What is the RING?

An Overview
What is the RING?

About Scality

Scality is the industry leader in software-defined storage for the information age, serving over 500 million users worldwide. Scality’s RING provides 100% reliable storage with unmatched performance, and is perfect for capacity-driven workloads such as cloud services, high-definition video, and enterprise archiving. It runs on any standard x86 servers powered by Linux, such as the ones of HP, Dell, Cisco, SuperMicro, or Seagate, and creates an unlimited storage pool for file, object and OpenStack applications.

Thanks to its underlying object storage architecture, the RING scales to exabytes of data and trillions of files. Seven of the top twenty telecommunication companies, petabyte-scale cloud services, and Global 2000 enterprises rely on the RING for mission-critical applications.
Contents

Typographic Conventions

Chapter 1
Introduction
Presenting Scality RING
By Design

Chapter 2
The Working RING
Web and Cloud Services
Content Distribution
Global Enterprise Cloud
Active Archiving

Chapter 3
Elements of the RING
Storage Nodes and IO Daemons
Connectors
Object (REST)
File (SOFS)
Supervisor
Supervisor GUI
RingSH Command Line Interface

Chapter 4
Data Routing
Protocols
Chord
Distributed Hash Table (DHT)
Universal Key Scheme (UKS)
Keyspace Provisioning
Disk Selection
Resource Failure Management
Disk
Server
Network
# Table of Contents

## Chapter 5
Data Durability and Protection ................................................................. 23

- Methodologies .................................................................................... 23
  - Data Replication ............................................................................. 23
  - ARC Erasure Coding ..................................................................... 24
- Mechanisms ....................................................................................... 26
  - Rebuild ......................................................................................... 26
  - Repair ............................................................................................ 26
  - Purge ............................................................................................. 26
  - Relocation ...................................................................................... 26
  - Removing Nodes and Servers ......................................................... 27
  - Proxy and Rebalance .................................................................... 27
  - Split Network Detection ................................................................. 27
- Geo-Replication .................................................................................. 27
  - Mirrored RING ............................................................................ 28
  - Stretched RING .......................................................................... 28

## Chapter 6
Maintaining the RING ........................................................................... 29

- Disk Management ............................................................................. 29
- Upgrading Software ......................................................................... 29
- Expanding RING Capacity ............................................................... 30

## Chapter 7
Monitoring the RING ........................................................................... 31

- Supervisor GUI Dashboard ............................................................... 31
  - Status Card .................................................................................. 31
  - Graphical Display Views ............................................................... 32
- RingSH Command Line Interface .................................................... 34
- SNMP-compliant MIB ..................................................................... 34

## Glossary
Typographic Conventions

Files and items that are entered into terminal windows and software fields are presented in a Courier New typeface.

Proper names for menus, menu commands, Toolbar buttons, etc. are presented in a Bold typeface.

Proprietary and non-conventional terms are presented initially in italic text, and occasionally thereafter for purposes of clarity.
Introduction

The $100 billion storage market is in an enormous state of flux, with the effects of the information age extending well beyond the Internet and into almost every industry. To thrive, businesses today need to store more data than ever before, and with mission-critical applications in the mix, high performance and reliability are paramount.

Scality built the RING on the precept that the future of storage is software. Software that can run on industry-standard servers, performing at an extremely high level, that becomes more reliable, and which – contrary to traditional storage appliances and systems – becomes simpler to operate as the system grows.

Presenting Scality RING

Scality RING is an SDS (Software-Defined Storage) solution for petabyte-scale data storage, designed to operate within the modern SDDC (Software-Defined Data Center). It creates scale-out storage systems that converge the storage of object-based and file-based data from multiple applications. The RING can scale, too, seamlessly, from a distributed system atop a minimum cluster of six (6) standard x86 servers to systems comprising thousands of physical storage servers with a total storage capacity running into hundreds of petabytes.
The underlying physical storage servers of the RING can be of any density, ranging from small storage servers with a few hard disk drives (HDDs), to very high-density servers containing dozens of HDDs as well as solid-state drives (SSDs). The RING abstracts the underlying physical servers and disk drives and, in the case of a mixed media deployment, can exploit the lower-latency of SSD storage to maintain its internal metadata and heighten overall performance.

The decoupling of the software from the underlying platform allows for the highest possible platform flexibility, from both a vendor perspective and a scaling and future-proofing perspective, which provides a quantum step in reducing the data center cost of ownership.

