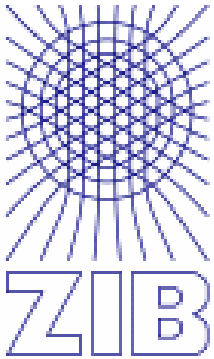
Transactions

D 3.1a First report on formal models for transactions over structured overlay networks



Taks description

- **T3.1 Formal models for transactions over a structured overlay network (ZIB:3, UCL:3, FT R&D:2):** This task will investigate how to do transactions over a structured overlay network. These will likely be different from classical transactions. One of the important research questions is to resolve the tension between the distributed system and the needs of the application. We will design different compromises and see which are appropriate for applications.

- **D3.1a** First report on formal models for transactions over structured overlay networks.

- **M12:** Understand how to do transactions over structured overlay networks

Results

- Framework for DHTs that provide strong data consistency
  - DHT with replication:
    - Operations on items include a majority of nodes being responsible for a replica
- **Transactions:**
  - Atomic commit: Adaption of Paxos Atomic Commit
    - Does not rely on a perfect failure detector
    - Non-blocking, tolerates a number of failures among nodes
    - Low number of communication rounds, low number of messages
    - Exploit symmetric replication technique to define set of nodes being responsible for the transaction management

Results

- **Transactions:**
  - Concurrency control:
    - Simple scheme, might be sufficient for a number of applications
    - Items get a version number
    - Transaction logic is executed in the TMs private workspace
    - Read and writes retrieve the items version number
    - At the end of the transaction, the commit protocol starts with the validation phase (prepare request)
  - Concurrency control dependent on application
  - Fits for the distributed wiki scenario

Results

- Stefan Plantikow: Diploma thesis, and paper submitted
- Different concurrency control models:
  - Optimization for long running transactions
  - Optimization for read only transactions
- Be able to adapt to the application requirements
- Cell model for replication and consistency
  - Different nodes form a cell
  - Constitute a replicated state machine

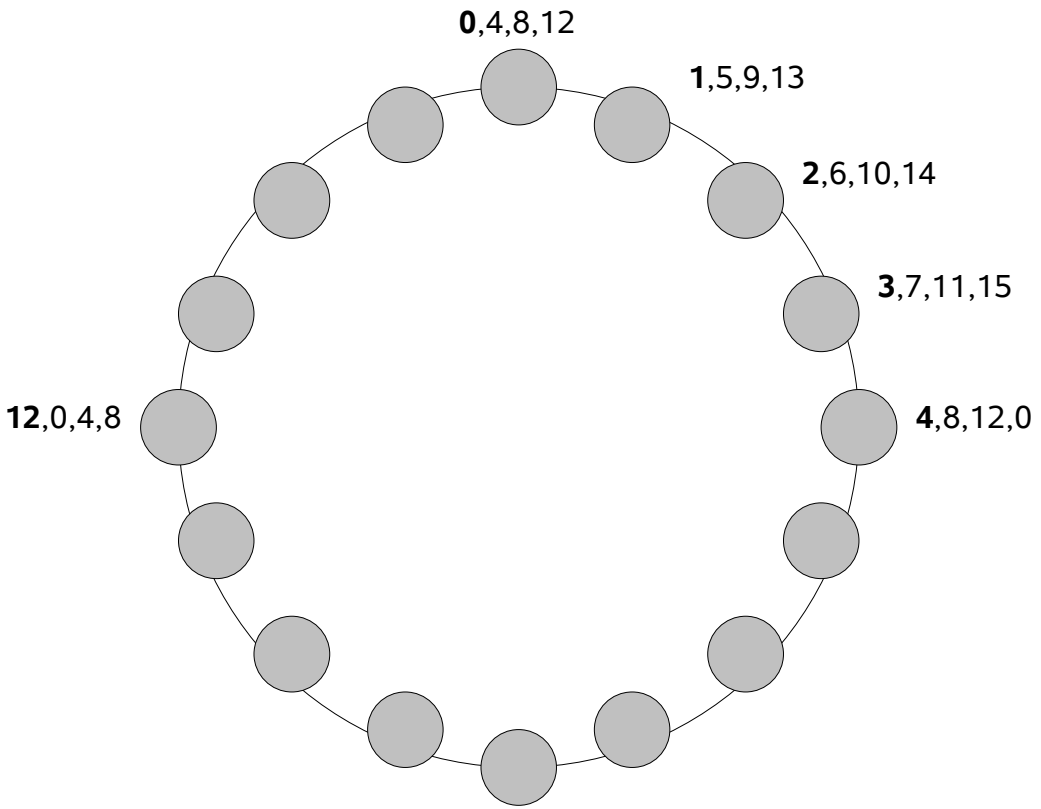
Summary

- What has been done:
  - Outline of the design of a transactional DHT
    - Atomic commit, which fits into a internet-based scenario
    - Concurrency control, simple but maybe sufficient
  - Different concurrency control schemes
- What has not been done:
  - Formalize all algorithms
  - Evaluation of the transactional DHT
  - Analyze needs of different applications (only Wiki scenario done)
  - More different models for transactions

Framework

- DHT with replicated items
  - Provide strong consistency
  - Supports transactions
- System
  - Items are replicated according to the DKS symmetric replication scheme
  - Failure model: crash-stop, nodes that rejoin are treated as new nodes
  - Imperfect failure detector, system is Internet-based

