User Annotation</th>
<th>Froid</th>
<th>Acorn</th>
</tr>
</thead>
</table>

20
<table>
<thead>
<tr>
<th>Correct</th>
<th>Program Synthesis</th>
<th>User Annotation</th>
<th>Froid</th>
<th>Acorn</th>
</tr>
</thead>
</table>

- ✓
- ✓
- ✓
- ✓
## UDF Translation

<table>
<thead>
<tr>
<th></th>
<th>Program Synthesis</th>
<th>User Annotation</th>
<th>Froid</th>
<th>Acorn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transparent</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## UDF Translation

<table>
<thead>
<tr>
<th></th>
<th>Program Synthesis</th>
<th>User Annotation</th>
<th>Froid</th>
<th>Acorn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transparent</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>General</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>(Java, Scala)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* ✓: Supported
  * X: Not Supported

* Correct
  * Transparent
  * General (Java, Scala)
## UDF Translation

<table>
<thead>
<tr>
<th>Feature</th>
<th>Program Synthesis</th>
<th>User Annotation</th>
<th>Froid</th>
<th>Acorn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transparent</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>General</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>General (Java, Scala)</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Fast</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
UDF Translation

- arbitrarily long or complex
- user-defined types (classes)
- anonymous functions

**translate via symbolic execution**

### Scala

```scala
def q6cond(shipDate: Long, disc: Double, qty: Int) = {
  val d1 = 757468799
  val d2 = 31536000
  if (shipDate < d1
    && shipDate >= d1 + d2
    && qty >= 24)
    return false
  val epsilon = .01
  val dec = .06
  var lower = dec - epsilon
  var upper = dec + epsilon
  return discount >= lower
  && discount <= upper
}
```

### Native Spark

**TPC-H**

```scala
If(LessThan(shipDate, 757468799),
If(GreaterThanOrEqualTo(shipdate, Add(757468799, 31536000)),
If(GreaterThanOrEqualTo(qty, Literal(24)), If
(GreaterThanOrEqualTo(discount, Subtract(Literal(.01),
Literal(.06))),
If(LessThanOrEqualTo(discount, Add(Literal(.01),
Literal(.06))),
true, false), false), false), false)
```

---

**Open Source**

```scala
_.births > 100
```
Step 1: Translate to an IR

person.filter(p => p.age > 18)

1  aload_1
2  invokeinterface
3  dload_1
4  ldc2_w
5  dcmpg
6  ifge 18
7  iconst_1
8  goto 10
9  iconst_0
10  aload_0
11  aload_1

1  Person r1 := @param0
2  double $d0 = r1.age()
3  int $d1 = 18
4  if $d0 < $d1
5  goto 8
6  boolean $zo = 1
7  goto 9
8  $zo = 0
9  return $zo
Step 2: Symbolic Execution

1 Person r1 := @param0
2 double $d0 = r1.age()
3 int $d1 = 18
4 if $d0 > $d1
5 goto 8
6 boolean $zo = 1
7 goto 9
8 $zo = 0
9 return $zo

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>class[Person]</td>
<td>this</td>
</tr>
</tbody>
</table>
Step 2: Symbolic Execution

1. Person r1 := @param0
2. double $d0 = r1.age()
3. int $d1 = 18
4. if $d0 > $d1
5. goto 8
6. boolean $zo = 1
7. goto 9
8. $zo = 0
9. return $zo

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>class[Person]</td>
<td>this</td>
</tr>
<tr>
<td>d0</td>
<td>double</td>
<td>Attribute(&quot;age&quot;)</td>
</tr>
</tbody>
</table>
Step 2: Symbolic Execution

1. Person r1 := @param0
2. double $d0 = r1.age()
3. int $d1 = 18
4. if $d0 > $d1
5. goto 8
6. boolean $zo = 1
7. goto 9
8. $zo = 0
9. return $zo

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>class[Person]</td>
<td>this</td>
</tr>
<tr>
<td>d0</td>
<td>double</td>
<td>Attribute(&quot;age&quot;)</td>
</tr>
<tr>
<td>d1</td>
<td>int</td>
<td>Literal(18)</td>
</tr>
</tbody>
</table>
Step 2: Symbolic Execution

