# DXRAM
# Log + Recovery

Florian Klein

Institut für Informatik
Abteilung Betriebssysteme
Heinrich-Heine-Universität Düsseldorf

11.12.2012

# Outline

1. Data-Structure
2. Log
3. Recovery
4. Conclusion

# Data-Structure

## ChunkInformation

| *Field* | *Type* | *Description* |
|---|---|---|
| `m_chunkID` | `ID` | unique Chunk-Identifier |
| `m_version` | `long` | version of the Chunk |
| `m_flags` | `int` | flags (present, foreign) |
| `m_nodeID` | `NodeID` | unique identifier of the hosting Node |

## Chunk extends ChunkInformation

| *Field* | *Type* | *Description* |
|---|---|---|
| `m_data` | `ByteBuffer` | binary data |

# Outline

1 Data-Structure

2 Log
- Current Log-Layout
- Update process

3 Recovery

4 Conclusion

# Types

## Block

- stores a Chunk
- default 1 MB

## Segment

- merges a number of Chunks
- backup entity
- default 64 MB

# Meta-Data I

## Segment-Header

- Segment-Index (4 Byte)
- Flags (4 Byte)
  - empty
  - full
  - foreign
  - number of free Blocks
- Bitmap (1 Bit per Block in the Segment)

## Bitmap (64 MB Segments and 1 MB Blocks)

⇒ 64 Blocks per Segment ⇒ 8 Byte Bitmap

## Bitmap (64 MB Segments and 1 KB Blocks)

⇒ 65536 Blocks per Segment ⇒ 8192 Byte Bitmap

# Meta-Data II

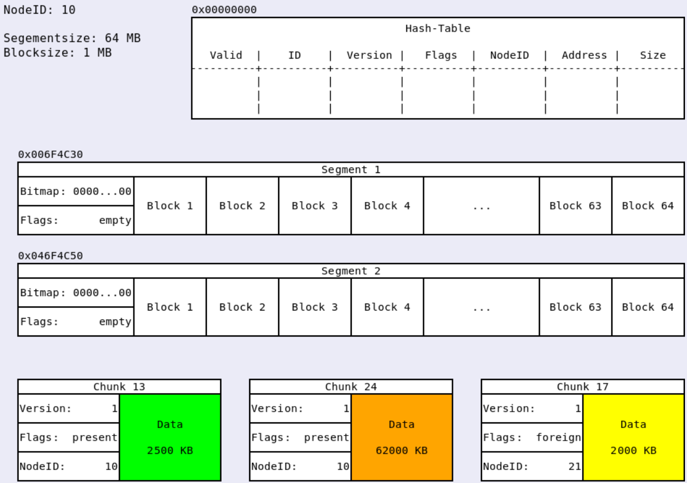
## Log-Header

Hash-Table with one entry per Block

| *Bytes* | *Content* | *Type* | |
|---|---|---|---|
| 1 | valid-flag | `boolean` | |
| 4 | ID length | `ID` | `ChunkInformation` |
| 20 | ID bytes | | |
| 8 | version | `long` | |
| 4 | flags | `int` | |
| 4 | host length | `NodeID` | |
| 15 | host | | |
| 2 | port | | |
| 8 | address | `long` | `StorageInformation` |
| 4 | size | `int` | |
| 70 | | | |

# Layout

## Initial situation



NodeID: 10

Segementsize: 64 MB
Blocksize: 1 MB

0x00000000

Hash-Table

| Valid | ID | Version | Flags | NodeID | Address | Size |
|---|---|---|---|---|---|---|
| | | | | | | |

0x006F4C30

| Segment 1 | | | | | | | |
|---|---|---|---|---|---|---|---|
| Bitmap: 0000...00<br>Flags: empty | Block 1 | Block 2 | Block 3 | Block 4 | ... | Block 63 | Block 64 |

0x046F4C50

| Segment 2 | | | | | | | |
|---|---|---|---|---|---|---|---|
| Bitmap: 0000...00<br>Flags: empty | Block 1 | Block 2 | Block 3 | Block 4 | ... | Block 63 | Block 64 |

| Chunk 13 | |
|---|---|
| Version: 1 | Data<br>2500 KB |
| Flags: present | |
| NodeID: 10 | |

| Chunk 24 | |
|---|---|
| Version: 1 | Data<br>62000 KB |
| Flags: present | |
| NodeID: 10 | |