The RING is remarkably resilient, with no SPOF (Single Point of Failure), and no downtime required for upgrading, scaling, or maintenance operations planned or unplanned. Also, the RING software is hardware-agnostic, and can be hosted on a wide spectrum of industry-standard (x86 based) hardware platforms, with support for a variety of popular Linux distributions. Hardware-dependencies and platform vendor lock-in concerns are eliminated as well, which frees Scality from having to maintain HCLs – hardware compatibility lists – other than those maintained by the specific Linux distributions.

**By Design**

Scality designed the RING along criteria spearheaded by such leading cloud-scale service providers as Google and Amazon, putting in play loosely-coupled, distributed systems designs that leverage off-the-shelf hardware along such key tenets as:

- 100% parallel design for metadata or data, enabling scaling of capacity and performance to limitless numbers of objects, with no SPOF, and no service disruptions or forklift upgrades as the system grows
- Multi-protocol access, enabling the widest range of object, file and host-based applications to leverage RING storage
- Flexible data protection mechanisms, to efficiently and durably protect a wide range of data types and sizes
- Self-healing, the RING expects and tolerates component failures and resolves them automatically, thus providing high levels of data durability and availability
- Hardware-agnostic for peak platform flexibility, to eliminate lock-in and reduce TCO

The RING incorporates these design principles across the board, to deliver the highest levels of data durability at the highest levels of scale and to achieve the most optimal data storage economics.
The Working RING

With so many divergent factors at play in its configuration, one can safely assume that each deployment of Scality RING is as unique as the entity for which it has been implemented. That said, Scality RING installations typically fall into at least one of four (4) broad use cases:

- Web and Cloud Services
- Content Distribution
- Global Enterprise Cloud
- Active Archiving

**Web and Cloud Services**

**FICTIONALIZED SCENARIO**

WatchIT is a (mostly) short video-sharing portal, definitively web 2.0, with a massive amount of immutable user-generated content. In just five years the company has taken on a mantle of ubiquity that extends beyond the Internet and into the lexicon of everyday life. They enjoy a hard-won reputation for providing instant and constant access to over 20 petabytes of video content, with an estimated doubling of growth year-over-year.

The RING deployed by WatchIT befits the extreme and relentless growth of the company’s product.

<table>
<thead>
<tr>
<th>WatchIT RING</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Servers</td>
<td>60</td>
</tr>
<tr>
<td>Drives Per Server/Size</td>
<td>72/6 TB</td>
</tr>
<tr>
<td>Drive Type(s)</td>
<td>HHD</td>
</tr>
<tr>
<td>Space Per Server (TB)</td>
<td>432</td>
</tr>
<tr>
<td>Total Available Space (PB)</td>
<td>26</td>
</tr>
<tr>
<td>Protection</td>
<td>ARC14+4, CoS2</td>
</tr>
<tr>
<td>Connector(s)</td>
<td>sproxyd</td>
</tr>
</tbody>
</table>

Revenues are steady at WatchIT, however they cannot compare to the incredible growth in the amount of content on offer. Thus, the company requires a cost-efficient, easily scalable storage solution architecture.
WatchIT understands well their unique position in a market they helped carve out. They consider their user community standing to be of paramount importance, and know well how fast and easy it is to lose customer goodwill to the competition.

To sustain and grow their business, Web application and Cloud service providers must put a storage solution in place similar to those deployed by the “Internet Big 5” (Amazon, Apple, Facebook, Google, and Microsoft). They need a solution that offers the operational cost benefits of deploying their own infrastructure, one that can scale virtually without limit and deliver Enterprise-level performance and reliability.

Scality RING is ideally suited to the needs of web-based businesses, such as video-sharing, offering key benefits that are akin to a homegrown storage architecture, and at a TCO that will not impede competition and profitability.

Content Distribution

CH55 ("The Chain") began life in 1985 as an upstart television network, with a plan to subtly and slowly grow their position to one of broadcast relevance. Thirty years later and a lot of creative (and not so creative) water under the bridge, CH55 is today an unassailable media powerhouse, its presence felt in every corner of information and entertainment delivery.

Chief among the many factors contributing to CH55’s remarkable growth was an early decision to offer their proprietary content – news, scripted series, movies and documentaries produced in-house, reality TV, etc. – over the Internet, in addition to television and radio. At first a novelty, this move has put the network in a position where today “The Chain” is synonymous with their advertising promise, to deliver “Free Quality Internet Content, Yours When You Want It”.