Symmetric Replication



- Identifier ring with size of ID space N=16, replication factor f=4

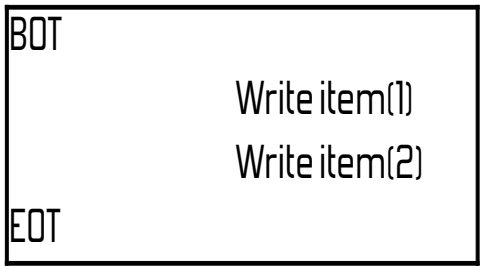
$$r(i,x)=i+(x-1)\frac{N}{f}, x=1..\mathrm{f}$$

Consistency of Replicas

- Main idea: Include a majority of replicas in all operations, where the majority is dependent on the replication factor
- Each operation maintains the invariant that a majority of replicas contains the latest version of an items
  - Data access: Read/Write need a majority of replicas or nodes responsible for the item
  - Maintenance of replication degree (handling of node failure): Restore replication degree by reading from a majority
  - Node join/leave won't violate the invariant

Transactions in a DHT

- Distributed Transactions on data stored in DHT, 2 levels of distribution:
  - Items are distributed
  - Items are replicated
- Main issues:
  - Isolate concurrent transactions from each other (Isolation)
  - Either all operations take place or none of them (Atomicity)
- Example:



Transaction Processing

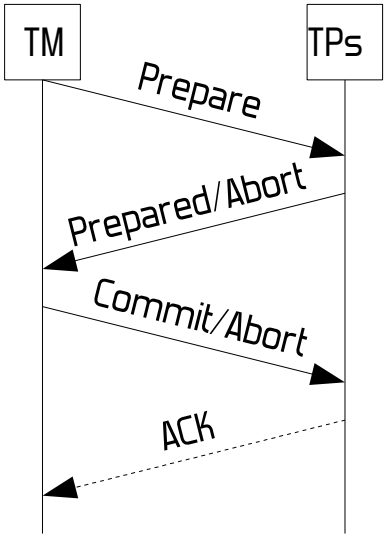
- One node acting as the transaction manager (TM), the one the client/application is issuing its requests from
- Processing of the transaction:
  - Client determines the operations which are part of the transaction
  - The TM will execute the transaction in its private workspace
    - Read remotely
    - For write operations, the current version number of the item is read remotely
  - At the end of the transaction the operations are distributed within the validation phase

Concurrency Control

- Prevent concurrent transactions from interfering with each other
- Simple scheme
  - Items are attached with a version number
  - During validation it is checked whether the versions of the items read are still valid
  - For write operations the proposed version number is checked to be larger than the current one
- Simple, but maybe sufficient
- Could end up in many aborts, dependent on application (or starvation)

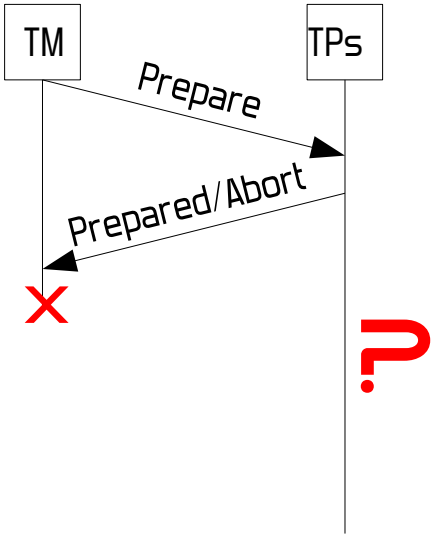
Atomic Commit in Distributed Systems

- Roles:
  - Transaction Manager (TM): coordinates transaction
  - Transaction Participant (TP): node responsible for an item involved in the transaction
- Most common commit protocol: 2-Phase-Commit



Atomic Commit in Distributed Systems

- Roles:
  - Transaction Manager (TM): coordinates transaction
  - Transaction Participant (TP): node responsible for an item involved in the transaction
- Most common commit protocol: 2-Phase-Commit



Blocking!

PAXOS Commit

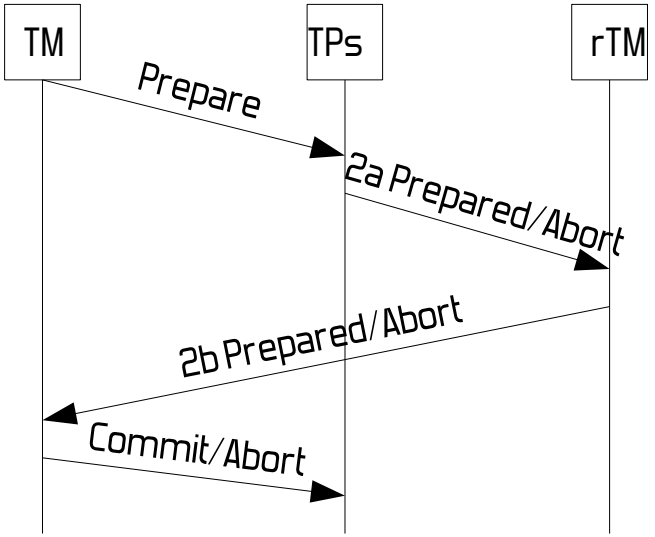
- Properties:
  - Non-blocking:
    - Uses a number of acceptors
    - Decision is recorded in acceptors' stable storage
  - Tolerates *F* failures among *2F+1* acceptors
  - TPs do not need to know each other, less overhead in information they have to maintain
  - No perfect failure detector needed (especially important in Internet based systems), algorithm can handle false suspections
  - Leader election mechanism needed for progress

