Trouble with transactions

Evan Jones

http://evanjones.ca

A love story


I fell in
love...

...with transactions

...with transactions

Transactions = correct programs

A four year romance

A four year romance

Main memory transactions

A four year romance

Main memory transactions

Automatic partitioning

A four year romance

Main memory transactions

Consolidating workloads

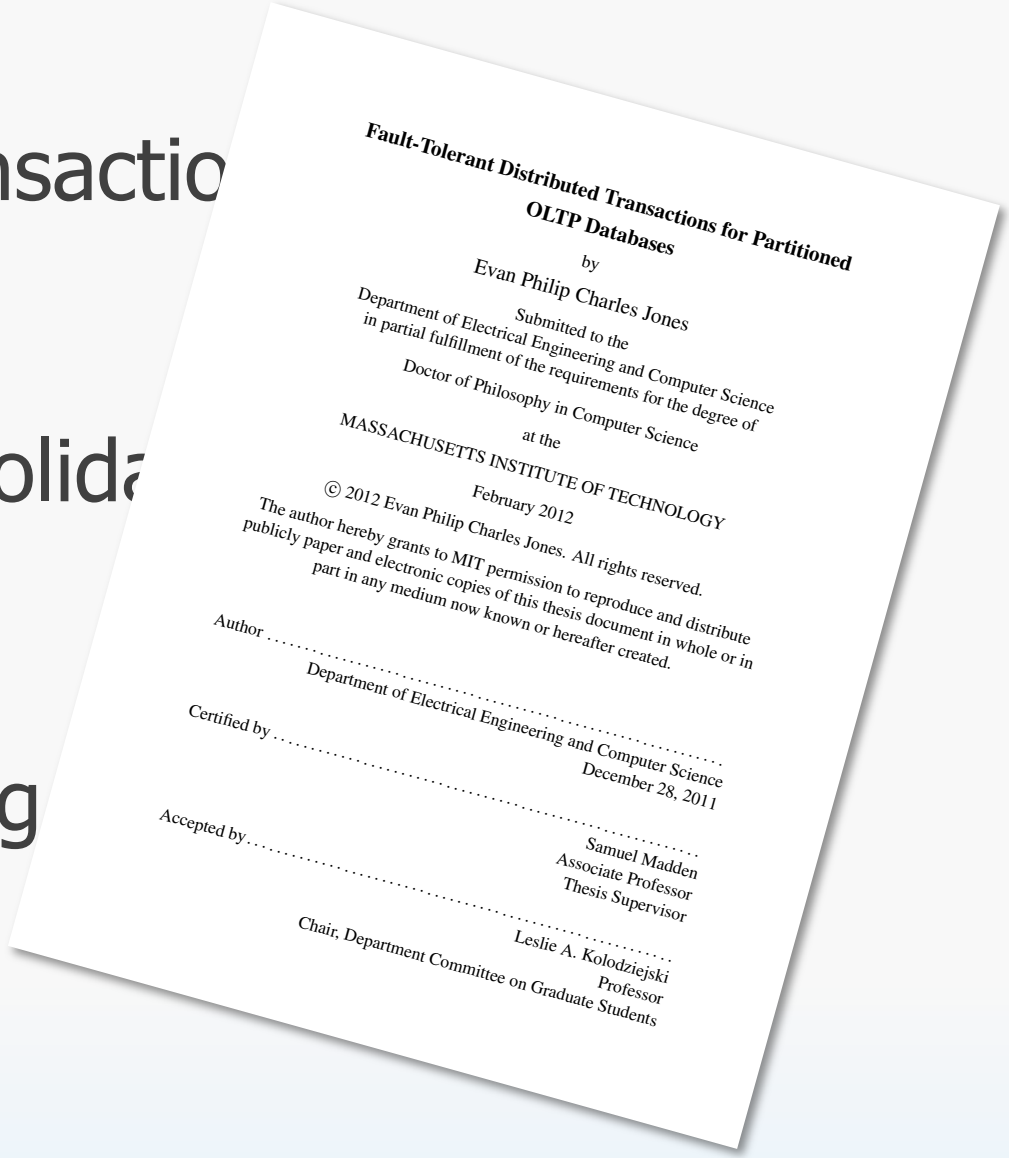
Automatic partitioning

A four year romance

Main memory transactio

Consolida

Automatic partitioning



Fault-Tolerant Distributed Transactions for Partitioned OLTP Databases

by

Evan Philip Charles Jones

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 2012

© 2012 Evan Philip Charles Jones. All rights reserved.
The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Author . . . . . .
Department of Electrical Engineering and Computer Science
December 28, 2011

