LINUXCON EUROPE 2012

GLUSTER Workshop

Disperse translator

Ramon Selga

rselga@datalab.es

Xavier Hernandez

xhernandez@datalab.es

data lab

Barcelona, November 8, 2012

- Introduction
- Features
- How it works
- Architecture
- Reliability
- Examples
- Current state
- Future

- **Introduction**
- Features
- How it works
- Architecture
- Reliability
- Examples
- Current state
- Future

Introduction

Main idea:

- Striped volumes offer a lot of space but do not support faulting bricks
- Replicated volumes allow a configurable degree of fault tolerance but eat a lot of space
- We want to build a volume that has a configurable degree of redundancy with a small space waste
- Solution: Disperse the data and add redundancy

Introduction

- Currently available volume types
  - Striped
  - Distributed
  - Replicated
  - Distributed+Replicated
  - Striped+Replicated
  - Distributed+Striped
  - Distributed+Striped+Replicated

Introduction

- New volume types for Gluster
  - Dispersed
    - Based on erasure codes
    - Configurable level of redundancy
    - Better utilization of physical storage space
    - Optimized bandwidth usage
    - Limited I/O performance
    - Small performance loss when degraded
  - Distributed+Dispersed
    - Improved I/O performance

- Introduction
- **Features**
- How it works
- Architecture
- Reliability
- Examples
- Current state
- Future

Features

- Configurable level of fault tolerance
  - Volumes can have any number of bricks (B)
  - A level of redundancy (R) must be defined
    - Minimum allowed value is 1

      At most 1 brick can fail at the same time without loss of service nor data.

      For a 0 redundancy, you can use **stripe** or **distribute**.
    - Maximum allowed value is $\left\lfloor \frac{B-1}{2} \right\rfloor$

      Almost half of the bricks can fail at the same time without loss of service nor data.

      To tolerate the failure of half of the bricks, you can use **replicate**.
    - The effective space is reduced (B-R)
  - Redundancy is distributed evenly amongst bricks
  - Tradeoff between reliability and available space

Features

- **Minimize storage space waste**
  - Each file is divided into chunks of size S
  - Each chunk is split into fragments
  - Additional redundancy fragments are generated
  - Each fragment is stored on one brick
  - The proportion of wasted space is $\frac{R}{B}$
    - You can make this value as small as desired

Example:

6 bricks (B=6) of 1 TB and redundancy 2 (R=2)

Total space:	6 TB
Wasted space:	2 TB (33%)
Effective space:	4 TB (67%)

Features

- Reduced bandwidth usage
  - Reads
    - Always read B - R fragments of size $\frac{S}{B-R}$
      - No overhead.
  - Writes
    - B – F fragments of size $\frac{S}{B-R}$ must be updated
      - If not a full chunk write, a read must be made (S bytes)
      - Always strictly less than 3S
      - On average it's commonly near 2S (or lower if read not needed)

Example:

6 bricks of 1 TB with redundancy 2. Read/Write size 4KB (S=4096)

Read overhead: 0 bytes (*)
Write overhead: 2048 bytes (6144 bytes if read needed)

Features

- Limited IOPS
  - Each brick stores a fragment of each chunk
  - Reads
    - R bricks do not need to be accessed
    - Some reads can be served in parallel
  - Writes
    - All alive bricks are accessed
    - No parallelism is possible
  - Degradation does not have a great impact
  - Distribute translator can improve that

