

# Study of Erasure Coding and its Application on Building Reliable Distributed File System

*EEL6706 Fault Tolerant Computer Architecture*  
*Class Project*  
*Spring 2007*

**Ming Zhao**  
**ming@acis.ufl.edu**



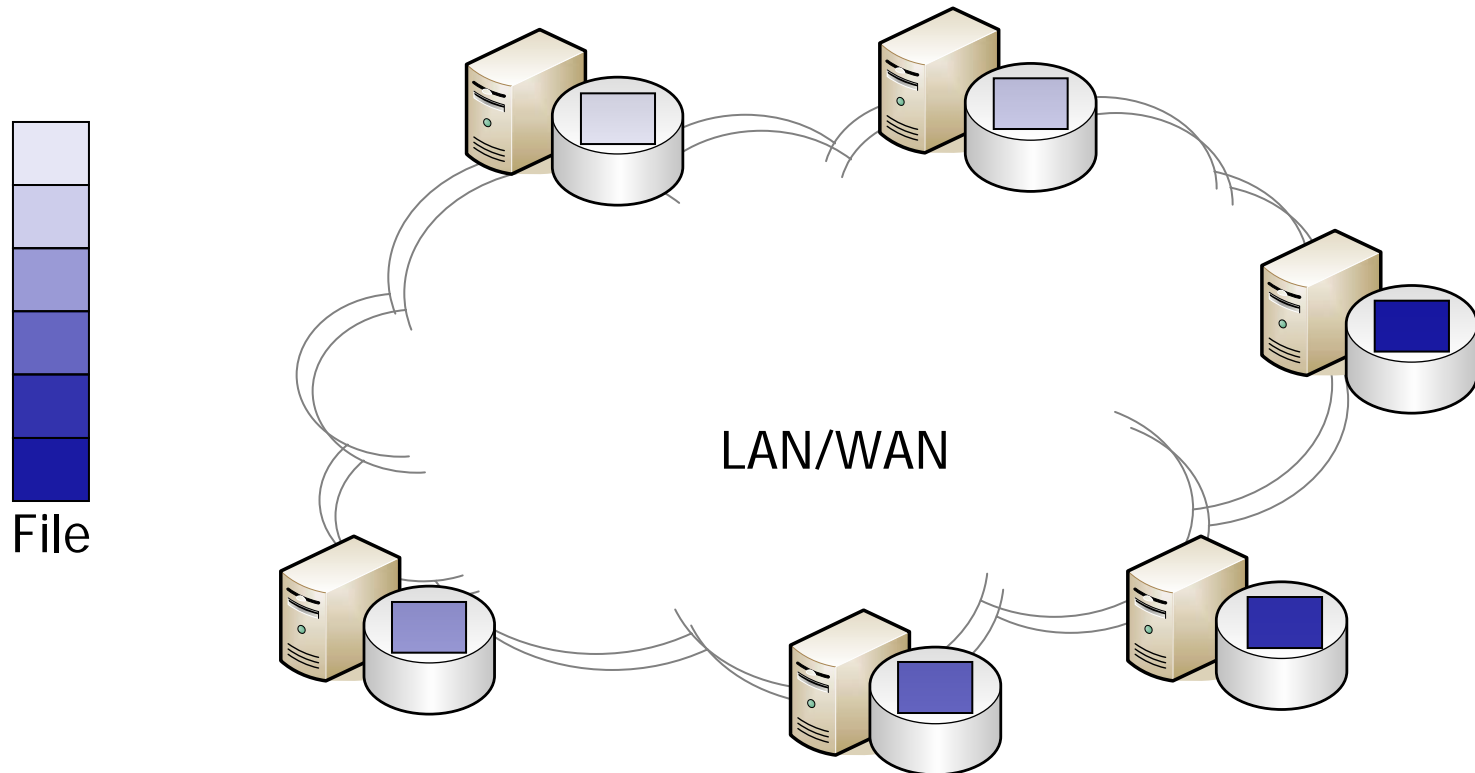
# Erasure Coding

- An Erasure Code divides a data block into  $p$  fragments and recode them into  $q > p$  fragments
  - The original block can be reconstructed from **any**  $p$  fragments
    - Sometimes referred as **optimal** erasure codes
    - Non-optimal: needs more than  $p$  fragments to recover the block
  - **Rate** of code:  $r = p/q$ 
    - Increases the storage cost by  $1/r$
- Example:
  - An  $r=1/4$  code divides a block into 16 fragments and encodes them into 64 fragments
  - Increasing storage cost by a factor of 4
  - The original can be recovered from any 16 fragments

