Understanding and Improving Database-backed Applications

Cong Yan
University of Washington
Real-World Application Performance
# Real-World Application Performance

<table>
<thead>
<tr>
<th>Application</th>
<th># github stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discourse (forum)</td>
<td>22k</td>
</tr>
<tr>
<td>Lobsters (forum)</td>
<td>2.4k</td>
</tr>
<tr>
<td>Gitlab (collaboration)</td>
<td>49k</td>
</tr>
<tr>
<td>Redmine (collaboration)</td>
<td>3.6k</td>
</tr>
<tr>
<td>Spree (E-commerce)</td>
<td>17k</td>
</tr>
<tr>
<td>ROR Ecommerce</td>
<td>1.7k</td>
</tr>
<tr>
<td>Fulcrum (task mgmt)</td>
<td>697</td>
</tr>
<tr>
<td>Tracks (task mgmt)</td>
<td>3.5k</td>
</tr>
<tr>
<td>Diaspora (social network)</td>
<td>18k</td>
</tr>
<tr>
<td>Onebody (social network)</td>
<td>1.2k</td>
</tr>
<tr>
<td>Openstreetmap (map)</td>
<td>8k</td>
</tr>
<tr>
<td>Fallingfruit (map)</td>
<td>1.1k</td>
</tr>
</tbody>
</table>

![Webpage load time distribution (CDF)](image)

(second)
Real-World Application Performance

<table>
<thead>
<tr>
<th>Application</th>
<th># github stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discourse (forum)</td>
<td>22k</td>
</tr>
<tr>
<td>Lobsters (forum)</td>
<td>2.4k</td>
</tr>
<tr>
<td>Gitlab (collaboration)</td>
<td>49k</td>
</tr>
<tr>
<td>Redmine (collaboration)</td>
<td>3.6k</td>
</tr>
<tr>
<td>Spree (E-commerce)</td>
<td>17k</td>
</tr>
<tr>
<td>ROR Ecommerce</td>
<td>1.7k</td>
</tr>
<tr>
<td>Fulcrum (task mgmt)</td>
<td>697</td>
</tr>
<tr>
<td>Tracks (task mgmt)</td>
<td>3.5k</td>
</tr>
<tr>
<td>Diaspora (social network)</td>
<td>18k</td>
</tr>
<tr>
<td>Onebody (social network)</td>
<td>1.2k</td>
</tr>
<tr>
<td>Openstreetmap (map)</td>
<td>8k</td>
</tr>
<tr>
<td>Fallingfruit (map)</td>
<td>1.1k</td>
</tr>
</tbody>
</table>

half of the users expect a page to load in less than 2 seconds
Real-World Application Performance

<table>
<thead>
<tr>
<th>Application</th>
<th># github stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discourse (forum)</td>
<td>22k</td>
</tr>
<tr>
<td>Lobsters (forum)</td>
<td>2.4k</td>
</tr>
<tr>
<td>Gitlab (collaboration)</td>
<td>49k</td>
</tr>
<tr>
<td>Redmine (collaboration)</td>
<td>3.6k</td>
</tr>
<tr>
<td>Spree (E-commerce)</td>
<td>17k</td>
</tr>
<tr>
<td>ROR Ecommerce</td>
<td>1.7k</td>
</tr>
<tr>
<td>Fulcrum (task mgmt)</td>
<td>697</td>
</tr>
<tr>
<td>Tracks (task mgmt)</td>
<td>3.5k</td>
</tr>
<tr>
<td>Diaspora (social network)</td>
<td>18k</td>
</tr>
<tr>
<td>Onebody (social network)</td>
<td>1.2k</td>
</tr>
<tr>
<td>Openstreetmap (map)</td>
<td>8k</td>
</tr>
<tr>
<td>Fallingfruit (map)</td>
<td>1.1k</td>
</tr>
</tbody>
</table>

20% of pages from real-world apps take longer, with ~1 GB of data.

Half of the users expect a page to load in less than 2 seconds.
Decades of Research on Optimizing Each Layer

- Web cache
- WebAssembly
- JavaScript JIT

- Dead code elimination
- Vectorization
- Object caching

- Query optimization
- Physical design
- Concurrency control

Diagram:
- Request to webpage
- Query to data
- Database
Key Insight: Leveraging Application Semantics
Leveraging Application Semantics To Optimize Each Layer

★ Leverage how query results are used in the application
Leveraging Application Semantics To Optimize Each Layer

★ Leverage how query results are used in the application
Leveraging Application Semantics To Optimize Each Layer

Leverage how query results are used in the application

request
webpage

prod=exec(Q1)

query
data
Leveraging Application Semantics To Optimize Each Layer

- Leverage how queries are connected in the application
- Leverage how query results are used in the application

request

webpage

query
data
Leveraging Application Semantics To Optimize Each Layer

★ Leverage how queries are connected in the application
★ Leverage how query results are used in the application
Leveraging Application Semantics To Optimize Each Layer

- Leverage how queries are connected in the application
- Leverage how query results are used in the application

Diagram:
- Q1 → Q2
- Q3 → Q4
- Request ➔ Webpage ➔ Query ➔ Data ➔ Database
Leveraging Application Semantics To Optimize Each Layer