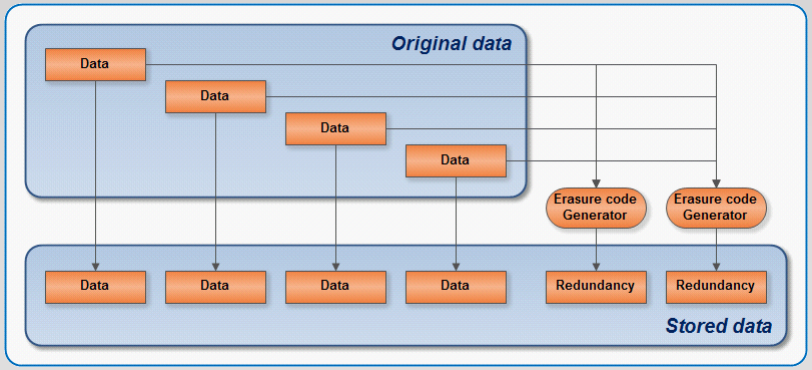
Features

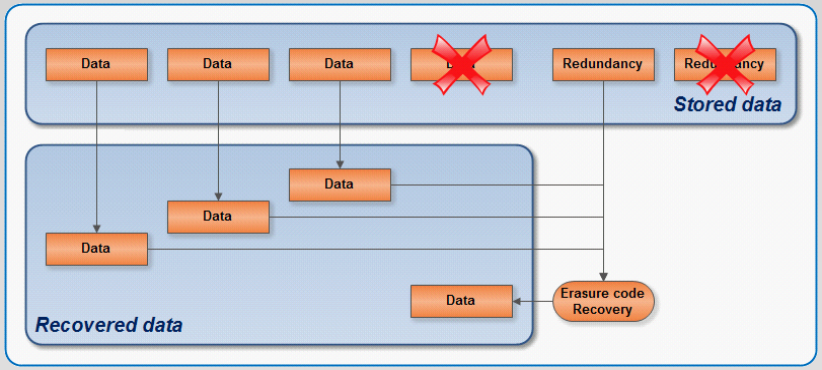
- **Lock-Free Self-Healing**
  - Based on a new healing translator running on each server
  - Managed only by one client per file basis
  - Data healing is handled without any lock held
    - Metadata requests are refused on the brick being healed
    - Read requests are only served if belong to an already healed area
    - Write requests are always handled and have priority over healing
    - The healing client is allowed to read/modify data or metadata

- Introduction
- Features
- **How it works**
- Architecture
- Reliability
- Examples
- Current state
- Future

How it works

- Based on erasure codes
  - Fast implementation of the Rabin IDA (Information Dispersal Algorithm)
  - R additional fragments are computed from a set of B – R data fragments
  - Any data fragment can be recovered from any subset of B – R fragments (data or redundancy)