For years CH55 contracted with the PointzOut Content Distribution Network (CDN) to push their programming content to PointzOut servers in fifteen different versions (five different formats, each with three different flavors of DRM). A costly setup. Recently, though, with the dual intent of cutting their TCO and taking more control of their content distribution, CH55 implemented a Scality RING.

CH55’s RING architecture serves up a single mezzanine format for each program in response to a pull order from PointzOut. The requested program file is then transcoded on the fly to match the

© 2016 Scality. All rights reserved
requesting entity or application (e.g., Apple iPhone, Windows PC) and is appended with the correct DRM, prior to its distribution to the CDN and ensuring delivery to the end-user.

**SOFS Connectors** for NFS and SMB are in use for populating the RING from CH55 storage archives (data input is always a copy, intended for distribution), with a CDMI Connector in place to fulfill pull orders from PointzOut.

<table>
<thead>
<tr>
<th>CH55 RING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Servers</td>
</tr>
<tr>
<td>Drives Per Server/Size</td>
</tr>
<tr>
<td>Drive Type(s)</td>
</tr>
<tr>
<td>Space Per Server (TB)</td>
</tr>
<tr>
<td>Total Available Space (PB)</td>
</tr>
<tr>
<td>Protection</td>
</tr>
<tr>
<td>Connector(s)</td>
</tr>
</tbody>
</table>

CH55 chose the RING with a steady eye on always making good on their customer promise, confident in its ability to provide service quality with 100% uptime.

So much more than the “Gee Whiz!” novelty of fifteen years ago, online video today is big business. One of the biggest, in fact, with content creators and producers making terabytes of new content available every day via Content Distribution Networks (CDNs) to a seemingly insatiable end-user audience. Everything from short-form news features to session-length cooking programs to movies clocking in at three-hours-plus.

CDNs play a large part in digital data delivery, serving content owners by storing their product and allocating that product to end-users by way of a high level architecture comprised of both Origin Servers (containing petabytes of data and typically deployed in a small number of data centers) and Edge Servers (smaller cache servers, deployed closer to end-users). This system, though, is costly for the content owners, and it suffers from limitations borne of trying to accurately forecast demand.

With Scality RING, organizations can build their own scalable distribution infrastructures that meet all the performance requirements for high definition media streaming. This at a fraction
of the cost of storage at CDN providers, while enabling new non-linear content distribution models such as NDVR/NPVR and VoD.
Traditionally, enterprises primarily built data centers to support transactional workloads. Over time, however, they have evolved into much more complex environments, made to perform multiple heterogeneous workloads (e.g., shared file storage, backup and archiving, B2C cloud offerings, a multitude of legacy applications).

The TCO of running such disparate storage environments rapidly adds up, with each system requiring trained/certified staff, the difficulties in achieving storage efficiency, and the scaling out of dedicated storage systems. As such, provident enterprises are looking to consolidate storage silos and replace monolithic storage appliances with more cost-efficient SDS solutions that function on commodity hardware. Solutions that are not only scalable, but which also support multi-protocol (file and object) mixed workloads across multiple applications.

With native interfaces available for file, object, and OpenStack applications, with Scality RING enterprises can run multiple applications on a single, high-performance, fully reliable storage infrastructure. In addition, as the RING functions on standard servers, it makes for simpler upkeep, deployment and hardware flexibility, while also providing multi-generation hardware support. This combination of software flexibility, multi-workload consolidation, and high scale and availability can reduce total costs by as much as 70%.

**Active Archiving**

**FICTIONALIZED SCENARIO**

SMPA (Selmantia Media Provision Association) is one of the five largest raw broadcast content aggregation companies worldwide. At the dawn of the first Internet boom, the group undertook a government-funded mission to collect, store, maintain, and make publicly available the whole of Selmantia’s vast video and audio production output and broadcast content, dating back to the country’s 1901 inception as a sovereign state.

The effort necessary to amass and digitize content aside, SMPA’s biggest struggle has long been storage management. For years, in fact, the company walked a simple – and unsustainable – path of “pouring hard disks on the fire”. Today, though, SMPA has Scality RING serving as an active archive for their immense media content.

To control media workflow, including the storing and reading of data from the RING, SMPA has set up a Media Asset Management (MAM) solution. Specifically, this system component handles file
ingestion, quality control, indexation, subtitling and captioning, and the appending of metadata forms for cataloguing.