Adapted PAXOS Commit

- Participants and Roles:
  - Transaction Manager (TM) + replicated TMs (rTMs) which act as acceptors
  - Transaction Participant (TP) act as proposer
- Each TP will run a separate instance of Paxos consensus together with the set of acceptors
- Each instance decides on the TP's decision, if the TP does not fail, otherwise it will decide on abort
- Each rTMs will have the outcome of all the consensus instances

Adapted PAXOS Commit

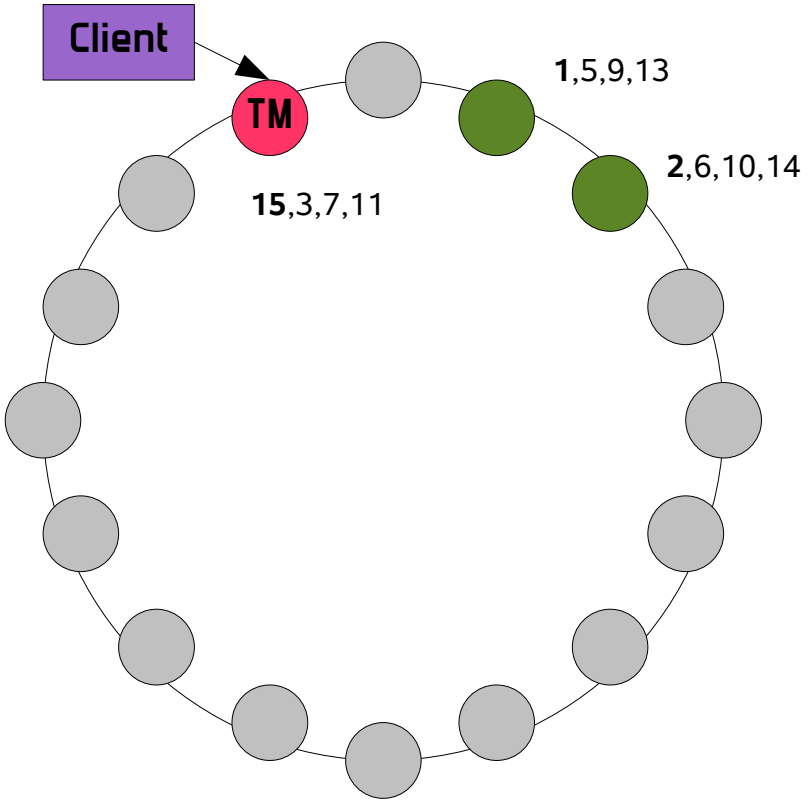
- Each TM decides independently on the outcome of the transaction. It will be
  - Commit, if for each item involved a majority of TPs storing a replica will vote prepared
  - Abort, if at least one of the items a majority votes to abort



Adapted PAXOS Commit

- Nodes involved in the transaction have to be known
- The responsibilities of the nodes will be fixed during the commit protocol
- Before starting the commit protocol, use bulk operation:
  - Initialize replicated TMs
  - Lookup nodes involved
  - Tell involved nodes to lock responsibilities
- With the prepare message:
  - Tell TPs which nodes act as rTMs

Atomic Commit in a DHT

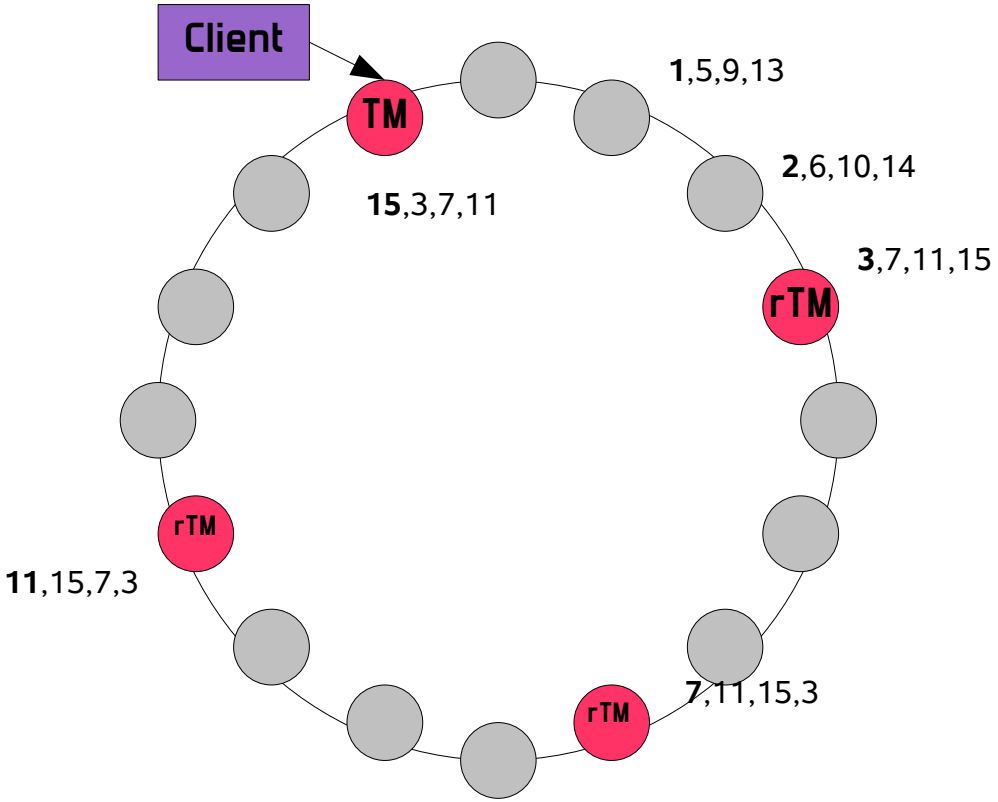


Client:

```
BOT
        Write item(1)
        Write item(2)
EOT
```

- Node 15 becomes the Transaction Manager (TM)
- TM creates a transaction item with a key for which it is responsible for (e.g. key = 15)