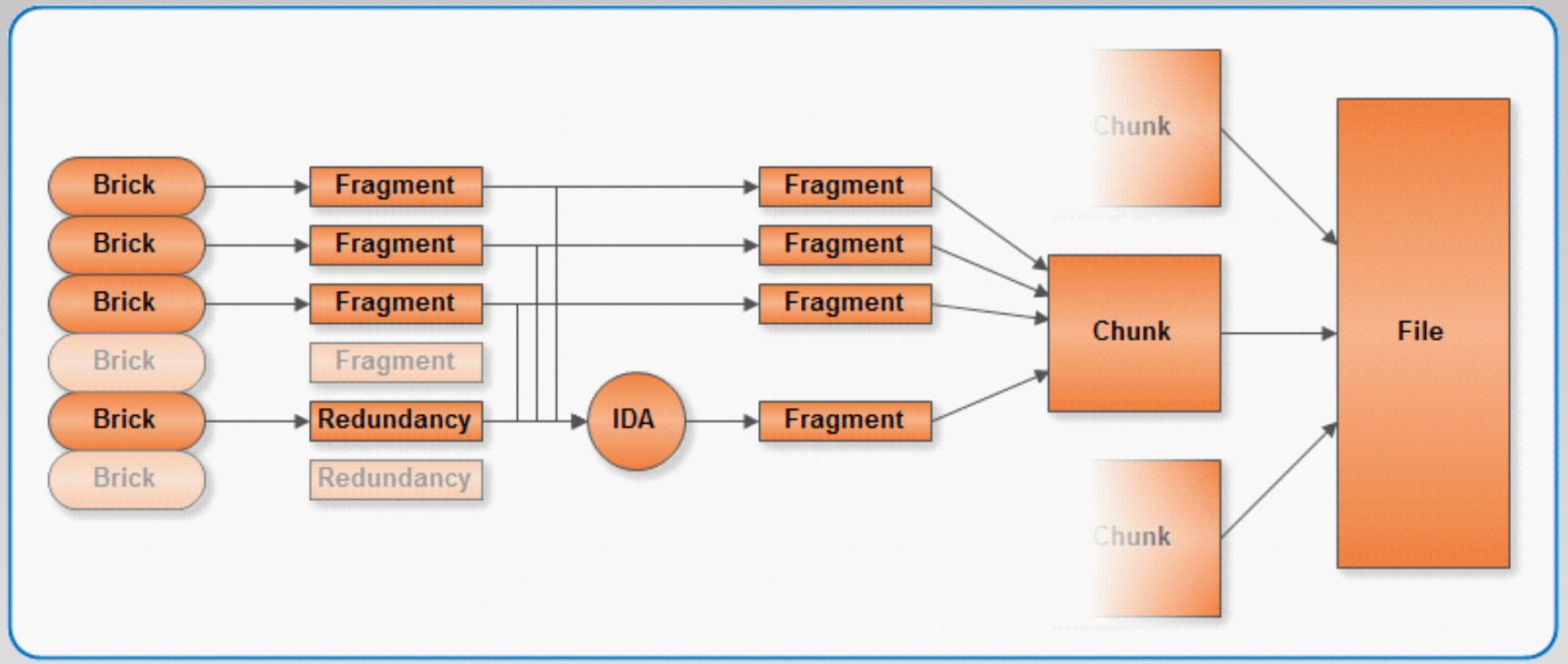


How it works

- Each request is mapped to the involved chunks of the file
  - The chunk size can be customized
  - The selected value may affect performance
  - It depends on access patterns and file sizes
- For read requests, B-R fragments of each chunk are read from B-R bricks
- For write requests, incomplete chunks are read and then updated
- If one or more bricks are down, their fragments are recovered using IDA