The MAM requires FTP and SMB access points to access the backend file system storage, FTP to push and pull media files and SMB to control space availability. To this end, four (4) **FUSE Connectors** are in place, each with a pure FTPd server and Samba on top of the FUSE mountpoint. These Connectors translate all of SMPA’s file system operations into data objects that are then sent to the RING.

<table>
<thead>
<tr>
<th>SMPA RING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Servers</td>
</tr>
<tr>
<td>Drives Per Server/Size</td>
</tr>
<tr>
<td>Drive Type(s)</td>
</tr>
<tr>
<td>Space Per Server (TB)</td>
</tr>
<tr>
<td>Total Available Space (PB)</td>
</tr>
<tr>
<td>Protection</td>
</tr>
<tr>
<td>Connector(s)</td>
</tr>
</tbody>
</table>

The physical makeup of the SMPA RING reflects the group’s mandate to harness the full history of Selmantia’s media output and to provide free Internet access to it. The company realizes significant TCO savings through the use of somewhat older server technology.

Data volumes are doubling roughly every two years, with growth expected to speed up due to such new technologies as high-resolution imaging, seismic data, and IoT (Internet of Things) sensor data. This remarkable growth is causing companies to shift their storage infrastructure to a simple two-tier architecture comprised of a small primary storage tier for hot data and a large, scalable active archive that provides immediate and constant access to archived data.

An active archive is a primary copy of less frequently accessed information that for cost purposes has been moved transparently off primary storage (i.e., without impact to the users or the application workflows). The data remains easily and quickly accessible, with the same durability as the data in primary storage.

With Scality RING organizations can build Exabyte-scale active archives that have all the performance benefits of disk storage, even presenting long-term cost advantages over tape.
Three (3) basic elements comprise each Scality RING regardless of its scale, configuration, geo-
distribution, data protection scheme, or the physical equipment in place: the Storage Nodes
and IO Daemons, the Connectors, and the Supervisor.

**Storage Nodes and IO Daemons**

Storage Nodes – also referred to as StoreNodes, or simply Nodes – lie at the core of the RING.
Configured to RING storage servers in multiples of six (6), each of these virtual processes is
responsible for data objects whose assigned keys fall into a Keyspace range that is established
for the Node at installation.

Nodes receive objects, files, and messages from the RING’s Connector interfaces, which
provide top-level access points and protocol services for the applications that engage the RING
for data storage. The Connectors leverage a limited list of Nodes in the RING – configured before use – for the purpose of uncovering the RING’s topology, thus enabling data exchange.

Extending beyond their own configured Keyspace, each Node also monitors the Nodes that precede and succeed it on the RING. As such, when a Node disappears from the RING for whatever reason, the system is able to automatically rebalance data replicas and system load to the neighboring Nodes without manual intervention. Of course, the reverse is true in the event that a Node rejoin the RING or is added to it. This makes the RING completely self-healing, and supplants the need for a master database or directory.

Nodes communicate operations to the standard physical disks that lie at the root of the RING via low-level I/O processes. Each of these processes manages the I/O operations to a particular physical disk drive on the server, and maintains the mapping of object indexes to the actual object locations on that disk drive.

Technical Tidbit: A Bizstorenode is the StoreNode process responsible for a given range of data. On each StoreNode, it is this process that communicates with the I/O processes to perform operations that involve access to disk storage.

The RING provides data safety by replicating data (and coding chunks, when ARC erasure coding is put to use) to different Nodes. The Storage Nodes may be distributed to dispersed data sites either in a single RING or through the deployment of separate RINGs (refer to Geo-Replication on page 27 for more information).

**Connectors**

Connectors are the interfaces to the RING, providing top-level access points and protocol services for the applications that leverage the RING for data storage. They receive requests from the applications and subsequently dispatch data to the StoreNodes within the RING, providing read, write, delete, and lookup storage services.
As stated with regard to Nodes, Connectors disseminate data to Nodes in the RING by leveraging a limited list of Nodes that is configured to the Connector prior to use. They use this pre-configured Node list to discover the RING’s topology and thus determine the location of the Node whose Keyspace includes the key for the data to be distributed or retrieved.

In addition to bridging the IO path from applications through to the StoreNodes on the RING, Connectors also implement the configured data protection storage policy.

