Efficiently Compiling Dynamic Code for Adaptive Query Processing

Tobias Schmidt, Philipp Fent, Thomas Neumann

Technische Universität München

ADMS’22
select count(*)
from lineitem
where commitdate < receiptdate
    and shipdate < commitdate
Adaptive Query Processing (AQP)

for l in lineitem:
    if not shipdate < commitdate:
        continue
    if not commitdate < receiptdate:
        continue

    counter++

select count(*)
from lineitem
where commitdate < receiptdate
    and shipdate < commitdate

Figure: Variant A
for l in lineitem:
    if not shipdate < commitdate:
        continue
    if not commitdate < receiptdate:
        continue

    counter++

**Figure:** Variant A

select count(*)
from lineitem
where commitdate < receiptdate
    and shipdate < commitdate

<table>
<thead>
<tr>
<th>branch-miss</th>
<th>instr.</th>
<th>exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.63</td>
<td>7.62</td>
</tr>
</tbody>
</table>

**Table:** Performance characteristics
Adaptive Query Processing (AQP)

```
for l in lineitem:
    if not shipdate < commitdate:
        continue
    if not commitdate < receiptdate:
        continue
    counter++

Figure: Variant A
```

```
select count(*)
from lineitem
where commitdate < receiptdate
    and shipdate < commitdate

<table>
<thead>
<tr>
<th>branch-miss</th>
<th>instr.</th>
<th>exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.63</td>
<td>7.62</td>
</tr>
</tbody>
</table>

Table: Performance characteristics

Tobias Schmidt, Philipp Fent, Thomas Neumann ADMS’22
Adaptive Query Processing (AQP)

```python
for l in lineitem:
    if not shipdate < commitdate:
        continue -- 51% taken
    if not commitdate < receiptdate:
        continue -- 75% taken
    counter++
```

**Figure:** Variant A

```sql
select count(*)
from lineitem
where commitdate < receiptdate
    and shipdate < commitdate
```

<table>
<thead>
<tr>
<th>branch-miss</th>
<th>instr.</th>
<th>exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.63</td>
<td>7.62</td>
</tr>
</tbody>
</table>

**Table:** Performance characteristics
Adaptive Query Processing (AQP)

for l in lineitem:
  if not commitdate < receiptdate:
    continue -- 37% taken
  if not shipdate < commitdate:
    continue -- 81% taken
  counter++

select count(*)
from lineitem
where commitdate < receiptdate
  and shipdate < commitdate

Figure: Variant B

<table>
<thead>
<tr>
<th></th>
<th>branch-miss</th>
<th>instr.</th>
<th>exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.63</td>
<td>7.62</td>
<td>18.4 ms</td>
</tr>
<tr>
<td>B</td>
<td>0.58</td>
<td>7.91</td>
<td>17.7 ms</td>
</tr>
</tbody>
</table>

Table: Performance characteristics

Tobias Schmidt, Philipp Fent, Thomas Neumann
ADMS’22
Adaptive Query Processing (AQP)

for l in lineitem:
    if not (shipdate < commitdate and commitdate < receiptdate):
        continue -- 88% taken

    counter++

Figure: Variant C

select count(*)
from lineitem
where commitdate < receiptdate
    and shipdate < commitdate

<table>
<thead>
<tr>
<th></th>
<th>branch-miss</th>
<th>instr.</th>
<th>exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.63</td>
<td>7.62</td>
<td>18.4 ms</td>
</tr>
<tr>
<td>B</td>
<td>0.58</td>
<td>7.91</td>
<td>17.7 ms</td>
</tr>
<tr>
<td>C</td>
<td>0.13</td>
<td>11.67</td>
<td>12.7 ms</td>
</tr>
</tbody>
</table>

Table: Performance characteristics
 Adaptive Query Processing (AQP)

```python
for l in lineitem:
    if not (shipdate < commitdate
            and commitdate < receiptdate):
        continue -- 88% taken