★ Leverage the data retrieval behind each webpage data
★ Leverage how queries are connected in the application
★ Leverage how query results are used in the application

Diagram:
- Request from webpage to database
- Query from database to data
- Interconnections showing data flow
Leveraging Application Semantics To Optimize Each Layer

- Leverage the data retrieval behind each webpage data
- Leverage how queries are connected in the application
- Leverage how query results are used in the application

```
<code>prod=exec(Q1)
```

Diagram:
- Computer
- Webpage
- Database
- Query
- Data
- Request

SELECT * FROM ...
Leveraging Application Semantics To Optimize Each Layer

Panorama: view-driven optimization
- Leverage the data retrieval behind each webpage data

Quro: reorder queries
- Leverage how queries are connected in the application

Chestnut: customize data layout
- Leverage how query results are used in the application

Diagram:
- Request ➔ Webpage ➔ Database ➔ Query ➔ Data
Outline

- Leveraging application semantics to optimize each layer
  [CIKM’17, FSE’18, ICSE’18, ICSE’19, CIDR’20]  [VLDB’16]  [VLDB’19]

Panorama: view-driven optimization

Quro: reorder queries

Chestnut: customize data layout
Outline

• Leveraging application semantics to optimize each layer
  [CIKM’17, FSE’18, ICSE’18, ICSE’19, CIDR’20]  [VLDB’16]  [VLDB’19]

Panorama: view-driven optimization

Quro: reorder queries

Chestnut: customize data layout

• Other projects

• Ongoing and future work
Outline

- Leveraging application semantics to optimize each layer
  - Panorama: view-driven optimization
  - Quro: reorder queries
  - Chestnut: customize data layout

- Other projects
- Ongoing and future work
Outline

- Leveraging application semantics to optimize each layer

Panorama: view-driven optimization

Quro: reorder queries

Chestnut: customize data layout

- Other projects
- Ongoing and future work
Query Reorder under 2PL

Execution time

--T1-- original --T2--
Query Reorder under 2PL

Execution time

--T1-- original --T2--

select(P, iphone)
Query Reorder under 2PL

Execution time

original

--T1--

select(P, iphone)
update(P, iphone)
select(C, Alice)
update(C, Alice)

--T2--

lock waiting time

select(P, iphone)
update(P, iphone)
select(C, Bob)
update(C, Bob)
Query Reorder under 2PL

Original order:
- T1: select(P, iphone), update(P, iphone)
- T2: select(C, Alice), update(C, Alice)

Reordered order:
- T1: select(P, iphone), update(P, iphone), select(C, Bob), update(C, Bob)
- T2: select(C, Alice)
Query Reorder under 2PL

Execution time

---T1--- original ---T2---

lock waiting time

---T1--- reorder ---T2---

select(C, Alice)

select(C, Bob)

select(P, iphone)

update(P, iphone)

select(C, Alice)

update(C, Alice)

select(P, iphone)

update(P, iphone)

select(C, Bob)

update(C, Bob)
Query Reorder under 2PL

Execution time

---T1--- original

select(P, iphone)
update(P, iphone)
select(C, Alice)
update(C, Alice)

lock waiting time

---T2---

select(P, iphone)
update(P, iphone)
select(C, Bob)
update(C, Bob)

---T1--- reorder

select(C, Alice)
update(C, Alice)
select(P, iphone)
update(P, iphone)

---T2---

select(C, Bob)
update(C, Bob)
select(P, iphone)
update(P, iphone)
Query Reorder Shortens Lock Wait Time

Execution time

original

lock waiting time

reorder
Quro: Compiler for Query Reorder

Input: C++ code with embedded SQL queries
Quro: Compiler for Query Reorder

Input: C++ code with embedded SQL queries

Analyze the control and data flow of the application code
**Quro: Compiler for Query Reorder**

- **Input:** application workload
  - Profile the application to get contentious queries

- **Input:** C++ code with embedded SQL queries
  - Analyze the control and data flow of the application code
Quro: Compiler for Query Reorder

Input: application workload
Profile the application to get contentious queries
Reorder queries while preserving application semantics
Output: C++ code with reordered, embedded queries

Input: C++ code with embedded SQL queries
Analyze the control and data flow of the application code
Quro: Compiler for Query Reorder

Profile the application to get contentious queries

Analyze the control and data flow of the application code

Reorder queries while preserving application semantics

Quantify contention level

- Calculate standard deviation of query running time
- Larger the deviation, more likely to have data conflict
Quro: Compiler for Query Reorder

Profile the application to get contentious queries

Quantify contention level

- Calculate standard deviation of query running time
- Larger the deviation, more likely to have data conflict

Analyze the control and data flow of the application code

Data dependency analysis

- Data dependency among program variables

Reorder queries while preserving application semantics

1. \(v1 = \text{select(table1)};\)
2. \(v2 = \text{select(table2, } v1);\)
3. \(\text{update(table1, } v1);\)
Quro: Compiler for Query Reorder

Profile the application to get contentious queries

Quantify contention level
- Calculate standard deviation of query running time
- Larger the deviation, more likely to have data conflict

Analyze the control and data flow of the application code

Data dependency analysis
- Data dependency among program variables
- Database constraints

