Multimedia Systems:
Lecture 3 -- Disk Arrays

- Disk Array Basics
- Disk Array Organizations/Terminology
- Cost/Performance
- Reliability

Disk Array Basics

- Disk arrays
  - Stripe or interleave data across disks

  - Large accesses:
    - » Higher transfer rates
  - Small accesses:
    - » High I/O rates

- Redundany needed for high reliability

Motivation for Disk Arrays

- Improving processor performance
- Amdahl's Law:
  - Improve only part of a system
  - Eventually overall performance limited by portion of system that is not improved
- Increasing gap between processor, I/O performance
  - CPUs: 50% per year
  - Disks: 10-20% per year
- Faster processor enable new applications
  - Video, multimedia: large storage requirements

Data Striping

- Data Striping
  - Transparently distributes data over disks
  - Appear as single fast, large disk
- Service I/O operations in parallel
  - Small operations on different disks
    - » Decreases queueing time for I/Os
  - Large operations use several disks in parallel
    - » Increases effective transfer rate

Granularity of Data Interleaving

- **Fine-grain:**
  - **Small unit of interleaving**
  - **Entire array involved in every I/O**
  - **At most one I/O at a time**
- **Coarse-grain:**
  - **Larger unit of interleaving**
  - **Small I/Os serviced by one disk**
  - **Larger I/Os use several disks in parallel**

Redundancy

- **Large disk arrays vulnerable to disk failures**
- **Single disk: mean time to failure (MTTF)**
  - **200,000 hours**
- **100 disks in array**
  - **MTTF(array) = 200,000 / 100 = 2000 hours**

- **Solution: Add redundancy**
  - **Error correction codes**
  - **Allow recovery from disk failures**

- **Performance of redundant arrays**
  - **All write operations must update redundancy**

Redundancy Issues

- Type of redundancy
  - Parity
  - Hamming codes (used for memory)
  - Reed-Solomon codes

- Method for distributing redundancy information
  - Disks dedicated to redundancy
  - Uniformly distributed

Non-redundant Arrays (Level 0)

- Employs no redundancy
- Highest write performance
  - No redundancy updates required
- No protection against disk failures

- Typical use: supercomputing environments (Cray)
  - Capacity, throughput important
  - Reliability less important

Mirrored Arrays (Level 1)

- **Maintain two copies of all disks**
- **Uses twice as many disks**
- **Best read performance**
  - **Read from disk with shortest queue, seek, rotation**
- **High availability**
  - **If any disk fails, another copy available**

- **Typical use: database applications**
  - **Availability, I/O rate important**
  - **Storage efficiency less important**

Memory-Style ECC (Level 2)

- **Traditionally, memory systems use Hamming codes**
- **Parity for distinct overlapping subsets of components**
- **Example: 4 data disks require 3 redundancy disks**
- **Capable of:**
  - **identifying failed component**
  - **reconstructing lost data**

- **In practice, not used for RAID**

Bit-Interleaved Parity (Level 3)

- Disk controllers can identify failed disks
- Only need to reconstruct failed data
- Parity: requires only a single redundancy disk
  - Recover from any single disk failure
- Fine-grain interleaving
  - Only one request serviced at a time
  - Writes involve all disks
  - Reads involve all but parity disk
- Easy to implement

- Typical use: applications requiring high throughput, not high I/O rates

Block-Interleaved Parity (Level 4)

- Similar to RAID Level 3
- Coarse-grained interleaving: stripe in large blocks
  - Small I/Os serviced by single disks
  - Large I/Os serviced by several disks in parallel
- Large writes:
  - Write entire group of disks
  - Calculate redundancy before writing to disk
- Small writes:
  - Require 4 I/O operations
  - Read old data, parity; write new data, parity

- Not built in practice: parity disk a bottleneck

Block-Interleaved Distributed Parity (Level 5)

- Similar to Level 4: Coarse-grain interleaving
- Parity distributed uniformly over all disks
- All disks participate in large read, write operations
- Small write requests: inefficient
- Good performance on other accesses
- More complicated to implement

- Typical uses: applications that need both high transfer rate and high I/O rate

P + Q Redundancy (Level 6)

- Tolerate multiple disk failures
- Logically: 2-dimensional parity
- Small write performance poor
  - 6 I/O operations
  - read and write data, P, and Q redundancy

Cost, Performance, Reliability Comparisons

- **Performance metrics:**
  - **I/Os per second**
  - **Bytes per second**
  - **Response time**
- **Best metric: depends on application**
  - **Databases: I/Os per second per dollar**
  - **Supercomputing: bytes per second per dollar**
  - **File servers: both I/Os/sec and bytes/sec**

- **Compare systems at same total user capacity**
  - **Does not include redundancy data**

Performance Comparisons

- **Small reads:**
  - **RAID 1, 5, 6: good**
  - **RAID 3: poor, diminishes with group size**
- **Small writes:**
  - **RAID 3: worst**
  - **RAID 6 worse than RAID 5: more updates**
  - **All perform poorly compared to RAID 0**
- **Large reads:**
  - **RAID 1, 5, 6 good**
  - **RAID 3: improves with larger group size**
- **Large writes:**
  - **All improve as group size increases**
  - **RAID 6 worse than RAID 5: more updates**

Reliability Comparisons

- First, consider only independent disk failures
- 100 disks, each with MTTF 200,000 hours
- Mean time to repair disk: one hour
- With parity, MTTF 3000 years
- With P + Q redundancy, MTTF 38 million years

- More realistic look:
  – System crashes
  – Uncorrectable Bit Errors
  – Correlated Disk Failures

System Crashes

- Interrupts I/O operation to disk array
  – power failure, operator error, software crash
- May interrupt write of parity or data
- Inconsistent parity

