Transactional replication: algorithms and properties

Tadeusz Kobus, Maciej Kokociński, Paweł T. Wojciechowski

Institute of Computing Science
Poznań University of Technology

Poznań, 24/05/2017

Problem

Forbes / Tech

AUG 19, 2013 @ 03:50 PM 19,645 VIEWS

Amazon.com Goes Down, Loses $66,240 Per Minute

Kelly Clay
CONTRIBUTOR
I write about social media, startups and technology trends.
FULL BIO >
Opinions expressed by Forbes Contributors are their own.

It's been a bad week for ecommerce. On Friday, Google GOOG +1.33% temporarily went dark, causing a 40% drop in web traffic. Today Amazon.com AMZN +0.56% went down for approximately 30 minutes, preventing shoppers from accessing the site via Amazon.com, mobile and Amazon.ca.

During the outage, users were hit with an error message: " **Oops!** We're very sorry, but we're having trouble doing what you just asked us to do. Please give us another chance–click the Back button on your browser and try your request again. Or start from the beginning on our homepage."

Providing high availability is crucial.

Data and services need to be replicated.

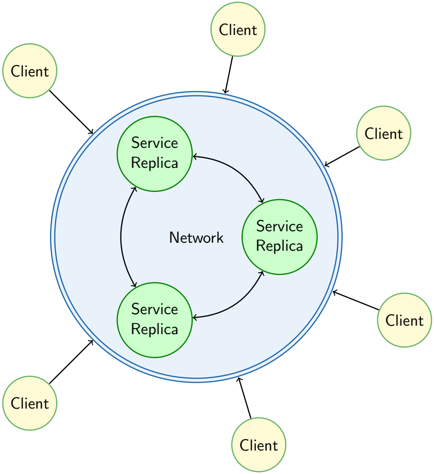
Replication – model

Client processes issue requests to the system.

N service replicas.

Updates disseminated among replicas through asynchronous links.

Strong consistency for all requests.



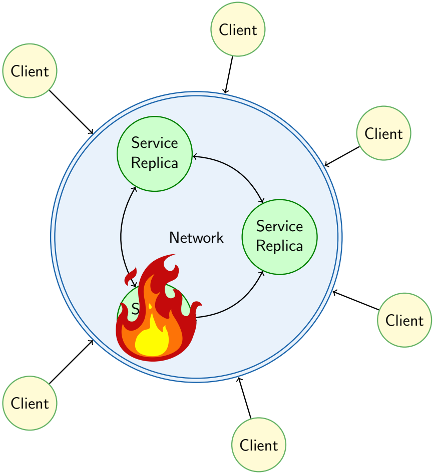
Replication – model

Client processes issue requests to the system.

N service replicas.

Updates disseminated among replicas through asynchronous links.

Strong consistency for all requests.



Transactional replication

Requests treated as transactions.

Transactions:

- all-or-nothing semantics,
- execution in isolation,
- arbitrary code.

Inconsistencies in uncommitted transactions are dangerous!

```
atomic {
  float amount = 100;
  if (accountA.balance() >= amount)
    accountA.withdraw(amount);
  else
    retry;
  accountB.deposit(amount);
}
```

Basic approaches to replication

State Machine Replication:

1. request reception,
2. replica coordination,
3. request execution,
4. response dispatch.

No transactional support.

Deferred Update Replication:

1. request reception,
2. request execution,
3. replica coordination,
4. response dispatch.

Transactional support.

Basic approaches to replication

State Machine Replication:

1. request reception,
2. replica coordination,
3. request execution,
4. response dispatch.

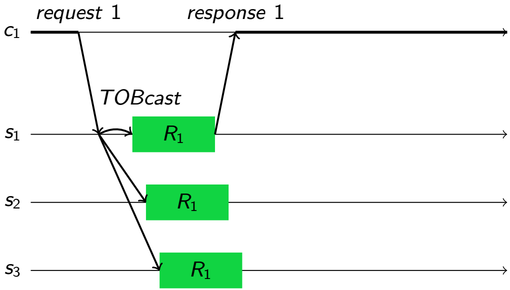
No transactional support.

Deferred Update Replication:

1. request reception,
2. request execution,
3. replica coordination,
4. response dispatch.

Transactional support.

