

# CHAPTER 6: DISTRIBUTED FILE SYSTEMS

## Chapter outline

- DFS design and implementation issues: system structure, file access, and sharing semantics
- Transaction and concurrency control: serializability and concurrency control protocols
- Replicated data management: one-copy serializability and coherency control protocols

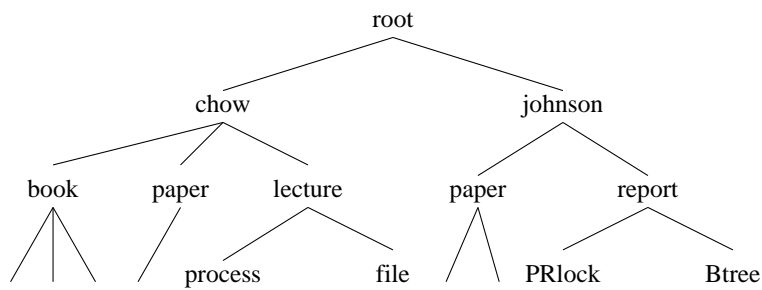
## Why is DFS important and interesting?

- It is one of the two important components (process and file) in any distributed computation.
- It is a good example for illustrating the concept of transparency and client/server model.
- File sharing and data replication present many interesting research problems.

## Structure of and access to a file system

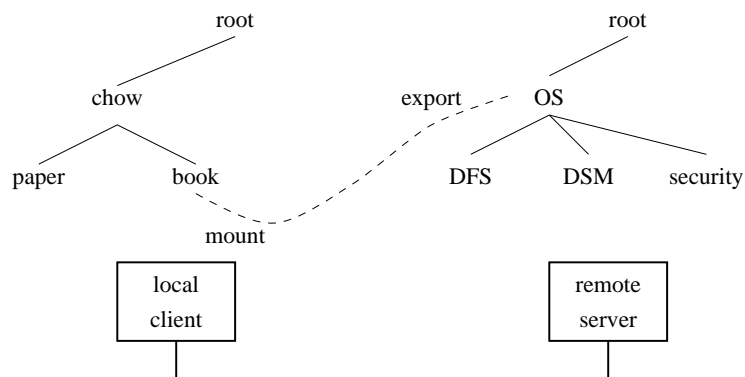
directory service		name resolution, add and deletion of files
authorization service		capability and / or access control list
file service	transaction	concurrency and replication management
	basic	read / write files and get / set attributes
system service		device, cache, and block management

- File: sequential, direct, indexed-sequential
- File System: flat, hierarchical



## Mounting protocols and NFS

- Explicit mounting
- Boot mounting
- Auto mounting



## Stateful and stateless file servers

- Opened files and their clients
- File descriptors and file handles
- Current file position pointers
- Mounting information
- Lock status
- Session keys
- Cache or buffer

## Semantics of sharing in DFS

space time	remote access	cache access	down/up load access
simple RW	no true sharing	coherency control	coherency control
transaction	concurrency control	coherency and concurrency	coherency and concurrency
session	not applicable	not applicable	ignore sharing

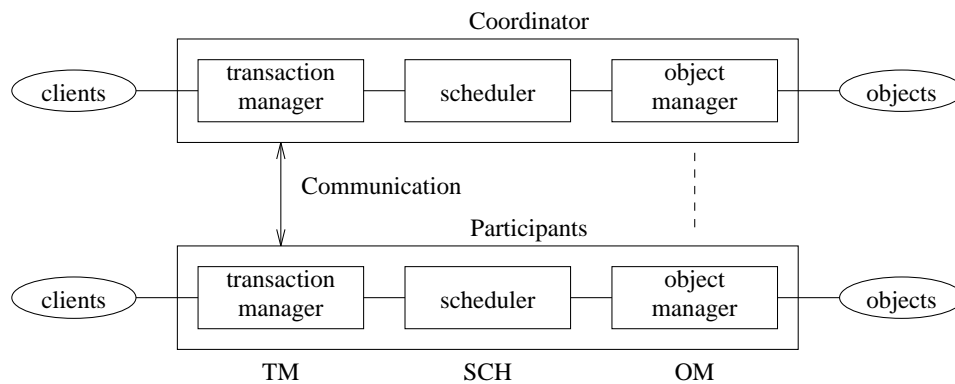
### sharing

- Unix semantics
- Transaction semantics
- Session semantics

### replication

- Write policies: write-through and write back
- cache coherence control: write-invalidate and write-update
- Version control (immutable files): ignore conflict, resolve version conflict, resolve serializability conflict

## Transaction and concurrency control



### transaction processing system (TPS)

- Transaction manager (TM)
- Scheduler (SCH)
- Object manager (OM)

### atomicity

- All or none: TM, two-phase commit
- Indivisible (serializable): SCH, concurrent control protocols
- Atomic update: OM, replica management

## Serializability

IF the interleaved execution of transactions is to be equivalent to a serial execution in some order, then all conflicting operations in the interleaved serializable schedule must also be executed in the same order at all object sites.