Atomic Commit in a DHT

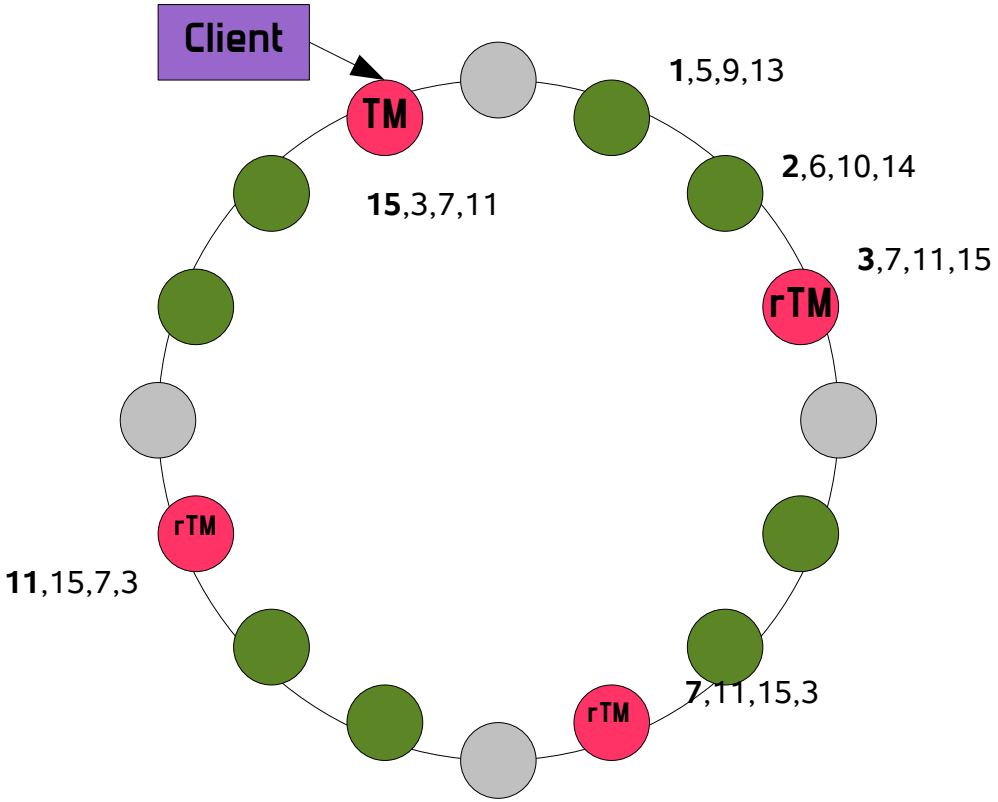


Client:

```
BOT
        Write item(1)
        Write item(2)
EOT
```

- Node 15 becomes the Transaction Manager (TM)
- Nodes 3, 7, 11 become replicated Transaction Managers (rTM), according to the replication of the transaction item

