Profiling Dataflow Systems on Multiple Abstraction Levels

Alexander Beischl, Timo Kersten, Maximilian Bandle
Jana Giceva, Thomas Neumann

Technische Universität München
### Compiling Dataflow Systems are Everywhere!

Dataflow systems in different areas

<table>
<thead>
<tr>
<th>Machine- and deep learning</th>
<th>Graph and stream-processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>Flink</td>
</tr>
<tr>
<td>PyTorch</td>
<td>Ligra</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Big-data processing</th>
<th>Relational DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Spark</td>
<td>Microsoft Hekaton</td>
</tr>
<tr>
<td>Microsoft Naiad</td>
<td>SAP HANA</td>
</tr>
<tr>
<td></td>
<td>UMBRA</td>
</tr>
</tbody>
</table>
Profiling a Compiling Dataflow System

Trying to optimize the system

Query

```python
df_sales.join(df_CPUs,
    col("df_sales.cpuID" ==
    col("df_CPUs.ID"), "inner")
```
Profiling a Compiling Dataflow System

Trying to optimize the system

```
for tuple t in table T
if t[1] > 5
...
load int32 %40, i64 %13
isnotnull ptr %12
...
mov rax, [4 * rbx]
cmp rax, 0
je ...
```

```
Query

df_sales.join(df_CPUs,
col("df_sales.cpuID" ==
col("df_CPUs.ID"), "inner")

Dataflow System

\[
\begin{align*}
\text{for tuple } t \text{ in table } T \\
\text{if } t[1] > 5 \\
\text{load int32 } %40, \text{ i64 } %13 \\
\text{mov rax, [4 * rbx]} \\
\text{cmp rax, 0} \\
\text{je } ...
\end{align*}
\]
```
Profiling a Compiling Dataflow System

Trying to optimize the system

Query

```
df_sales.join(df_CPUs,
col("df_sales.cpuID" ==
col("df_CPUs.ID"), "inner")
```

Dataflow System

```
# for tuple t in table T
load int32 %40, i64 %13
isnotnull ptr %12
mov rax, [4 * rbx]
cmp rax, 0
je ...
```

Execution Time

5067ms
Profiling a Compiling Dataflow System

Trying to optimize the system

Query

```
df_sales.join(df_CPUs,
col("df_sales.cpuID" ==
col("df_CPUs.ID"), "inner")
```

Dataflow System

```
\[
\begin{align*}
&\text{for tuple } t \text{ in table } T \\
&\text{load int32 } %40, i64 %13 \\
&\text{mov rax, } [4 \times rbx] \\
\end{align*}
\]
```

Execution Time

```
5067ms
```
Profiling a Compiling Dataflow System

Trying to optimize the system

Query
```
df_sales.join(df_CPUs, col("df_sales.cpuID" == col("df_CPUs.ID"), "inner")
```

Dataflow System
```
^ for tuple t in table T
  load int32 %40, i64 %13
  mov rax, [4 * rbx]
```

Execution
```

```
Execution Time
```

5067ms

Query
```
df_sales.join(df_CPUs, col("df_sales.cpuID" == col("df_CPUs.ID"), "inner")
```

Dataflow System
```
^ for tuple t in table T
  load int32 %40, i64 %13
  mov rax, [4 * rbx]
```

.Execution
```

```
Execution Time
```

567ms
Profiling a Compiling Dataflow System

Trying to optimize the system

Query
```
df_sales.join(df_CPUs,
col("df_sales.cpuID" ===
col("df_CPUs.ID"), "inner")
```

Dataflow System
```
for tuple t in table T
  load int32 %40, i64 %13
  mov rax, [4 * rbx]
```

Execution
```
5067ms
```

Perf report
```
(15%)
```

```
567ms
```

```
32%
```

```
567ms
```

```
(15%)
```

```
567ms
```

```
32%
```

```
567ms
```

```
(15%)
```

```
567ms
```

```
32%
```

```
567ms
```

```
(15%)
```

```
567ms
```
Profiling a Compiling Dataflow System

Trying to optimize the system

```
df_sales.join(df_CPUs, col("df_sales.cpuID" == col("df_CPUs.ID"), "inner"))
```

```
for tuple t in table T
  load int32 %40, i64 %13
  mov rax, [4 * rbx]
```

```
\[
\text{for tuple } t \text{ in table } T \\
\text{load int32 } %40, i64 %13 \\
\text{mov rax, [4 * rbx]}
\]