State Machine Replication (SMR)

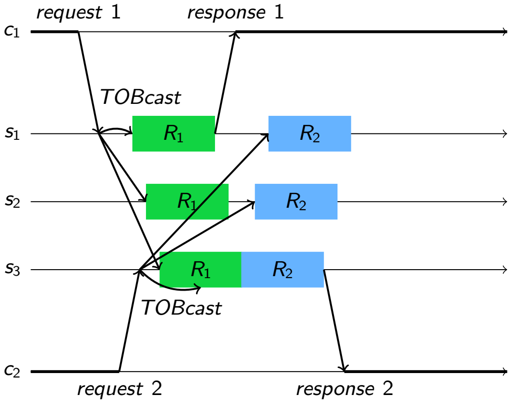


Total Order Broadcast (TOBcast) used to disseminate a request.

Every replica executes the request independently.

Requests need to be deterministic!

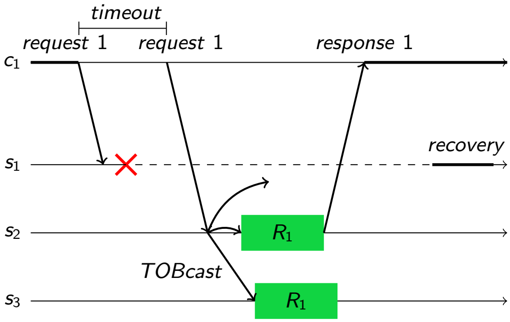
SMR – concurrently submitted requests



Requests execute sequentially.

TOB guarantees that replicas change state in the same way.

SMR – crash



Crashes are tolerated.

Typically majority of nodes has to be up → efficient implementation of TOBcast.

Client might have to reissue the request.

Basic approaches to replication

State Machine Replication:

1. request reception,
2. replica coordination,
3. request execution,
4. response dispatch.

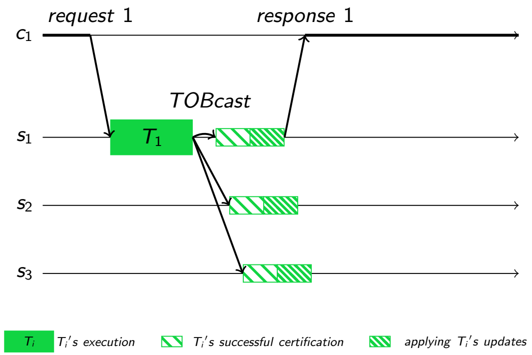
No transactional support.

Deferred Update Replication:

1. request reception,
2. request execution,
3. replica coordination,
4. response dispatch.

Transactional support.

Deferred Update Replication (DUR)

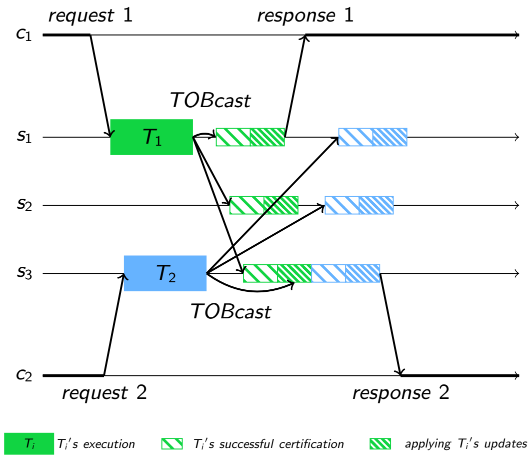


Transactions execute locally and optimistically.

TOB is used for synchronization (other atomic commitment protocols possible).

Certification (conflict detection) is performed independently.

DUR – no conflict

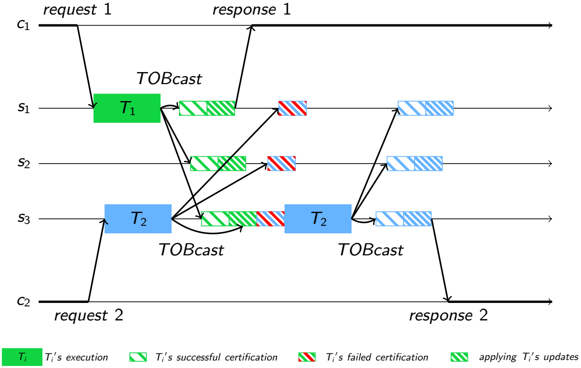


Transactions can execute concurrently.

Certification and applying updates is sequential.

TOB guarantees that replicas change state in the same way.