- Failure model
  - **Erasure**: a failed device is recognized by the system in advance
  - As opposed to error: data block is stored or retrieved incorrectly
  - Needs some embedded coding to recognize corrupted blocks
    - Eg. A verifying hash
    - Not considered in this study
- Improves both performance and reliability
  - RAID-like storage systems
  - P2P file sharing: BitTorrent
  - Wide-area file systems: OceanStore

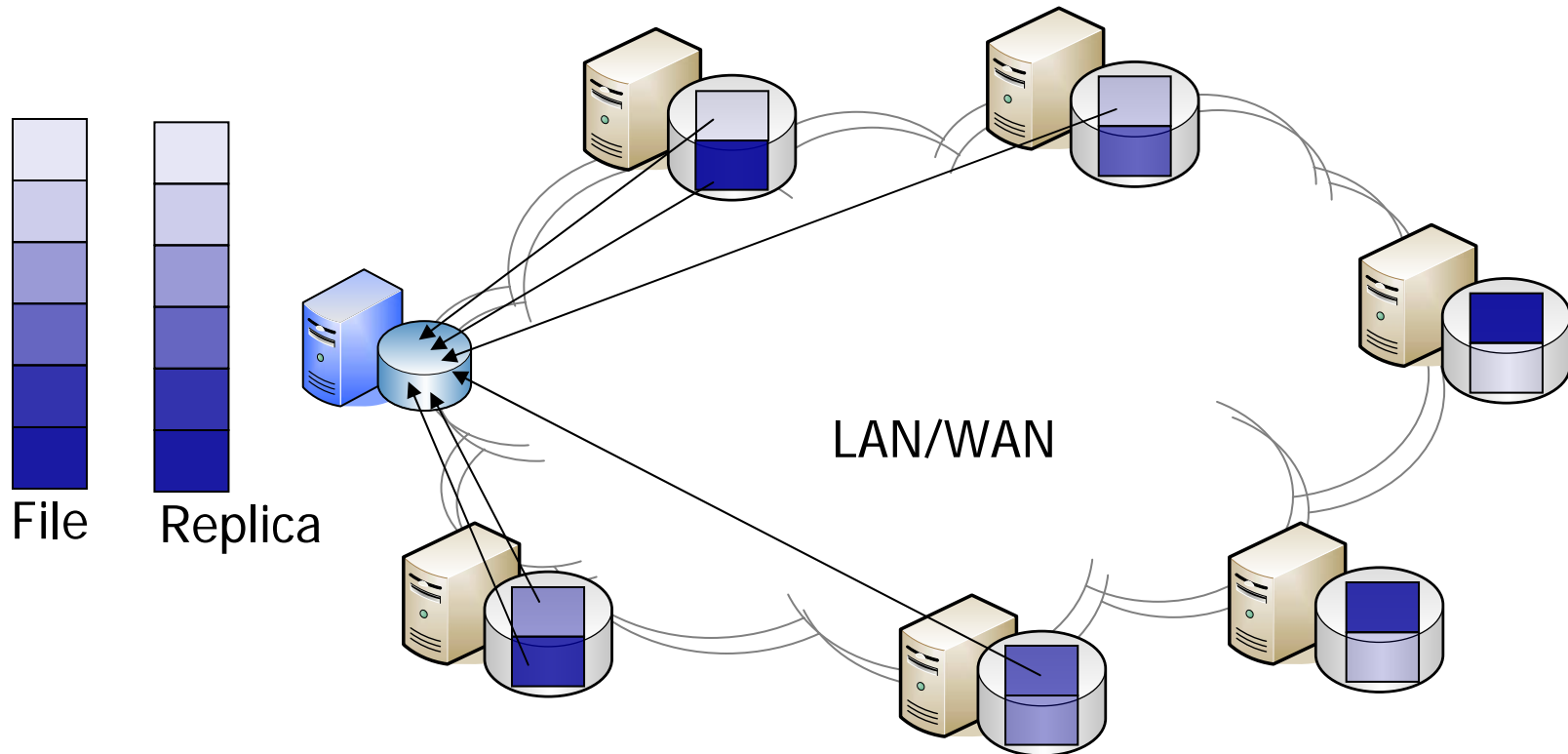
# Example

- Server farm, P2P network, Grid system, ...
- Splits a large file into  $n$  small blocks and distributes on the network



