Strict Serializability is Harmless: A New Architecture for Enterprise Applications

Sérgio Fernandes and João Cachopo

October 25, 2011, Portland, Oregon, USA
Context

Consistency

max

Performance
Consistency

max

Performance

Context

😊
Context

Consistency

max

Performance

😊
Why Strict Serializability?

• Concurrent programming is difficult
• Reduce programming effort
• Fewer bugs
Can we live without it?
Can we live without it?

- Yes, but...
Can we live without it?

- Yes, but...
Can we live without it?

- Yes, but...
Can we live without it?

• Yes, but...
Can we live without it?

- Yes, but...
Can we live without it?

• Yes, but...
Can we live without it?

• Yes, but...
Can we live without it?

- Yes, but...
Can we live without it?

- Yes, but...
Can we live without it?

• Yes, but...
Can we live without it?

- Yes, but...
So... why don’t we always enforce it?
“To be sure of correctness you should always use the serializable isolation level.”

-- Martin Fowler, PoEAA, 2002
“To be sure of correctness you should always use the serializable isolation level.”

“(…) choosing serializable really messes up the liveness of a system, (…) you often have to reduce serializability (…) to increase throughput.”

-- Martin Fowler, PoEAA, 2002
“To be sure of **correctness** you should always use the **serializable** isolation level.”

“(…) choosing serializable really messes up the liveness of a system, (…) you often have to reduce serializability (…) to increase throughput.”

“You have to decide what risks you want take and make your own trade-off of **errors** versus **performance**.”

-- Martin Fowler, PoEAA, 2002
Serializability is a performance killer!
Strict Serializability is a performance killer!
Strict Serializability is a performance killer, with current architectures!
Strict Serializability is a performance killer, with current architectures!

Why?
Not so long ago

Client \rightarrow RDMBS
Not so long ago

User Interface

Client

Application logic
Transactions
Persistence

RDMBS
More clients
More clients

User Interface

Client

RDMBS

Application logic

Transactions

Persistence
Even more clients

User Interface

Client

Client

Application logic

App Server

Transactions Persistence

RDMBS
Even more clients

User Interface

Client

Client

Application logic

Simple

App Server

Transactions Persistence

RDMBS
Even more clients

User Interface

Client

Client

Application logic

Simple

App Server

Transactions Persistence

RDMBS

1 request

10 round-trips
Even more clients

User Interface

Client

Client

Simple

App Server

Transactions Persistence

RDMBS

Application logic

Latency ➪ Latency
Even more clients

User Interface

Client

Client

Application logic

Complex

App Server

Transactions Persistence

RDMBS
Even more clients

User Interface

1 request

Complex

Transactions Persistence

1000 round-trips

Client

App Server

Client

RDMBS

Application logic
Even more clients

User Interface

Client

Client

Application logic

App Server

Complex

Transactions Persistence

RDMBS

Latency

Latency

Latency
Caching

User Interface

Client

Client

Application logic

App Server

Cache

Transactions Persistence

RDMBS
“[cache-based] application server implementations are incorrect when used with databases providing snapshot isolation.”

-- Perez-Sorrosal, Middleware’07
Application logic

User Interface

Client

Client

App Server

Transactions Persistence

RDMBS
New Architecture

User Interface

Client

Client

Application logic

Transactions

App Server

Persistence

RDMBS
Enabling Technology

- Recent hardware
- Transactional Memories
Target Applications

- Concurrent
- Transactions
- Persistence
- Rich domain model
- Complex logic
Target Applications

• Concurrent
• Transactions
• Persistence
• Rich domain model
• Complex logic

AKA
Enterprise Applications
Fénix Framework: Key Features

- STM-based
- Transactional Domain Objects
- Domain Object Cache
- Transparent Persistence
- Distributed
JVSTM

- First **multi-version** STM (2005)
- Designed for **very large** transactions with **high read/write ratio**
- Read-only transactions **never** conflict
- Commit-time locking (for R/W only)
- **Strict Serializability**
# Transactional Domain Objects

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID: 324</td>
</tr>
<tr>
<td>name: John</td>
</tr>
<tr>
<td>contact: ...</td>
</tr>
</tbody>
</table>
Transactional Domain Objects

Person

<table>
<thead>
<tr>
<th>OID: 324</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: John</td>
</tr>
<tr>
<td>contact: ...</td>
</tr>
</tbody>
</table>
Transactional Domain Objects

Person
OID: 324
name: John
contact: ...