DUR – conflict (1)



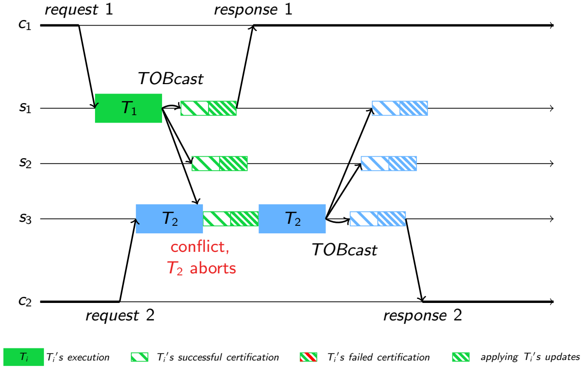
T_1 and T_2:

- concurrent,
- access the same data,
- committing both $\rightarrow$ inconsistency.

Conflict:
$T_1.writeset \cap T_2.readset \neq \emptyset$

T_2 has to be rolled back and restarted.

DUR – conflict (2)



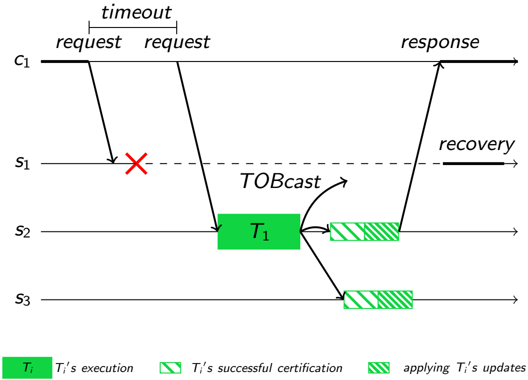
T_1 and T_2:

- concurrent,
- access the same data,
- committing both $\rightarrow$ inconsistency.

Conflict:
$T_1.writeset \cap T_2.readset \neq \emptyset$

T_2 has to be rolled back and restarted.

DUR – crash



Crashes are tolerated.

Typically majority of nodes has to be up.

Replication

State Machine Replication:

- a request is executed by all replicas,
- all requests delivered in the same order by every replica,

Deferred Update Replication:

- a request is executed by a single replica,
- all updates delivered in the same order by every replica,

Replication

State Machine Replication:

- a request is executed by all replicas,
- all requests delivered in the same order by every replica,

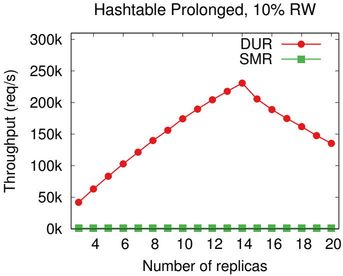
+ easy to implement,
+ performs surprisingly well,
– no parallelism,
– service needs to be deterministic.

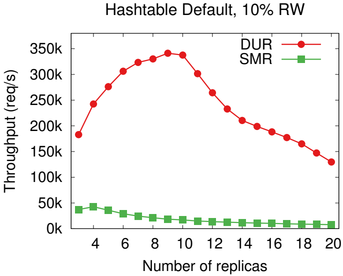
Deferred Update Replication:

- a request is executed by a single replica,
- all updates delivered in the same order by every replica,

+ powerful transactional semantics,
+ scalable,
– suffers under contention,
– suffers when messages are large.

SMR vs DUR – performance (1)



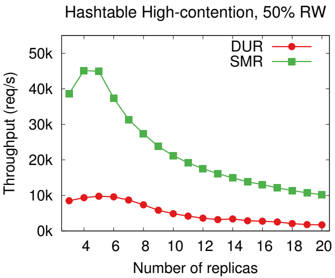


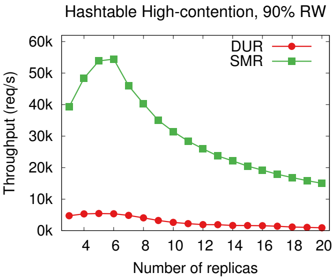
Sometimes gains resulting from parallel execution of requests in DUR are apparent.

They are especially visible in execution dominant workloads → processing power is the limiting factor.

Performance breaks when network gets saturated.

SMR vs DUR – performance (2)





High contention is deadly for DUR.

DUR reexecutes a transaction multiple times before it eventually commits.

Illusive SMR scalability: for low number of replicas, TOB is not running at 100%.

SMR vs DUR – performance

Our comparison[1,2]:

- a lot depends on the workload,
- always exists a dominant factor: TOBcast or request execution time,
- the dominant factor varies across cluster configurations,
- substantial overhead in DUR caused by transactional processing,
- there is no clear winner.

[1] Paweł T. Wojciechowski, Tadeusz Kobus, and Maciej Kokociński. "Model-Driven Comparison of State-Machine-based and Deferred-Update Replication Schemes". In: *Proc. of SRDS '12*. Oct. 2012.