Atomic Commit in a DHT

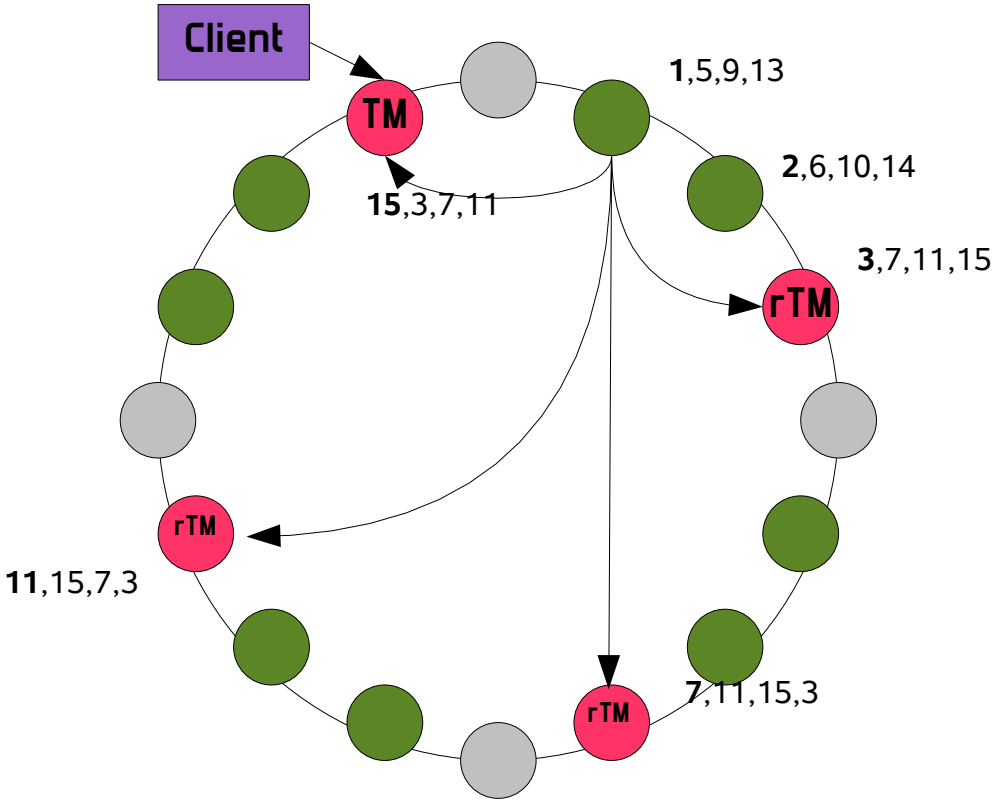


Client:

```
BOT
        Write item(1)
        Write item(2)
EOT
```

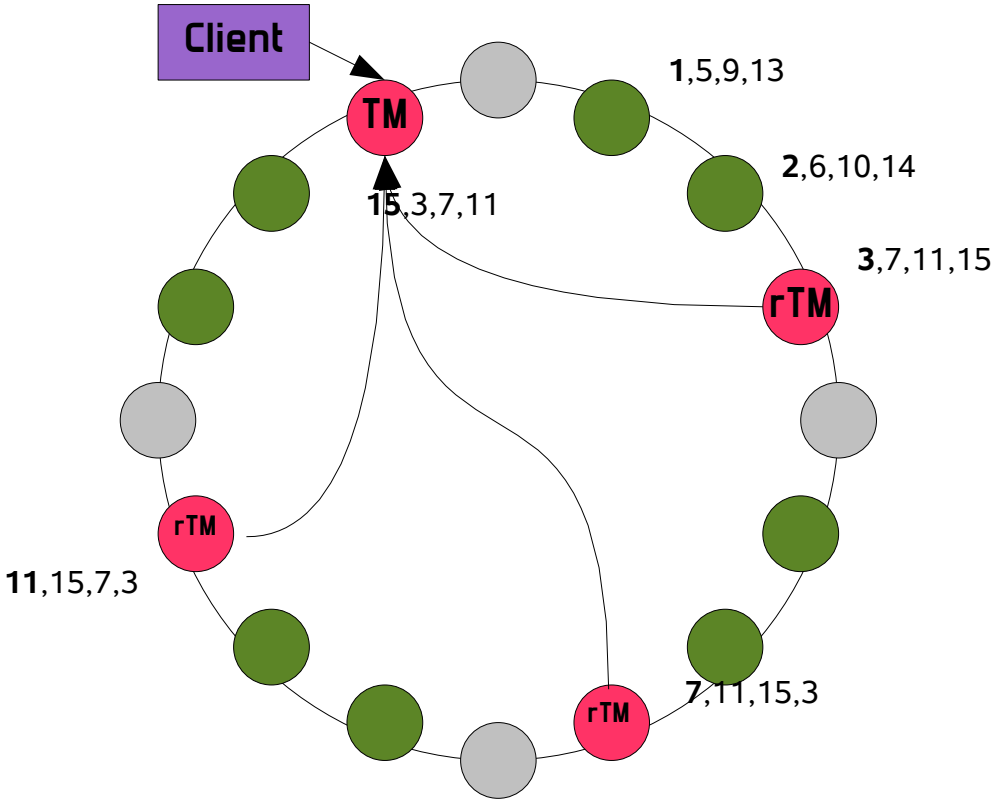
- Node 15 becomes the Transaction Manager (TM)
- Nodes 3, 7, 11 become replicated Transaction Managers (rTM)
- Nodes 1,2,5,6,9,10,13,14 become Transaction Participants(TP)

Atomic Commit in a DHT



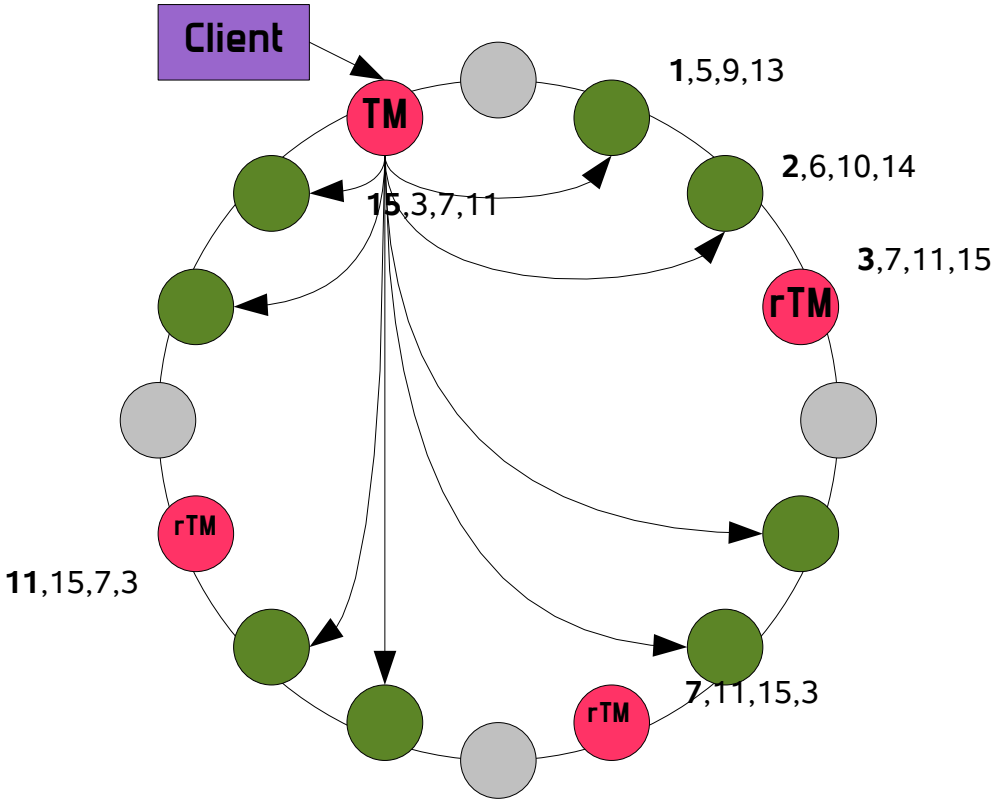
- After having received "Prepare" from the TM, each TP sends it's Prepared message to all rTMs, if it can successfully validate the operation on its item (check version of an item)