1. Person r1 := @param0
2. double $d0 = r1.age()
3. int $d1 = 18
4. if $d0 > $d1
   5. goto 8
6. boolean $zo = 1
7. goto 9
8. $zo = 0
9. return $zo

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>class[Person]</td>
<td>this</td>
</tr>
<tr>
<td>d0</td>
<td>double</td>
<td>Attribute(&quot;age&quot;)</td>
</tr>
<tr>
<td>d1</td>
<td>int</td>
<td>Literal(18)</td>
</tr>
</tbody>
</table>
Person r1 := @param0
double $d0 = r1.age()
int $d1 = 18
if $d0 > $d1
goto 8
boolean $zo = 1
goto 9
$zo = 0
return $zo

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>class[Person]</td>
<td>this</td>
</tr>
<tr>
<td>d0</td>
<td>double</td>
<td>Attribute(&quot;age&quot;)</td>
</tr>
<tr>
<td>d1</td>
<td>int</td>
<td>Literal(18)</td>
</tr>
</tbody>
</table>
Step 2: Symbolic Execution

1 Person r1 := @param0
2 double $d0 = r1.age()
3 int $d1 = 18
4 if $d0 > $d1
5 goto 8
6 boolean $zo = 1
7 goto 9
8 $zo = 0
9 return $zo

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>class[Person]</td>
<td>this</td>
</tr>
<tr>
<td>d0</td>
<td>double</td>
<td>Attribute(&quot;age&quot;)</td>
</tr>
<tr>
<td>d1</td>
<td>int</td>
<td>Literal(18)</td>
</tr>
</tbody>
</table>
Step 2: Symbolic Execution

```java
Person r1 := @param0
double $d0 = r1.age()
int $d1 = 18
if $d0 > $d1
goto 8
boolean $zo = 1
goto 9
$zo = 0
return $zo
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>class[Person]</td>
<td>this</td>
</tr>
<tr>
<td>d0</td>
<td>double</td>
<td>Attribute(&quot;age&quot;)</td>
</tr>
<tr>
<td>d1</td>
<td>int</td>
<td>Literal(18)</td>
</tr>
</tbody>
</table>
IF

GreaterThan(Attribute("age"),
Literal(18))

cast (1) as boolean

cast(0) as boolean

filterUDF{ p => p.age > 18 }

select age

table people

person.filter(p => p.age > 18)
Step 3: Rewriting

\[
\text{select age} \quad \mid \quad \text{filter(If(GreaterThan("age", 18), cast 0 as boolean, cast 1 as boolean))} \\
\text{table people}
\]

\[
\text{person.filter(p => p.age > 18)}
\]
Step 3: Rewriting

```
select *
from table people
filter If(GreaterThan("age", 18), cast 0 as boolean, cast 1 as boolean))
```

```
select *
from table people
filter "age" > 18

Partial Optimizer
```

```
person.filter(p => p.age > 18)
```

```
person.filter(age > 18)
```
Inserted UDFs into TPC-H

No caching opportunities

Same benchmark used by Microsoft's Froid

UDF translation is faithful to native SQL
partial optimization + udf translation
Open Source Applications: Caching+UDFs

connected components on BerkStan and Twitter networks contains UDFs and iteration

3X improvement over the baseline

hand-optimized workload, 1.4X improvement over baseline
Open Source Applications: Caching+UDFs

belief propagation on BerkStan and Twitter networks

contains UDFs and iteration

3X improvement over the baseline

hand-optimized workload, 1.4X improvement over baseline
Acorn

Aggressive caching in big data systems

First Java/Scala UDF ➔ SQL Translator

Integrated into Spark 2.3.2

lana@cs.ucla.edu
Comparing Pipelines

- comparing proposed pipelines on TPC-DS benchmark
- time each stage in the pipeline
- **Acorn pipeline** minimizes time spent in optimizer without impacting execution time
Translation Limits

- Inherent limitation
  - unbounded loops (including recursion)
  - bounded loops (ex: foreach) are ok
- Limited by target language
- Non-determinism