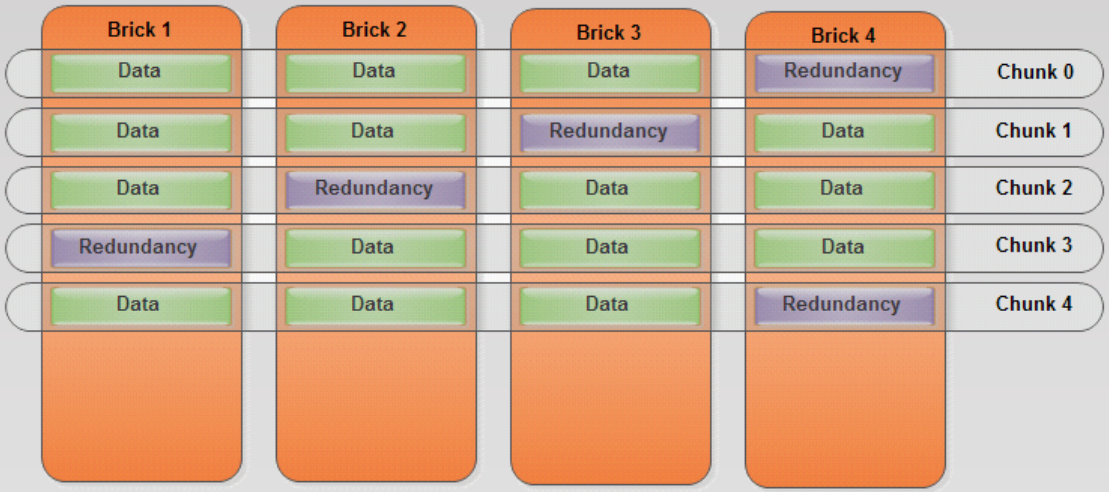
How it works

- Read operation



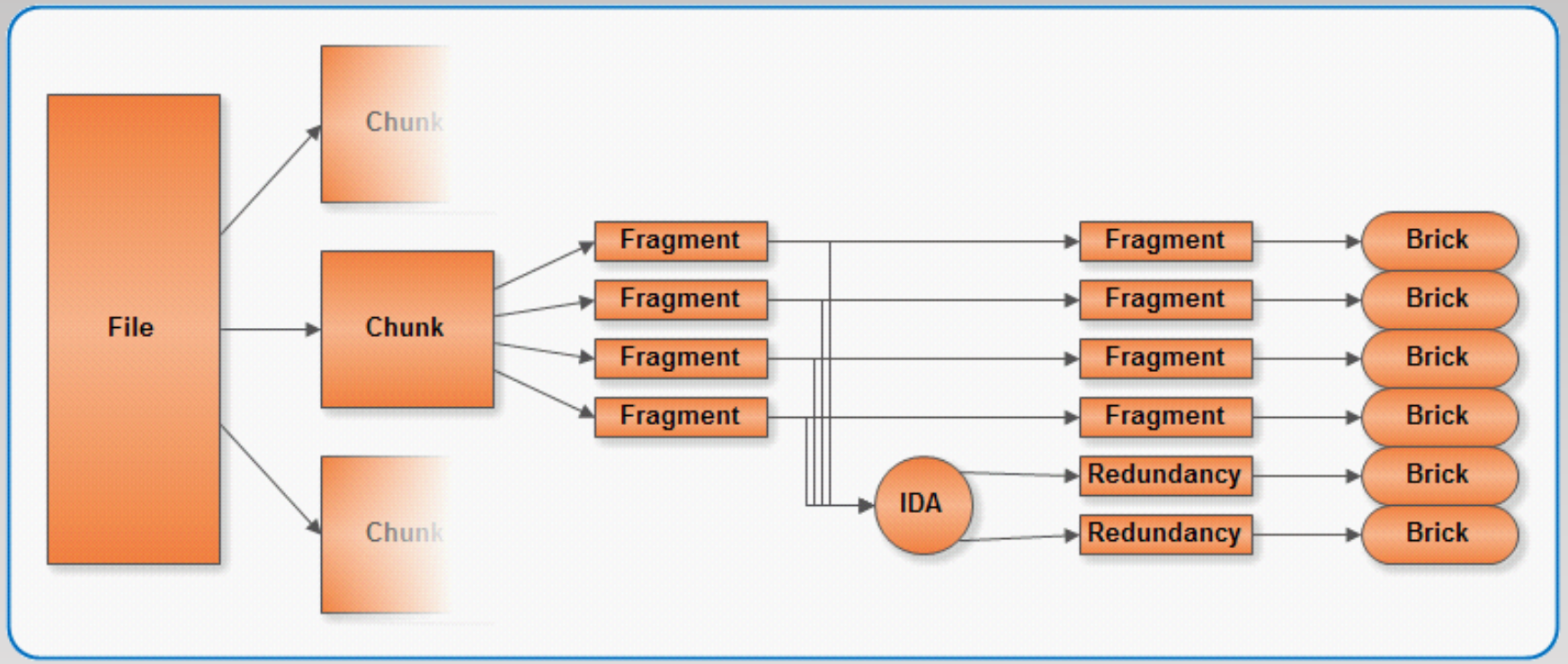
How it works

- Read operation
  - When possible, all fragments are read from data fragments, not redundancy, to avoid using IDA
  - Redundancy is spread over the bricks in a way that, in average, it distributes the load