Certified by . . . . . .
Samuel Madden
Associate Professor
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor
Chair, Department Committee on Graduate Students

Mitro

Mitro

Applications, not databases

Example: Read bank balance

```
connection.begin_transaction()
print connection.query_balance()
# ... do some computation ...
print connection.query_balance()
connection.commit()
```

Example: Read bank balance

```
connection.begin_transaction()
print connection.query_balance()  = $1000
# ... do some computation ...
print connection.query_balance()
connection.commit()
```

Example: Read bank balance

```
connection.begin_transaction()
print connection.query_balance()  = $1000
# ... do some computation ...
print connection.query_balance()  = $1500
connection.commit()
```

Example: Read bank balance

```
connection.begin_transaction()
print connection.query_balance()  = $1000
# ... do some computation ...
print connection.query_balance()  = $1500 💔
connection.commit()
```

💔#1: Weak defaults

Transactions = Serializability

Postgres: Read committed

MySQL/InnoDB: “Repeatable read”

(reality: something weird)

💔#1: Weak defaults

Transactions = Serializability 💖

Postgres: Read committed

MySQL/InnoDB: “Repeatable read”

(reality: something weird)

💔#1: Weak defaults

Transactions = Serializability 💖

Postgres: Read committed

MySQL/InnoDB: “Repeatable read”

(reality: something weird)

Set SERIALIZABLE by default

Example 2: Withdrawal

```
try:
  connection.begin_transaction()
  balance = connection.query_balance()
  connection.update_balance(balance – 500)
  some_function(connection)
  connection.commit()
except e:
  connection.rollback()
  print connection.query_balance()
```

Example 2: Withdrawal

```
try:
  connection.begin_transaction()
  balance = connection.query_balance()      = $1000
  connection.update_balance(balance - 500)
  some_function(connection)
  connection.commit()
except e:
  connection.rollback()
  print connection.query_balance()
```

Example 2: Withdrawal

```
try:
  connection.begin_transaction()
  balance = connection.query_balance()        = $1000
  connection.update_balance(balance - 500)  = okay!
  some_function(connection)
  connection.commit()
except e:
  connection.rollback()
  print connection.query_balance()
```

Example 2: Withdrawal

```
try:
  connection.begin_transaction()
  balance = connection.query_balance()        = $1000
  connection.update_balance(balance - 500)    = okay!
  some_function(connection)                   Exception
  connection.commit()
except e:
  connection.rollback()
  print connection.query_balance()
```

Example 2: Withdrawal

```
try:
  connection.begin_transaction()
  balance = connection.query_balance()        = $1000
  connection.update_balance(balance - 500)    = okay!
  some_function(connection)                   Exception
  connection.commit()
except e:
  connection.rollback()
  print connection.query_balance()            = $500
```

Example 2: Withdrawal

```
try:
  connection.begin_transaction()
  balance = connection.query_balance()          = $1000
  connection.update_balance(balance - 500)      = okay!
  some_function(connection)                     Exception
  connection.commit()
except e:
  connection.rollback()
  print connection.query_balance()              = $500
```

Example 2: Withdrawal

```
try:
  connection.begin_transaction()
  balance = connection.query_balance()          = $1000
  connection.update_balance(balance - 500)      = okay!
  some_function(connection)                     Exception
  connection.commit()
except e:
  connection.rollback()
  print connection.query_balance()              = $500
```

💔#2: Implicit begin

`some_function` committed (my fault)

DB automatically started new txn

💔#2: Implicit begin

`some_function` committed (my fault)

DB automatically started new txn

If you begin, you commit

Like memory in C/C++

Nested transactions can help

Don’t implicitly start transactions

Transactions: Use with care

- Communication with external systems
- Accidental long running transactions
- Retry loops for concurrency errors

Transactions: Use with care

- Communication with external systems
- Accidental long running transactions
- Retry loops for concurrency errors

Make it hard to use systems incorrectly

http://evanjones.ca/