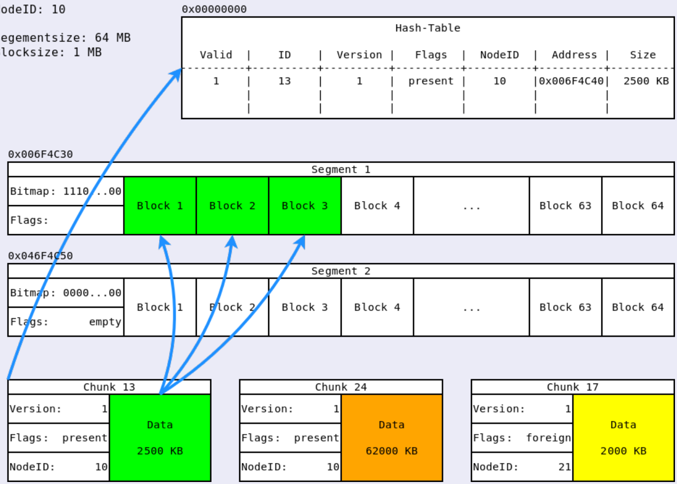
| Chunk 17 | |
|---|---|
| Version: 1 | Data<br>2000 KB |
| Flags: foreign | |
| NodeID: 21 | |

# Layout

## Inserting Chunk 13

NodeID: 10

Segementsize: 64 MB
Blocksize: 1 MB



0x00000000

Hash-Table

| Valid | ID | Version | Flags | NodeID | Address | Size |
|---|---|---|---|---|---|---|
| 1 | 13 | 1 | present | 10 | 0x006F4C40 | 2500 KB |

0x006F4C30

Segment 1

| Bitmap: 1110...00<br>Flags: | Block 1 | Block 2 | Block 3 | Block 4 | ... | Block 63 | Block 64 |
|---|---|---|---|---|---|---|---|

0x046F4C50

Segment 2

| Bitmap: 0000...00<br>Flags: empty | Block 1 | Block 2 | Block 3 | Block 4 | ... | Block 63 | Block 64 |
|---|---|---|---|---|---|---|---|

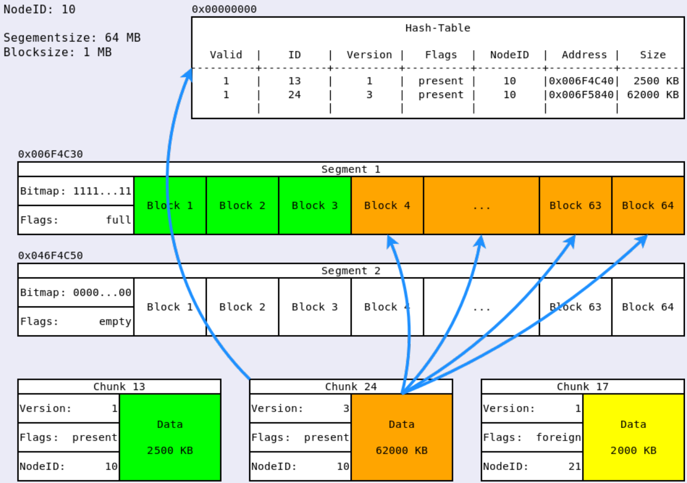
Chunk 13
Version: 1
Flags: present
NodeID: 10
Data
2500 KB

Chunk 24
Version: 1
Flags: present
NodeID: 10
Data
62000 KB

Chunk 17
Version: 1
Flags: foreign
NodeID: 21
Data
2000 KB

# Layout

## Inserting Chunk 24



NodeID: 10

Segementsize: 64 MB
Blocksize: 1 MB

0x00000000

Hash-Table

| Valid | ID | Version | Flags | NodeID | Address | Size |
|---|---|---|---|---|---|---|
| 1 | 13 | 1 | present | 10 | 0x006F4C40 | 2500 KB |
| 1 | 24 | 3 | present | 10 | 0x006F5840 | 62000 KB |

0x006F4C30

Segment 1

| Bitmap: 1111...11<br>Flags: full | Block 1 | Block 2 | Block 3 | Block 4 | ... | Block 63 | Block 64 |
|---|---|---|---|---|---|---|---|

