Visualizing Transactional Algorithms for DHTs

Boris Mejías
Université catholique de Louvain
boris.mejias@uclouvain.be

Mikael Högqvist
Zuse Institute Berlin
hoegqvist@zib.de

Peter Van Roy
Université catholique de Louvain
peter.vanroy@uclouvain.be

1. Introduction

Distributed Hash Tables (DHT) provide interesting properties for storing and retrieving data in decentralized systems. They are usually built on top of Structured Overlay Networks (SON) which has self-organizing and fault-tolerant properties. A DHT offers a simple interface to store and lookup elements associated with a key. The operations are basically $put(key, value)$ and $get(key)$. Every peer is responsible for a set of keys and if a peer fails, another peer takes over its responsibility. But what happens with the data of the crashed peer? Either the data must be re-inserted into the system or it can be replicated and recovered on the new responsible node. A replication mechanism must guarantee that the recovered replicated value is the same as the last value stored before the failure.

Data replication becomes more complex when the application running on top of the peer-to-peer network requires the update of several values stored on the DHT at the same time. This is typically done as a *transaction* involving keys belonging to different sets, and hence, involving different peers. How are the different peers coordinated in order to decide if the whole transaction must *commit* or *abort*? How do the replicas of these peers get the last valid data?

The two-phase commit protocol (2PC) is one of the most popular choices for implementing distributed transactions, being used since the 1980s. Unfortunately, its use on peer-to-peer networks is very inefficient because it relies on the survival of the transaction manager, as explained further in section 2. A three-phase commit protocol (3PC) has been designed in order to overcome the limitation of 2PC. However, 3PC introduces an extra round-trip which results in higher latency and increased message load. We advocate the use of an algorithm based on Paxos consensus [4, 2]. This algorithm is especially adapted for the requirements of a DHT and can survive a crash of the coordinator during a transaction. Compared to 3PC, it reduces latency and overall message load by requiring less message round-trips.

Demonstrator

We implement two-phase commit and the Paxos consensus-based algorithm on top of a Chord-like structured overlay network [5], extending the PEPINO network inspector [3] for visualization. By introducing arbitrary failures, the demonstrator shows why two-phase commit does not work on peer-to-peer networks. Then, the robustness of Paxos consensus is tested by injecting failures on a certain amount of transaction managers and participants, showing the failure recovery mechanism of this protocol.

2. Two-phase commit

The pseudo-code below implements a swap operation within a transaction. The objective is that the instructions from the beginning of the transaction (BOT) until its end (EOT) are executed atomically to avoid race conditions with other concurrent operations. The values of $item\_i$ and $item\_j$ are stored on different peers. The operators put and get are replaced by $read$ and $write$ in order to differentiate a regular DHT from a transactional DHT.

```
BOT
    x = read(item_i);
    y = read(item_j);
    write(item_j, x);
    write(item_i, y);
EOT
```

In order to guarantee atomic commit of a transaction on a decentralized storage, two-phase commit uses a *validation* phase and a *write* phase, coordinated by a *transaction manager* (TM). All peers responsible for the items involved in the transaction, as well as their replicas, become *transaction participants* (TP). Initially, the TM sends a request to every TP to *prepare* the transaction. If the item is available, the TP will lock it and acknowledge the *prepare* request. Otherwise, it will reply *abort*. The *write* phase follows *validation* once the replies are collected by the TM. If none of the participants voted *abort*, then the decision will be *commit*. When the participants receive the commit message from the TM, they will make the update permanent and release the lock on the item. An abort message will discard any update and release the item locks.

The problem with the 2PC protocol is that relies too much on the survival of the transaction manager. If the TM fails during the validation phase, it will block all the TPs that acknowledged the prepare message. A very reliable TM is required for this protocol, but it cannot be guaranteed on peer-to-peer networks.

3. Paxos Consensus Algorithm

The 3PC protocol avoids the blocking problem of 2PC at the cost of an extra message round-trip. This solution might be acceptable for cluster-based applications but not for peer-to-peer networks, where it is better to have less rounds with more messages than adding extra rounds to the protocol. This problem lead to the recent introduction of [4] based on Paxos consensus [2].