1. \( v_1 = \text{select}(\text{table1}) \)
2. \( v_2 = \text{select}(\text{table2}, v_1) \)
3. \( \text{update}(\text{table1}, v_1) \)

Reorder queries while preserving application semantics
Quro: Compiler for Query Reorder

Profile the application to get contentious queries

- Quantify contention level
  - Calculate standard deviation of query running time
  - Larger the deviation, more likely to have data conflict

Analyze the control and data flow of the application code

- Data dependency analysis
  - Data dependency among program variables
  - Database constraints

Reorder queries while preserving application semantics

- Formalize reorder problem
  - Formalize into ILP problem
  - Constraint: data dependency constraints
  - Goal: make contentious query appear later in a transaction

1. \( v_1 = \text{select}(\text{table1}) \)
2. \( v_2 = \text{select}(\text{table2}, v_1) \)
3. \( \text{update}(\text{table1}, v_1) \)
Evaluation: Quro Vs. Original

Workload: TPC-C payment transaction

Latency: decrease up to 70%
Quro: Compiler for Query Reorder

Profile the application to get contentious queries

Quantify contention level
- Calculate standard deviation of query running time
- Larger the deviation, more likely to have data conflict

Analyze the control and data flow of the application code

Data dependency analysis
- Data dependency among program variables
- Database constraints

Reorder queries while preserving application semantics

Formalize reorder problem
- Formalize into ILP problem
- Constraint: data dependency constraints
- Goal: make contentious query appear later in a transaction

\[
1. v_1 = \text{select}(\text{table1}), \\
2. v_2 = \text{select}(\text{table2}, v_1), \\
3. \text{update}(\text{table1}, v_1)
\]
Evaluation: Quro Vs. Original

Workload: TPC-C payment transaction

Throughput: txn/sec

Latency: decrease up to 70%
Evaluation: Quro Vs. Original

Workload: TPC-C payment transaction

Latency: decrease up to 70%
Evaluation: Quro Vs. Other CC Schemes

Workload: TPC-C 50% payment and 50% new order
Evaluation: Quro Vs. Other CC Schemes

Workload: TPC-C 50% payment and 50% new order

- OCC
- MVCC
- 2PL
- 2PL-quro-reordered
• The order of queries has large impact on transaction performance.

• We build Quro, a compiler that leverages information about query contention, and automatically reorders the queries.

• Quro-generated code improves throughput up to 3.5x, and outperforms other concurrency control schemes under high contention.

• Reordering is implemented in critical transactions of Taobao in Alibaba
Outline

• Leveraging application semantics to optimize each layer

Panorama: view-driven optimization

Quro: reorder queries

Chestnut: customize data layout

• Other projects

• Ongoing and future work
Data Representation Mismatch
Data Representation Mismatch

- Nested object vs. tabular data
Data Representation Mismatch

- Nested object vs. tabular data
Data Representation Mismatch

- Nested object vs. tabular data
Data Representation Mismatch

- Nested object vs. tabular data
Data Representation Mismatch

- Nested object vs. tabular data
Data Representation Mismatch

- Nested object vs. tabular data
Data Representation Mismatch

- Nested object vs. tabular data
Data Representation Mismatch

- Nested object vs. tabular data
Data Representation Mismatch

- Nested object vs. tabular data

Website Project
- Noelle

Channels

#website-feedback

Noelle Keyy
Do you have any feedback on the new website?

Steve Young
Could you modify the price to $39?

Website
msg, "Hey,"
file, a.pdf
Keyy
Young

Dev
file, b.pdf
Young

<table>
<thead>
<tr>
<th></th>
<th>website</th>
<th>dev</th>
<th>brunch</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 10K| msg, "Hey,"
    | file, a.pdf | file, b.pdf |
| 5K | Keyy | Young | Eve |

1.7 sec
Data Representation Mismatch

- Nested object vs. tabular data

55 sec

1.7 sec
Different App Queries Use Different Layout

- Nested object vs. tabular data
Different App Queries Use Different Layout

- Nested object vs. tabular data
How to Run Queries Faster?
How to Run Queries Faster?
How to Run Queries Faster?

(in-mem cache)
How to Run Queries Faster?

(in-mem cache)

(in-mem cache)

(in-mem cache)
How to Run Queries Faster?

(in-mem cache)

(in-mem cache)

(in-mem cache)
How to Run Queries Faster?

Chestnut

(in-mem cache)

(in-mem cache)

(in-mem cache)
How to Run Queries Faster?

(chestnut-generated in-mem cache)
How to Run Queries Faster?

read query → query result

(chestnut-generated in-mem cache)
How to Run Queries Faster?

write query

(chestnut-generated in-mem cache)
How to Run Queries Faster?

write query

(chestnut-generated in-mem cache)

write query

Database
How to Run Queries Faster?

Key challenges:
1. How to decide the data layout?
2. How to answer the queries?

write query

(chestnut-generated in-mem cache)

write query
Chestnut Step 1: Layout Enumeration
How to Run Queries Faster?

