Why do big data and cloud systems slow down and stop?

Shan Lu
What are?

Why do big data and cloud systems slow down and stop?
Big data & cloud systems
Big data & cloud systems

- DB-backed web applications
- Cloud services

- MongoDB
- Amazon Aurora
- PostgreSQL
- SQLite
- Apache HDFS
- Apache Zookeeper
- Apache HBase
- Apache Cassandra
- Apache Spark
DB-backed web applications

HTTP request

Application server

Database query

DBMS
Performance is critical for web applications

- Low latency is critical
  
  Nearly half of the users expect a site to load in less than 2 seconds

- Low latency is challenging given the data size
Cloud services
Cloud services in Azure, AWS, GCP
Reliability is critical for cloud services

<table>
<thead>
<tr>
<th>Compute</th>
<th>Web and Mobile</th>
<th>Developer Services</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytics</td>
<td>Application Integration</td>
<td>AR &amp; VR</td>
<td>AWS Cost Management</td>
</tr>
<tr>
<td>Security &amp; Management</td>
<td></td>
<td>Customer Engagement</td>
<td>Database</td>
</tr>
<tr>
<td>Portal</td>
<td></td>
<td>Internet of Things</td>
<td>Developer Tools</td>
</tr>
<tr>
<td>Active Directory</td>
<td></td>
<td></td>
<td>Blockchain</td>
</tr>
<tr>
<td>Multi-Factor Authentication</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key Vault</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store / Marketplace</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VM Image Gallery &amp; VM Depot</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtual Machines</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satellite</td>
<td>Security, Identity &amp; Compliance</td>
<td>Storage</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile</td>
<td></td>
<td>Machine Learning</td>
<td>Management &amp; Governance</td>
</tr>
<tr>
<td>Networking &amp; Content Delivery</td>
<td></td>
<td></td>
<td>Robotics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reliability is critical for cloud services

Network Interruptions Are the Biggest Culprit

Monday, January 25, 2016 9:01 am EST

AWS's S3 outage was so bad Amazon couldn't get into its own dashboard to warn the world

Websites, apps, security cams, IoT gear knackered

By Shaun Nichols in San Francisco 1 Mar 2017 at 03:00
Outline

- What slows down (big data) web applications [ICSE’18]
  - What can we do about it? [CIKM’17, FSE’18, ICSE’19, CIDR’20]
    - 1000+ bugs found

- What stops cloud systems? [HotOS’19]
  - What can we do about it? [ASPLOS’16, ASPLOS’17, ASPLOS’18, PLDI’19, SOSP’19]
    - 1000+ bugs found
What Slowed Down Database-Backed Web Applications

hyperloop.cs.uchicago.edu

Shan Lu

View-Centric Performance Optimization for Database-Backed Web Applications. ICSE’19
How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
PowerStation: Automatically detecting and fixing inefficiencies of database-backed web applications in IDE. FSE’18
Common Web-app Architecture

HTTP request

Application server

Database query

DBMS
Common Web-app Architecture

```
class BlogsController
  def index
    user_id = 1
    myblogs = Blog.retrieve(user_id)
  end
end

class Blog
  def retrieve(user_id)
    SELECT * FROM blogs where uid = id
  end
end

http://www.xxx.com/blogs/index
```
class BlogsController
  def index
    user_id = 1
    myblogs = Blog.retrieve(user_id)
  end
end

View

Model

Controller

Application server

DBMS

app/views/blogs/index.html.erb

@myblogs.each do |blog|
  blog.content<br/>
end

http://www.xxx.com/blogs/index

1001 unread blogs

- Arriving at Zurich
- Stopping by Bern
- One day at Luzern
- Love love Berner Oberland
- Love Berner Oberland
- Back to Lausanne
Potential sources of inefficiencies

Object Relational Mapping Framework

Model

SELECT * FROM blogs where uid = id

class Blog
    Blog.where(uid = user_id)
end

DBMS
Potential sources of inefficiencies

MVC Design Pattern

Controller

View

Model

Object Relational Mapping Framework

class Blog

  Blog.where(uid = user_id)

end

app/views/blogs/index.html.erb

@myblogs.each do |blog|
  blog.content<br/>
end

SELECT * FROM blogs where uid = id

<table>
<thead>
<tr>
<th>blogs</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
</tr>
<tr>
<td>contents</td>
</tr>
</tbody>
</table>
Outline