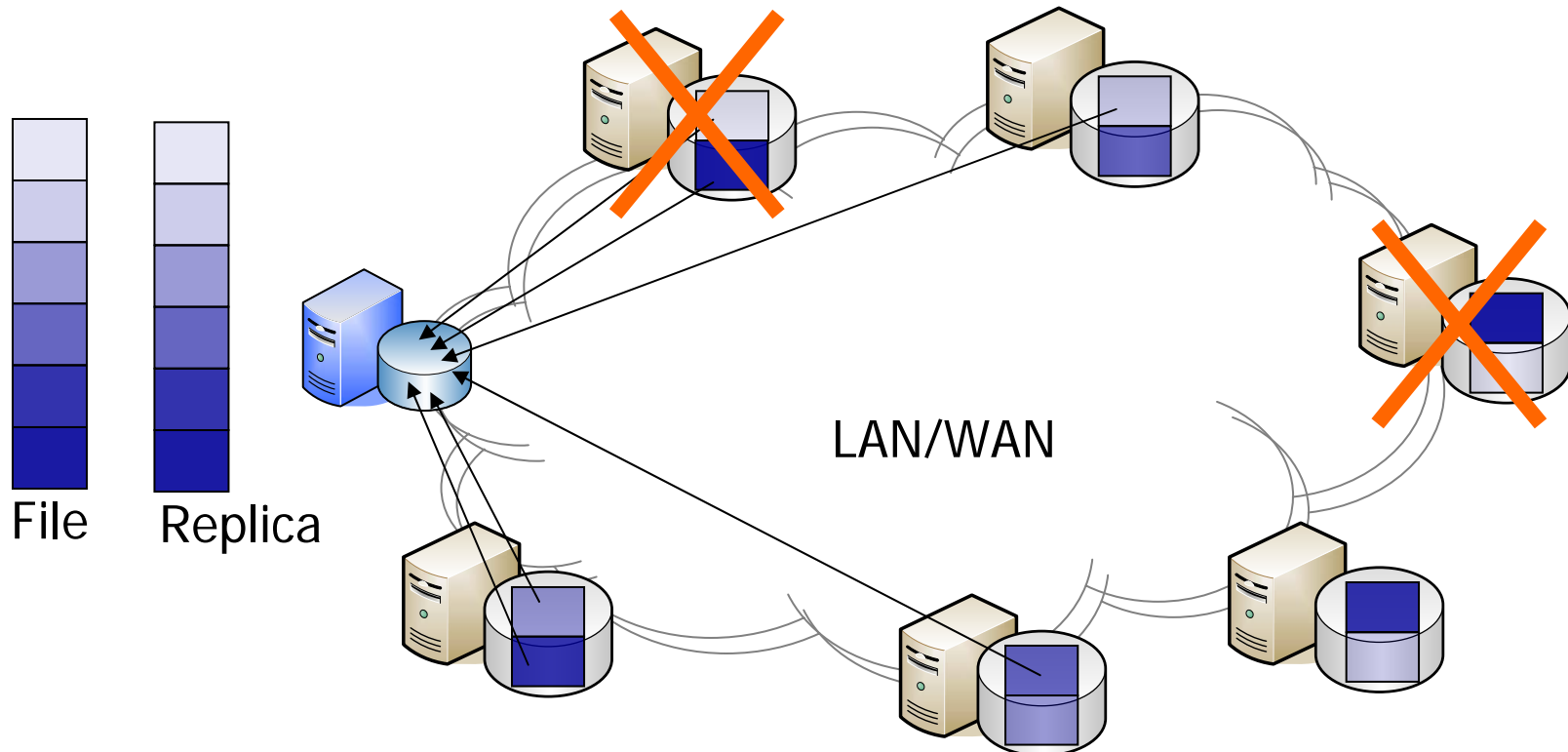
# Full-file Replication

- A client/peer can download each block from the closest copies
- Replication tolerates failures