0x046F4C50

Segment 2

| Bitmap: 0000...00<br>Flags: empty | Block 1 | Block 2 | Block 3 | Block 4 | ... | Block 63 | Block 64 |
|---|---|---|---|---|---|---|---|

Chunk 13

Version: 1
Flags: present
NodeID: 10
Data
2500 KB

Chunk 24

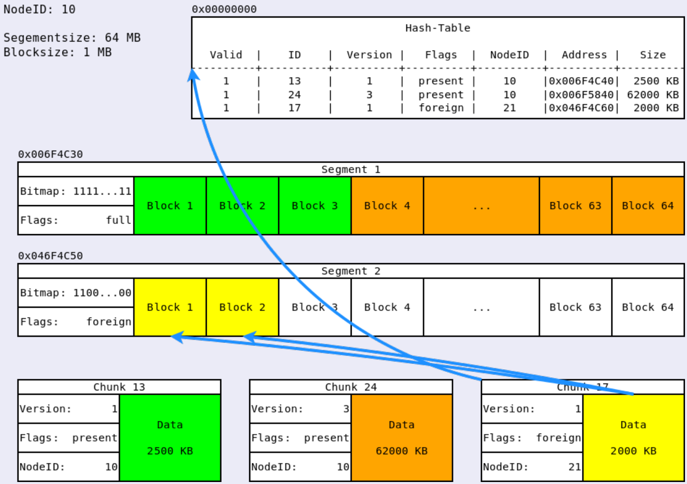
Version: 3
Flags: present
NodeID: 10
Data
62000 KB

Chunk 17

Version: 1
Flags: foreign
NodeID: 21
Data
2000 KB

# Layout

## Inserting Chunk 17



NodeID: 10

Segementsize: 64 MB
Blocksize: 1 MB

0x00000000

```
                              Hash-Table

 Valid   |    ID    |  Version |   Flags   |  NodeID  |  Address |   Size
---------+----------+----------+-----------+----------+----------+---------
   1     |    13    |    1     |  present  |    10    |0x006F4C40|  2500 KB
   1     |    24    |    3     |  present  |    10    |0x006F5840| 62000 KB
   1     |    17    |    1     |  foreign  |    21    |0x046F4C60|  2000 KB
```

0x006F4C30

| Segment 1 | | | | | | | |
|---|---|---|---|---|---|---|---|
| Bitmap: 1111...11 | Block 1 | Block 2 | Block 3 | Block 4 | ... | Block 63 | Block 64 |
| Flags: full | | | | | | | |

0x046F4C50

| Segment 2 | | | | | | | |
|---|---|---|---|---|---|---|---|
| Bitmap: 1100...00 | Block 1 | Block 2 | Block 3 | Block 4 | ... | Block 63 | Block 64 |
| Flags: foreign | | | | | | | |

| Chunk 13 | |
|---|---|
| Version: 1 | Data<br>2500 KB |
| Flags: present | |
| NodeID: 10 | |

| Chunk 24 | |
|---|---|
| Version: 3 | Data<br>62000 KB |
| Flags: present | |
| NodeID: 10 | |

| Chunk 17 | |
|---|---|
| Version: 1 | Data<br>2000 KB |
| Flags: foreign | |
| NodeID: 21 | |

# Layout

## Final situation

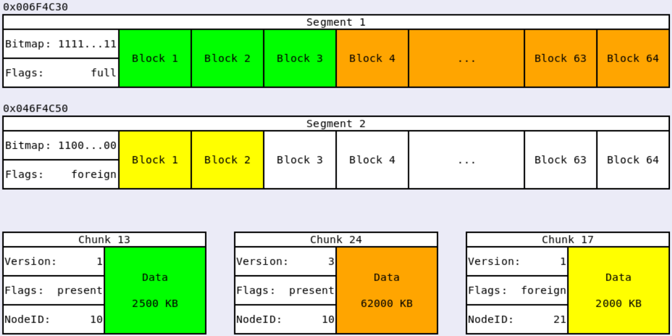
NodeID: 10

Segementsize: 64 MB
Blocksize: 1 MB

0x00000000

Hash-Table

| Valid | ID | Version | Flags | NodeID | Address | Size |
|---|---|---|---|---|---|---|
| 1 | 13 | 1 | present | 10 | 0x006F4C40 | 2500 KB |
| 1 | 24 | 3 | present | 10 | 0x006F5840 | 62000 KB |
| 1 | 17 | 1 | foreign | 21 | 0x046F4C60 | 2000 KB |