$t_0$  : **bt** Write A=100, Write B=20 **et**

$t_1$  : **bt** Read A, Read B, 1: Write sum in C, 2: Write diff in D **et**

$t_2$  : **bt** Read A, Read B, 3: Write diff in C, 4: Write sum in D **et**

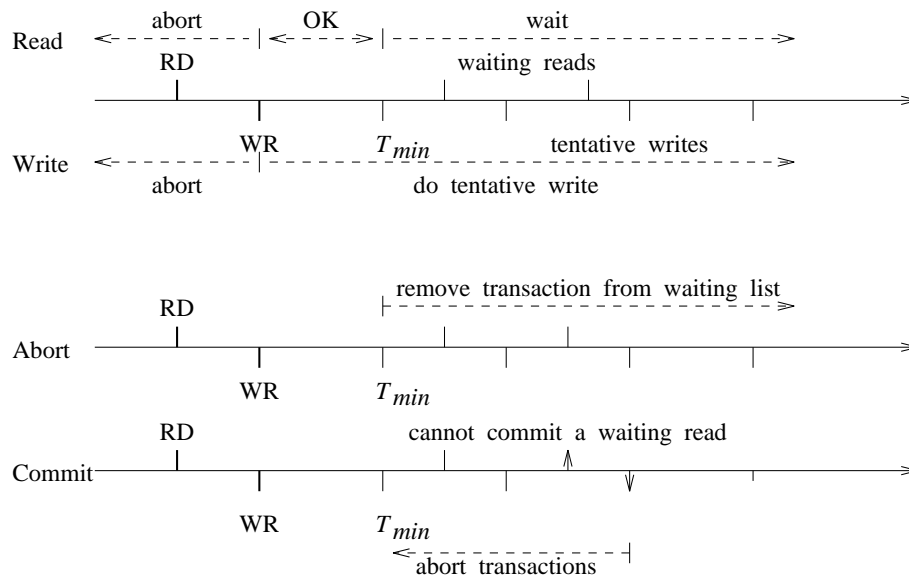
<i>Sched</i>	<i>Interleave</i>	<i>Log in C</i>	<i>Log in D</i>	<i>Result (C,D)</i>	<i>2PL</i>	<i>Timestamp</i>
1	1, 2, 3, 4	$W_1 = 120$ $W_2 = 80$	$W_1 = 80$ $W_2 = 120$	(80, 120) consistent	feasible	feasible
2	3, 4, 1, 2	$W_2 = 80$ $W_1 = 120$	$W_2 = 120$ $W_1 = 80$	(120, 80) consistent	feasible	$t_1$ aborts and restarts
3	1, 3, 2, 4	$W_1 = 120$ $W_2 = 80$	$W_1 = 80$ $W_2 = 120$	(80, 120) consistent	not feasible	feasible
4	3, 1, 4, 2	$W_2 = 80$ $W_1 = 120$	$W_2 = 120$ $W_1 = 80$	(120, 80) consistent	not feasible	$t_1$ aborts and restarts
5	1, 3, 4, 2	$W_1 = 120$ $W_2 = 80$	$W_2 = 120$ $W_1 = 80$	(80, 80) inconsistent	not feasible	cascade aborts
6	3, 1, 2, 4	$W_2 = 80$ $W_1 = 120$	$W_1 = 80$ $W_2 = 120$	(120, 120) inconsistent	not feasible	$t_1$ aborts and restarts

## Concurrency control protocols

### Two-phase locking

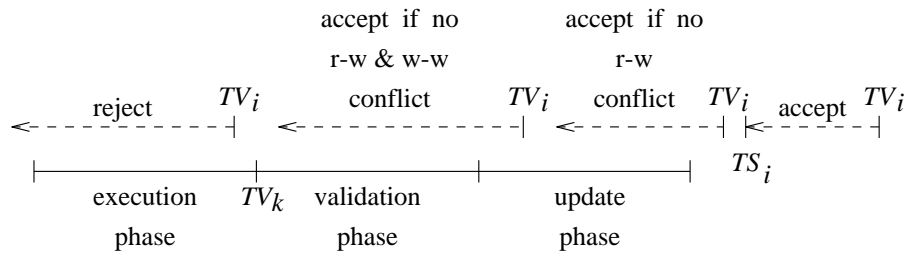
- locking and shrinking phases of requesting and releasing objects
- concurrency versus serializability
- rolling abort, strict two-phase locking