[2] Paweł T. Wojciechowski, Tadeusz Kobus, and Maciej Kokociński. "State-Machine and Deferred-Update Replication: Analysis and Comparison". In: *IEEE Transactions on Parallel and Distributed Systems* 28.3 (2017), pp. 891–904.

SMR vs DUR – semantics

State Machine Replication:

- no transactional support,
- service needs to be deterministic.

Deferred Update Replication:

- transactional constructs: *commit*, *rollback*, *retry*,
- no irrevocable operations (i.e., system calls) inside transactions.

SMR vs DUR – guarantees[3,4] (1)

Informally: what are the *grammar rules* of the replicated system?

State Machine Replication:

- satisfies real-time linearizability,
- all requests appear as if they were executed on a single (reliable) machine.

Deferred Update Replication:

- satisfies update-real-time opacity,
- all updating committed requests appear as if they were executed on a single (reliable) machine,
- read-only and live requests can observe stale (but still consistent) data.

[3] Tadeusz Kobus, Maciej Kokociński, and Paweł T. Wojciechowski. "The Correctness Criterion for Deferred Update Replication". In: *Program of TRANSACT '15*. 2015.

[4] Tadeusz Kobus, Maciej Kokociński, and Paweł T. Wojciechowski. "Relaxing Real-time Order in Opacity and Linearizability". In: *Elsevier Journal on Parallel and Distributed Computing* 100 (2017), pp. 57–70.

SMR vs DUR – guarantees (2)

The satisfied properties are part of the $\diamond$-linearizability and $\diamond$-opacity families of properties:

- base: well known properties:
  - linearizability $\rightarrow$ e.g., concurrent collections in Java,
  - opacity $\rightarrow$ golden property of transactional memory (TM),
- can be used to formalize guarantees of a wide range of (transactional and non-transactional) replicated systems,
- formal relationship between the families.

DUR provides strictly weaker guarantees compared to SMR:

- DUR $\rightarrow$ unlike SMR, no illusion of a single reliable machine for, e.g., read-only requests,
- consequences: programmers have to be more careful when using DUR.

Hybrid Transactional Replication (HTR)[5,6]

Transactional semantics.

Coexisting transaction execution modes:

- Deferred Update → *DU transactions*,
- State Machine → *SM transactions*.

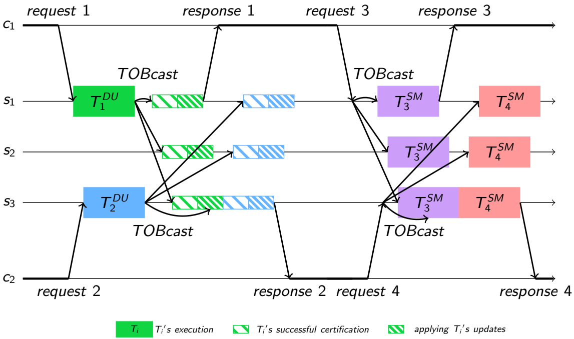
Oracle chooses the execution mode for each transaction's run

- read-only transactions always executed as DU transactions.

[5] Tadeusz Kobus, Maciej Kokociński, and Paweł T. Wojciechowski. "Hybrid Replication: State-Machine-based and Deferred-Update Replication Schemes Combined". In: *Proc. of ICDCS '13*. July 2013.

[6] Tadeusz Kobus, Maciej Kokociński, and Paweł T. Wojciechowski. "Hybrid Transactional Replication: State-Machine and Deferred-Update Replication Combined". Under review, ArXiv preprint: arXiv:1612.06302 [cs.DC]. 2017.

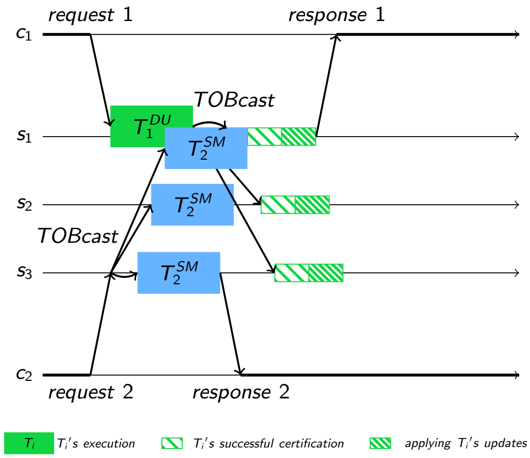
HTR – trivial case



DU transactions execute fully concurrently.

One SM transaction at a time; guaranteed to commit.

HTR – interesting case



SM transaction executes concurrently with DU transactions.

SM transaction never conflicts.

