AVAMAR™

# Design and Implementation of a Storage Repository Using Commonality Factoring

IEEE/NASA MSST2003

April 7-10, 2003

Eric W. Olsen

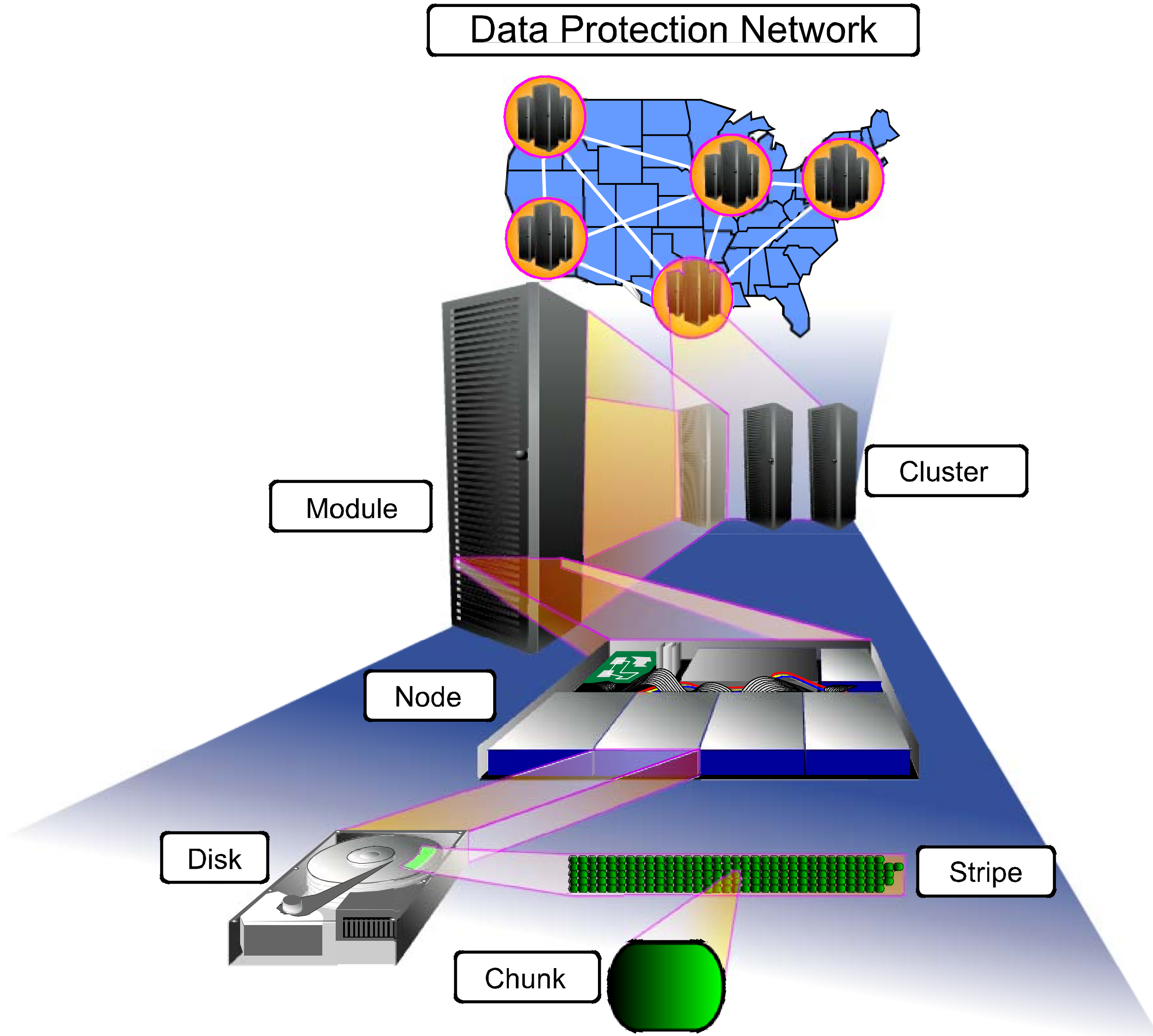
# Axion Overview

AVAMAR

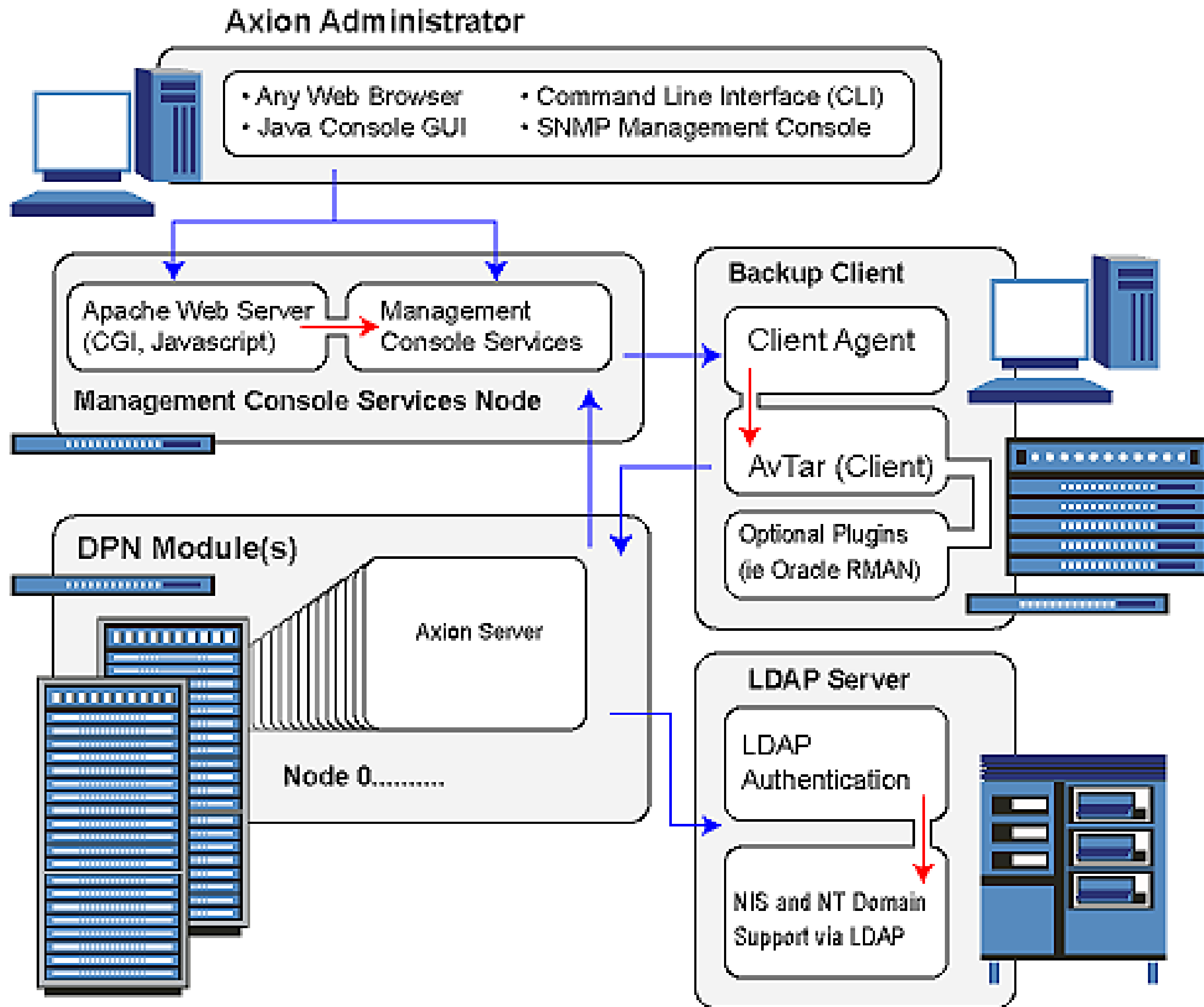
- Potentially infinite historic versioning for rollback and file access
- Block-level commonality factoring
  - Adds to storage only small chunks of changed data that has not already been stored
- Redundancy
- Scalability
- Efficient resource utilization
  - Takes less time to restore than tape
  - Provides higher integrity and no degradation over time compared to tape
  - Provides remote access