0x006F4C30

Segment 1

Bitmap: 1111...11
Flags: full
Block 1 | Block 2 | Block 3 | Block 4 | ... | Block 63 | Block 64

0x046F4C50

Segment 2

Bitmap: 1100...00
Flags: foreign
Block 1 | Block 2 | Block 3 | Block 4 | ... | Block 63 | Block 64

Chunk 13
Version: 1
Flags: present
NodeID: 10
Data 2500 KB

Chunk 24
Version: 3
Flags: present
NodeID: 10
Data 62000 KB

Chunk 17
Version: 1
Flags: foreign
NodeID: 21
Data 2000 KB

# Overhead

## Log-Header

- 70 Bytes per Block

## Segment-Header

- 1 Bit per Block
- 8 Bytes per Segment

# Overhead examples

### 100 GB Log-File, 64 MB Segments and 1 MB Blocks

$\Rightarrow$ 1.600 Segements each with 64 Blocks
$\Rightarrow$ 102.400 Blocks

$$1.600 * 8B + 102.400 * 1Bit + 102.400 * 70B$$
$$= 12.800B + 12.800B + 7.168.000B = 7.193.600B$$
$$\approx 7MB$$

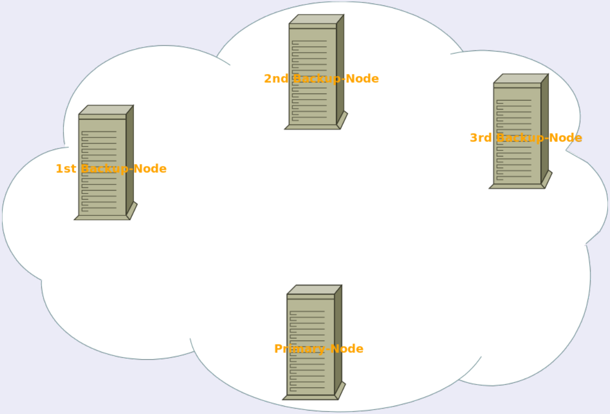
### 100 GB Log-File, 64 MB Segments and 1 KB Blocks

$\Rightarrow$ 1.600 Segements each with 65536 Blocks
$\Rightarrow$ 104.857.600 Blocks

$$1.600 * 8B + 104.857.600 * 1Bit + 104.857.600 * 70B$$
$$= 12.800B + 13.107.200B + 7.340.032.000B = 7.353.152.000B$$
$$\approx 7GB$$

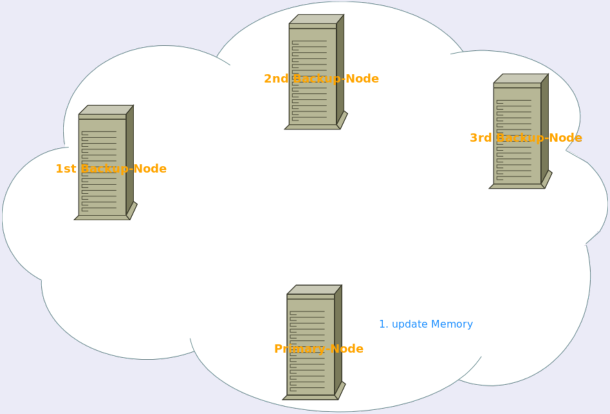
# Update process (Overview)

## Initial situation



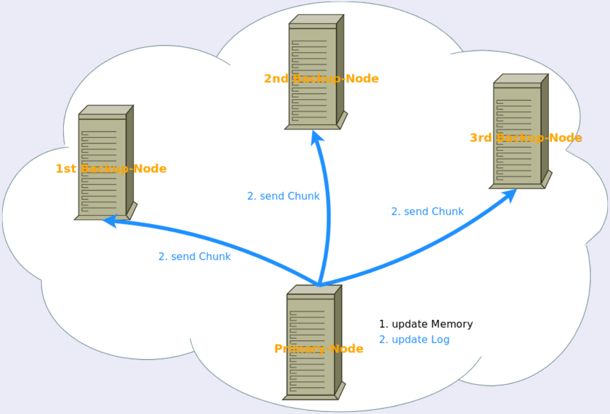
# Update process (Overview)