How it works

- Write operation

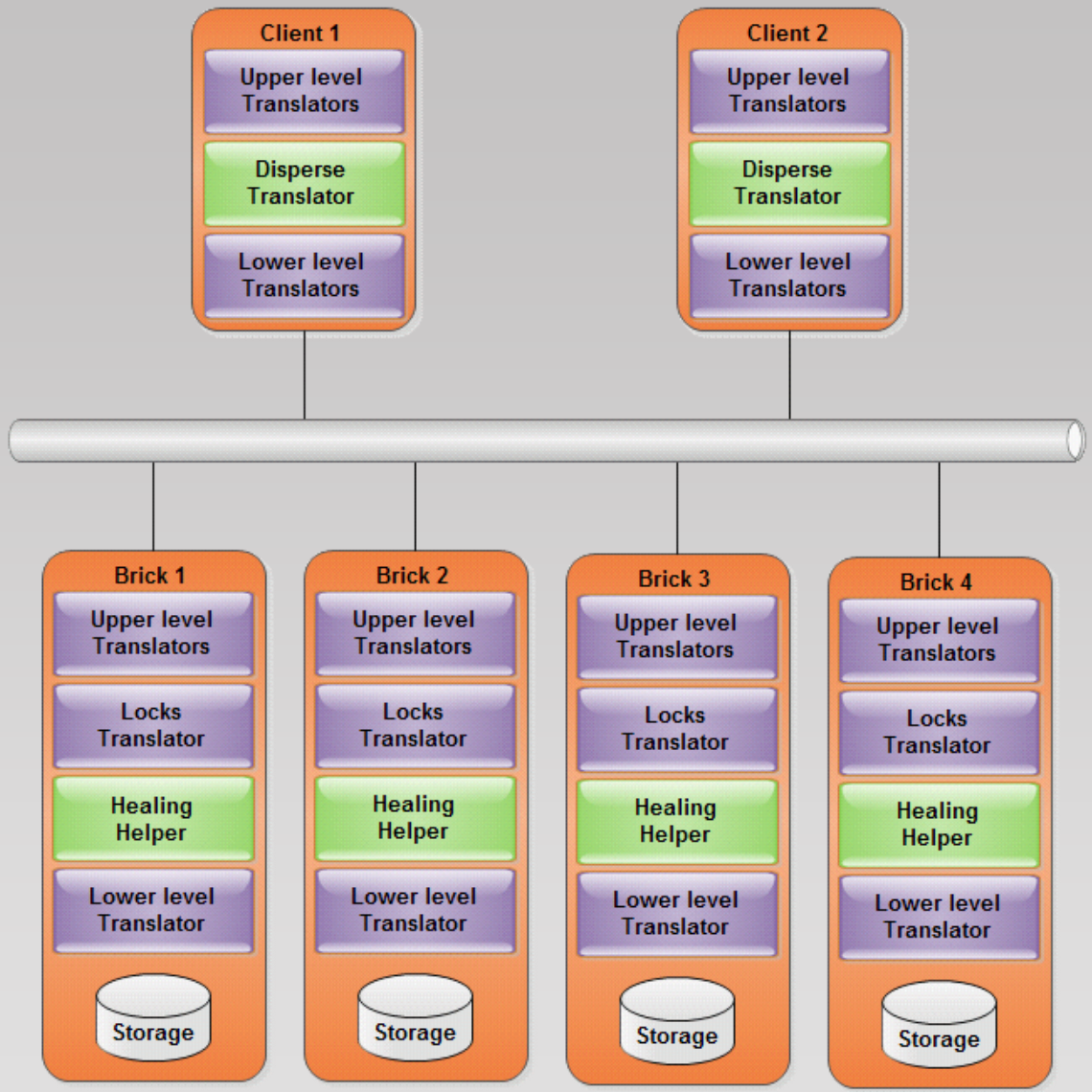


How it works

- Self-Healing
  - Clients detect inconsistencies using metadata
  - Then initiate a healing session using the healing translator (only one is allowed to heal a single file)
  - Initially `entrylk()` and `inodelk()` are held
    - Healing client handshake
    - Healing preparation
    - Metadata healing
  - No lock is held during data healing
  - Finally `inodelk()` is held
    - Final synchronization of metadata (xattr)
    - Gracefully finalize the healing process