# Storage Components





# Axion System Architecture



# Axion Software Architecture

AVAMAR



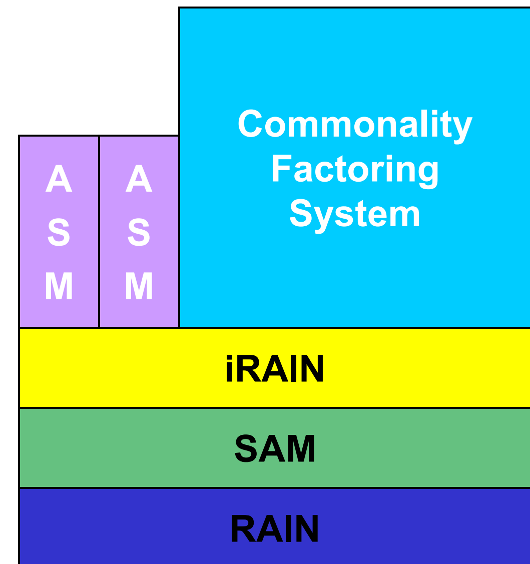
## Software Architecture

- RAIN™ (Redundant Array of Inexpensive Nodes)
- SAM (Storage Allocation Manager)
- iRAIN™ (Intelligent RAIN)
- CFS (Commonality Factoring System)

**Distributed QoS Manager**

**Storage Allocation Manager (Storage OS)**

**Hardware Layer**



# Commonality Factoring

AVAMAR

- Beyond fixed content: extends support for changing files
- Storage efficiency
  - Sub-file commonality can dramatically decrease storage requirements for certain applications
  - Reduces 'storage under management' costs
- Network efficiency
  - Remote or local signature processing
  - Can dramatically reduce network traffic
  - Optimized with hash cache
- Supercomputer functionality: highly parallel in operation
  - Better load balanced
  - Parallel operations across all resources for faster read and write
- Support for directories and file systems
- Large file support: no file size limitations

# Commonality Factoring Implementation

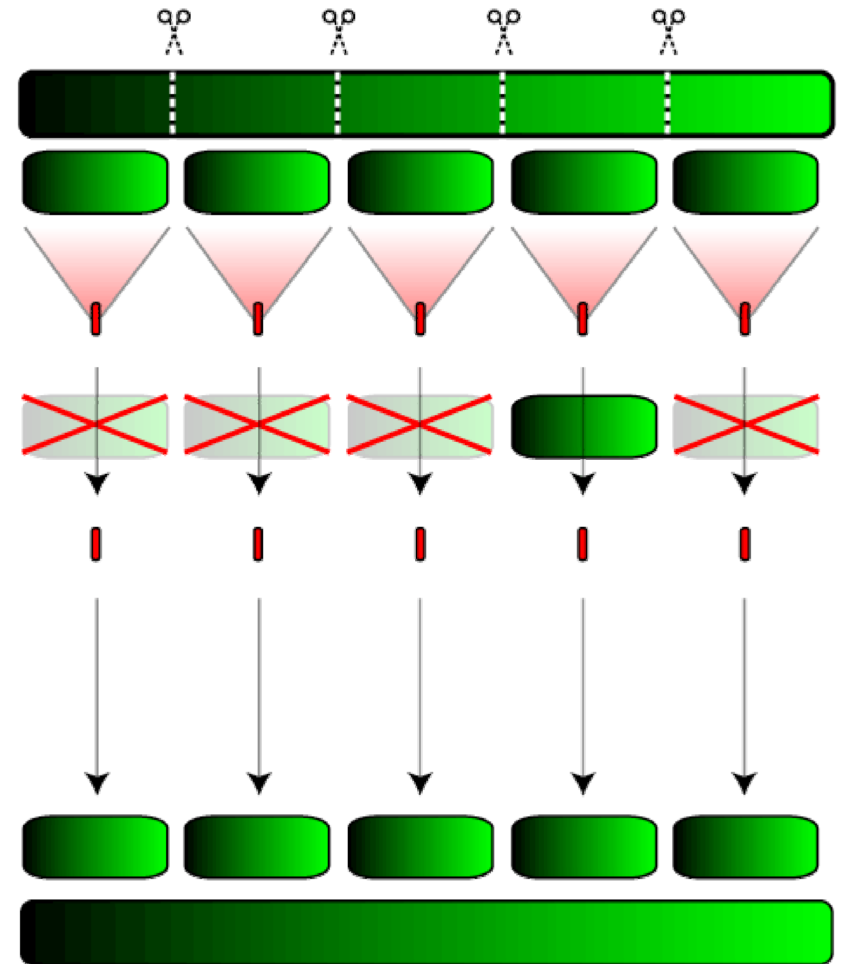
- Searches for commonality at the granular variable data chunk level for all users of the system
- Stores data chunks for all users of the system and assigns a hash address to each data block
- Reduces:
  - Bandwidth requirements
  - Cost of storage
  - Storage requirements
- Increases:
  - Data availability
  - Data integrity
  - Efficiency of storage
- Uses unique data hashes and stripes to recreate quickly any version of a user's system or any file version at any point in the past
- Uses TCP/IP for data transmission during snapup and restore
- Performs full backups at the speed of incremental backups