DU transactions' certification might be delayed.

HTR – semantics and guarantees

Transactional semantics – constructs: *commit*, *rollback*, *retry* (as in DUR).

DU transaction:

- may abort at any moment,
- non-deterministic operations allowed.

SM transaction:

- guaranteed to commit,
- suitable for executing irrevocable operations,
- sometimes used to boost performance.

HTR satisfies update-real-time opacity (similarly to DUR).

HTR satisfies real-time linearizability (similarly to SMR), when there are only SM transactions.

HTR – the oracle

Decision made on per transaction execution basis.

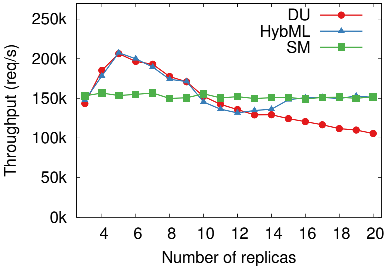
Used parameters:

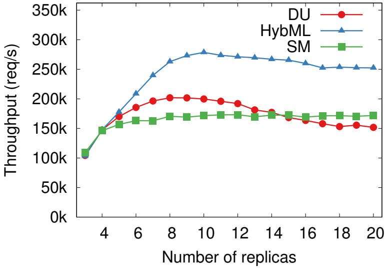
- abort rate – ratio of aborted to all runs,
- duration of TOBcast phase,
- duration of transaction's execution,
- network saturation,
- ...

Oracle policy is tailored to the application and may rely on machine learning techniques → HybML.

Dynamic adaptation to the changing workload.

HTR – performance (1)





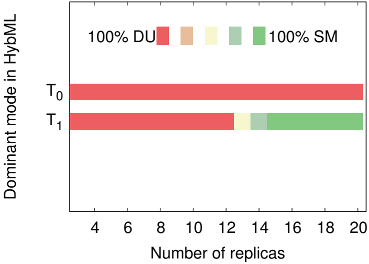
Hashtable benchmark:

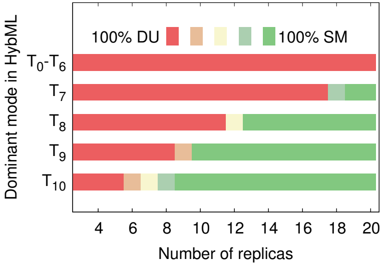
- simple and complex scenarios,
- 3 oracles: SM, DU and HybML.

HybML oracle is at least as good as either SM-only or DU-only oracle (when the difference between SM and DU mode is large enough).

Often HybML gives much better performance!

HTR – performance (2)





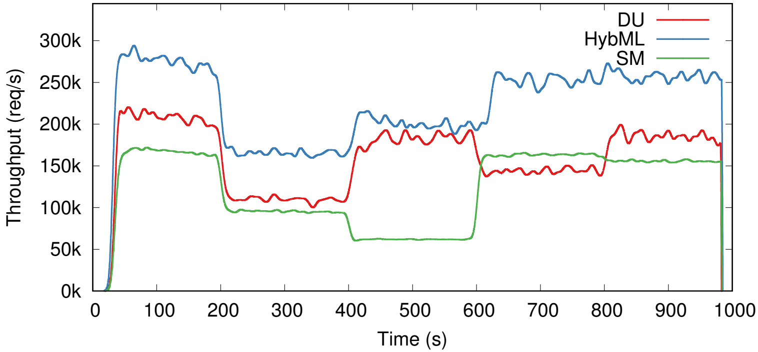
HybML learns in which mode to execute a transactions of a given type.

Implementing by hand an oracle analogous to HybML would be very difficult.

Modelling transaction execution to predict the optimal execution mode would be extremely difficult too:

- very short transactions (from start to commit 50-500μs),
- arbitary code, high-level programming language.

HTR – performance (3)



201–400s - 2x as many reads and writes in updating transactions

401–600s - additional 0.1 ms sleep in updating transactions

601–800s - 2x smaller ranges for updating transactions

HybML quickly adopts to changing conditions.

Experimental evaluation

All tests have been carried on the Eagle cluster at Poznań Supercomputing and Networking Center:

- grant no. 272 *Magazyn danych*, dr hab. inż. Paweł T. Wojciechowski,
- 3 different implementations (SMR, DUR, HTR),
- each implementation tested with dozens of workload types and cluster configurations (3-20 nodes),
- each test: 15-50 min,
- total time: almost 600.000 CPU hours,
- tests on the Infiniband network – network is no longer the bottleneck.

http://www.cs.put.poznan.pl/tkobus