- Introduction
- Features
- How it works
- **Architecture**
- Reliability
- Examples
- Current state
- Future

Architecture



- Introduction
- Features
- How it works
- Architecture
- **Reliability**
- Examples
- Current state
- Future

Reliability of metadata

- All metadata is replicated over all bricks
  - This makes metadata highly reliable
  - Metadata is used to detect inconsistencies
- A minimum quorum of matching metadata is needed
  - The data of a file is only considered valid if metadata of at least B – R bricks matches
  - Split-Brain is not possible
    - It will never have two valid versions of the same data
- Special files are handled as metadata

Reliability of data

- Many concepts are similar to RAID5/6
- User selectable level of reliability (configuring R at creation time)
  - Up to R bricks (any subset of B) may fail without service interruption or data loss
- Redundancy is spread uniformly over the bricks
- A fast implementationof Rabin IDA is used when a volume is degraded

- Introduction
- Features
- How it works
- Architecture
- Reliability
- **Examples**
- Current state
- Future

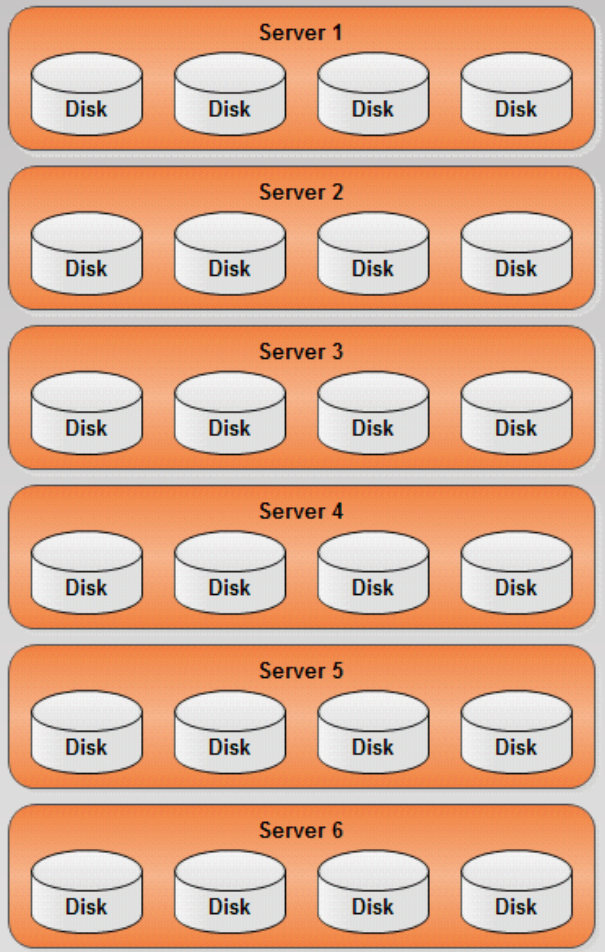
Examples

- Scenario:
  - 6 servers with 4 SATA disks 4TB each and capable of 90 IOPS
  - Each disk is configured as one brick

- Alternatives considered:
  - Striped volume
  - Distributed + Replica 2 volume
  - Distributed + Replica 3 volume
  - Distributed + Disperse 6.2 (B=6, R=2) volume