### Timestamp ordering



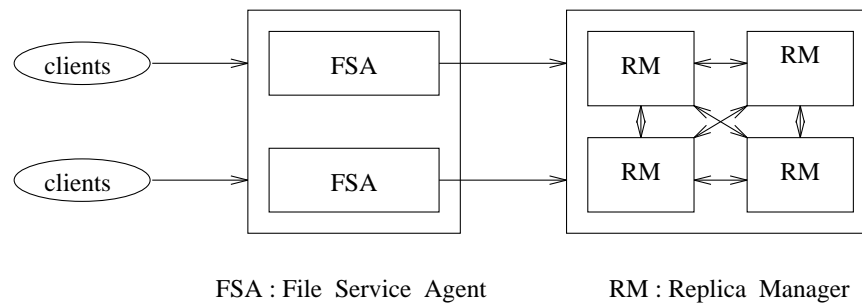
## Optimistic concurrency control

- Execution phase
- Validation phase:
  1. Validation of transaction  $t_i$  is rejected if  $TV_i < TV_k$ . All transactions must be serialized with respect to  $TV$ .
  2. Validation of transaction  $t_i$  is accepted if it does not overlap with any  $t_k$ .  $t_i$  is already serialized with respect to  $t_k$ .
  3. The execution phase of  $t_i$  overlaps with the update phase of  $t_k$ , and  $t_k$  completes its update phase before  $TV_i$ . Validation of  $t_i$  is accepted if  $R_i \cap W_k = \phi$ .
  4. The execution phase of  $t_i$  overlaps with the validation and update phases of  $t_k$ , and  $t_k$  completes its execution phase before  $TS_i$ . Validation of  $t_i$  is accepted if  $R_i \cap W_k = \phi$  and  $W_i \cap W_k = \phi$ .
- Update phase



## Data and file replication

### Architecture of a replica manager



#### read operations

- Read-one-primary
- Read-one
- Read-quorum

#### write operation

- Write-one-primary
- Write-all
- Write-all-available
- Write-quorum
- Write-gossip

## One-copy serializability

The execution of transactions on replicated objects is equivalent to the execution of the same transactions on nonreplicated objects. Some approaches:

- read-one-primary/write-one-primary
- read-one/write-all
- read-one/write-all-available

### failures

Failures can cause problems with one-copy serializability. For example:

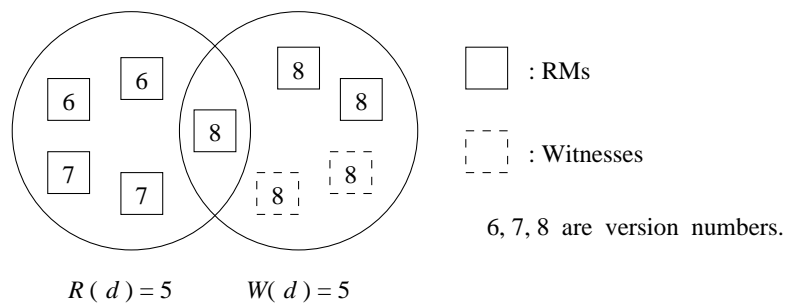
$t_0$  : **bt**  $W(X)$   $W(Y)$  **et**  
 $t_1$  : **bt**  $R(X)$   $W(Y)$  **et**  
 $t_2$  : **bt**  $R(Y)$   $W(X)$  **et**

$t_1$  : **bt**  $R(X_a)$  ( $Y_d$  fails)  $W(Y_c)$  **et**  
 $t_2$  : **bt**  $R(Y_d)$  ( $X_a$  fails)  $W(X_b)$  **et**

## Quorum voting

From read-one/write-all-availble to read-quorum/write quorum:

1. **Write-write conflict:**  $2 * W(d) > V(d)$ .
2. **Read-write conflict :**  $R(d) + W(d) > V(d)$ .



## Gossip update propagation

Lazy update propagation: read-one/write-gossip

- Basic gossip protocol: read/overwrite
- Causal order gossip protocol: read/modify-update