# Operation Overview

AVAMAR

## Encode:

- Start with original data stream
- Partition stream into chunks
- Hash each chunk
- Eliminate the redundant
- Store the unique
- Hashes now substitute for data

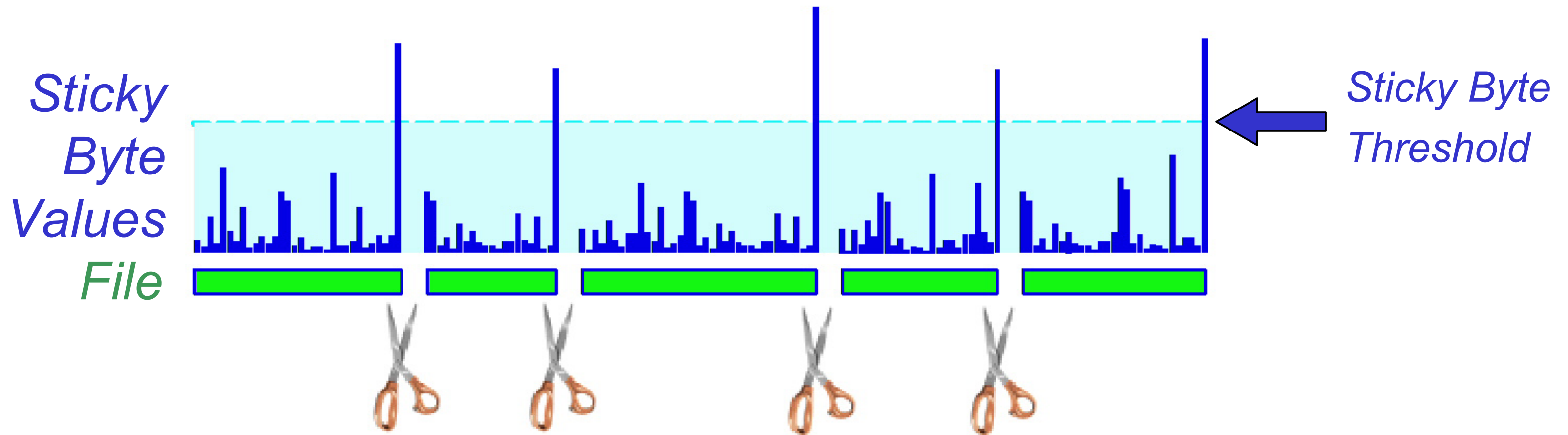


## Decode:

- Use hash to lookup chunks
- Join chunks to recreate data

# Sticky Byte Factoring

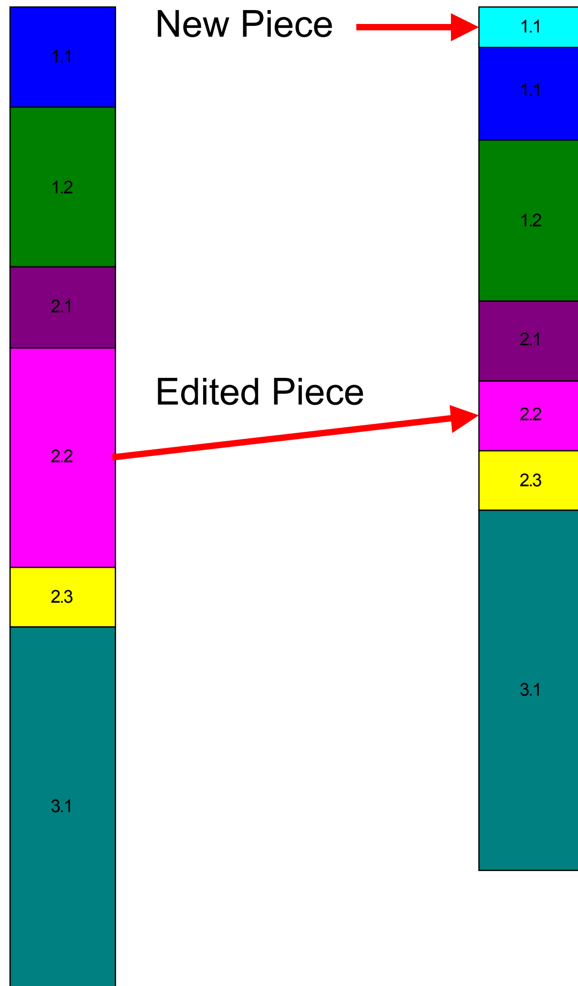
AVAMAR




*Sticky Byte Factoring* orchestrates commonality by cutting large files into variable length chunks in a deterministic way.

# More Sticky Byte Factoring

AVAMAR




*Sticky Byte Factoring* orchestrates commonality using traits from probability theory to divide input text into consistent data elements, reaping commonality across:

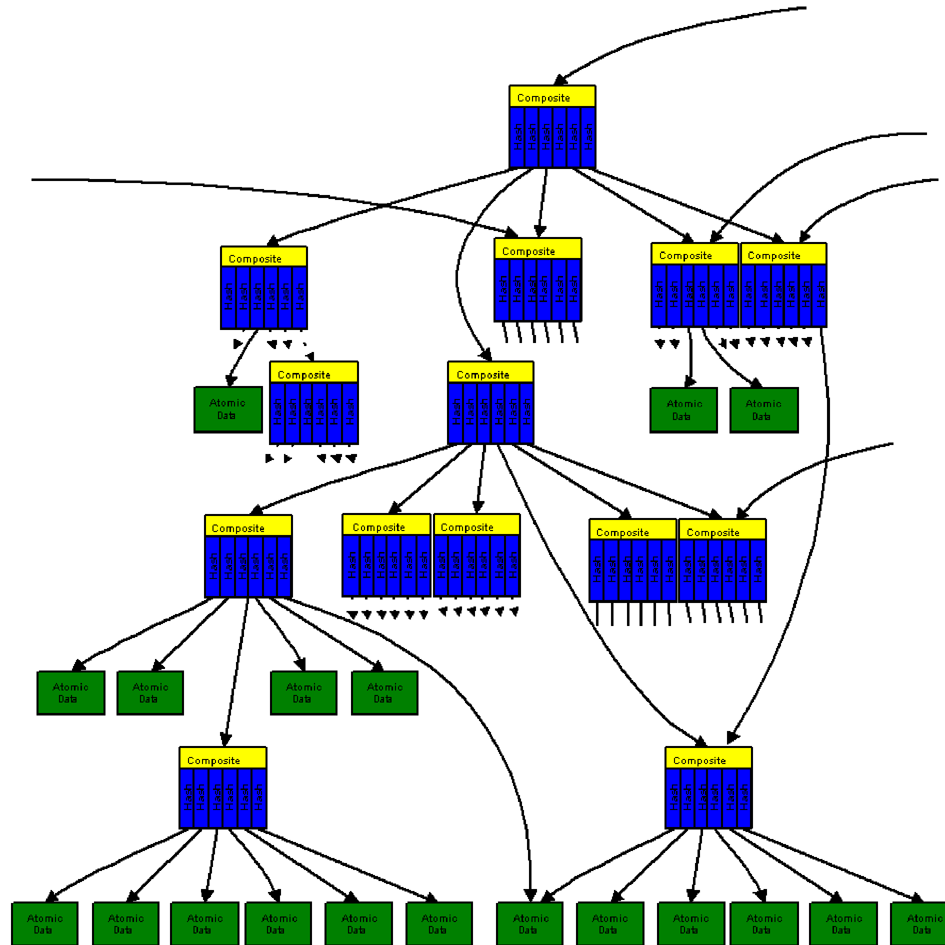
- Different applications
- Different operating systems
- Different authors
- Different times