Atomic Commit in a DHT



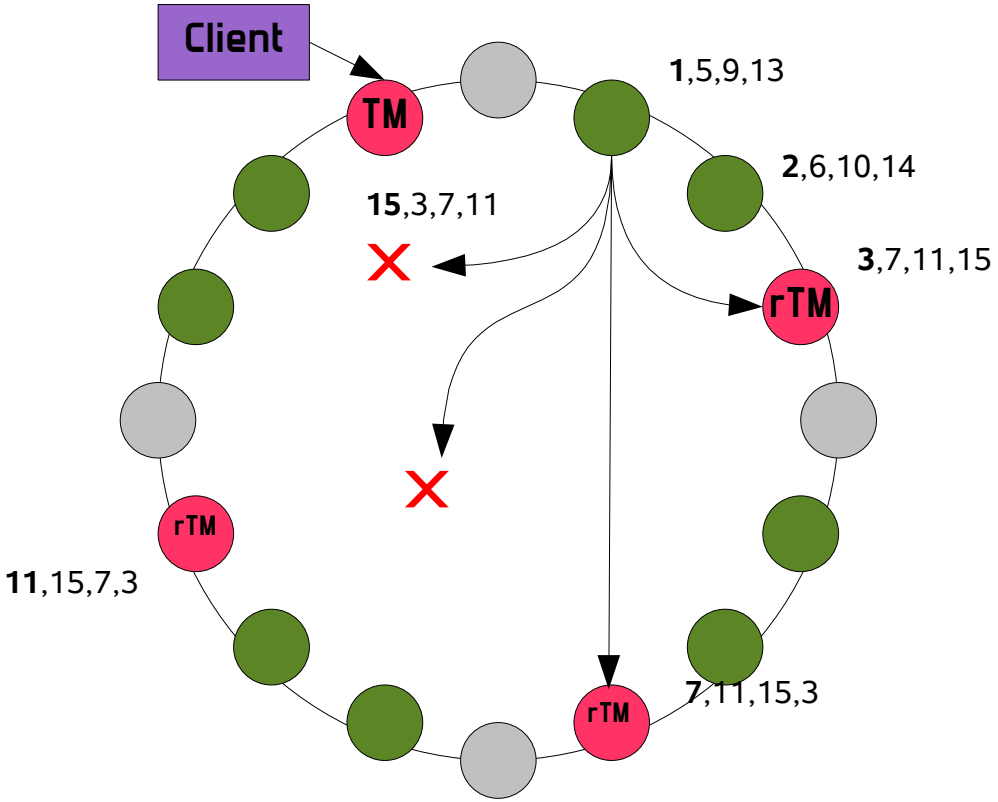
- Each TP sends it's Prepared message to all rTMs
- Each rTM sends the 2b Prepared message to the leading TM

Atomic Commit in a DHT



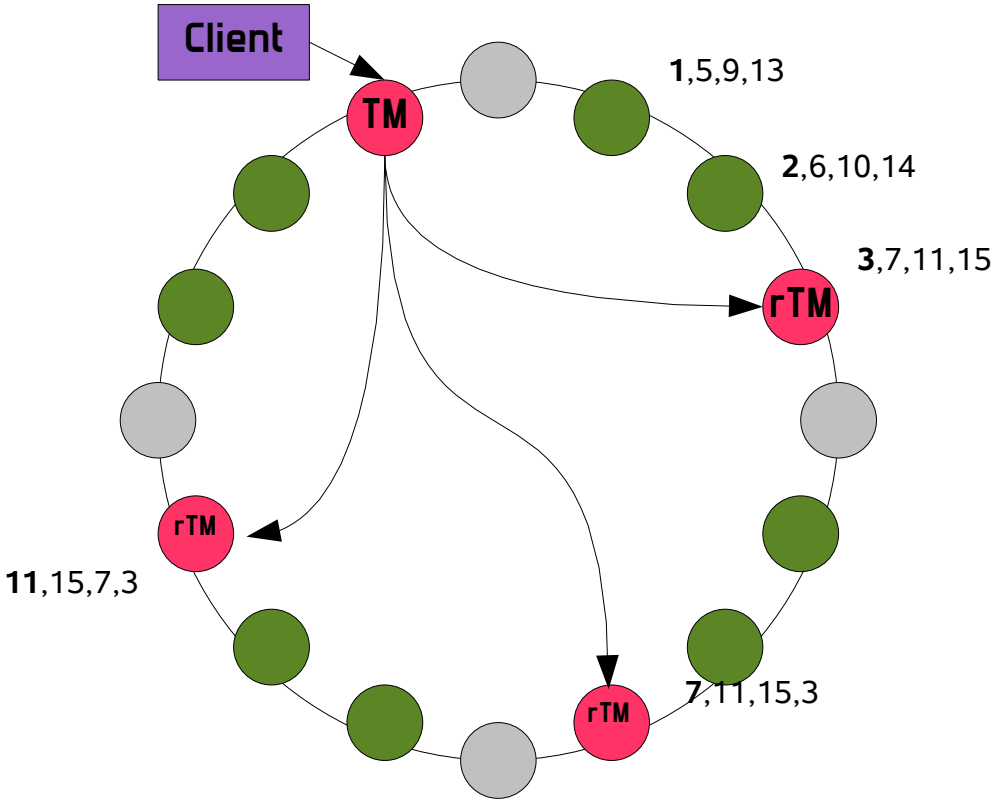
- Each TP sends it's Prepared message to all rTMs
- Each rTM sends the 2b Prepared message to the leading TM
- TM sends the decision to all TPs