Person
OID: 324
name: John
contact: ...

Tx1  Tx2
Transactional Domain Objects

STM protected

Person
- OID: 324
- name: John
- contact: ...

Person
- OID: 324
- name: John
- contact: ...

Tx1 Tx2
Transactional Domain Objects

Person

- OID: 324
- name: John
- contact: ...

STM protected

Tx1  Tx2
Transactional Domain Objects

STM protected

Person

- OID: 324
- name: John
- contact: ...

Tx1  Tx2

✓
Domain Object Cache

Person

- OID: 324
- name: John
- contact: ...

Tx1  Tx2
Domain Object Cache

Cache

| 324 | SoftRef( ) |

Person

OID: 324
name: John
contact: ...

Tx1 Tx2
Domain Object Cache

- Domain objects have identity
- DB round-trips reduced
Evolution of the Fénix original architecture
<table>
<thead>
<tr>
<th>Presentation Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Services</td>
</tr>
<tr>
<td>OJB + DAOs</td>
</tr>
</tbody>
</table>
Presentation Logic

Services

Views

OJB + DAOs

Anemic Domain
Presentation Logic

Rich Domain Model
<table>
<thead>
<tr>
<th>Presentation Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>DML + Java + JVSTM</td>
</tr>
<tr>
<td>Presentation Logic</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>DML + Java</td>
</tr>
<tr>
<td>DML Compiler + JVSTM</td>
</tr>
</tbody>
</table>
Presentation Logic

DML + Java

Fénix Framework
<table>
<thead>
<tr>
<th>Presentation Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Domain</td>
</tr>
<tr>
<td>Fénxix Framework</td>
</tr>
</tbody>
</table>
Other Applications

Fénix Framework
O projecto tem 84 sugestões

**Suporte a Flash Video**

Novo

#59 | Por: João Manuel Brisson Lopes em: 30/09/2009 | 2 comentários

Existem já bastantes disciplinas com elementos de estudo / demonstração em vídeo. O formato flash permite que esses vídeos sejam de pequeno tamanho e o seu streaming, o que é mais eficiente do q (…)

**Pautas de Avaliação**

Novo

#43 | Por: Alexandre Jorge Salgueiro Chambel em: 08/07/2009 | 6 comentários

As páginas com as pautas de avaliação das cadeiras só deveriam estar disponíveis para pessoas pertencentes ao IST, ou seja, deveriam pedir autenticação.

**Votar Online para a AEIST (e outras eleições)**

Novo

#45 | Por: Pedro Manuel Romão Ferreira Pereira da Silva em: 21/07/2009 | 6 comentários

Parece-me ser uma boa ideia que se pudesse votar no fénix nas listas da AEIST, em vez de se exigir exclusivamente o voto presencial. Seria semelhante do que se já faz (e bem!) para eleger o d (…)

**Histórico do Aluno**

Novo

#71 | Por: Diogo Alexandre dos Santos Nunes em: 12/12/2009 | 5 comentários

No seguimento desta ideia (https://fears.ist.utl.pt/#Project3&viewFeature55) aqui vai outra: Implementar no Fénix, uma opção que permitisse consultar o histórico/aproveitamento de um aluno (…).
NEW COMER?
JOIN MAMBA NATION!
IT'S FREE!

CONNECT!

E-mail

Lost your password?

ENTER

CONNECT WITH FACEBOOK

Discover Mamba Nation! Create the coolest avatar on the internet! Come go wild with your friends and meet people!
Evaluation
Evaluation

• Real-world applications
  • Fénix EDU
  • dot.IST
  • FeaRS
  • ...