    counter++

Figure: Variant C

⇒ 50% performance improvement
```

```sql
select count(*)
from lineitem
where commitdate < receiptdate
    and shipdate < commitdate
```

<table>
<thead>
<tr>
<th></th>
<th>branch-miss</th>
<th>instr.</th>
<th>exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.63</td>
<td>7.62</td>
<td>18.4 ms</td>
</tr>
<tr>
<td>B</td>
<td>0.58</td>
<td>7.91</td>
<td>17.7 ms</td>
</tr>
<tr>
<td>C</td>
<td>0.13</td>
<td>11.67</td>
<td>12.7 ms</td>
</tr>
</tbody>
</table>

Table: Performance characteristics

Tobias Schmidt, Philipp Fent, Thomas Neumann  ADMS’22
Vision: evaluate variations of the pipeline (*exploration*) and choose the best one (*exploitation*) during execution.
Vision: evaluate variations of the pipeline (exploration) and choose the best one (exploitation) during execution.

Challenge 1: compiling pipelines multiple times is prohibitively expensive

Challenge 2: support high-level plan changes and low-level optimizations

Challenge 3: minimize the runtime overhead and avoid slowdowns
AQP in Umbra

**Vision:** evaluate variations of the pipeline (*exploration*) and choose the best one (*exploitation*) during execution.

**Challenge 1:** compiling pipelines multiple times is prohibitively expensive

**Challenge 2:** support high-level plan changes and low-level optimizations

**Challenge 3:** minimize the runtime overhead and avoid slowdowns

⇒ Dynamic Blocks code generation framework
Dynamic Blocks Framework

for l in lineitem:
    if not shipdate < commitdate:
        continue
    if not commitdate < receiptdate:
        continue
    counter++

Variant A

for l in lineitem:
    if not commitdate < receiptdate:
        continue
    if not shipdate < commitdate:
        continue
    counter++

Variant B

for l in lineitem:
    if not (shipdate < commitdate and commitdate < receiptdate):
        continue
    counter++

Variant C

Tobias Schmidt, Philipp Fent, Thomas Neumann  ADMS’22
Dynamic Blocks Framework

for l in lineitem:
    if not shipdate < commitdate:
        continue
    if not commitdate < receiptdate:
        continue
    counter++

Variant A

for l in lineitem:
    if not commitdate < receiptdate:
        continue
    if not shipdate < commitdate:
        continue
    counter++

Variant B

for l in lineitem:
    if not (shipdate < commitdate and commitdate < receiptdate):
        continue
    counter++

Variant C
for l in lineitem:
    if not shipdate < commitdate:
        continue
    if not commitdate < receiptdate:
        continue
    counter++

Variant A

for l in lineitem:
    if not commitdate < receiptdate:
        continue
    if not shipdate < commitdate:
        continue
    counter++

Variant B

for l in lineitem:
    if not (shipdate < commitdate
        and commitdate < receiptdate):
        continue
    counter++

Variant C

⇒ compile once with all code fragments
Dynamic Blocks Framework

Optional Block

Alternative Block

Reorder Block

Tobias Schmidt, Philipp Fent, Thomas Neumann ADMS’22
for l in lineitem:

Alternative Block

if not (shipdate < commitdate and commitdate < receiptdate):
    continue

Alternative Block

Reorder Block

Variant 2

if not commitdate < receiptdate:
    continue

Reorder Block

Variant 2.1

if not commitdate < receiptdate:
    continue

Reorder Block

Variant 2.2

if not shipdate < commitdate:
    continue

counter++
Dynamic Blocks Framework

1. Code Generation

```python
for l in lineitem:
    if not (shipdate < commitdate and commitdate < receiptdate):
        continue
    if not commitdate < receiptdate:
        continue
    if not shipdate < commitdate:
        continue
    counter++
```

⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒⇒

integrate Dynamic Blocks into Umbra IR

Alternative Block

Variant 1

```c
%3156 = cmpult i32 %3116, %2736
%3180 = getelementptr int8 %2551, i64 3670016
%3202 = load int32 %3180, i32 %localTid
%3224 = cmpult i32 %3202, %3116
%3238 = and bool %3152, %3224
condbr %3238 %cont %contScan
%3292 = cmpult i32 %3116, %2736
condbr %3292 %cont2 %contScan
%3354 = getelementptr int8 %2551, i64 3670016
%3376 = load int32 %3354, i32 %localTid
%3398 = cmpult i32 %3376, %3116
condbr %3398 %cont3 %contScan
```

Variant 2

```c
%3292 = cmpult i32 %3116, %2736
condbr %3292 %cont2 %contScan
%3354 = getelementptr int8 %2551, i64 3670016
%3376 = load int32 %3354, i32 %localTid
%3398 = cmpult i32 %3376, %3116
condbr %3398 %cont3 %contScan
```

Variant 2.1

```c
%3292 = cmpult i32 %3116, %2736
condbr %3292 %cont2 %contScan
```

Variant 2.2

```c
%3354 = getelementptr int8 %2551, i64 3670016
%3376 = load int32 %3354, i32 %localTid
%3398 = cmpult i32 %3376, %3116
condbr %3398 %cont3 %contScan
```
Dynamic Blocks Framework

2. Compilation

Alternative Block

1 Alternative Block
2 %3156 = cmpult i32 %3116, %2736
3 %3180 = getelementptr int8 %2551,
   i64 3670016
4 %3202 = load int32 %3180, %localTid
5 %3224 = cmpult i32 %3202, %3116
6 %3238 = and bool %3152, %3224
7 condbr %3238 %cont %contScan
8
9
10
11
12
13
14
15
16

Variant 1

Reorder Block

9 %3292 = cmpult i32 %3116, %2736
10 condbr %3292 %cont2 %contScan
11
12
13
14
15
16

Variant 2.1

11 %3354 = getelementptr int8 %2551,
   i64 3670016
12 %3376 = load int32 %3354, %localTid
13 %3398 = cmpult i32 %3376, %3116
14 condbr %3398 %cont3 %contScan
15
16

Variant 2.2

13
14
15
16
17
18

Flying Start

Compiler

Alternative Block

1 Alternative Block
2 cmp r14d, ebx
3 setb r13b
4 mov r15, [rsp+80]
5 mov r15d, [r15+r12*4+3670016]
6 cmp r15d, r14d
7 setb r15b
8 and r13b, r15b
9 cmp r13b, 1
10 jnz contScan
11
12
13
14
15
16
17
18

Variant 2

Reorder Block

11 cmp r14d, ebx
12 jae contScan
13
14
15
16
17
18

Variant 2.1

13 mov r12, [rsp+80]
14 mov r13, [rsp+32]
15 mov r12d, [r12+r13*4+3670016]
16 cmp r12d, r14d
17 jae contScan
18
19

Variant 2.2
Dynamic Blocks Framework

3. Rewriting