Key challenges:
1. How to decide the data layout?
2. How to answer the queries?

write query

(chestnut-generated in-mem cache)

write query
Chestnut Step 1: Layout Enumeration
Chestnut Step 1: Layout Enumeration

website  dev  brunch


Keyy  Young  Eve

layout 1
Chestnut Step 1: Layout Enumeration

layout 1

layout 2
Chestnut Step 1: Layout Enumeration

layout 1

layout 2

layout 3

...
Chestnut Step 1: Layout Enumeration

index on channel(name)
- website
- dev
- brunch

index on activity(id), only on msg activity
- msg, "Hey"
- file, a.pdf
- file, b.pdf
- Keyy
- Young
- Eve

layout 1

layout 2

layout 3

...
Chestnut Step 1: Layout Enumeration

index on channel(name)
- website
- dev
- brunch

index on activity(id), only on msg activity
- msg
- file
- “Hey”
- a.pdf
- b.pdf

index on activity.user(name)
- msg
- “Hey”
- file
- a.pdf
- Young
- file
- b.pdf
- Young

layout 1
- Keyy
- Young
- Eve

layout 2
- Keyy
- Young

layout 3
- dev
- file
- b.pdf
Chestnut Step 1: Layout Enumeration

Index on channel(name)
- website
- dev
- brunch

Index on activity(id), only on msg activity
- msg, "Hey"
- file, a.pdf
- file, b.pdf

Index on activity.user(name)
- Keyy
- Young

Index on channel(name), only on channel with msg activity
- website
- dev
- brunch

Layout 1

Layout 2

Layout 3

...
Chestnut Step 2: Plan Enumeration

Data layout:

Expected query result:
Chestnut Step 2: Plan Enumeration

Data layout:


Keyy  Young  Young

website  dev  brunch

Expected query result:

website  msg."Hey,"  file, a.pdf

Keyy  Young

relational query plans cannot be applied
Chestnut Step 2: Plan Enumeration

Data layout:

```
  Keyy    Young    Young

website  dev  brunch
```

--- array_activity

----- array_channel

Expected query result:

```
for c in array_channel:
    result.append(c)
```
Chestnut Step 2: Plan Enumeration

Data layout:

- msg:"Hey"
- file, a.pdf
- file, b.pdf

- Keyy
- Young

--- array_activity

- website
- dev
- brunch

----- array_channel

Expected query result:

- msg:"Hey"
- file, a.pdf

- Keyy
- Young

symbolic verification

```python
for c in array_channel:
    result.append(c)
```
Chestnut Step 2: Plan Enumeration

Data layout:

Keyy Young Young

website dev brunch

------ array_channel

Expected query result:

website msg."Hey," file, a.pdf
Keyy Young

symbolic verification

for c in array_channel:
    for a in array_activity:
        if a.channel_id==c.id:
            c.activities.add(a)
    result.append(c)
Chestnut Step 2: Plan Enumeration

Data layout:

```
msg"Hey," file, a.pdf file, b.pdf Young
Keyy
website dev brunch
```

--- array_activity

```
for c in array_channel:
    result.append(c)
```

----- array_channel

Expected query result:

```
msg"Hey," file, a.pdf
Keyy Young
```

```
for c in array_channel:
    for a in array_activity:
        if a.channel_id==c.id:
            c.activities.add(a)
    result.append(c)
```

symbolic verification
Chestnut Step 2: Plan Enumeration

**Data layout:**

```
| Keyy       | Young      | Young       |
```

```
website  dev  brunch
```

--- array_activity

```
for c in array_channel:
    result.append(c)
```

----- array_channel

```
for c in array_channel:
    for a in array_activity:
        if a.channel_id==c.id:
            c.activities.add(a)
    result.append(c)
```

**Expected query result:**

```
website  msg."Hey,"  file, a.pdf
| Keyy | Young |
```

symbolic verification

```
for c in array_channel:
    if c.name == 'website':
        for a in array_activity:
            if a.channel_id==c.id:
                c.activities.add(a)
    result.append(c)
```
Chestnut Step 2: Plan Enumeration

Data layout:

```
msg:"Hey," file, a.pdf file, b.pdf
Keyy Young Young

website dev brunch

----- array_channel

--- array_activity

Expected query result:

```
website
msg:"Hey," file, a.pdf
Keyy Young

= symbolic verification


code:

```python
for c in array_channel:
    result.append(c)

for c in array_channel:
    for a in array_activity:
        if a.channel_id==c.id:
            c.activities.add(a)
    result.append(c)

for c in array_channel:
    if c.name == "website":
        for a in array_activity:
            if a.channel_id==c.id:
                c.activities.add(a)
        result.append(c)
```
Chestnut Step 3: Update Query
Chestnut Step 3: Update Query

write query:

```java
Message.where(content="Hey").update(content=>"Hi")
```
Chestnut Step 3: Update Query

write query:
Message.where(content="Hey").update(content=>"Hi")

read query to identify the objects to update:
Chestnut Step 3: Update Query

**write query:**
```java
Message.where(content="Hey").update(content=>'Hi')
```

**read query to identify the objects to update:**
```java
Channel.where(exists(msg, where(content="Hey")))
```
Chestnut Step 3: Update Query