It is a more powerful method for reducing storage consumption than any conventional compression method.

# More Commonality Factoring








Commonality factoring removes duplicate sequences from any data image:

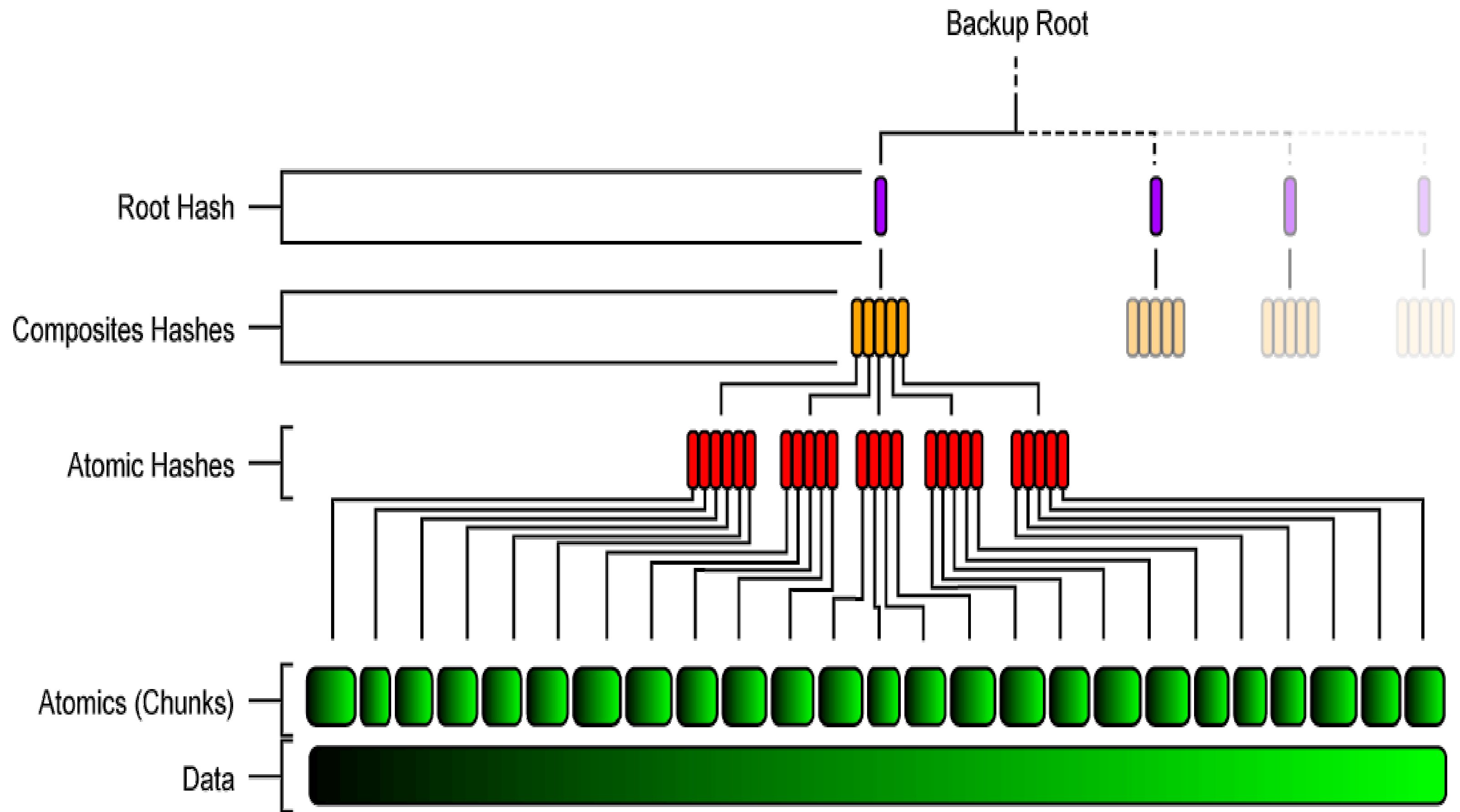
- File systems – directories and files across operating systems
- Disk devices – raw database images, arbitrary block devices
- Network transmissions – email, remote file systems, BCVs can be deeply factored

Storage for historical snapshots or backups generally reduced to less than 2% original size.

Query operations accelerated by factoring, architecture ideal for distributed search operations.