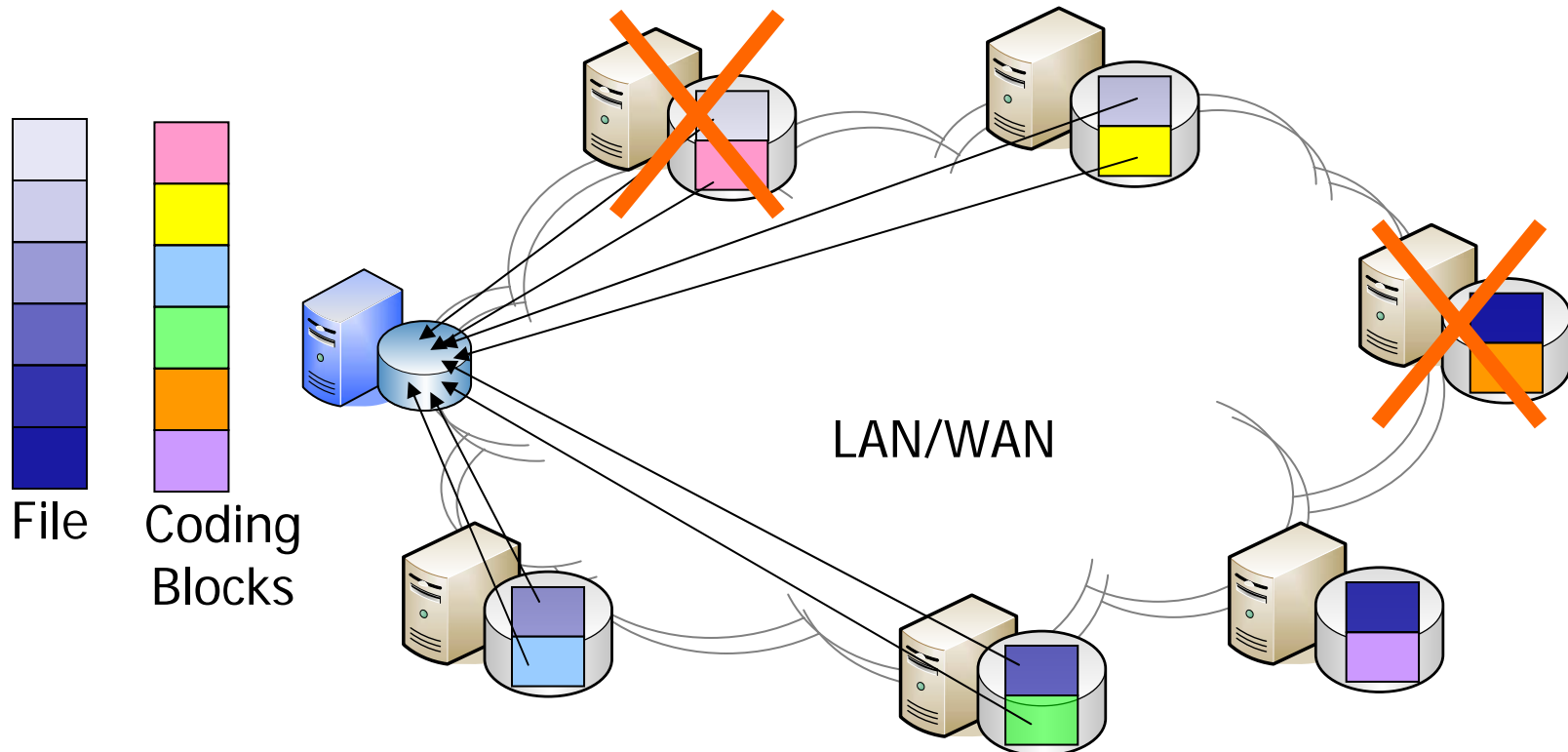
# Full-file Replication

- Replication cost is too high in both space and performance
- Cannot recover if a block's both copies are failed



# Erasure Coding

- Calculates  $m$  coding blocks and distributes on the network
- A client can download the closest  $n$  blocks and recover the file



# Reed-Solomon Codes

- The canonical erasure code
  - Uses Vandermonde matrix to calculate and maintain checksum codes
  - Uses Gaussian Elimination to recover from failures
  - Uses Galois Fields to perform arithmetic
- Suppose
  - Each data block is divided into words of size  $w$  ( $2^w > n+m$ )
  - $N$  data words ( $d_1, \dots, d_n$ ),  
 $m$  coding words ( $c_1, \dots, c_m$ )
  - Vandermonde matrix:

$$\begin{bmatrix} 0^0 & 0^1 & 0^2 & \dots & 0^{n-1} \\ 1^0 & 1^1 & 1^2 & \dots & 1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (n+m-1)^0 & (n+m-1)^1 & (n+m-1)^2 & \dots & (n+m-1)^{n-1} \end{bmatrix}$$

