* Google File System (GFS)
  + Inverted Index - index is from words to pages, not which pages contain what words.
  + sustained bandwidth more important than latency
  + most modifications are appends
  + files are 3 way mirrored, labeled with 64-bit unique global IDs
  + GFS master, application, GFS chunkservers
  + uses linux filesystem to store files on the chunkservers.
  + GFS Master
    - only answers chunk locations
    - only single master, you can have backups of master if needed.
  + chunkservers
    - only know what chunks they have. no idea what files are in those chunks.
    - Only master knows what to do with the chunks.
    - sends out heartbeat, includes state changes.
  + operation logs
    - metadata updates are logged
    - global snapshots are taken every so often to truncate logs
    - recovery = latest checkpoint + subsequent log files.
  + Relaxed consistency
    - writes are done at least once, sometimes twice, creating duplicates
  + master grants a chunk lease to a replica - leases are only for updates
    - replica holding the lease determines the order of updates to all replicas
    - leases have 60 second timeouts, can be infinitely extended on heartbeats.
* 10-12 pages is about the size of typical conference paper
* ZFS - the last word in file systems. Solaris
  + Pooled storage
    - Abstracts away volumes.
  + Transactional object system
    - Universal consistency
  + Data integrity
    - Detects and corrects silent data corruption
  + Simple administration
  + Pooled storage
    - Abstraction for malloc/free
    - No partitions to manage
    - grow/shrink automatically
    - Bandwidth always available
    - Shared storage
  + Transactions are built into the file system, no separate log writing
    - Copy-on-write transactions. Never in place.
    - Called waffle (write anywhere file system).
    - Old copies are like snapshots, which is better than old snapshot infrastructure.
    - ZFS birth times keep track of versions.
    - DAG - Directed Acyclic graph, different from a tree. ZFS uses DAG to keep track of snapshots/file system. Multiple trees.
  + Data Integrity
    - Zfs validates entire I/O path
    - Checksum stored in metadata instead of with data. Locality separation.
  + RAID-Z
    - All writes are full-stripe writes
    - Bundles writes so that there are no small writes. No small write problem for RAID-5
    - Checksumming is not hardware- so can be done with any type of disk
  + Top-down resilvering
    - ZFS resilvers the storage pool’s block tree from the root down
  + ZFS scalability
    - Tons of capacity
    - Concurrent everything
      * Byte-range locking: good for parallel
    - Copy-on-write design
      * Hot-spot-free by design
    - Pipelined I/O
      * 24 stage pipeline
      * Maximum possible I/O
      * Architecture -scoreboard
  + Dynamic striping
    - Number of drives can change at any time, old copies stay, no problem for RAID-Z
  + Intelligent Prefetch
    - Assumes multiple threads. Old systems assume 1 thread.
  + Variable block size
  + ZFS clone
  + ZFS send/receive backup
  + Native cifs support (Common Internet File System)
  + Zfs and zones, Zfs root
* Data deduplication
  + Advantages
    - Less disk
    - Less bandwidth
    - Less power
  + Disadvantages
    - More time
    - More complex
    - Latent channels
  + Eager v. Lazy
    - Time/Space tradeoff
    - If you have a lot of temporary files, go with lazy, because you may delete them
  + Fixed v Variable block size
  + A Low-Bandwidth Network File System (good paper)
    - System for slow/large networks
    - Exploits similarities between files/versions
    - Also uses conventional compression and caching
  + On slow networks
    - Make local copies
    - Remote login for text-based things
    - Use LBFS instead
  + LBFS
    - Exploits cross file similarities, especially with previous versions
    - LBFS divides files into chunks, then indexes chunks by hash value
    - Never transfers chunks that recipient already has, identified by hash
    - Provides close-to-open consistency
    - Sets chunk boundaries based on data (variable length)
    - Rabin fingerprints to identify chunks, sha-1 hash for the file on top for the chunkserver
  + Issues
    - Small chunks
      * So small the hashes of chunks are as big as the chunks
    - Large chunks
      * Can not be sent in a single RPC (remote procedure call)
    - LBFS imposes minimum and maximum chunk sizes.
  + File Consistency
    - Based on leases on files
    - No need for write leases
    - Last to close overwrites everything
  + LBFS is designed for low bandwidth networks, If you have a LAN/fast network, the fast network will usually beat out the slow networks.
  + LBFS’s benefits do not really increase as bandwidth increases.
* Business continuity
  + B.C. is