**write query:**

```java
Message.where(content="Hey").update(content=>"Hi")
```

**read query to identify the objects to update:**

```java
Channel.where(exists(msg, where(content="Hey")))
```
Chestnut Step 3: Update Query

**write query:**
```
Message.where(content=="Hey").update(content=>'Hi')
```

**read query to identify the objects to update:**
```
Channel.where(exists(msg, where(content=="Hey"))
.include(msg, where(content=="Hey"))
```

Diagram:
- Website: "Hey"
- File: a.pdf
- Key: Young
- Dev: File: b.pdf
- Young
Chestnut Step 4: Handling Multiple Queries
Chestnut Step 4: Handling Multiple Queries

Q1
layout1
layout2
layout3
plan1
plan2
plan3

Q2
layout1
layout2
layout3
plan1
plan2
plan3

(infinite memory)
Chestnut Step 4: Handling Multiple Queries
Chestnut Step 4: Handling Multiple Queries

Diagram showing the relationship between Q1, Q2, and Q3 layouts and plans, with a note on limited memory.
Chestnut Step 4: Handling Multiple Queries

Q1
- layout1
- layout2
- layout3

Q2
- layout1
- layout2
- layout3

Q1_layout4
Q2_layout7
Q3_layout5

(limited memory)
Chestnut Workflow

Constraint:
- Used data structures within memory bound

Optimization goal:
- Minimize $\sum$ estimated query time
Chestnut Workflow

Output:
- Which data structure to use
- Which query plan to use
Chestnut Workflow

Constraint:
- Used data structures within memory bound

Optimization goal:
- Minimize $\sum estimated\ query\ time$
Chestnut Step 3: Update Query

**write query:**
```java
Message.where(content="Hey").update(content=>'Hi')
```

**read query to identify the objects to update:**
```java
Channel.where(exists(msg, where(content="Hey"))).include(msg, where(content="Hey"))
```
Chestnut Step 3: Update Query

**write query:**
```
Message.where(content="Hey").update(content=>"Hi")
```

**read query to identify the objects to update:**
```
Channel.where(exists(msg, where(content="Hey"))).include(msg, where(content="Hey"))
```
Chestnut Workflow

Constraint:
- Used data structures within memory bound

Optimization goal:
- Minimize $\sum \text{estimated query time}$
Chesnut Workflow

Output:
- Which data structure to use
- Which query plan to use
Evaluation

- 4 open-source popular web applications built with Rails
  - kandan: chatting app (2.8k ★)
  - redmine: project management app (3.6k ★)
  - lobsters: forum app (2.4k ★)
  - huginn: web-scraping app (22.6k ★)
Evaluation

• 4 open-source popular web applications built with Rails
  • kandan: chatting app (2.8k ★)
  • redmine: project management app (3.6k ★)
  • lobsters: forum app (2.4k ★)
  • huginn: web-scraping app (22.6k ★)

• Compare against app impl with 3 relational databases:
  • Original setting with MySQL (in-memory)
  • PostgreSQL (in-memory)
  • Commercial in-memory column-store
Evaluation: Web Applications

- 4 open-source popular web applications built with Rails

- Top-10 pages
- ~26 queries per app
- Include read & write

(avg query time, with the same memory)
Evaluation: Web Applications

- 4 open-source popular web applications built with Rails

![Graph showing query times for different web applications and database engines.]

- top-10 pages
- ~26 queries per app
- include read & write

(avg query time, with the same memory)
Evaluation: Web Applications

- 4 open-source popular web applications built with Rails

Chestnut running time:
- kandan: 1 min
- redmine: 10 min
- lobsters: 54 min
- huginn: 3 min

(avg query time, with the same memory)
• Object-oriented database applications processes data in nested format, different from the tabular data layout used in DB.

• We propose Chestnut, a data layout designer that customizes data layout and query execution for each application.

• It uses enumeration (layout + query plan) and ILP solver to search for the best data layout given an application.

• Evaluation shows chestnut works for real-world applications, speeding up app queries up to 9.8x.
Outline

- Leveraging application semantics to optimize each layer

Panorama: view-driven optimization
Quro: reorder queries
Chestnut: customize data layout

- Other projects
- Ongoing and future work
Anti-patterns In Web Apps

- A comprehensive study on 12 popular open-source Rails Applications

[ICSE’18]
Anti-patterns In Web Apps

- A comprehensive study on 12 popular open-source Rails Applications

[ICSE’18]

140 performance issues from bug tracking system
Anti-patterns In Web Apps

- A comprehensive study on 12 popular open-source Rails Applications

[ICSE’18]

140 performance issues from bug tracking system

64 performance issues from profiling
Anti-patterns In Web Apps

- A comprehensive study on 12 popular open-source Rails Applications

140 performance issues from bug tracking system + 64 performance issues from profiling = 9 anti-patterns

[ICSE’18]
Anti-patterns In Web Apps

• A comprehensive study on 12 popular open-source Rails Applications

[ICSE’18]

140 performance issues from bug tracking system + 64 performance issues from profiling = 9 anti-patterns

.any? ✗ .exists? ✓
Anti-patterns In Web Apps

- A comprehensive study on 12 popular open-source Rails Applications

[ICSE ’18]