How severe is the problem?
Profile 12 apps from 6 common categories

What are the common inefficiency patterns?
Build performance-bug taxonomy

How to solve the problem?
Design automated bug detection & fixing
Outline

Profile 12 apps from 6 common categories

Build performance-bug taxonomy

Design automated bug detection & fixing

64 issues in 40 pages
Profiling methodology

Table 1: Details of the applications chosen in our study

<table>
<thead>
<tr>
<th>Category</th>
<th>Abbr.</th>
<th>Name</th>
<th>Stars</th>
<th>Commits</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forum</td>
<td>Ds</td>
<td>Discourse</td>
<td>21238</td>
<td>22501</td>
<td>568</td>
</tr>
<tr>
<td></td>
<td>Lo</td>
<td>Lobster</td>
<td>1304</td>
<td>1000</td>
<td>48</td>
</tr>
<tr>
<td>Collaboration</td>
<td>Gi</td>
<td>Gitlab</td>
<td>19255</td>
<td>49810</td>
<td>1276</td>
</tr>
<tr>
<td></td>
<td>Re</td>
<td>Redmine</td>
<td>2399</td>
<td>13238</td>
<td>6</td>
</tr>
<tr>
<td>E-commerce</td>
<td>Sp</td>
<td>Spree</td>
<td>8331</td>
<td>17197</td>
<td>709</td>
</tr>
<tr>
<td></td>
<td>Ro</td>
<td>Ror_ecommerce</td>
<td>1109</td>
<td>1727</td>
<td>21</td>
</tr>
<tr>
<td>Task-management</td>
<td>Fu</td>
<td>Fulcrum</td>
<td>1502</td>
<td>697</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Tr</td>
<td>Tracks</td>
<td>835</td>
<td>3512</td>
<td>62</td>
</tr>
<tr>
<td>Social</td>
<td>Da</td>
<td>Diaspora</td>
<td>11183</td>
<td>18734</td>
<td>335</td>
</tr>
<tr>
<td>Network</td>
<td>On</td>
<td>Onebody</td>
<td>1592</td>
<td>1220</td>
<td>6</td>
</tr>
<tr>
<td>Map</td>
<td>OS</td>
<td>Openstreetmap</td>
<td>664</td>
<td>8000</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>FF</td>
<td>Fallingfruit</td>
<td>41</td>
<td>1106</td>
<td>7</td>
</tr>
</tbody>
</table>

Synthesize DB content based on real-world website statistics

Top 2 Apps in 6 popular categories

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
Profiling End-to-end Page Time

6 apps have pages > 3s
11 apps have pages > 2s

40 problematic pages
Server takes most time

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
Why is it slow?

There are inefficiency bugs!

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
Why is it slow?

- We manually fix the 64 issues we found across 39 pages

There are bugs!

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
Outline

Profile 12 apps from 6 common categories

Build performance-bug taxonomy

Design automated bug detection & fixing

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
Common Performance Anti-patterns

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

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
ORM API Misuse

- Inefficient Computation
  26 issues across 8 apps

- Unnecessary Data Access
  44 issues across 11 apps

- Inefficient Data Retrieval
  9 issues across 4 apps

- Inefficient Rendering
  5 issues across 4 apps

- Unnecessary Computation
  22 issues across 10 apps

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
ORM API Misuse

Inefficient Computation
26 issues across 8 apps

Unnecessary Computation
22 issues across 10 apps

Inefficient Data Access
44 issues across 11 apps

Unnecessary Data Retrieval
9 issues across 4 apps

Inefficient Rendering
5 issues across 4 apps

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
ORM API Misuse: inefficient computation