```

```
\text{Execution Time: 5067ms}
```

```
\text{Profiling Dataflow Systems on Multiple Abstraction Levels}
```
Profiling a Compiling Dataflow System

Trying to optimize the system

---

Query

\[
\text{df\_sales.join(df\_CPUs, \\
\text{  col("df\_sales.cpuID" \text{==} \\
\text{  col("df\_CPUs.ID"}, "inner")}}
\]

Dataflow System

```java
\[
\text{\(\begin{align*}
&\text{for tuple t in table T} \\
&\text{  load int32 \%40, i64 \%13} \\
&\text{  mov rax, [4 * rbx]} \\
\end{align*}\)}
\]

Execution

5067ms

---

Perf report

```java
\[
\text{\begin{align*}
&\text{0% \%localTid = phi [\%1, \%loopBlocks \%2, \%contScan]} \\
&\text{0.1\% \%3 = getelementptr int8 \%state, i64 320} \\
&\text{0.1\% \%4 = getelementptr int8 \%3, i64 262144} \\
&\text{2.2\% \%5 = load int32 \%4, \%localTid} \\
&\text{2.3\% \%7 = crc32 i64 5961697176435608501, \%5} \\
&\text{1.5\% \%8 = crc32 i64 2231409791114444147, \%5} \\
&\text{1.2\% \%9 = rotl i64 \%8, 32} \\
&\text{2.3\% \%10 = xor i64 \%7, \%9} \\
&\text{2.2\% \%11 = mul i64 \%10, 2665821657736338717} \\
&\text{1.2\% \%12 = shr \%11, 16} \\
&\text{2.4\% \%13 = getelementptr int8 \%5, i64 \%12} \\
&\text{32.1\% \%14 = load int32 \%40, i64 \%13} \\
&\text{0.2\% \%15 = isnotnull ptr \%12} \\
&\text{0.3\% \%15 = loopHashChain %nextTuple} \\
\end{align*}\)}
\]
Why do we have this problem?

Identifying the gap

```
Query
↓
Dataflow System
   ↓
Dataflow Graph
   ↓
Imperative Prog.
   ↓
Machine IR
   ↓
x86 Assembly
```
Why do we have this problem?

Identifying the gap

Query

Dataflow System

- Dataflow Graph
- Imperative Prog.
- Machine IR
- x86 Assembly

```
df_sales.join(df_CPUs,
  col("df_sales.cpuID")
...)
```
Why do we have this problem?

Identifying the gap

**Query**

**Dataflow System**

- Dataflow Graph
- Imperative Prog.
- Machine IR
- x86 Assembly
Why do we have this problem?

Identifying the gap

```
for tuple t in table T
  if t[1] > 5
  ...
```
Why do we have this problem?

Identifying the gap

Dataflow System

- Dataflow Graph
- Imperative Prog.
- Machine IR
- x86 Assembly

Query

---

```python
for tuple t in table T
    if t[1] > 5
...```

```c
load int32 %40, i64 %13
isnotnull ptr %12
...```
Why do we have this problem?

Identifying the gap

```
mov rax, [4 * rbx]
cmp rax, 0
je  ...  
```
Why do we have this problem?

Identifying the gap

Query

Dataflow System

Dataflow Graph

Imperative Prog.

Machine IR

x86 Assembly

Profiling Samples

Execution

Profiling

Result
Why do we have this problem?

Identifying the gap

Query

Dataflow System

Dataflow Graph
Imperative Prog.
Machine IR
x86 Assembly

Machine IR Results
Profiling Samples

Execution

Profiling

Result
Why do we have this problem?
Identifying the gap

Query → Dataflow System

Dataflow Graph
Imperative Prog.
Machine IR
x86 Assembly

Profiling Samples
Machine IR Results
CodeGen Dev

Execution → Profiling

Result
Why do we have this problem?

Identifying the gap

![Diagram showing the process of dataflow systems profiling on multiple abstraction levels.](image)
Why do we have this problem?

Identifying the gap

Query → Dataflow System

- Dataflow Graph
- Imperative Program
- Machine IR
- x86 Assembly

Execution → Profiling

Result → Profiling Samples

Machine IR Results

32%

for tuple t in table T
if t[1] > 5...

Domain Expert
Optimizer Dev
CodeGen Dev

beischl@in.tum.de
Why do we have this problem?