```
1 ... 

Alternative Block  Variant 1
2 cmp r14d, ebx
3 setb r13b
4 mov r15, [rsp+80]
5 mov r15d, [r15+r12*4+3670016]
6 cmp r15d, r14d
7 setb r15b
8 and r13b, r15b
9 cmp r13b, 1
10 jnz contScan

Reorder Block  Variant 2
11 cmp r14d, ebx
12 jae contScan
13 mov r12, [rsp+80]
14 mov r13, [rsp+32]
15 mov r12d, [r12+r13*4+3670016]
16 cmp r12d, r14d
17 jae contScan
18 ...
```

```
1 ... 

Alternative Block  Variant 2
2 mov r12, [rsp+80]
3 mov r13, [rsp+32]
4 mov r12d, [r12+r13*4+3670016]
5 cmp r12d, r14d
6 jae contScan

Reorder Block  Variant 2.2
7 cmp r14d, ebx
8 jae contScan
9 ...
```
Dynamic Blocks Framework

4. Execution

![Graph showing execution times for different variants and threads.](image)

- Variant A (explore)
- Variant B (explore)
- Variant C (exploit)

Thread 1, 2, 3, 4

Time [ms]

0 1 2 3 4 5 6

LLVM compilation

Tobias Schmidt, Philipp Fent, Thomas Neumann

ADMS’22
Dynamic Blocks Framework
Optimization: Dynamic Predicates

- Adapt n-ary conjunctions in joins $\land$ and selections $\sigma$
- Reorder the terms or evaluate them together

```plaintext
for l in lineitem:
    if not (shipdate < commitdate and commitdate < receiptdate):
        continue
    if not commitdate < receiptdate:
        continue
    if not shipdate < commitdate:
        continue
    counter++
```

---

Tobias Schmidt, Philipp Fent, Thomas Neumann ADMS’22
Dynamic Blocks Framework
Optimization: Dynamic Join Probes

- Reorder hash joins $\bowtie$, early probes $\bowtie$, and selections $\sigma$
- Limited to operators in the same pipeline
- Pullup predicates during execution
Dynamic Blocks Framework

Optimization: Dynamic Join Probes

- Reorder hash joins $\bowtie$, early probes $\bowtie$, and selections $\sigma$
- Limited to operators in the same pipeline
- Pullup predicates during execution

Tobias Schmidt, Philipp Fent, Thomas Neumann
ADMS’22
Dynamic Blocks Framework
Optimization: Dynamic Join Probes

- Reorder hash joins $\bowtie$, early probes $\bowtie$, and selections $\sigma$
- Limited to operators in the same pipeline
- Pullup predicates during execution

Tobias Schmidt, Philipp Fent, Thomas Neumann
ADMS’22
Evaluation

Setup

▶ Four OLAP benchmarks: TPC-H, TPC-DS, SSB, and JOB
▶ Dynamic optimizations are used by 163 queries

System:
▶ Intel Xeon Gold 6338 (Icelake, 2.0 GHz - 3.2 GHz)
▶ 32 cores
▶ 256 GB memory
▶ Ubuntu 21.10
Evaluation

- more than 2× speedup, almost no overhead, and up to 25% performance improvement on average
Conclusion

Dynamic Blocks combine the benefits of Adaptive Query Processing and Query Compilation.
Conclusion

*Dynamic Blocks combine the benefits of Adaptive Query Processing and Query Compilation.*

- postpone query optimization decision and perform low-level optimizations.
- generate dozens variations in a single compilation pass.
- find the best-performing implementation at run-time.
Evaluation

Tobias Schmidt, Philipp Fent, Thomas Neumann

ADMS’22
Evaluation

<table>
<thead>
<tr>
<th>Scale Factor</th>
<th>TPC-H minimum</th>
<th>TPC-H geometric mean</th>
<th>TPC-H maximum</th>
<th>TPC-DS minimum</th>
<th>TPC-DS geometric mean</th>
<th>TPC-DS maximum</th>
<th>SSB maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>-25 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+25 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+50 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+75 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+100 %</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tobias Schmidt, Philipp Fent, Thomas Neumann ADMS'22
Evaluation

Flying Start compiler

rewriting

computed gotos

Overhead

LLVM

Tobias Schmidt, Philipp Fent, Thomas Neumann
ADMS’22