- Especially a problem for block-interleaved arrays

- Maintain a log of write operations
- After crash, use log to assure consistency

Uncorrectable Bit Errors

- Caused by incorrectly-written data or later damaged
- One in 10^{14}
- Detected on reads
- Subject to interpretation
  - Rate at which errors are detected during reads during normal operation
- Problem when a disk fails
  - During reconstruction of failed data, may encounter uncorrectable bit errors
  - Make it impossible to reconstruct the data
- One solution: detect when disks are about to fail

Correlated Disk Failures

- Environmental and manufacturing factors
  - Frequently cause correlated disk failures
- Disasters
- Power failures/turning on/off
- Shared hardware (e.g., host controllers)
- More likely to fail very early and late in lifetimes
  - Bad batches of disks: fail close together in time
    - » problems with manufacturing process

- Increased likelihood that additional disk will fail before old data is reconstructed

Main Failure Modes

- **Double disk failure**
- **System crash followed by a disk failure**
- **Disk failure followed by uncorrectable bit error during reconstruction**

- **Hard to protect against them without harming performance**

RAID Level 5 Reliability

- **Double Disk Failure**
  - **MTTDL 285 years**
  - **Prob. of Data Loss in First 10 years: 3.4%**
- **System Crash + Disk Failure**
  - **MTTL 154 years**
  - **Prob. Loss in First 10 yrs: 6.3%**
- **Disk Failure + Uncorrectable Bit Error**
  - **MTTL 36 years**
  - **Prob. Loss in First 10 yrs: 24.4%**

RAID Level 6 Reliability

- Double Disk Failure
  - MTTDL 38052 years
  - Prob. of Data Loss in First 10 years: 0.03%
- System Crash + Disk Failure
  - MTTL 144 years
  - Prob. Loss in First 10 yrs: 7.7%
- Disk Failure + Uncorrectable Bit Error
  - MTTL 47697 years
  - Prob. Loss in First 10 yrs: 0.02%

- System crashes the biggest problem

Implementation Issues

- Maintain state information on data, parity blocks
  - Which hold valid blocks

- After a crash, regenerate parity
  - Avoid having to regenerate all parity
  - Before writing data, mark parity sectorns inconsistent
  - Maintain state of parity sectors in stable storage
  - On restart: regenerate only inconsistent parity blocks

Implementation Issues

- Operating with a failed disk
  - Risky: subsequent system crash may lose data
- Maintain some form of logging
- Prevent loss of data after crash
  - Demand Reconstruction
    - » When try to access a failed sector,
    - » immediately reconstruct onto spare disk
    - » Reconstruct entire disk "soon"
  - Parity sparing
    - » Reconstruct lost sector, replace parity sector
    - » No more parity -- not affected by crashes
    - » Later, replace disk, regenerate parity

Implementation Issues

- Orthogonal RAID
  - Calculate redundancy in groups perpendicular to disk strings
  - Then if controller lost, don't lose whole parity group

Improving Small Writes (RAID5)

- Caching
  - Write buffering: tell O.S. write is complete after written to buffer (not disk)
  - Group sequential writes together
    - » May turn small writes into full-stripe writes
  - Read caching: if old data in buffer, small write penalty only 3 I/Os
  - Cache parity: reduce penalty to 2 I/Os
- Floating parity
  - Optimizes writes of parity to minimize disk rotation operations
  - Updating parity only takes a little longer than reading old parity

Improving Small Writes (RAID5)

- Parity Logging
  - Delays read of old parity, write of new parity
  - Temporarily write update image to log
  - Difference between old and new parity
  - Later, group parity updates into large accesses

Declustered parity

- Reconstruction of failed data degrades performance
- Normal operation: load uniformly distributed
- Reconstruction: non-uniform; parity group becomes a bottleneck
- Declustered parity
  - Divide disk array into redundancy groups
  - Parity calculated over redundancy groups
  - Logically distributed over all disks

Exploiting On-Line Spare Disks

- On-Line spares
  - Immediate reconstruction of data
  - When idle, capacity and bandwidth wasted
- Distribute spare capacity over collection of disks
  - Can use disk bandwidth
- Parity Sparing
  - Uses spare capacity to store parity information
  - Second copy of parity information
    - » Faster updates of write operations
      - Pick fastest parity update

Tape Striping

- **Within or between tape robots**
- **Interleave data from file across tapes**
  - **Can increase throughput on large, sequential accesses**
  - **Does not reduce response time**
- **Requires loading multiple tapes into tape drives**
- **Most libraries have small number of tape drives**
- **Contention**
- **Tape switch operations**
  - **May take minutes**

Performance

- **Sequential accesses**
  - **One request active at a time**
  - **Can use all tape drives in parallel**
  - **Increases throughput, decreases response time**
  - **Very effective technique**

Performance

- **Ranomly distributed accesses**
- **Higher concurrency of simultaneous requests**
  - **Performance significantly worse**
  - **Each access spans several tapes**
  - **All have to be loaded**
  - **Small number of tape drives**
  - **Increase contention**
  - **Longer response times**

Issues

- **Fine-grain vs. Coarse-grain interleaving**
  - **Fine-grain easier to implement**
  - **Coarse-grain hard because append-only**
- **Striping within a robot**
  - **Easy to keep logically connected tapes together**
  - **May be contention for single robot arm**
- **Striping among tape robots**
  - **May reduce contention for robot arm at lower concurrencies**
  - **If small robots, better ratio of tapes to drives**

Issues

- **Synchronization**
  - **Harder than in disks**
  - **Long delays if no free tape drives**
  - **Write errors: rewrite data**
- **Future tape drives**
  - **Faster access times**
    - » **Make tape switches faster**
  - **Higher data rates**
    - » **Less need for striping**