Identifying the gap

Query

Dataflow System

Dataflow Graph
Imperative Prog.
Machine IR
x86 Assembly

Connection lost

Execution

Profiling

Result

Machine IR Results
Profiling Samples

for tuple t in table T
if t[1] > 5
...
Tailored Profiling

Closing the gap

- Track connection between components of all abstraction levels down to generated code
- Map profiling samples back to higher abstraction levels

- Ingredients
  - Tagging Dictionary & Register Tagging
① *Connection tracking* of abstraction components for each lowering step (top-down)
Tailored Profiling

Tagging Dictionary

② Store mapping in the Tagging Dictionary
Tailored Profiling

Tagging Dictionary

③ Map profiling results to each abstraction level’s components (bottom-up)
Tailored Profiling

Tagging Dictionary

Aggregate the data for profiling results
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

Generated Query Code

for each tuple t in sales

...  

if t.price > 500  

...
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

\[ \text{Scan sales} \]
\[ \text{Filter}_{\text{price}>500} \]
\[ \ldots \]

Generated Query Code

\[
\text{for each tuple } t \text{ in sales} \\
\text{...} \\
\text{if } t.\text{price} > 500 \\
\text{...}
\]

\{for each tuple } t \text{ in sales } s \rightarrow \text{Scan sales}\}
Tailored Profiling

Tagging Dictionary and Register Tagging

**Dataflow Graph**
- ...  
  - Filter\(_{price>500}\)
  - Scan sales

**Generated Query Code**
```c
for each tuple t in sales
  ...
  if t.price > 500
  ...
{if t.price > 500 -> Filter}
```

*Tagging Dictionary*
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

```
...  
|        |
Filter_{price>500}  
|        |
Scan sales
```

Generated Query Code

```
for each tuple t in sales
    call malloc(...)
    if t.price > 500
        call malloc(...)
```
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

```
...  
Filter_{price>500}  
Scan sales
```

Generated Query Code

```
for each tuple t in sales
  call malloc(...)
  if t.price > 500
    call malloc(...)  
```

Profiling Sample

```
Source Line
malloc(...)  
```

Scan, Filter?
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

Generated Query Code

for each tuple \( t \) in sales

\[
\text{call malloc(…)}
\]

if \( t.\text{price} > 500 \)

\[
\text{call malloc(…)}
\]

Profiling Sample

Source Line

malloc(…)

Tagging Dictionary

Scan, Filter?

Call-Stack Sample

Recorded Call-Stack

malloc(…)

Scan: call malloc()

...
Tailored Profiling

Tagging Dictionary and Register Tagging

**Dataflow Graph**

```
... 
Filter_{price>500} 
Scan sales 
```

**Generated Query Code**

```
for each tuple t in sales 
  call malloc(...) 
  if t.price > 500 
    call malloc(...) 
```

**Machine Register**
Tailored Profiling

Tagging Dictionary and Register Tagging

**Dataflow Graph**

```
...                          
|                             |
Filter price > 500           
|                             |
Scan sales                    
```

**Generated Query Code**

```
for each tuple t in sales
  setTag(Scan)
  call malloc(...)  
  unsetTag()        
  if t.price > 500
     setTag(Filter)
     call malloc(...) 
     unsetTag()      
```

**Machine Register**

...
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

Generated Query Code

```python
for each tuple t in sales
  setTag(Scan)
  call malloc(...)
  unsetTag()
  if t.price > 500
    setTag(Filter)
    call malloc(...)
    unsetTag()
```

Machine Register

Scan
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

```
... 
<table>
<thead>
<tr>
<th>Filter t.price &gt; 500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan sales</td>
</tr>
</tbody>
</table>
```

Generated Query Code

```
for each tuple t in sales
  setTag(Scan)
  call malloc(...)
  unsetTag()
  if t.price > 500
    setTag(Filter)
    call malloc(...)
    unsetTag()
```

Machine Register

```
Scan
```

Profiling Sample

```
<table>
<thead>
<tr>
<th>Source Line</th>
<th>Register Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc(...)</td>
<td>Scan</td>
</tr>
</tbody>
</table>
```
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

... 