# Data Image Reconstruction

AVAMAR



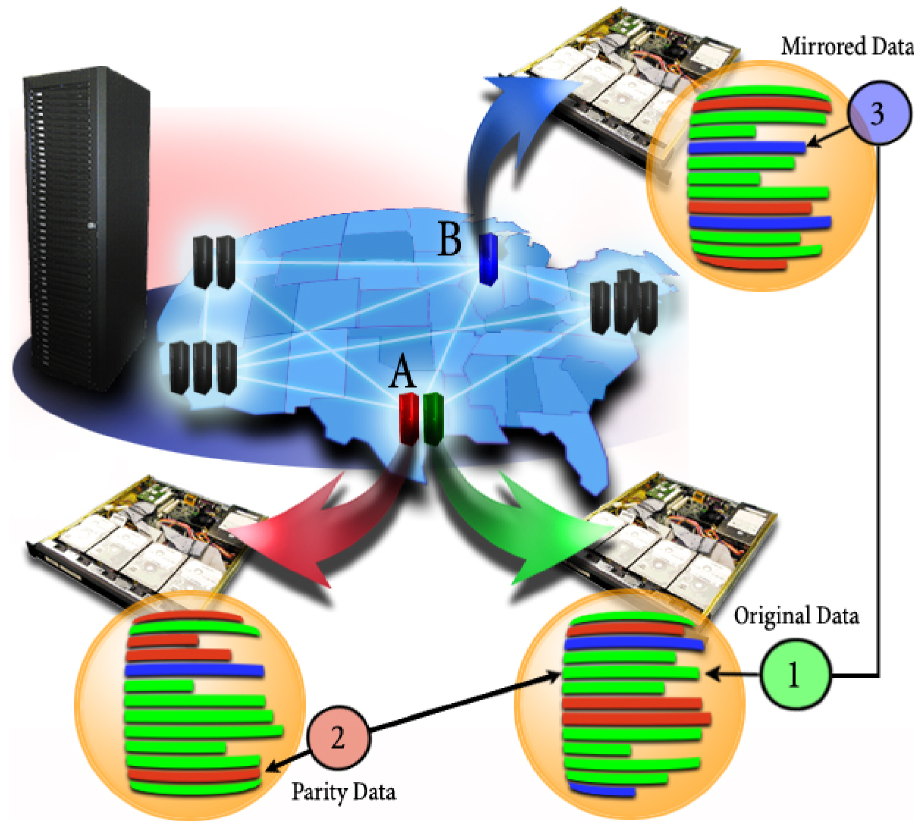
# Compression vs. CFS

AVAMAR

| Characteristics | Compression | Commonality Factoring |
|---|---|---|
| **Scale of operation** | **Small scale** | **Large Scale** |
| Low end: | Thousand bytes | Billion bytes |
| High end: | Million bytes | Trillion trillion bytes |
| **Scope of operation** | **Within a single file** | **Across files**<br>**Across systems**<br>**Across time** |
| **Applications** | **Data set specific – Jpeg for still images, Mp3 for audio, Zip for text and binary.** | **Works on all data sets** |
| **Theory of Operation** | **Dictionary based scan for token substitutions *(i.e. LZW, RLE, DCT, Wavelets, MP3, Mpeg).* Lossy adds filtering to improve substitution rates.** | **Unorchestrated substitution, hashing eliminates comparisons and scaling issues, lossless conversion.** |
| **Typical Substitutions** | **10s of bytes for 2 or 3** | **1K..1M of bytes for 20** |
| **Typical Reduction Factors** | **2:1 up to 4:1 for lossless**<br>**10:1 up to 100:1 for lossy** | **Arbitrarily high, always lossless**<br>**Typically 100:1 to 1000:1** |

# Local and Global Fault Tolerance

AVAMAR




In this example a primary data image (1) is being protected with one dimension of local parity (2) and one dimension of global mirroring (3).

Data is depicted being stored in several ways inside the system:

- In the lowest (A), the right module holds a node containing the primary image.
- The left module in that same cluster holds a copy of the primary in local parity *(XORed with other images within the same cluster).*
- In the upper right a mirror image of the primary is being held in a separate cluster (B).

# Performance

AVAMAR

- Hash lookup is simple and direct: O(1)
- With 8K average chunk size, 1 Gbyte of unique data becomes 128K hash table entries
- Storage layout can be tuned for speed, capacity, locality, bandwidth, fault tolerance
- Fault tolerance is synchronous on write
- Commonality dominates write performance
- Client performance usually dominates read performance

# Recent Read Benchmark

AVAMAR

- Standard Axion 2x7 server was able to achieve an average sustained restore performance of 20.8 MB/sec. This would restore 500 GB in approximately six hours, 41 minutes.
- Test results indicate the performance of the application client processor (Dell Model PE 2500 server, with dual P3-1266 Mhz processors with 1 GB RAM) restricted the restore performance to 20.8 MB/sec. A more powerful application server will support faster restore performance.

# Recent Write Benchmark

Performance by Chunk Size

Server: Cobalt1 --parity=N8,F4 --matchbits=8 with jfs and 4K NIC ring buffer

Client: Core, 5 runs: v0.9.47-6 avtar --nocache --randchunk=20000 --maxpending=800

Average Throughput (MB/sec)

15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

0 4000 8000 12000 16000 20000 24000 28000

Chunk size (bytes)

1x11 N8
1x9 N8
1x7 N8
2x7 N8+F4
1x5 N8
2x5 N8+F4
1x1 none
1x3 N8+F4
2x3 N8+F4

Cobalt1-1x1-none-core-v0.9.54-0-021003-0116a
Cobalt1-1x3-N8-core-v0.9.54-0-021003-0116a
Cobalt1-1x5-N8-core-v0.9.54-0-021003-0116a
Cobalt1-1x7-N8-core-v0.9.54-0-021003-0116a
Cobalt1-core-v0.9.54-0-1x9-N8-021003-1140a
Cobalt1-core-v0.9.54-0-1x11-N8-021003-1140a
Cobalt1-2x3-N8F4-core-v0.9.54-0-021003-0116a
Cobalt1-2x5-N8F4-core-v0.9.54-0-021003-0116a
Cobalt1-2x7-N8F4-core-v0.9.54-0-021003-0116a

# Impact of Network Latency on Two Module Axion Server

**Client Initialization with 100% Unique Data**
**Axion Release 1.1**
**5 Mbps Simulated Link (Dummynet)**

FTP windowing minimizes impact of latency to a point

Axion minimizes impact by performing many operations simultaneously

SCP synchronous transfer slows in proportion to latency

Average Transfer Rate (KB/sec)