False Suspection



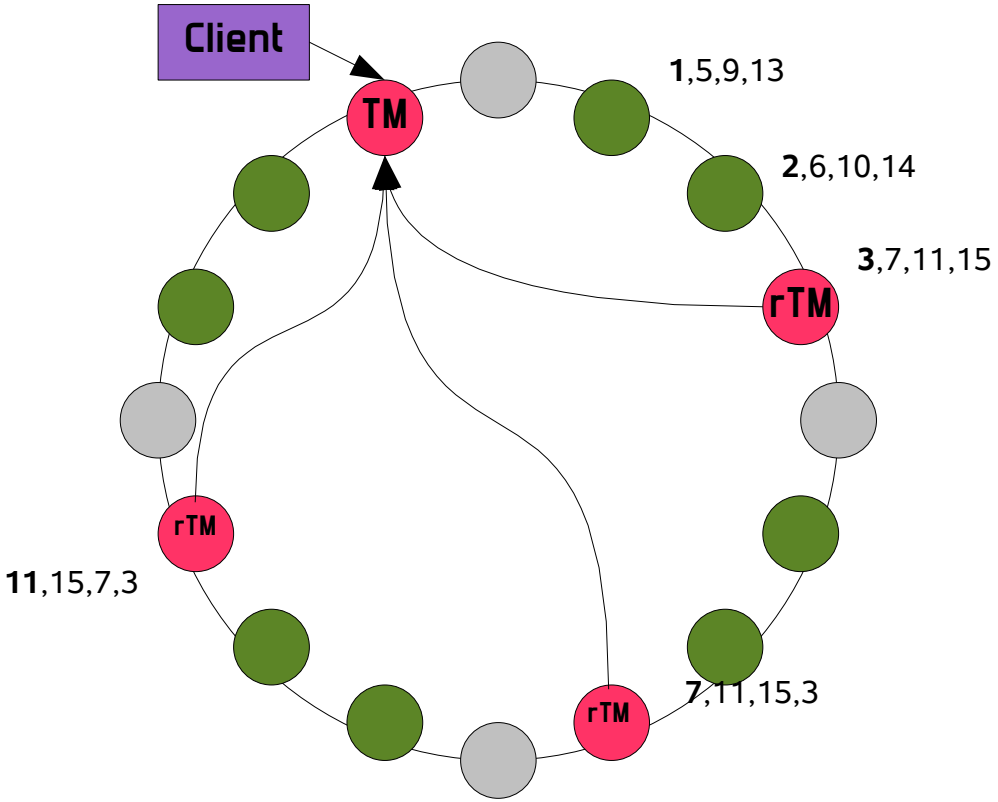
- **Failure Handling:**
- **Node 1 suspected**
-

False Suspection



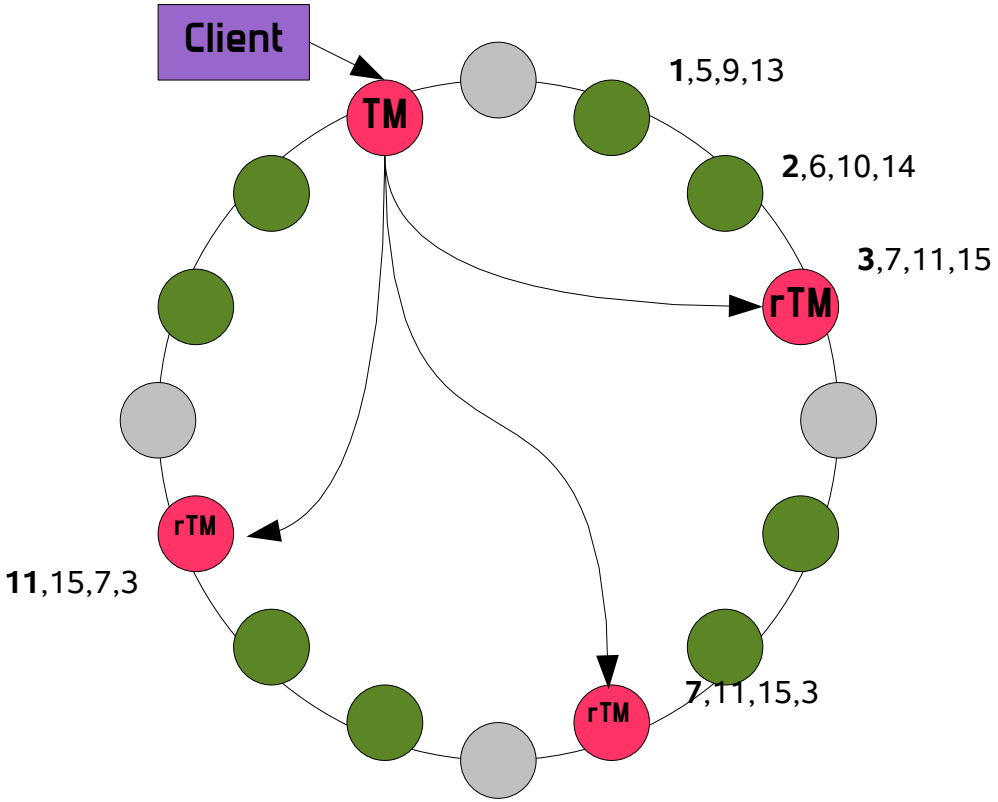
- **Failure Handling:**
- Node 1 suspected
- TM tries to vote in Node1's instance of Paxos to abort,
- By sending a Phase 1a message to all rTMs
- With ballot = 1