<table>
<thead>
<tr>
<th>ORM Method</th>
<th>SQL Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>project.issues.count&gt;0</td>
<td>SELECT COUNT(*) FROM issues WHERE project_id = ?</td>
</tr>
<tr>
<td>project.issues.any?</td>
<td>SELECT COUNT(*) FROM issues WHERE project_id = ?</td>
</tr>
<tr>
<td>project.issues.exists?</td>
<td>SELECT 1 AS ONE FROM issues WHERE project_id = ? LIMIT 1</td>
</tr>
</tbody>
</table>

inefficient inefficient efficient

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
ORM API Misuse: unnecessary computation

```sql
values.each do |value|
  u.issues.include? value
end
```

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
Values each do |value|
- u.issues.include?value
end

+ rans = u.issues
values.each do |value|
+ rans.include?value
end

20X speed up

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
ORM API misuses that affect memory consumption

- `map (:id) VS pluck (:id)`
- `pluck(size).sum VS sum(size)`
- `pluck + pluck VS SQL UNION`
- `...`
How to tackle API Misuses?

- Why cannot existing compiler handle this?

- Can we extend compiler to
  - Understand ORM APIs and queries?
  - Detect the problem?
  - Solve the problem?
Database-aware PDG

```
v1 = u
v2 = values
values.reject |val|
u.issues.include?val
end
end
```

SQL: `SELECT * from issues WHERE user_id=?`

(b) PDG

Copy: v1 = u
Copy: v2 = values
val = v2[
Call: v3=v1.issues
Call:v3.include?val
query node
data edge
control edge
Detect and Fix

Copy: v1 = u

Copy: v2 = values

val = v2[

Call: v3=v1.issues

Call:v3.include?val

Loop-invariant query

PowerStation: Automatically detecting and fixing inefficiencies of database-backed web applications in IDE. FSE’18
PowerStation: Automatically detecting and fixing inefficiencies of database-backed web applications in IDE.
Try our Powerstation!

- 12 real world apps
- 1221 inefficiencies found

PowerStation: Automatically detecting and fixing inefficiencies of database-backed web applications in IDE. FSE’18
Common Performance Anti-patterns

1. Database Design
   - 41 issues across 10 apps

2. ORM API Misuse
   - 106 issues across 12 apps

3. Application Design Tradeoff
   - 47 issues across 12 apps
Database Design Problem

- Missing fields (8 issues across 5 apps):
  - fields derivable from other fields and not persistently stored

<table>
<thead>
<tr>
<th>id</th>
<th>longitude</th>
<th>latitude</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Missing index (33 issues across 10 apps)

How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
How not to structure your database-backed web applications: a study of performance bugs in the wild.

ICSE'18.
How not to structure your database
backed web applications: a study of performance bugs in the wild.
ICSE’18.
SELECT count(*) FROM moderations JOIN stories where stories.user_id = @user.id AND moderations.created_at > 5.days.ago

Whether to show this guideline
How to tackle application design tradeoffs?

- Can we do automated optimization?

- Help developers make informed decision, by providing
  - Cost information
  - Alternative display/functionality options
def index
  @blogs = Blog.all
  render "index"
end

app/views/blogs/index.html.erb

@blogs.each do |blog|
  blog.content<br/>
end

Arriving at Zurich
Stopping by Bern
One day at Luzern
Love love Berner Oberland
Love Berner Oberland
Back to Lausanne
def index
  @blogs = blog.all
  render "index"
end

app/views/blogs/index.html.erb

@blogs.each do |blog|
  blog.content<br/>
end

View-Centric Performance Optimization for Database-Backed Web Applications. ICSE’19
def index
  @blogs = Blog.all.paginate(...)
  render "index"
end

app/controllers/blogs_controller.rb

app/views/blogs/index.html.erb

@blogs.each do |blog|
  blog.content<br/>
end

will_paginate @blogs

View-Centric Performance Optimization for Database-Backed Web Applications. ICSE'19
def index
    @blognum = blog.count
    render "index"
end

http://blogs/index

There are @blognum blogs

1001 unread blogs
- remove
- approximation
- async loading

Arriving at Zurich
Stopping by Bern
One day at Luzern
Love love Berner Oberland
Love Berner Oberland

View-Centric Performance Optimization for Database-Backed Web Applications. ICSE’19
more than 20 unread blogs

app/controllers/blogs_controller.rb

```ruby
def index
  @blognum = blog.limit(21).count
  render "index"
end
```

app/views/blogs/index.html.erb

```erb
There are @blognum>20?'more than 20':@blognum blogs
```