Evaluation

• Real-world applications
  • Fénix EDU
  • dot.IST
  • FeaRS
  • ...
• Benchmark
  • TPC-W
Fénix EDU

2001: Started as a typical web application

2005: New architecture
Code Base

1.2 million LOC
12 full-time engineers
3,600 web pages
8,000 classes (1,200 domain)
NOT ‘Hello World!’
User Base

6,000 BSc
4,000 MSc & PhD
1,100 faculty members
Transactions / quarter

Number of transactions

Quarter

Q4'06 Q1'07 Q2'07 Q3'07 Q4'07 Q1'08 Q2'08 Q3'08 Q4'08 Q1'09 Q2'09 Q3'09 Q4'09 Q1'10 Q2'10 Q3'10 Q4'10 Q1'11

1e+08 9e+07 8e+07 7e+07 6e+07 5e+07 4e+07 3e+07 2e+07 1e+07 0
Daily @ Feb 2011

Number of transactions vs Day of month

- Number of transactions on Day 9 is significantly higher than on other days.
- There is a general trend of decreasing numbers of transactions after Day 9.
Daily @ Feb 2011

Write Tx
Conflicts

Rate

Day of month

0% 2% 4% 6% 8% 10%
1 3 5 7 9 11 13 15 17 19 21 23 25 27
Some Fénix statistics
Normal week
January 2010
February 2010
No way we could get this before

17th to 19th
September 3rd
Enrollment counts
Hardware

2 nodes (fault-tolerance)

2 quad-core CPU

32GB RAM (6GB used)

Underused!
2008: Already new architecture

Fewer users (no students)

No public pages
Transactions / quarter

Number of transactions

Quarter

Q4'08 Q1'09 Q2'09 Q3'09 Q4'09 Q1'10 Q2'10 Q3'10 Q4'10 Q1'11
Writes & conflicts

Rate

Quarter

Write Tx
Conflicts
TPC-W

- Web commerce application: Bookstore
- JDBC vs. Fénix Framework
- Web Interactions Per Second (WIPS)
- Workloads:
  - Read-only
  - Browsing (5% writes)
  - Shopping (20% writes)
- Up to 8 app servers
Single server

- Java 6
- MySQL 5.1
- Tomcat 6
- 48-core: 10 clients
- 128GB RAM: <15GB used
Single server

![Graph showing Web Interactions Per Second (WIPS) for different scenarios with varying number of items.]
Single server

![Graph showing speed-up FF/JDBC with number of items and different activities (Read-only, Browsing, Shopping). The graph indicates a speed-up of 3.7x at a certain point.](image-url)
# TPC-W specs

<table>
<thead>
<tr>
<th>#Emulated Clients</th>
<th>#DB Customers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,880</td>
</tr>
<tr>
<td>10</td>
<td>28,880</td>
</tr>
<tr>
<td>60</td>
<td>172,800</td>
</tr>
</tbody>
</table>
Single server

![Graph showing speed-up FF/JDBC for different numbers of clients.](chart)

- **Read-only**
- **Browsing** (22x speed-up)
- **Shopping**

- Number of clients: 10, 20, 30, 40, 50, 60
- Speed-up FF/JDBC: 1, 5, 10, 15, 20, 25
Clustered server

- 10 machines (client + 8 x server + DB)
- 40 clients evenly split
- LAN
- 8 cores
- 8GB RAM
Clustered server

![Graph showing WIPS with varying number of application servers]

- FF Read-only
- JDBC Read-only

Number of Application Servers:
- 2
- 4
- 6
- 8
Clustered server

![Bar graph showing WIPS for different numbers of application servers and browsing/shopping scenarios.]

- Number of Application Servers: 2, 4, 6, 8
- WIPS: From 0 to 600
- Scenarios: FF Browsing, FF Shopping, JDBC Browsing, JDBC Shopping

Legend:
- FF Browsing: Red
- FF Shopping: Red
- JDBC Browsing: Green
- JDBC Shopping: Green
Conclusions

• Current Tx away from App Logic
  • DB access is expensive
• Recent hardware + STM
• Strict Serializability & Performance
• Best with high read/write ratio
Future Work

- Lock-free commit
- Increase % writes
- Increase scalability