False Suspection



- **Failure Handling:**
- Node 1 suspected
- TM tries to vote in Node1's instance of Paxos to abort, by sending a Phase 1a message to all rTMs, with ballot = 1
- Responses:
  - rTM(3): 1b (1, commit)
  - rTM(7): 1b (1, commit)
  - rTM(11): 1b (1, -)

False Suspection



- **Failure Handling:**
- Node 1 suspected
- TM tries to vote in Node1's instance of Paxos to abort, by sending a Phase 1a message to all rTMs, with ballot = 1
- Responses:
  - rTM(3): 1b (1, prepared)
  - rTM(7): 1b (1, prepared)
  - rTM(11): 1b (1, -)
- TM: 2a message (1, prepared)

Costs of the Protocol

- Costs of atomic commit without DHT operations, Failure-free

  I: number of Items involved, f:replication factor

  - Number of messages:

Step	Paxos Commit	2PC
Prepare	I*f	I*f
2a Prepared	I*f*f	I*f
2b Prepared	f-1	
Commit	I*f	I*f

  - "2b Prepared" is optimized: all votes received by a rTM are bundled and sent to the TM

Costs of the Protocol

- Costs of atomic commit without DHT operations

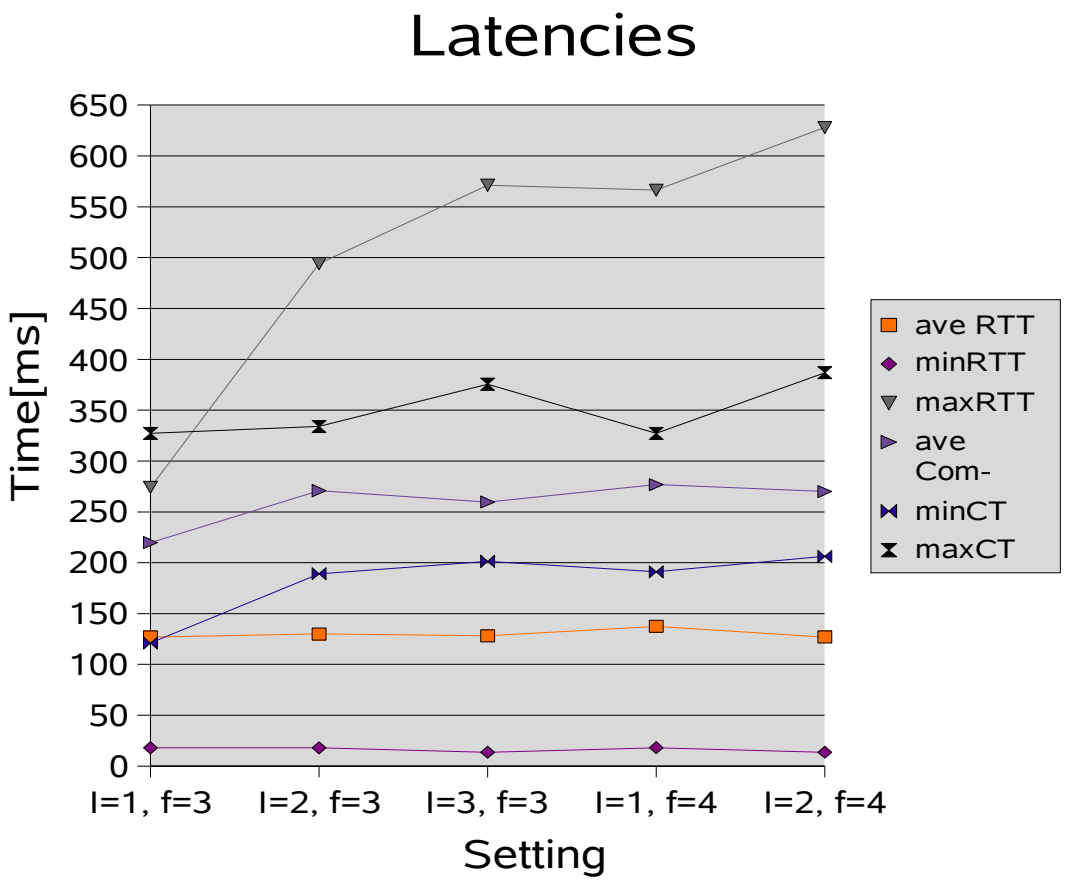
  Duration:

Step	Delay
Prepare	max(TM -> TP)
2a Prepared	max(fastest Majority(TP -> rTM))
2b Prepared	max(fastest Majority(rTM->TM))
Commit	TM->TP

  Decision can be distributed after having received “2b Prepared” from a majority of rTMs.

  - Further possible optimizations: Send 2b Prepared to TPs directly

Delays on PlanetLab



- Simulation of the failure free case (without sending decision)
- Hosts are distributed worldwide (China, USA, Canada, Norway, Switzerland ...)