140 performance issues from bug tracking system

64 performance issues from profiling

9 anti-patterns

.any? ✗ .exists? ✓

select * from... ✗ select f1 from .. ✓
Inefficiencies Fixed by Manual Patch

• We manually fix the 64 issues found across 40 pages from 12 apps
• Avg speedup of each anti-pattern fix

[ICSE’18]
Automatic Fix of Anti-patterns

Powerstation

[CIKM’17, FSE’18]

It fixes 6 anti-patterns

Static analysis on the source code

Pattern-match to find anti-patterns
Automatic Fix of Anti-patterns

Powerstation

[CIKM’17, FSE’18]

It fixes 6 anti-patterns

Static analysis on the source code

Pattern-match to find anti-patterns

Semantic-preserving!
Automatic Fix of Anti-patterns

Powerstation
[CIKM’17, FSE’18]

It fixes 6 anti-patterns

Static analysis on the source code

Pattern-match to find anti-patterns

Semantic-preserving!
Automatic Fix of Anti-patterns

It fixes 6 anti-patterns

Static analysis on the source code

Pattern-match to find anti-patterns

Non-semantic-preserving!

Semantic-preserving!

Webpage design-performance tradeoff
Automatic Fix of Anti-patterns

Powerstation

[CIKM’17, FSE’18]

It fixes 6 anti-patterns

Static analysis on the source code

Pattern-match to find anti-patterns

Semantic-preserving!

Panorama

Non-semantic-preserving!

Webpage design-performance tradeoff
Anne has 10001 visitors

- The 41st ICSE is held in Montreal! ★ (42 comments)

- My paper is accepted. ★ (56 comments)

- My poster won the student competition.

- I have attended the student volunteer party!

- The banquet is super nice and awesome!

- Look forward to next year’s conference
Anne has 10001 visitors

- The 41st ICSE is held in Montreal! ★ (42 comments)
- My paper is accepted. ★ (56 comments)
- My poster won the student competition.
- I have attended the student volunteer party!
- The banquet is super nice and awesome!
- Look forward to next year’s conference
Anne has 10001 visitors

- The 41st ICSE is held in Montreal! ★ (42 comments)
- My paper is accepted. ★ (56 comments)
- My poster won the student competition.
- I have attended the student volunteer party!
- The banquet is super nice and awesome!
- Look forward to next year’s conference
Anne has 10001 visitors

- The 41st ICSE is held in Montreal! ★ (42 comments)
- My paper is accepted. ★ (56 comments)
- My poster won the student competition.
- I have attended the student volunteer party!
- The banquet is super nice and awesome!
- Look forward to next year’s conference
Anne has 10001 visitors

The 41st ICSE is held in Montreal! ★ (42 comments)

My paper is accepted. ★ (56 comments)

My poster won the student competition.

I have attended the student volunteer party!

The banquet is super nice and awesome!

Look forward to next year’s conference

The 41st ICSE is held in Montreal! ★ (42 comments)

My paper is accepted. ★ (56 comments)

My poster has won the student competition

I have attended the student volunteer party!

The banquet is super nice and awesome!
Anne has 10001 visitors

The 41st ICSE is held in Montreal!
⭐ (42 comments)

My paper is accepted.
⭐ (56 comments)

My poster won the student competition.

I have attended the student volunteer party!

The banquet is super nice and awesome!

Look forward to next year’s conference
Anne has 10001 visitors

The 41st ICSE is held in Montreal!

My paper is accepted.

My poster won the student competition.

I have attended the student volunteer party!

The banquet is super nice and awesome!

Look forward to next year’s conference

(42 comments)

(56 comments)

Anne has 10001 visitors

The 41st ICSE is held in Montreal!

My paper is accepted.

My poster has won the student competition

I have attended the student volunteer party!

The banquet is super nice and awesome!
Anne has 10001 visitors

The 41st ICSE is held in Montreal!

My paper is accepted.

My poster won the student competition.

I have attended the student volunteer party!

The banquet is super nice and awesome!

Look forward to next year’s conference

The 41st ICSE is held in Montreal!

My paper is accepted.

My poster has won the student competition

I have attended the student volunteer party!

The banquet is super nice and awesome!
Making Design-Performance Tradeoff is Not Trivial

• Changing code across layers
Making Design-Performance Tradeoff is Not Trivial

- Changing code across layers
Making Design-Performance Tradeoff is Not Trivial

• Changing code across layers
Making Design-Performance Tradeoff is Not Trivial

- Changing code across layers
Making Design-Performance Tradeoffs is Not Trivial

- Changing code across layers
Making Design-Performance Tradeoff is Not Trivial

- Changing code across layers
Making Design-Performance Tradeoff is Not Trivial

- Changing code across layers
- Understanding the tradeoff
Making Design-Performance Tradeoff is Not Trivial

- Changing code across layers
- Understanding the tradeoff
Making Design-Performance Tradeoff is Not Trivial