Arriving at Zurich

Stopping by Bern

One day at Luzern

Love love Berner Oberland

Love Berner Oberland

View-Centric Performance Optimization for Database-Backed Web Applications. ICSE'19
def index
  @blognum = blog.count
  render "index"
end

app/controllers/blogs_controller.rb

app/views/blogs/index.html.erb

@blognum.unread_blogs

Arriving at Zurich

Stopping by Bern

One day at Luzern

Love love Berner Oberland

Love Berner Oberland

View-Centric Performance Optimization for Database-Backed Web Applications. ICSE’19
Try our Panorama!

- 12 real world apps
- 149 performance-enhancing opportunities identified for 119 costly HTML tags
- 4.5X average page-load time speedup
- User study agrees!
Slow downs in web applications

- Real world database-backed applications perform poorly
- Data-related performance anti-patterns exist
- Automatic tools are built to detect and fix performance issues

View-Centric Performance Optimization for Database-Backed Web Applications. ICSE’19
How not to structure your database-backed web applications: a study of performance bugs in the wild. ICSE’18.
PowerStation: Automatically detecting and fixing inefficiencies of database-backed web applications in IDE. FSE’18
What stopped cloud services?

Efficient and Scalable Thread-Safety Violation Detection --- Finding thousands of concurrency bugs during testing. SOSP’19
DFix: Automatically Fixing Timing Bugs in Distributed Systems. PLDI’19
FCatch: Automatically detecting time-of-fault bugs in cloud systems. ASPLOS’18
DCatch: Automatically Detecting Distributed Concurrency Bugs in Cloud Systems. ASPLOS’17
What Bugs Cause Production Cloud Incidents? HotOS’19
Need to study real-world cloud incidents
Existing studies for cloud incidents

- **Cause**
  - Unknown
  - Unknown
- **Data source constraints**
- **Software**
- **Hardware**
- **Others**

**Handling**

- **Unknown**

**Handling**

**Bugzilla**

**NEWS**

[6] Leesatapornwongsa. TaxDC. In ASPLOS’16
Our work

Handling

Cause

6-month high-severity incidents in Microsoft Azure services
What causes incidents in non-cloud software?

- Concurrency bugs
- Semantic bugs
- Memory bugs
Our findings

Handling

6-month high-severity incidents in Microsoft Azure services

Cause

Software

Hardware

Others

Concurrent bugs

Semantic bugs

Memory bugs

What Bugs Cause Production Cloud Incidents? HotOS’19
Our findings

Handling

Cause

6-month high-severity incidents in Microsoft Azure services

Software Hardware Others

Concurrency bugs

Semantic bugs

Memory bugs

What Bugs Cause Production Cloud Incidents? HotOS’19
Our findings

Handling

Cause

6-month high-severity incidents in Microsoft Azure services

- Software
- Hardware
- Others

Concurrency bugs
Semantic bugs
Memory bugs

What Bugs Cause Production Cloud Incidents? HotOS’19
Our findings

What Bugs Cause Production Cloud Incidents? HotOS’19
Our findings

Handling

- Concurrency bugs (50% persistent races)
- Semantic bugs
- Resource (memory) leaks

Cause

6-month high-severity incidents in Microsoft Azure services

Software

Hardware

Others
Our findings

What Bugs Cause Production Cloud Incidents? HotOS'19
Our findings

>50% through mitigation **without** patches

6-month high-severity incidents in Microsoft Azure services

- Fault-handle bugs
- Data-format bugs
- Concurrency bugs (50% persistent races)
- Semantic bugs
- Resource (memory) leaks

What Bugs Cause Production Cloud Incidents? *HotOS’19*
What can we do?

Handling

mitigation

Cause

Concurrency bugs
(50% persistent races)

What Bugs Cause Production Cloud Incidents? / HotOS ’19

github.com/microsoft/TSVD
Conclusions

- Software bugs widely exist in big data & cloud systems
- Software bugs are taking on new forms in big data & cloud systems
  - Memory data $\leftrightarrow$ Persistent data
- A lot of bug fighting can be done and to be done
- Our are making our bug set and tools open source!
Thanks!