0
100
200
300
400
500
600
700

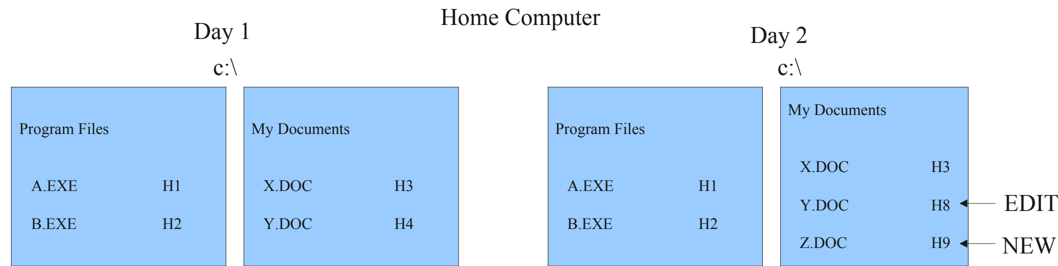
0 20 40 60 80 100 120 140 160 180 200

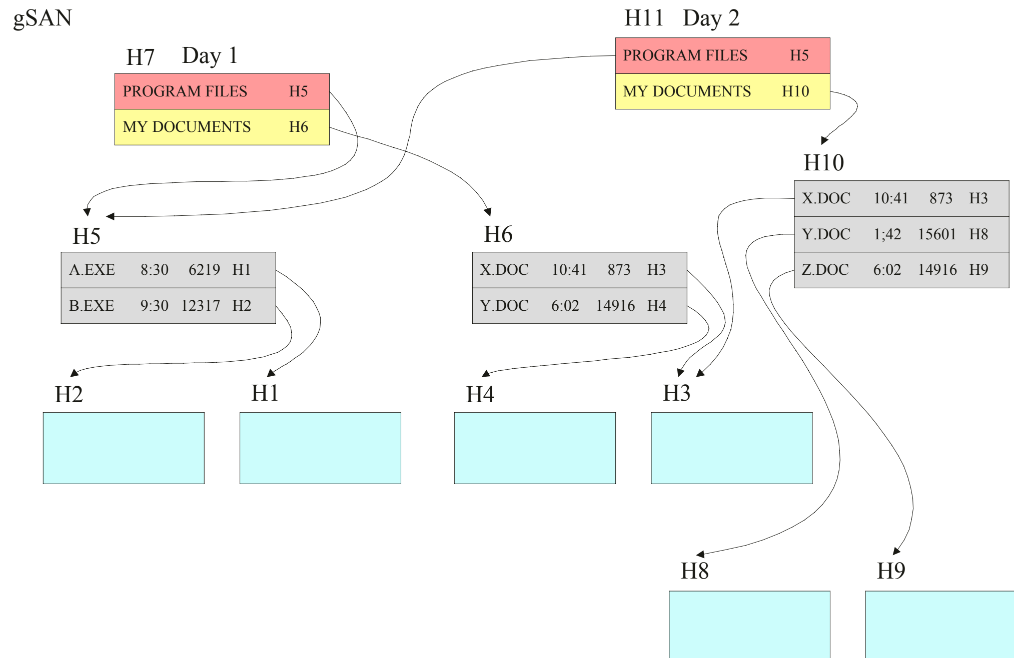
Roundtrip Network Delay (ms)

- FTP: From Client to Remote Module
- SCP: From Client to Remote Module
- Axion Snapup: Client+Module Colocated, Other Module Remote
- Axion Snapup: Both Modules Colocated, Client Remote