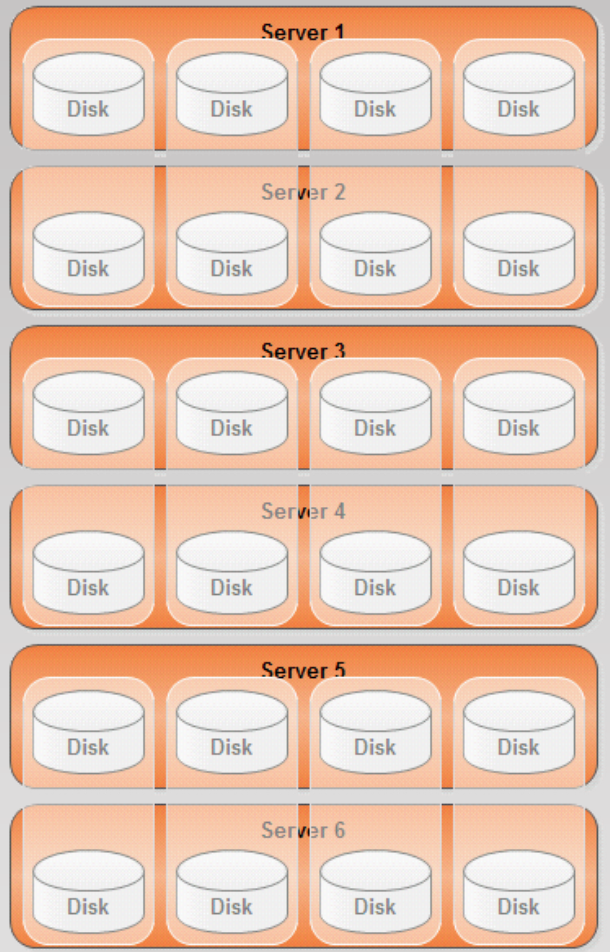
Examples

- Striped volume
  - Effective capacity: 96 TB
  - Read IOPS: 2160
  - Write IOPS: 2160
  - Read bandwidth ratio: 1
  - Write bandwidth ratio: 1
  - Maximum failed bricks: 0
  - Maximum failed servers: 0



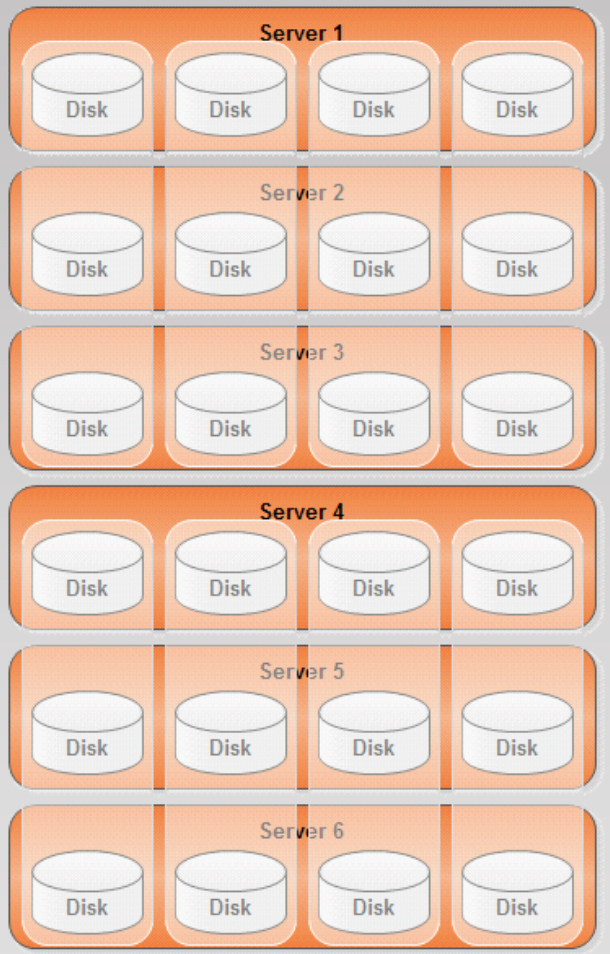
Examples

- Distribued + Replica 2 volume
  - Effective capacity: 48 TB
  - Read IOPS: 2160
  - Write IOPS: 1080
  - Read bandwidth ratio: 1
  - Write bandwidth ratio: 2
  - Maximum failed bricks: 1
  - Maximum failed servers: 1