    - The preparation for,Response to and,Recovery from,Application outage that adversely affects business operations
  + Solutions address
    - Systems unavailability,Degrade application performance, and Recovery strategies
  + Why
    - Lost productivity
    - Damaged reputation
    - Lost revenue
  + Information availability
  + Recovery Point Objective/Recovery Time Objective
    - RPO, the time to the restore point before the crash. Measured from the crash backwards.
    - RTO, the time it takes to get back online.
    - RPO mediums
      * Tape = days/weeks to recover
      * Periodic = hours/days
      * Asynchronous replication = seconds/minutes
      * Synchronous replication = seconds or less
    - RTO
      * Fault detection,Recover the data, Restart
    - RTO mediums
      * Tape restore = days/weeks
      * Manual migration = hours/days
      * Global cluster = minutes
  + Disaster recovery or Disaster resistant
    - Most business critical apps have some level of data interdependencies
    - Recovery
      * Restoring previous copy of data (applying loss to bring to consistent point)
      * Implies B/U
      * Requires manual intervention during restore and/or recovery
    - Restart
      * Process of restarting mirrored consistent copies of data & apps
      * Restart time is comparable to restart after power off
      * No manual (if desired)
  + Data unavailability
    - Planned outages - 87% - upgrades
    - Unplanned outages - 13% - h/w failure, db corruption, human errors
    - Disaster - less than 1%- floods and such
  + Technology solutions
    - Zero-RPO has to be baked in from the beginning.
* Backup + recovery
  + What is a BU?
    - Additional copy
    - Create for use when primary is lost
  + Implement
    - Copy data
    - Mirror - then copy
    - Remote backup
  + Retention Periods
    - Operational
      * Data on primary
      * After restore move off
    - Disaster recovery
      * failure/restore
    - Archival
      * Regulatory
  + Hot backup
  + Cold backup
  + Backup granularity & levels
    - Full Backup
      * Everything
      * Coarse, compared to Cumulative and incremental.
      * Longest RPO,  Shortest RTO
    - Cumulative Backup
      * Everything+full mini-backup since then
      * Short RPO, long RTO
    - Incremental backup
      * Everything+application of all incrementals since the last checkpoint.
      * Usually takes longer than Cumulative because incrementals total usually are more since most changes are not mutually exclusive.
      * Shortest RPO, Longest RTO
  + Differences between Backup/recovery VS archive
    - Secondary copy vs primary
    - Recovery vs retrieval
    - Improve availability vs operational
    - Short term vs long term
    - Overwritten vs maintained/appended
    - RC vs regulatory compliance
  + Replica
    - A copy
    - Uses
      * backup(reliability),recovery(availability),Decision support,Test ,platform,Migrate,Compliance
    - Kind of replica
      * PIT - point in time
      * Continuous - zero RPO
    - Local replication
      * Array do it’s own replication, through hardware or software
    - Remote replication
      * Synchronous replication, where all backups are written before write is considered down. Synchronous is the only way to get zero RPO. Synchronous is usually also within 200km from each other.
      * Asynch replication, where first copy is written and the system handles the backup.
  + Remote replication
    - Log shipping
      * Aggregates transactions to lower costs. RPO of a few seconds.
    - Array based
* Hard disk power
  + Three states
    - Active,standby,off
  + Power dominated by motor
  + Save power by turning off disk
  + Fine grained states. Can be simulated
    - Active
      * Seek energy+read/write energy
    - Idle
      * Idle transition energy - additional delay introduced
      * Idle energy
  + Two-Trace method
    - To estimate average power for stage s, construct two traces that differ only in stage S
    - P=measured energy difference/measured time difference.
  + Experimental results
    - Real-world(minutes/hours) vs synthetic traces(small I/O benchmarks, milliseconds).
  + Dempsey disk power simulator
    - More accurate than 2 or 3 parameter power predictions
  + Long periods
    - Turn hard disk off during expected idle periods.
    - More volumes mean that more drives can be off for longer since they aren’t use as often.
* SNIA
  + Storage Networking Industry Association
  + Common architectural vocabulary
  + Cleaner design
  + SNIA shared storage model
    - Application
    - file/record layer
      * database/file system
    - Block layer
      * Block aggregation
        + Host (software raid)
        + Network (sans)
        + Device (raid array)
      * Storage devices
  + file/record layer
    - database/file system
    - Caching and coherency
    - Byte mapping
      * + Host side
        + SN-based