## 1. Step



# Update process (Overview)

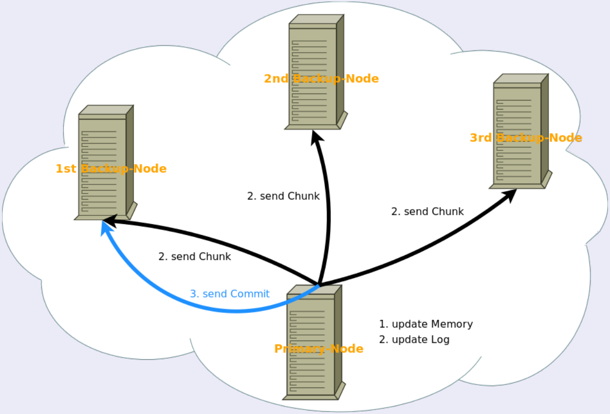
## 2. Step



1. update Memory
2. update Log

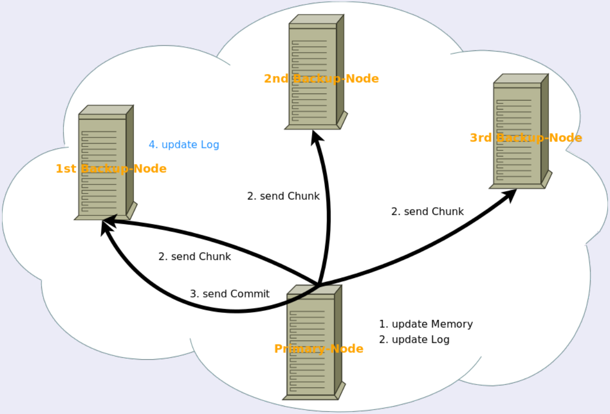
# Update process (Overview)

## 3. Step



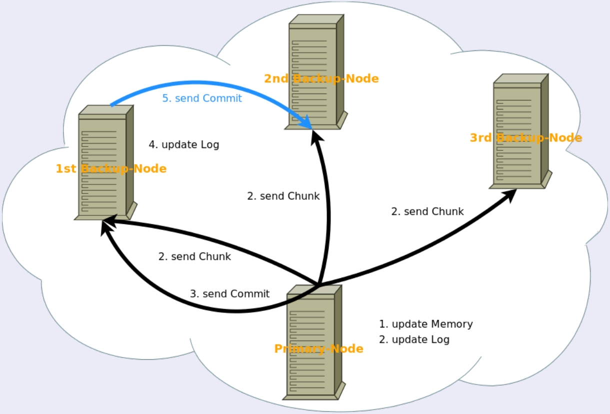
# Update process (Overview)

## 4. Step



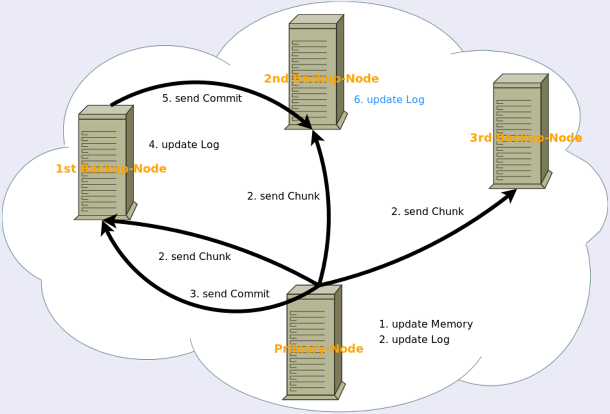
# Update process (Overview)

## 5. Step



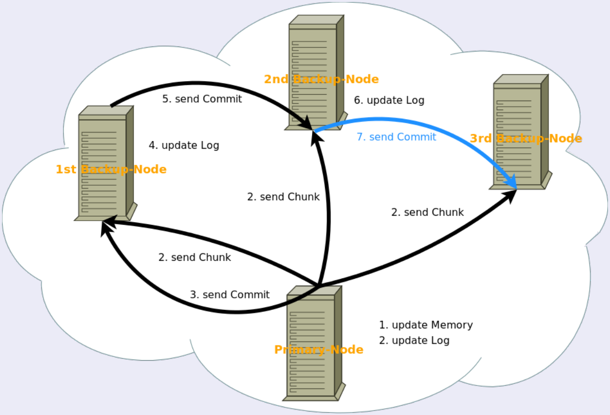
# Update process (Overview)