- Uses elementary transformations to construct information dispersal matrix

$$\begin{bmatrix} 0^0 & 0^1 & 0^2 & \dots & 0^{n-1} \\ 1^0 & 1^1 & 1^2 & \dots & 1^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ (n+m-1)^0 & (n+m-1)^1 & (n+m-1)^2 & \dots & (n+m-1)^{n-1} \end{bmatrix} \Rightarrow \begin{bmatrix} I \\ X \end{bmatrix} = B$$

- Calculates the code words

$$\begin{bmatrix} I \\ X \end{bmatrix} * \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

# Decoding

- Downloads any  $n$  words ( $f_1, \dots, f_n$ )
- Creates  $B'$  using the corresponding rows from  $B$ 
  - Any  $n$  rows from  $B$  is guaranteed to be linearly independent
  - The values of  $(d_1, \dots, d_n)$  can be calculated using Gaussian elimination

$$B' * \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

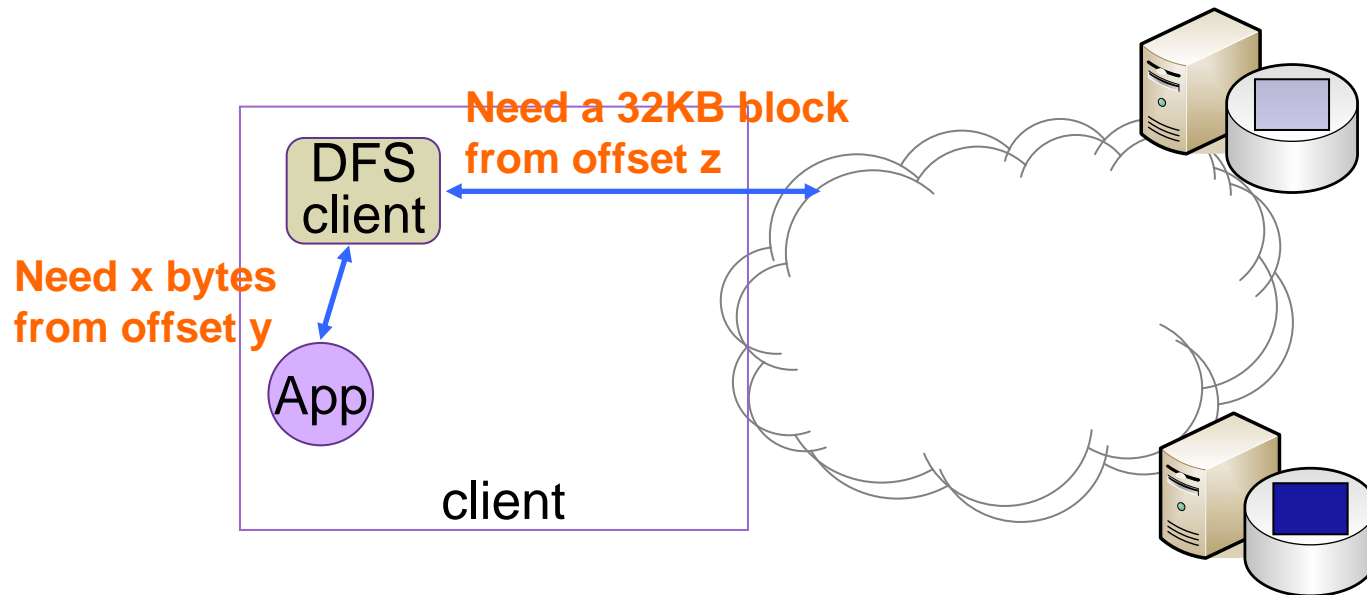
- Inverts  $B'$
- Calculates the data words

$$B'^{-1} * \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}$$

- Must use Galois Field arithmetic  $GF(2^w)$ 
  - Addition (exclusive or) is cheap
  - Multiplication is expensive
- Suppose  $x$  words per data block
  - Encoding:  $O(mnx)$
  - Decoding:  $O(n^3)+O(n^2x)$
- Very expensive for large  $n$  and  $m$ 
  - Studied here to help understanding of using erasure codes in distributed file systems (DFS)
  - Suitable for use in small/medium-scale systems