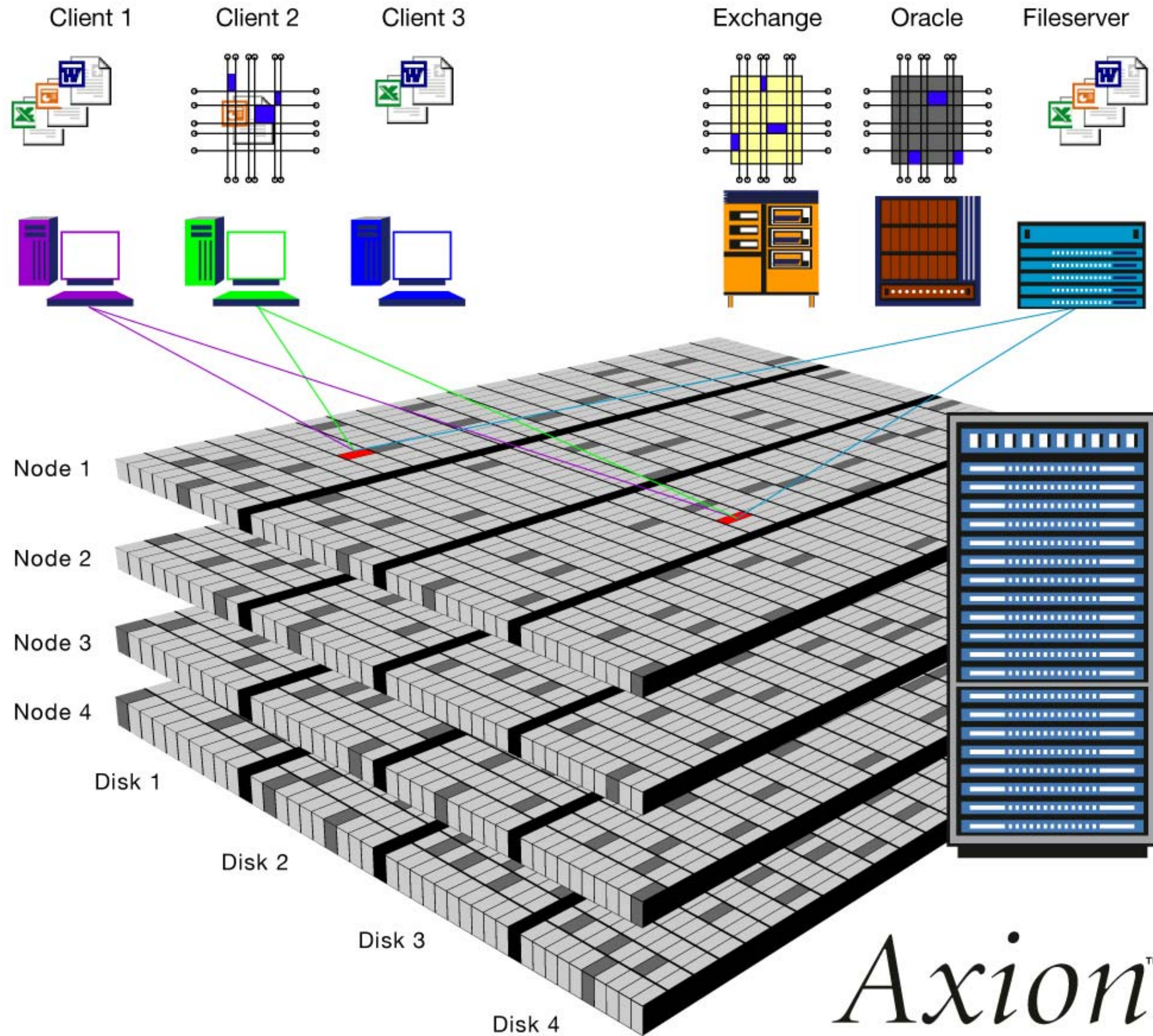
# Day-to-Day Commonality

NHFS: Backup Over 2 Days



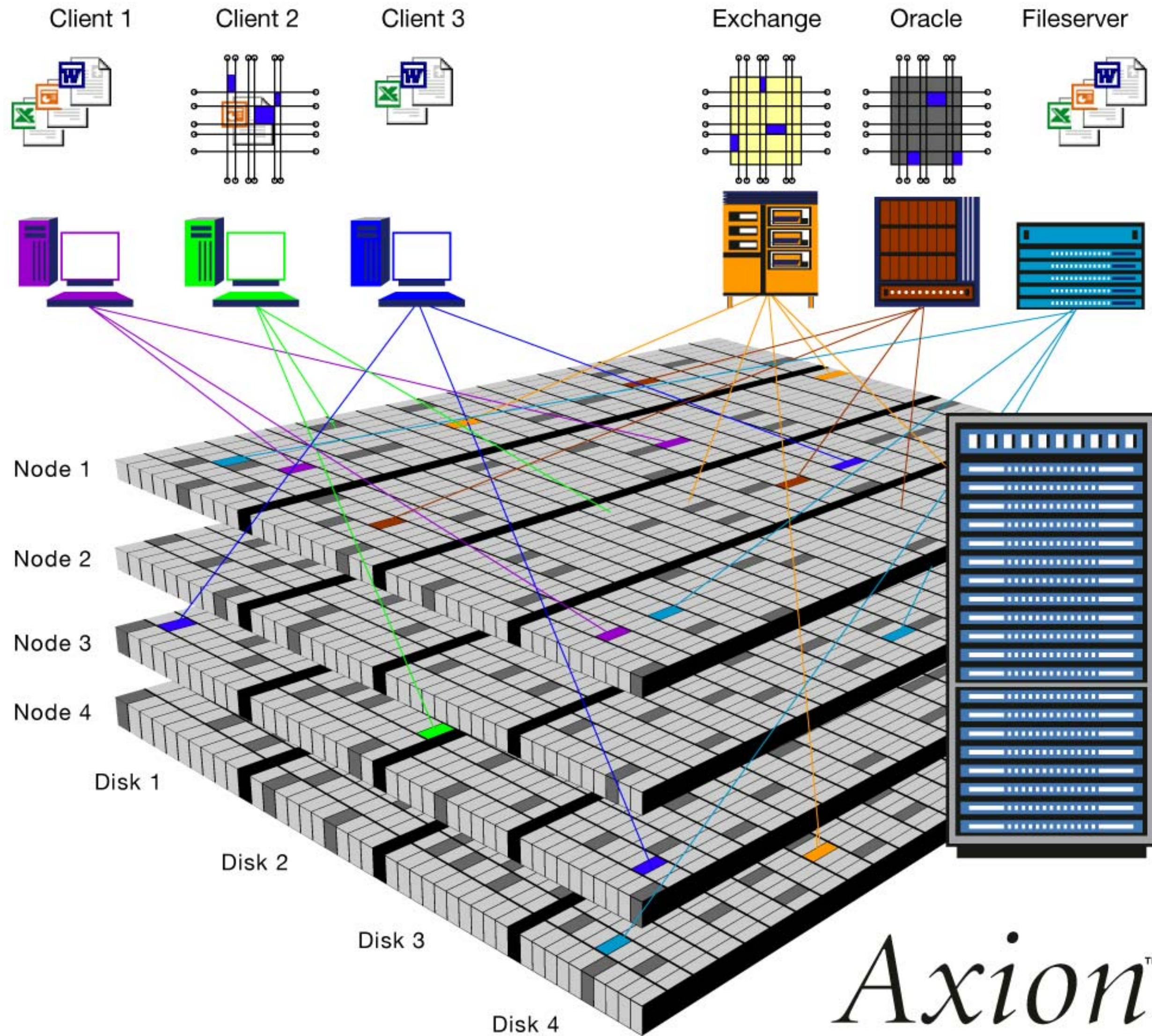
# Day-to-Day Changes






AVAMAR™

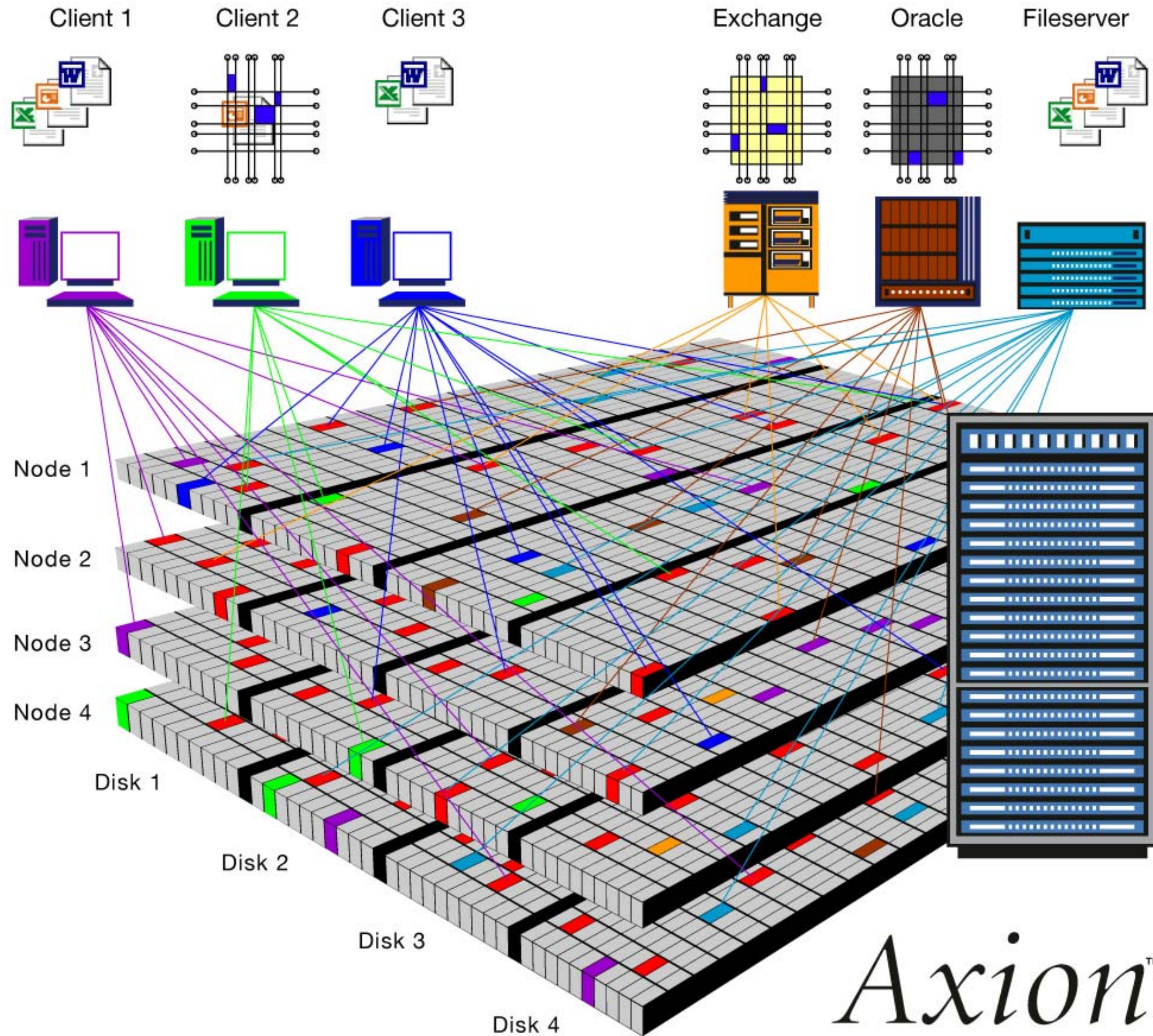
# Day-to-Day Changes





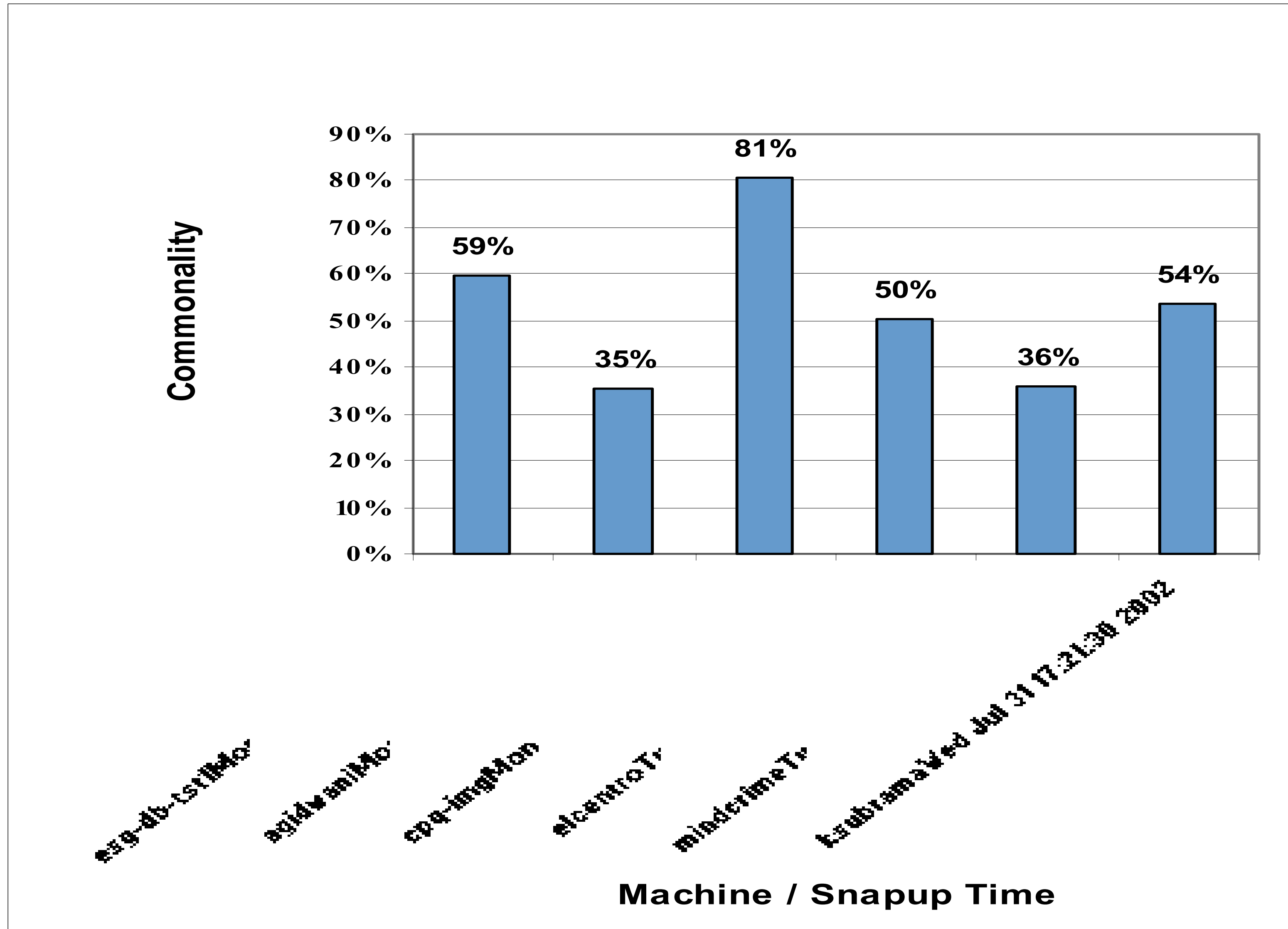

AVAMAR™

# Day-to-Day Changes



# Initial Snapup Commonality





# Average Tape Compared to Average Snapup

Average Tape (Full & Incremental) to Average Snapups (Including Overhead)

Average Tape Full and Incremental
Average Snapup Data Sent to DPN

MB

100,000
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0

19,728
168
16,607
457
50,669
1,009
59,679
39
6,063
146
20,030
703
44,709

esg-db-tst1 Windows NT
agidwaniWindows NT
cpq-imgWindows NT
elcentroSolaris
mindcrimeSolaris
ksubramaWindows NT
Weighted Average

Machine

| % Snapup to Tape | 0.082% | 1.010% | 0.902% | 1.691% | 0.643% | 0.731% | 1.572% |
|---|---|---|---|---|---|---|---|

# Commonality Across Systems

Commonality Across Systems: Windows

| | |
|---|---|
| Average Initial Snapup | 65.80% |
| Average Snapup | 99.88% |

Commonality Across Systems: UNIX

| | |
|---|---|
| Average Initial Snapup | 48.99% |
| Average Snapups | 99.77% |

Average Snapup to Tape Incremental Savings

| | |
|---|---|
| Average Snapups Windows Clients | 79.76% |
| Average Snapups Unix Clients | 86.54% |

Commonality Across Systems: Combined

| | |
|---|---|
| Average Initial Snapup | 59.64% |
| Average Snapups | 99.88% |