## 6. Step



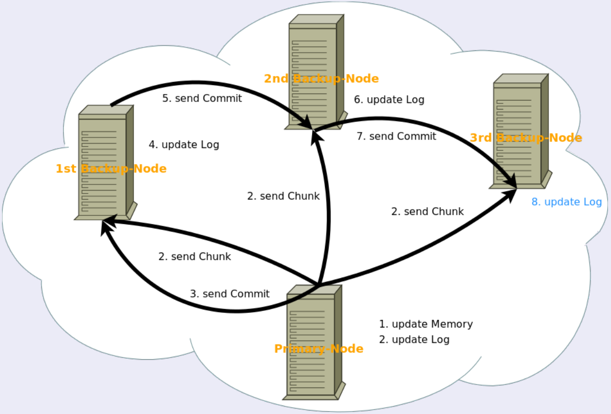
# Update process (Overview)

## 7. Step



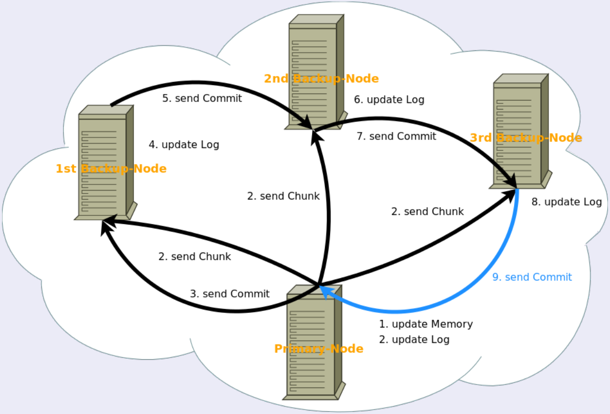
# Update process (Overview)

## 8. Step



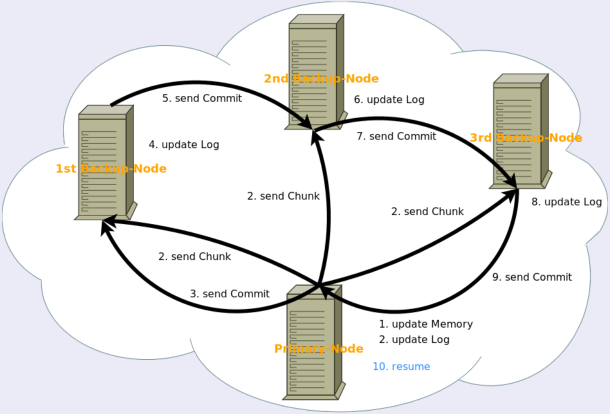
# Update process (Overview)

## 9. Step



# Update process (Overview)

## 10. Step



# Update process I

## Memory (Primary-Node)

(a) set dirty-bit

(b) update chunk in-place

(c) increment version

(d) clear dirty-bit

## Log (Primary-Node)

(a) send chunk to the backup nodes

(b) update chunk

Solution 1: in-place
Solution 2: copy-on-write (RAMCloud)
Solution 3: twins

(c) send Commit-Message to 1$^{st}$ Backup-Node

(d) wait for Commit-Message from 3$^{rd}$ Backup-Node

# Update process II

## Log (Backup-Node)

(a) receive chunk from Primary-Node

(b) buffer chunk

(c) wait for Commit-Message from predecessor

(d) update chunk (like Primary-Node)