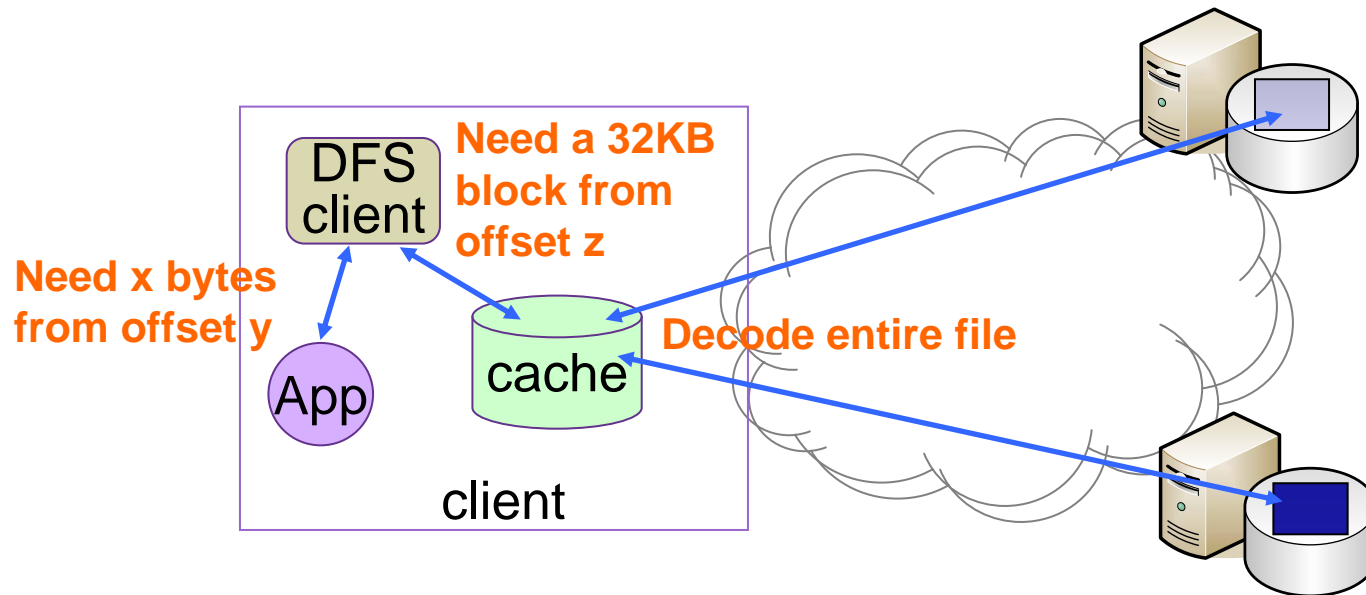
# Using Erasure Codes in DFS

- DFS needs to support on-demand data access
  - Different from file-swapping
  - Applications cannot explicitly download and decode files