Filter \_price > 500

Scan sales

Generated Query Code

for each tuple t in sales
  setTag(Scan)
  call malloc(...)
  unsetTag()
  if t.price > 500
    setTag(Filter)
    call malloc(...)
    unsetTag()
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

... 
Filter_{price>500} 
Scan sales

Generated Query Code

for each tuple t in sales 
setTag(Scan) 
call malloc(...) 
unsetTag() 
if t.price > 500 
setTag(Filter) 
call malloc(...) 
unsetTag()

Machine Register

Filter
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

```plaintext
... 
Filter \text{price} > 500 
Scan sales
```

Generated Query Code

```plaintext
for each tuple \textit{t} in sales
    setTag(Scan)
    \textbf{call malloc(\ldots)}
    \textbf{unsetTag()}
    \textbf{if} \textit{t.price} > 500
        setTag(Filter)
    \textbf{call malloc(\ldots)}
    \textbf{unsetTag()}
```

Machine Register

<table>
<thead>
<tr>
<th>Machine Register</th>
<th>Filter</th>
</tr>
</thead>
</table>

Profiling Sample

<table>
<thead>
<tr>
<th>Source Line</th>
<th>Register Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc(\ldots)</td>
<td>Filter</td>
</tr>
</tbody>
</table>
Insights with Tailored Profiling
Insights with Tailored Profiling

\[
\Gamma_{l\text{\_orderkey,avg}(\ldots)} (65.1\%)
\]
\[
\land_{o\_ord=l\_ord\ldots} (32.4\%)
\]
\[
\sigma_{o\_ord<\text{’1995\’}} (0.3\%)
\]
\[
\text{Tablescan lineitem (1.6\%)}
\]
\[
\text{Tablescan orders (0.6\%)}
\]

Time per operator

beischl@in.tum.de
Insights with Tailored Profiling

Time per operator

Context-aware profiling over time
We begin the use cases with the domain expert and proceed.

were conducted on a machine with an Intel Core i7-7700K (TSC) 

the throttling and handed the samples to Tailored Pro

perf version 5.2 

of 4.4 GHz), 64 GB DRAM and Ubuntu 20.04. We used Linux

cchine had an Intel Core i9-9900X with 3.5 GHz (turbo boost

running at 4.2 GHz (turbo boost of 4.5 GHz), 32 GB DRAM

ality, e.g., to avoid locking and other side-

effects. Hence, we all queries single-threaded with Umbra

and scale factor 10 (dataset size 10 GB) to measure perfor-

success metrics, we show the value of Tailored Pro

is hard to quantify and very subjective. Thus, instead of

In this section we evaluate the advantages of Tailored Pro

5.5 Precise Timestamps for Pro

structions through the task level back to the original data

partitioning and aggregation. Thus, we can map LLVM IR in-

Profiling Dataflow Systems on Multiple Abstraction Levels EuroSys '21, April 26–29, 2021, Online, United Kingdom

However, the samples' timestamps provided by the Linux

le costs and operator activity, we used

if the tuple count is a decent approximation, our sampling
izes the statistics in an operator tree. However, even though

counts how many tuples each operator processes and visual-

may decide to take computational shortcuts and add a sam-

reduce the cost of the join computation. Alternatively, they

decision on whether to, e.g., introduce index structures to

for experimental clar-

After this hint, further investigation reveals that lineitem

con

over time in the probing pipeline (cf. Figure 11). The report

the cause and re

slower runtime than the alternative.

consequently reduce the number of tuples that also probe

seems like a good option as the query plan

could choose either plan. Choosing the left one (Figure 10a)

result sizes, so with the standard cost function the optimizer

shown in Figure 10. Both plans have identical intermediate

Optimizer Developer.

the tuple count is a decent approximation, our sampling

izes the statistics in an operator tree. However, even though

counts how many tuples each operator processes and visual-

may decide to take computational shortcuts and add a sam-

reduce the cost of the join computation. Alternatively, they

decision on whether to, e.g., introduce index structures to

for experimental clar-

After this hint, further investigation reveals that lineitem

con

over time in the probing pipeline (cf. Figure 11). The report

the cause and re

slower runtime than the alternative.

consequently reduce the number of tuples that also probe

seems like a good option as the query plan

could choose either plan. Choosing the left one (Figure 10a)

result sizes, so with the standard cost function the optimizer

shown in Figure 10. Both plans have identical intermediate

Optimizer Developer.

the tuple count is a decent approximation, our sampling
Impact of Tailored Profiling

Where can you apply it?

- Preserve connection information to close gap
- Profiling results on high abstraction levels
Impact of Tailored Profiling

Where can you apply it?

- Preserve connection information to close gap
- Profiling results on high abstraction levels
- Lightweight, high accuracy
- Easy to integrate
- Applicable to many systems
Impact of Tailored Profiling

Where can you apply it?

- Preserve connection information to close gap
- Profiling results on high abstraction levels
- Lightweight, high accuracy
- Easy to integrate
- Applicable to many systems

- Already supported: profiling code on CPUs (multi-socket and multicore)
- Future work: heterogenous compute resources, distributed systems
Thank you for watching!