(e) send Commit-Message to successor

## Problems

- Primary-Node crashes before sending the Commit-Message
- Backup-Node crashes

# Outline

1 Data-Structure

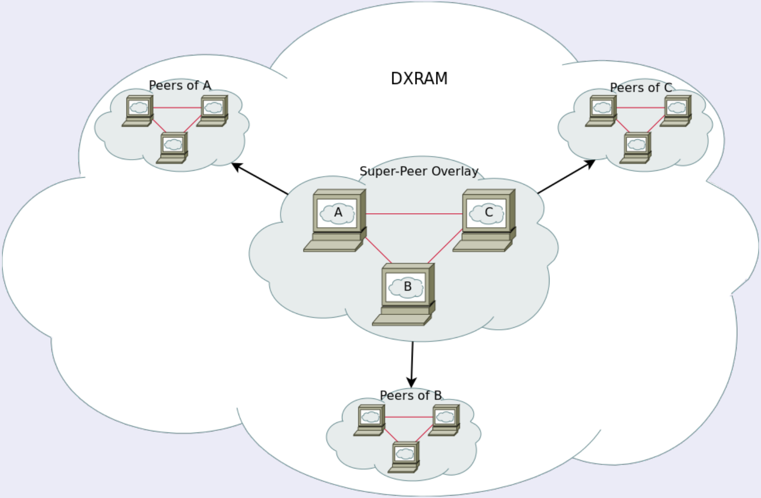
2 Log

3 Recovery
- Peers and Super-Peers
- Recovery process

4 Conclusion

# Super-Peer Overlay

## Overview



# Tasks

## Peer

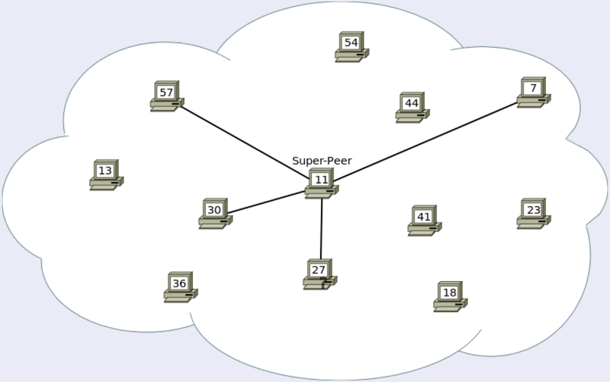
- stores Chunks in RAM
- logs own Chunks on SSD
- backups Chunks of other nodes on SSD

## Super-Peer

- also a Peer
- monitors its Peers (Heart-Beat)
- handles Meta-Data of Chunks (DHT-Node)

# Recovery process (Overview)

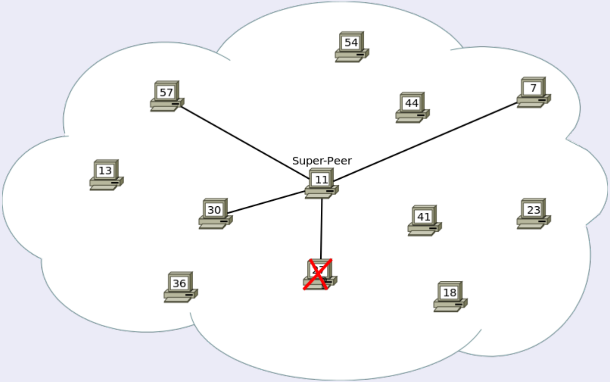
## Initial situation



- Super-Peer is connected with its Peers

# Recovery process (Overview)

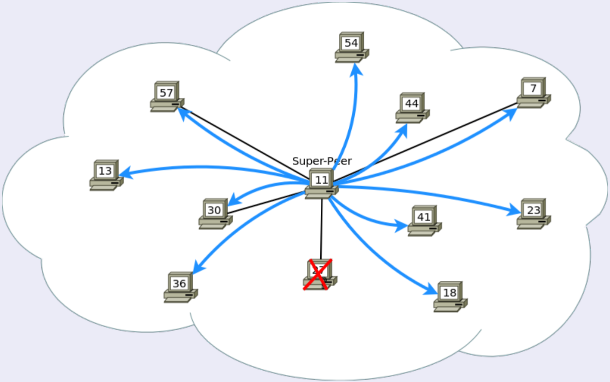
## Node failure



- Node 27 crashes

# Recovery process (Overview)

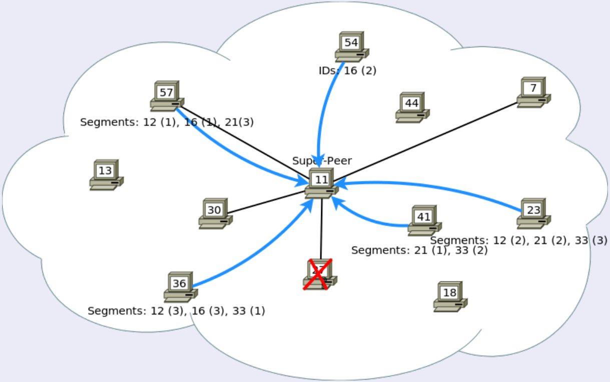
## Broadcast



- Super-Peer detects and broadcasts node failure
- Super-Peer becomes the Recovery Coordinator