# On-demand File-based Encoding/Decoding

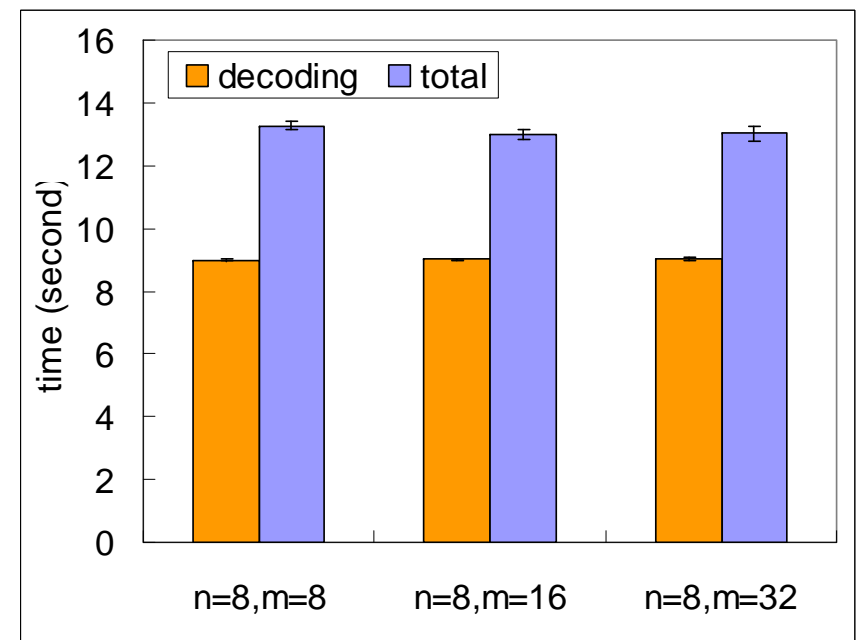
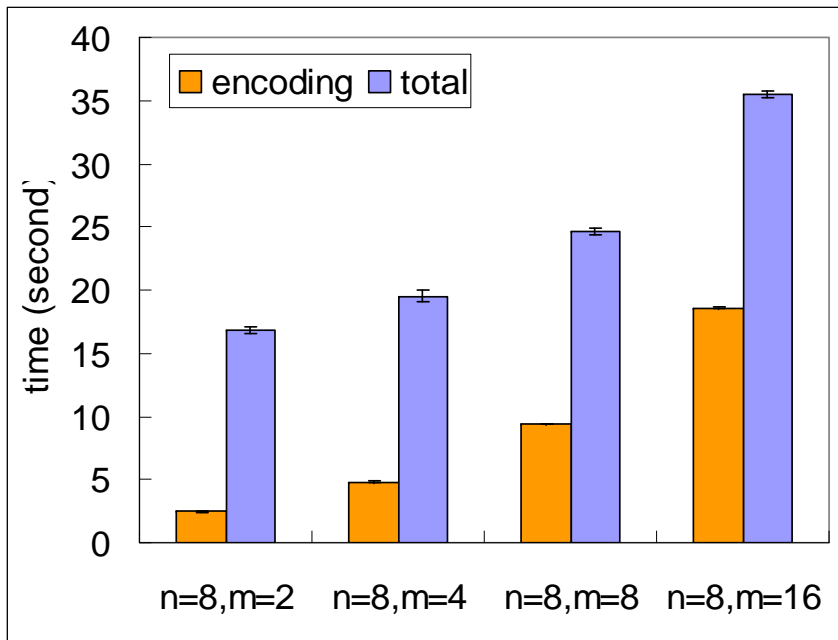
- DFS client downloads and decodes files in local disk cache
- Satisfies application requests from cached files
- Re-encoding and distributes files afterwards



# On-demand File-based Encoding/Decoding

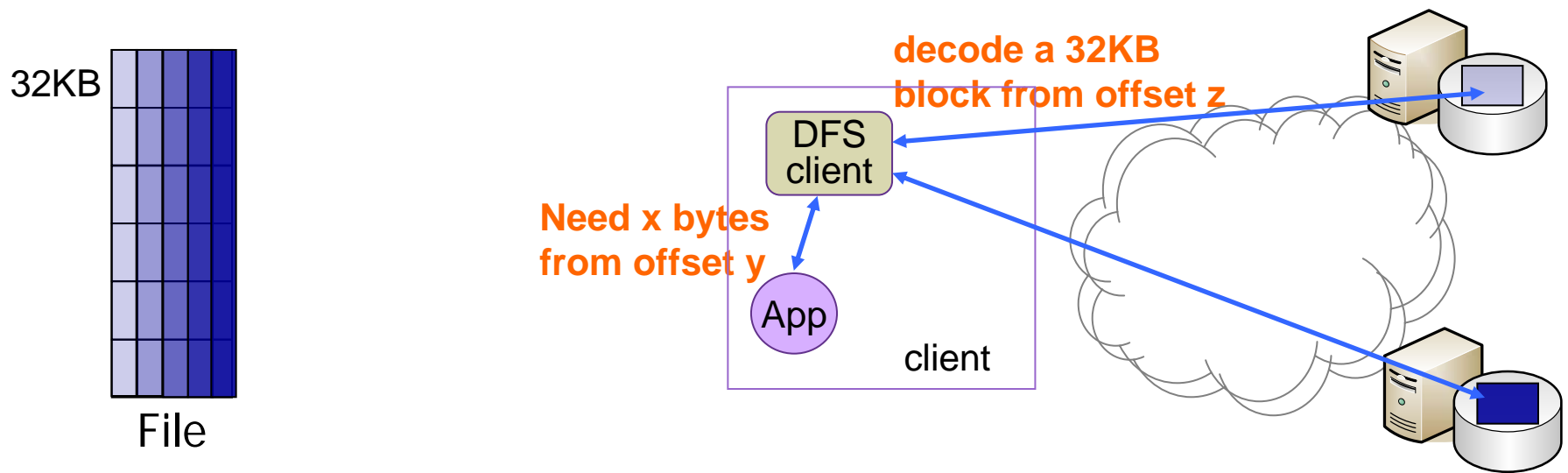
## ■ Implementation

- Algorithms developed based on Dr. Plank's Galois Field library
- Integrated with GVFS – a user-level grid/wide-area file system