Within the NAS network.

* + - * + Device-based

On the device side

* + Block layer
    - Block aggregation
      * + Address mapping
        + Striping, raid
    - Storage devices. Disks and such.
    - Blocks on top of blocks.
      * + host , network, device, storage devices.
  + Network layers
    - Application,Operating system,File layer,Block layer,Storage device
    - network/interface between each of the above steps
* WAFL
  + Write anywhere file layout
  + Use inodes
  + Metadata lives in files
  + 3 metadata files
    - Inode file - many inodes
    - Blockmap - free block
    - Inode map - free inodes
  + WAFL can write metadata anywhere
  + WAFL has lots of trees to keep track of everything. Uses links to reference data.
  + Copy on write. Lazy copy, slower write. Only write when versions change.
  + File consistency + noRAM
  + Consistency point
    - Special snapshot
    - Cp created every few second
  + Restart
    - Hard disk resets to most recent cp
    - Fast
  + Between CPs
    - 1000s of requests
    - Writes are to unused blocks
    - On disk image will not advance until next CP
    - Store log of requests in NVRAM
    - 2 part NVRAM
      * + When one part fills
        + Flush to disk - write to other
        + Write log - faster / less
* DAFS
  + Direct access file system
  + Netapp research paper, didn’t really get any traction but still neat
  + Takes advantage of RDMA (remote direct memory access)
  + Storage can
    - Shared
    - Managed
    - Scaled
  + Independently from
    - Applications,Oss,Architecture (MP)
  + four basic
    - Raw block
      * + Hi perf
        + Application must provide all data management
    - Local FS
      * + Good performance
        + Sharing and supported
    - Cluster FS
      * + On each node execute block protocols
    - File access protocol (NFS, CIFS)
      * + Benefits FA

Virtualization

Fine-grained data management

Data sharing with protection

* + - * + Costs FA

CPU overhead

* + General parallel file system (gpfs) native raid
    - High throughput because of wide striping
    - Parallel computing, raid, general file system
    - External raid controller becomes the limiting factor for standard solutions
    - Declustered RAID 6 makes critical errors(2 failures) less common by spreading them out across the servers.
* DFS design
  + NFS - sun
    - Used by a lot of NAS devices
  + AFS - andrew file system
  + Views (hierarchical file system)
    - Global
      * All clients have same view of filesystem
      * Single global directory structure
      * AFS
    - Local
      * Clients can have different views
        + /etc/hosts
      * Mount remote directories locally
  + Comparison
    - Global is easier to administer
    - Local is more flexible
  + Location transparency
    - You can see/specify which server is holding a file.
  + State
    - AFS - stateful
    - NFS - stateless
    - Stateful
      * Client opens file on server
      * Client access file
      * Client closes file
    - Stateless
      * Client resolves path via host
      * Client gets/puts blocks of a file
    - Stateful vs stateless
      * Stateless recovers way faster than stateful because there is no state to recover.
      * Stateful can manage consistency with locks. Stateless can do leases, but it’s not a true lock.
      * Stateless is more scalable in terms of users, because stateful needs to remember the state of all clients.
  + Caching
    - For performance
    - At any level, client memory or disk. Server memory or disk.
    - For stateful, it’s easy for server to notify clients of changes. Client caches can stay fresh because of this.
    - AFS - full file transfer
  + Centralization
  + Replication
* Hadoop
  + Reference architecture
    - Hadoop 1
      * MR (map reduce)
      * HDFS (hadoop file system)
    - Hadoop 2
      * Yarn (yet another resource negotiator) - reliable distributed computer of COTS (commodity off the shelf)
      * Containers provide reliability
      * MR2, storm, spark.
      * HDFS
    - Near-data
      * Local storage, lots of data, move computation to the data.
    - Map-reduce
      * Map splits things apart
      * Reduce solves the parts
      * Embarrassingly parallel, map independent.
      * Reduce size must be much smaller than map size
    - Every node is a compute node and a data node
      * Triple replication
      * Optimized for sequential mostly read
      * HBase - noSQL
        + Using HDFS
        + Not normalized
  + HPCC systems
    - Distributed computer, like hadoop
    - Lexis nexus
    - Created in 2001, open sourced on 2011
    - Thor - for computing. Single large bandwidth
    - Roxie - for storing. Low latency many connections
    - Declarative language ECL