- Changing code across layers
- Understanding the tradeoff

Design-perf tradeoff:
143 days

Overall avg:
29 days

Issue resolving time
Panorama User Interface: Heatmap
How does Panorama generate heatmap?
How does Panorama generate heatmap?

```ruby
blogs_controller.rb
...
@bcound=blogs.joins(comments)
  .exclude_self(user).count
  ...

index.html.erb
...
user.blogs.each |b|
  if @bcound[b] > 100:
    ...

blog.rb
class Blog
  def exclude_self(user):
    where('comments.user_id != user.id')

select count(*) from blogs join ... where ... group by ...
```
How does Panorama generate heatmap?

```
blogs_controller.rb
...
@bcount = blogs.joins(comments)
 .exclude_self(user).count
...
```

```
index.html.erb
...
user.blogs.each |b| if @bcount[b] 100:
...
```

```
blog.rb
class Blog
def exclude_self(user):
  where('comments.user_id = user.id')

```
How does Panorama generate heatmap?

```
index.html.erb
...,
user.blogs.each |b| 
if @bcount[b] > 100: 
...,
```

```
blogs_controller.rb
...
@bcount = blogs.joins(comments)
 .exclude_self(user).count
...
```

```
blog.rb
class Blog
  def exclude_self(user):
    where('comments.user_id' == user.id)
  
select count(*) from blogs join ... where ... group by ...
```
How does Panorama generate heatmap?

N_{blogs} \times N_{comments}

blogs_controller.rb
...
@bcount = Blog.joins(comments).exclude_self(user).count
...

index.html.erb
...
user.blogs.each |b|
  if @bcount[b] > 100:
    ...

blog.rb
class Blog
  def exclude_self(user):
    where('comments.user_id = user.id')

select count(*) from blogs join ... where ... group by ...
Panorama: Suggests Design Change and Refactors Code Automatically

Anne has 10001 visitors

- The 41st ICSE is held in Montreal! ★ (42 comments)

- My paper is accepted. ★ (56 comments)

- My poster won the student competition.

- I have attended the student volunteer party!

- The banquet is super nice and awesome!

- Look forward to next year’s conference
Panorama: Suggests Design Change and Refactors Code Automatically

Anne has 10001 visitors

The 41st ICSE is held in Montreal! ★ (42 comments)

My paper is accepted. ★ (56 comments)

My poster won the student competition.

I have attended the student volunteer party!

The banquet is super nice and awesome!

Look forward to next year’s conference

removal async load approximation

pagination

high cost low cost
Panorama: Suggests Design Change and Refactors Code Automatically

Anne has 10001 visitors

The 41st ICSE is held in Montreal!

My paper is accepted.

My poster won the student competition.

I have attended the student volunteer party!

The banquet is super nice and awesome!

Look forward to next year’s conference

The 41st ICSE is held in Montreal!

My paper is accepted.

My poster has won the student competition

I have attended the student volunteer party!

The banquet is super nice and awesome!

removal
async load approximation

pagination

high cost

low cost

<< 1 2 3 ... >>
Panorama as An IDE Plugin

The Most Intelligent Ruby and Rails IDE

Rocky Emsor
Belkis Jaskolski
Mrs. Darrin Kazey
Kenny Hettinger

There are 114 blogs

Design Choices

pagination

Remove approximation
Panorama Evaluation

end-to-end speedup

Original page vs. new design by Panorama

End-to-end webpage time distribution (ms)
(14 webpages across 4 apps)
Panorama Evaluation

end-to-end speedup

(upto 17x)

original page  new design by Panorama

End-to-end webpage time distribution (ms)
(14 webpages across 4 apps)
Panorama Evaluation

end-to-end speedup

- Original page
- New design by Panorama

(up to 17x)

End-to-end webpage time distribution (ms)
(14 webpages across 4 apps)

user’s satisfaction

User survey result on page preference
Panorama Evaluation

**end-to-end speedup**

![Box plot showing end-to-end speedup](image)

- Original page
- New design by Panorama

End-to-end webpage time distribution (ms)
(14 webpages across 4 apps)

**user’s satisfaction**

![Bar chart showing user’s satisfaction](image)

- Prefer original page
- Prefer new page

User survey result on page preference
Panorama Evaluation

end-to-end speedup

(up to 17x)

original page  new design by Panorama

End-to-end webpage time distribution (ms)
(14 webpages across 4 apps)

user’s satisfaction

Avg: 22% prefer original, 20% prefer new

Approximate  Asynch  Paginate  Removal

user survey result on page preference

Prefer original page  Prefer new page
Press Coverage

- Uchicago news

Award-Winning ICSE Paper Offers Web Developers a “Panorama” View of Slowdowns and Fixes

June 24, 2019
Press Coverage

- Uchicago news
- Morning Paper

How not to structure your database-backed web applications: a study of performance bugs in the wild

View-centric performance optimization for database-backed web applications
Press Coverage

- Uchicago news
- Morning Paper
- Ruby Weekly

How Not to Structure Your Database-Backed Webapps — A breakdown of a paper that studies 12 of the most popular Rails apps looking for and fixing ORM issues.

ADRIAN COLYER

View-centric performance optimization for database-backed web applications
Press Coverage

• Uchicago news
• Morning Paper
• Ruby Weekly
• Hackernews
Press Coverage

- Uchicago news
- Morning Paper
- Ruby Weekly
- Hackernews
- Consultancy
Change webpage design to speed up a page is a common practice, yet making design-performance tradeoff is not trivial.