# On-demand Block-based Encoding/Decoding

- Files are encoded/decoded block-by-block
  - Splits a file horizontally
- DFS client downloads and decodes blocks on-demand

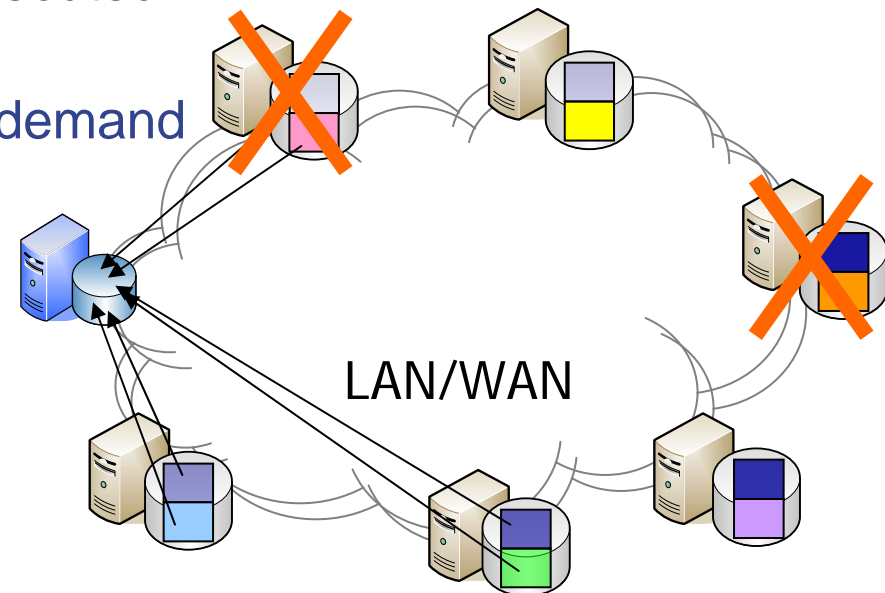


# On-demand Block-based Encoding/Decoding

- Implementation
  - Developed encoding, decoding, update algorithms (based on GFLib)
  - Integrated with GVFS
- Advantages
  - Less startup overhead: no need to download entire files
  - Supports fine-grained data sharing
- Disadvantages
  - Less efficient for bulk-data accesses
  - Requires more time for encoding
    - File is not used sequentially

# Experiments on Fault-tolerance

- A real medium-scale testbed built with virtual machines
  - A 512MB file encoded and distributed across 16 file servers
    - $n=8, m=8$
  - An application is continuously executed
    - Reads the entire file sequentially
  - GVFS client decodes the file on-demand for every iteration: full-file based
- Injects failures on file servers
  - Independent Poisson processes
    - Same MTTF and MTTR
  - Varies MTTF and MTTR
    - Studies the system's reliability



# Results from an Extreme Case

- MTTF=MTTR=60 seconds

16:47:01: esx12 failed

16:47:06: esx13 failed

.....

16:47:19: application started

16:47:21: esx6 failed

16:47:25: esx5 failed

.....

16:48:15: decoding completed

16:48:23: esx8 failed

16:48:24: esx11 failed

16:48:25: application finished

16:48:25: application started  
mission impossible!

Totally 16 nodes (0 ~ 15)

Alive nodes

0 1 2 4 5 6 7 8 9 10 13 14 15

0 1 2 5 6 7 8 9 10 11 13 15

0 1 5 6 8 11 13  
(only 7 nodes alive)

# Conclusion

- Studied erasure coding and RS codes in details
- Developed algorithms based on GFLib and integrated with GVFS
- Evaluated the performance of encoding/decoding
- Evaluated the reliability on a real, medium-scale testbed
- Future work
  - Fast, near-optimal erasure codes
  - Large-scale distributed file systems