# Recovery process (Overview)

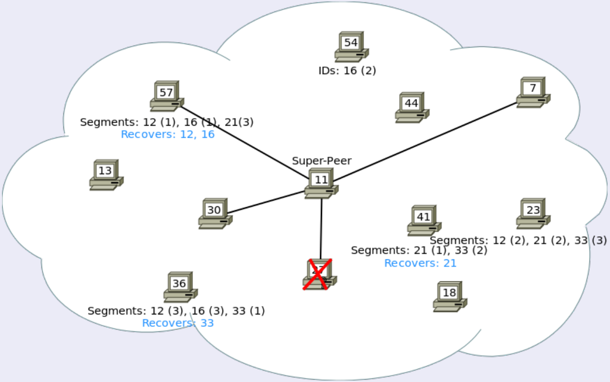
## Info-Message


54
IDs: 16 (2)
57
Segments: 12 (1), 16 (1), 21(3)
44
7
13
Super-Peer
11
30
41
23
Segments: 12 (2), 21 (2), 33 (3)
Segments: 21 (1), 33 (2)
27
36
Segments: 12 (3), 16 (3), 33 (1)
18

- every node deletes cached Meta-Data of the failed node
- every Backup-Node of the failed node sends information about the backuped data to the Recovery Coordinator
- Recovery Coorinator gathers this information

# Recovery process (Overview)

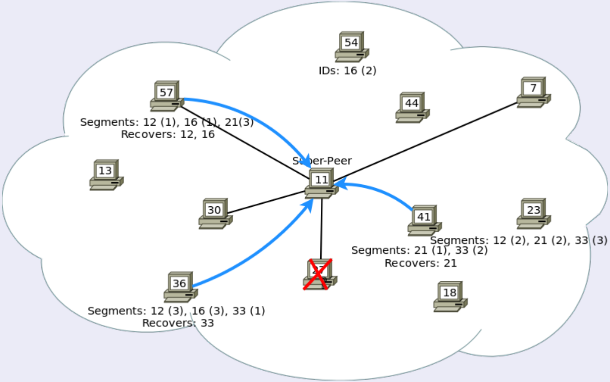
## Chunk recovery



- the 1$^{st}$ Backup-Node of each segment recovers the segment

# Recovery process (Overview)

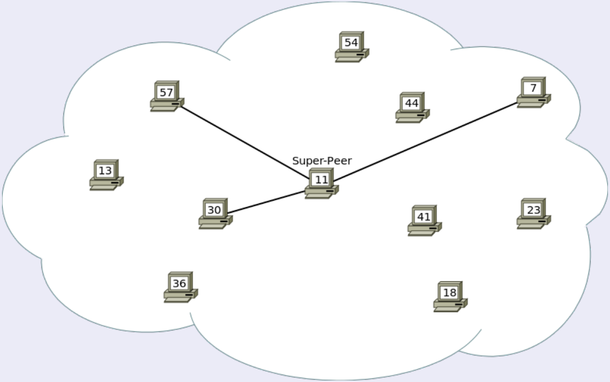
## Commit-Message



- the 1$^{st}$ Backup-Node of each segment send the result to the Recovery Coordinator

# Recovery process (Overview)

## Final situation



- Recovery Coordinator updates Meta-Data for the recovered Segments/Chunks
- the failed node is removed
- an application callback can be executed (optional)

# Conclusion

## Log

- little overhead for objects $\geq 1MB$ (e.g. Map & Reduce)
- huge overhead for Objects $\leq 1KB$ (e.g. Facebook)

⇒ a new log layout have to be disigned for small objects

- the three update solutions must be compared

## Recovery

- distributed recovery dependent on segment distribution
  - high distribution allows fast recovery
  - low distribution preserves locality
- ignores workload of the Backup-Nodes
- Problem: Not all data could be locally recovered (IO failure or insufficent memory space)
- Problem: Realisation of the application callback

Questions?