Although all of the data stored on the RING is in the form of objects, through its family of Connector interfaces the system is fully equipped to handle both data objects and data files. Consequently, Scality’s Connectors are able to handle a vast set of applications promoting a wide variety of data types.

Connectors are designed to be stateless, allowing incoming requests to be load-balanced to any number of Connectors and implementing different protocols to access the underlying RING storage cloud.

**Object (REST)**

The RING is accessible as a native object storage platform via a selection of REST/HTTP protocols, each of which provides simple object key/value storage semantics through basic PUT, GET, DELETE calls with flat (non-hierarchical) and scalable name spaces.

Scality provides a selection of Connectors that implement REST APIs, including the native, high-performance sproxyd, the Amazon S3-compatible REST Storage Server (RS2) family of Connectors, and OpenStack Swift.

**sproxyd**

Scality’s sproxyd RING Connector implements a pure object REST API designed to meet extreme scalability and performance requirements. The Connector provides basic PUT, GET, HEAD and DELETE API calls, along with standard and Scality-defined optional headers that serve to customize behavior. sproxyd typically uses an Apache fastcgi interface, and can be deployed directly on the application server, the Storage Nodes, or on a dedicated server functioning as a proxy.

As a basic object Connector, sproxyd leaves such additional features as key management (catalogs), authentication, file system sharing or multi-tenancy capabilities for the application to manage.

**scloned**

The use of the scloned Connector is exclusive to geo-distribution scenarios, providing a means of synchronization between a local RING and a remote RING. The Connector allows local and remote RINGs to employ different CoS or ARC data protection schemes. scloned is installed on the local RING, which may be configured for either replication or erasure coding (without necessarily using the same ARC coding scheme as the remote RING).
RS2
Modeled after the Amazon Web Services (AWS) S3 object API, Scality’s REST Storage Server Connector (RS2) offers features that make it optimal for public cloud deployments (e.g., MD5 signatures for integrity validation, a Bucket container mechanism). As a trade-off for such high functionality, the RS2 Connector is sometimes eschewed for applications that require especially high scalability or performance.

RS2 processes HTTP requests and returns HTTP responses to applications based on the RFC 2616 protocol. Recommended for Storage-as-a-Service (STaaS) applications, the Connector can be configured to upload, retrieve, and manage data on an Amazon S3 (Simple Storage Service) backend.

RS2 Light
RS2 Light is the RS2 Connector in light mode, meaning without indexing, authentication, billing, or access control, but with additional benefits for scaling and performance. From a practical use perspective, the RS2 Light Connector can be put to use in situations in which the data stored is immutable, and access and security are not an overriding concern.

OpenStack Swift
Scality’s OpenStack Swift Connector provides a scalable data storage system for OpenStack Swift. It plugs in underneath OpenStack Swift, and is completely interoperable with OpenStack Accounts, Containers and Keystore authentication mechanisms.

The OpenStack Swift Connector operates as a back-end replacement to the normal Swift storage layer, and provides all of the RING’s features including erasure coding, scalability, and high-performance functionality.

File [SOFS]
SOFS (Scale-Out-File-System) is POSIX-based, and thus it provides file storage services without the need for the external file gateways that are common in other object storage solutions. SOFS provides the possibility of unlimited space for content storage, and permits unlimited root directories within the same volume.

Via the file system Connectors, SOFS offers a file system view and a set of protocols that enable file-based access to the RING, supporting such file system protocols as FUSE, NFS, SMB, and CDMI.

Scality RING employs a distributed NewSQL database called MESA to provide file system semantics and views. An internal distributed database, MESA sits on top of the RING’s storage services that store file system directories and inode structures to provide the virtual file system hierarchy and assure transactional consistency of file system data.

SOFS file systems can be scaled-out in capacity across however many Storage Nodes are required to support application requirements, and they can be accessed by a scalable number of eligible Connectors to support application load requirements.
What is the RING?

FUSE

The sfused (Scality FUSE daemon) Connector is the FUSE interface to SOFS, providing a regular file system interface for the Scality RING.

FUSE is a file system that is supported across all major Linux distributions. It provides local file system access to the RING, in support of a variety of application deployments.

Technical Tidbit: The FUSE model is capable of handling $2^{64}$ inodes per volume, thus the number of objects FUSE can process is virtually unlimited.

SOFS can be configured to support quotas, as well as parallel IO to the back-end RING servers, while optimizing access to very large files that are striped across multiple back-end servers. As such, it provides a unified approach that simplifies management and failover scenarios, while guaranteeing the consistency of stored messages and metadata.