- We build Panorama, a tool that suggests non-semantic-preserving changes.
- It provides an interface for developers to understand the webpage performance and tradeoff by only interact with the webpage.
- Panorama speeds up a page up to 17x, proving that the tradeoff is worthwhile.
Panorama Evaluation

**End-to-end speedup**

- (up to 17x)

- Original page
- New design by Panorama

**User’s satisfaction**

- Avg: 22% prefer original, 20% prefer new

**End-to-end webpage time distribution (ms)**
- 14 webpages across 4 apps

**User survey result on page preference**
How does Panorama generate heatmap?

```
blogs_controller.rb
...
@bcnt = blogs.joins(comments)
  .exclude_self(user).count
  ... 
```

```
index.html.erb
...
user.blogs.each |blog|
  if @bcnt[b] > 100:
    ...
```

```
blog.rb
class Blog
  def exclude_self(user):
    where('comments.user_id = user.id')
```

```
select count(*) from blogs join ... where ... group by ... 
```
How does Panorama generate heatmap?

index.html.erb

```ruby
N_blogs * N_comments
```

blogs_controller.rb

```ruby
@bcount = blogs.joins(comments).exclude_self(user).count
```

blog.rb

```ruby
class Blog
  def exclude_self(user):
    where('comments.user_id = user.id')
end
```

select count(*) from blogs join ... where ... group by ...
How does Panorama generate heatmap?

index.html.erb

```ruby
user.blogs.each |b| if @bcount[b] > 100:
  ...
```

blogs_controller.rb

```ruby
@bcount = blogs.joins(comments)
  .exclude_self(user).count
  ...
```

blog.rb

```ruby
class Blog
  def exclude_self(user):
    where('comments.user_id' => user.id)
  end
end
```

select count(*) from blogs join ... where ... group by ...
Panorama Evaluation

end-to-end speedup

original page  new design by Panorama

End-to-end webpage time distribution (ms)
(14 webpages across 4 apps)
Panorama Evaluation

end-to-end speedup

(up to 17x)

original page  new design by Panorama

End-to-end webpage time distribution (ms)
(14 webpages across 4 apps)

user's satisfaction

Avg: 22% prefer original, 20% prefer new

Approximate  Asynch  Paginate  Removal

Prefer original page  Prefer new page

user survey result on page preference
Outline

• Leveraging application semantics to optimize each layer

Panorama: view-driven optimization

Quro: reorder queries

Chestnut: customize data layout

• Other projects

• Ongoing and future work
Outline

- Leveraging application semantics to optimize each layer

  Panorama: view-driven optimization

  Quro: reorder queries

  Chestnut: customize data layout

- Other projects

- Ongoing and future work
Building More Intelligent Data Preparation Systems

- Intern work at DMX (2017, 2019)
- Leveraging open source code to help data scientists with their work

**AutoType** [SIGMOD’18]
- Automatic validation of semantic data types (e.g., zipcode, credit card) using open source code
- Find functions for 84 data types across different domains, with 90% precision

**AutoSuggest** [under submission]
- Learn from open source Jupyter notebooks, and recommend data operations based on data, including op parameters (e.g., which column to join/pivot) and op types
- Achieves much higher precision (>16% higher) than existing commercial vendors
Outline

• Leveraging application semantics to optimize each layer

  Panorama: view-driven optimization

  Quro: reorder queries

  Chestnut: customize data layout

• Other projects

• Ongoing and future work
Outline

• Leveraging application semantics to optimize each layer

Panorama: view-driven optimization

Quro: reorder queries

Chestnut: customize data layout

• Other projects

• Ongoing and future work
Ongoing Work: Maintaining Data Integrity

• Data constraint
  • Constraints among persistent data (e.g., password length, only 1 default option, return and purchase have the same order id, etc.)
  • Defined in HTML/ruby/SQL
  • Very common: 74% data fields involved in constraint, ~1.5 per field

• Constraint issue study [ICSE’20]
  • >100 (per app) constraints missing in DB
  • >13 (per app) conflict constraints
  • >12 (per app) constraints defined incorrectly (easily violated)

• We are building a tool to fix data constraint issues
Future Work

• More chances to leverage application semantics
  • E.g., using data constraint for query optimization
  • using client-level statistics (e.g., click rate) to help query optimization
  • …

• Building a “smarter” middleware that:
  • Incorporate static/dynamic cross-stack optimizations
  • Provide analysis interface to support adding optimization
Summary

- Leveraging application semantics to optimize each layer
Summary

- Leveraging application semantics to optimize each layer
Summary

- **Leveraging application semantics** to optimize each layer

![Diagram showing Quro: reorder queries and Chestnut: customize data layout.](image-url)
Summary

- Leveraging application semantics to optimize each layer

Panorama: view-driven optimization

Quro: reorder queries

Chestnut: customize data layout
Summary

• **Leveraging application semantics** to optimize each layer

Panorama: view-driven optimization

Quro: reorder queries

Chestnut: customize data layout

• Other projects
  • Building more intelligent data preparation systems

• Ongoing and future work
  • Maintaining data integrity (fix data constraint bugs)
  • More chances to leverage application semantics; building a “smarter” middleware