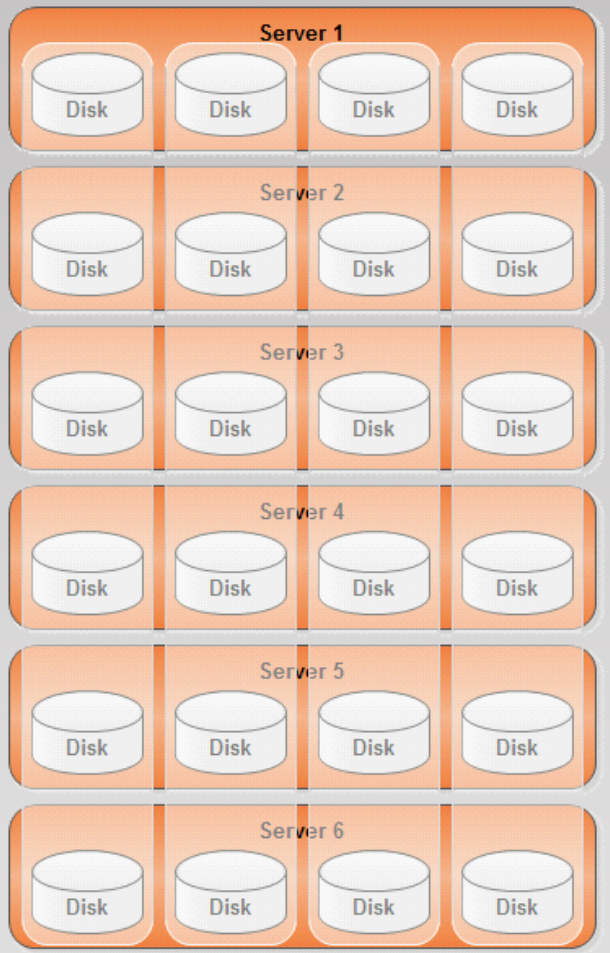
Examples

- Distribued + Replica 3 volume
  - Effective capacity: 32 TB
  - Read IOPS: 2160
  - Write IOPS: 720
  - Read bandwidth ratio: 1
  - Write bandwidth ratio: 3
  - Maximum failed bricks: 2
  - Maximum failed servers: 2



Examples

- Distribued + Disperse 6.2 volume
  - Effective capacity: 64 TB
  - Read IOPS: 540
  - Write IOPS: 360
  - Read bandwidth ratio: 1
  - Write bandwidth ratio: 1..2
  - Maximum failed bricks: 2
  - Maximum failed servers: 2



- Introduction
- Features
- How it works
- Architecture
- Reliability
- Examples
- **Current state**
- Future

Current state

- Disperse translator
  - It's implemented and operational with all features enabled
  - No optimizations applied yet
- Heal helper translator
  - It's implemented and operational with a minimal set of features to allow lock-free healing
  - More features can be added to improve healing capabilities
- An alpha version is being tested in our labs

- Introduction
- Features
- How it works
- Architecture
- Reliability
- Examples
- Current state
- **Future**

Future

- Add cli support for managing the new kind of volume
- Analyze the possibility and advantages of using a RAID5-like striping
  - Worse network performance
  - Better IOPS
- Analyze the possibility of allowing per file chunk size definition (using xattrs)

That's it

Thank you very much

About us

- Who we are
  - Company in the IT services sector
  - 32 years of experience
  - Expertise in a wide range of fields
  - Aware of the latest technology trends
  - Partner of the leading technology companies
  - We support Open Source
  - Involved in some european funded projects

About us

- What we do
  - Provide support in decision making to our customers in a wide range of areas
  - Smoothly integrate different technologies to achieve the best solution for a given problem
  - Develop custom applications for client/server, web or mobile environments
  - Install network and system components
  - Virtualization

About us

- Our customers
  - Mid-sized companies
  - Some in the public sector
  - Healthcare area
  - Research departments
- Our customers needs (some of them)
  - Need large amounts of storage space