STAYING IN SYNC.

from **TRANSACTIONS** to **STREAMS**

@martinkl  martinkl.com /gcon
UNIVERSITY OF CAMBRIDGE
Web FE

Users

Products

Mobile API

Billing
SAME DATA in DIFFERENT FORM

Denormalisation

Caching

Indexes

Aggregations
set \( X = A \)  

set \( X = B \)  

set \( X = A \)  

set \( X = B \)
set $X = A$  

set $X = A...$

set $X = B$  

set $X = B...$
Лепра
@leprasorium
Добро пожаловать отсюда
Default City

Лепра @leprasorium · 2h
Викторианские советы
Часть 2 pic.twitter.com/21PraRYBaO
Details

Лепра @leprasorium · 2h
Викторианские советы
Часть 1 pic.twitter.com/BVE6ao8711
Details

Go to full profile
insert into emails
  (mailbox_id, unread, body)
values(42, true, 'Hello!');
update mailboxes
set unread_count += 1
where id = 42;
begin transaction;

insert into emails
(mailbox_id, unread, body)
values(42, true, 'Hello!');

update mailboxes
set unread_count += 1
where id = 42;

commit;
2-Phase commit?
2-PHASE COMMIT

1. Begin transaction

- Web app
- Coordinator
- DB
- Cache
- Index

Submit data
2-PHASE COMMIT

2. Read & write data

- Coordinator
- DB
- Cache
- Index

Submit data to Web app, which accesses Coordinator for write operation and caches the data. The index is updated as well.
2-PHASE COMMIT

3. Prepare
2-PHASE COMMIT

4. Commit

- Coordinator
- Web app
- DB
- Cache
- Index

User submits data to Web app, which commits to Coordinator. Coordinator commits to DB, Cache, and Index.
$X = A \quad X = A \quad \text{PREPARE} \quad \text{COMMIT}$

- $x =$ large
- $x =$ large
- $x =$ large
- $x =$ large

- $\text{OK}$
- $\text{LOCKED}$
- $\text{OK}$
- $\text{OK}$

$X = B$
$X = A$, $X = A$, PREPARE

BOOM!

$X = B$
Cannot safely time out lock!
BETWEEN THE DEVIL AND THE DEEP BLUE SEA

Distributed transactions
    poor performance, operational problems, ...

OR

Eventual consistency
    more like perpetual inconsistency, amirite?
set $X = A$

set $X = B$

set $X = A$

set $X = B$
set $X = A$

ORDERING

set $X = B$

set $X = A$
Stupidly simple solutions are the best.

\[
\begin{array}{cccc}
  x &= 5 & y &= 8 \\
  x &= 6 & x &= 7 & y &= 9 \\
\end{array}
\]
Stupidly simple solutions are the best.

totally ordered sequence
**STUPIDLY SIMPLE SOLUTIONS ARE THE BEST**

| $x=5$ | $y=8$ | $x=6$ | $x=7$ | $y=9$ |

- totally ordered sequence
- append-only, persistent
State machine replication

(Lamport 1978; Schneider 1990)
State machine replication

(Lamport 1978; Schneider 1990)

Input → State machine (deterministic) → Output
State machine replication
(Lamport 1978; Schneider 1990)

Input

State machine (deterministic)

Output

Single writer principle
(Thompson 2011)
State machine replication
(Lamport 1978; Schneider 1990)

Input

\[ \rightarrow \]

State machine (deterministic)

\[ \rightarrow \]

Output

Single writer principle
(Thompson 2011)

Event sourcing
(Vernon 2013)
REPPLICATION

Leader

Follower

reads & writes

reads & writes

reads

reads
REPLICATION

Leader

replication

Follower

replication
Log file

Append

↑

tail -f
Leader

Follower applies writes in order

Current log position

Follower
Leader

Follower applies writes in order

Current log position

network interruption

X=5  Y=8  X=6  X=7  Y=9  Z=3  X=8
Kafka

Web servers

Log aggregation

Analytics

Recommendation systems

Ops support
Log file/stream

request: GET
url: /hello.html
user: 123456
client_ip: 12.34.5.6
browser: Chrome 40
referrer: google.com
timestamp: 14255006
stream

oldest events

most recent events
stream

new events added here

← oldest events

... → most recent events
stream

new events added here

oldest events

most recent events

consumer position

(close to head of stream)
Log file

Append
tail -f
Comparison: AMQP

Consumer 1
Consumer 2
Comparison: AMQP

Consumer 1

Consumer 2

failed
Comparison: AMQP

Consumer 1

Consumer 2

failed

out-of-order redelivery
Job queue

"please send this email for me"
"please charge this credit card"

vs.

Event log

"user viewed a web page"
"customer X purchased product Y"
Stupidly simple solutions are the best.

\[
\begin{array}{ccc}
  x &=& 5 \\
  y &=& 8 \\
  x &=& 6 \\
  x &=& 7 \\
  y &=& 9 \\
\end{array}
\]

totally ordered sequence

append-only, persistent
BETWEEN THE DEVIL AND THE DEEP BLUE SEA

Distributed transactions
poor performance, operational problems, ...

OR

Eventual consistency
more like perpetual inconsistency, amirite?
Enforcing invariants

Integrity constraints

unique username

account balance positive
register "jane"
register "jane"

username claims

ok!

ok

registrations

register "jane"
register "jane"

register "jane" ok!
register "jane"  "jane" is taken

username claims

registrations

ok!
taken!
ok
error

register "jane"  ok!
STOP DOING THIS.
Instead, embrace the log.

Log

(Kafka recommended)
INSTEAD, EMBRACE THE LOG

Consumers independently apply writes in LOG ORDER
like UNIX pipes
but for distributed data
mysql | elasticsearch
kafka | sed | awk | memcached
stream  new events added here

oldest events  most recent events

re-processing historical events starts here

consumer position (close to head of stream)
stream

new events added here

← oldest events

most recent events →

complete history
New index (starts off empty)
New index (starts off empty)
oldest events

newest events

New index

read

1 2 3

...
"Reducing irreversibility"

(Bartlett and Fowler, 2015)
stream

new events added here

oldest events

most recent events

complete history
(using compaction to collect garbage)
Kafka changelog compaction
Tools:
Kafka Connect
Bottled Water (PostgreSQL)

Stream processing:
Kafka Streams / Samza / Storm / Spark / Flink / ...
Web FE

Mobile API

Users

Products

Billing

Recommendations

Search
Interactive R/W transactions

- Lots of network round trips
- Lots of locking — or poor consistency
- Global coordination, 2-phase commit

Ordered event log

- All consumers see events in the same order
- Pipelined, non-blocking
- Idempotent operations for fault tolerance
STUPIDLY SIMPLE SOLUTIONS ARE THE BEST

 totally ordered sequence

append-only, persistent
References (1)