The idea is to add replicated transaction managers (rTM) that can take over the responsibility of the TM in case of failure. The other advantage is that decisions can be made considering a majority of the participants reaching consensus, and therefore, not all participants needs to be alive or reachable to commit the transaction. This means that as long as the majority of participants survives, the algorithm terminates even in presence of failures of the TM and TPs, without blocking the involved items.

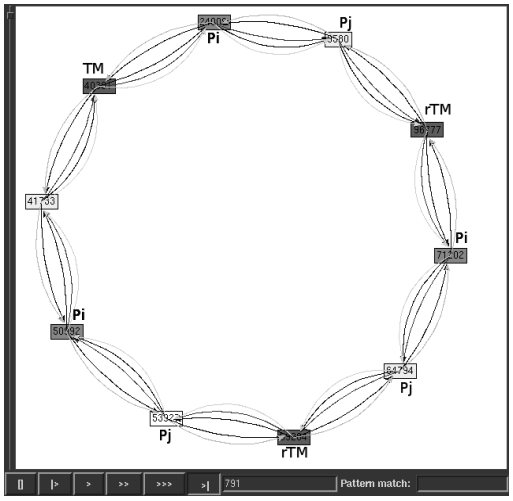

Figure 1. Network during a transaction with replicated manager and participants.

Figure 1 depicts a network visualized by the demonstrator. Different colours are assigned to TMs and TPs depending on the item they are responsible for. The labels in the figure are not in the simulator, but where added here for clarity. The TMs and TPs in the protocol are replicated using symmetric replication as described in [1]. Figure 2 shows the initial effect of introducing an arbitrary failure on the transaction manager, breaking the connection of the ring. The demonstrator continues with the recovery of the ring, and the election of a new TM from the rTMs.

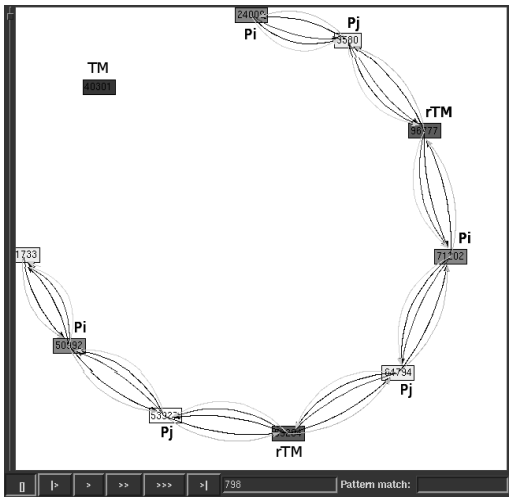

Figure 2. Failure of the transaction manager

4. Summary

The focus of this demonstrator is on the study of algorithms for implementing transactions on peer-to-peer networks. Their visualization contributes to the analysis and test of the protocols, verifying their tolerance to failures. In particular, we show a DHT running two-phase commit and the Paxos consensus algorithm.

5. Acknowledgements

The authors would like to thank Monika Moser and Seif Haridi for their help on the understanding of the Paxos consensus algorithm. This work is mainly funded by project SELFMAN (contract number: 034084), with additional funding by CoreGRID (contract number: 004265).

References

[1] A. Ghodsi. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. PhD dissertation, KTH — Royal Institute of Technology, Stockholm, Sweden, Dec. 2006.

[2] J. Gray and L. Lamport. Consensus on transaction commit. *ACM Trans. Database Syst.*, 31(1):133–160, 2006.

[3] D. Grolaux, B. Mejías, and P. Van Roy. PEPINO: PEer-to-Peer network INspectOr. In *The Seventh IEEE International Conference on Peer-to-Peer Computing*, 2007.

[4] M. Moser and S. Haridi. Atomic commitment in transactional DHTs. In *Proceedings of the CoreGRID Symposium*, 2007.

[5] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.