NFS [sfused for NFS]

NFS (Network File System) is supported, available as a client interface on nearly all operating systems – Linux, Mac OS X, and Microsoft Windows. Client applications that draw on this protocol can use the RING for storage operations through the Scality NFS Connector using the sfused daemon for NFS, which can be installed on NFS v3 servers with support for either the TCP or UDP transport layer protocols. RING data can be written or accessed through the NFS Connector using IP addresses or DNS names.

SMB [sfused for CIFS]

Server Message Block (SMB) client applications can use the RING for storage operations via servers running the Samba Windows-Unix interoperability program. Samba transforms the view of the RING provided by a Scality SMB Connector installed on the Samba server, making the POSIX filesystem used by the RING storage servers accessible to the SMB client applications running on Windows platforms.

SMB was previously known as Common Internet File System (CIFS).

CDMI

The Scality CDMI Connector attaches to the RING via a File Access Layer (Ring FAL) that leverages the open standard CDMI protocol, thus allowing files and objects to be shared between the CDMI REST protocol and such other file protocols as NFS and FUSE. The Connector provides a global namespace for all files and objects in storage, and the data appears together in a single view regardless of whether it is exposed locally (through the CDMI/SOFS Connector) or remotely.

The CDMI Connector includes performance-boosting features such as transparent compression and local read and write caches. It can be used or modified for private, public or hybrid cloud connections.
All SOFS file system based Connectors (FUSE, NFS, SMB, and CDMI) can be scaled-out across multiple servers, to provide scalable read performance to high-numbers of client applications that require simultaneous read access to the same file system data.

**Supervisor**

The Supervisor is the central management utility for the RING, employed to configure and monitor the entire system. It is independent of the core of the RING, and thus does not need not be running to ensure the integrity of the system or the flow of storage operations traffic.

The Supervisor can manage multiple RINGs, and RINGs can be added and removed from its purview.

**Only one Supervisor should be actively running per data center.**

The Supervisor acquires RING component data by way of the `sagentd` instance hosted on each Scality managed Node and Connector server.

Technical Tidbit: The Scality Monitoring Agent daemon – `sagentd` – serves as a single point of communication between the RING’s Nodes, Connectors, and the Supervisor, for purposes of statistics and health metrics collection. Via the Supervisor – which by default automatically detects the presence of sagentd on each server to which it connects – the daemon supplies data processing information for each RING component running on the server on which it resides.

Scality offers two means for accessing Supervisor function, the Supervisor GUI and the RingSH command line tool.

**Supervisor GUI**

The Supervisor GUI is a Web-based visual interface for the RING Supervisor, providing point-and-click style monitoring and management of the RING software and the underlying physical platform layer. The full functionality of the GUI is made available via three distinct windows – **Volumes**, **Rings**, and **Hardware** – which are always available.

**Volumes Window**

Specific to SOFS, the **Volumes** window in the Supervisor GUI provides a simple UI that can be used to easily create Connector Volumes. Managed in this manner, multiple SOFS Connectors can be simultaneously configured and started, ready for access by end-use applications.
Rings Window

The entry point for the Supervisor GUI — the Rings window — offers the Ring list, an inventory of all of the RING installations that can be managed from the tool.

The total number of Nodes and a State description is displayed for each listed RING, along with links to four key information tabs: Dashboard, Operation, Administration, and Provisioning.
Elements of the RING

<table>
<thead>
<tr>
<th>Tab</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dashboard</td>
<td>Presents a selection of graphical views used to monitor and manage RING Nodes and view bandwidth statistics for the RING, accessible via a menu that includes buttons labeled <strong>Topology, Keyspace, Main, System, Ring, Connectors, Tiering, Disk Health, and Custom.</strong></td>
</tr>
<tr>
<td>Operation</td>
<td>Offers an interface for monitoring RING Nodes by key or server, for monitoring Connectors, and for performing various actions as they pertain to Nodes, Connectors, and tasks.</td>
</tr>
<tr>
<td>Administration</td>
<td>Features a full array of configuration settings for the RING and its Connectors, accessible via a menu that includes buttons labeled <strong>General, Tasks, Tiering, Connectors, and RS2 templates.</strong></td>
</tr>
<tr>
<td>Provisioning</td>
<td>Allows for key range assignment to the Nodes, thus making the Nodes available to join the RING.</td>
</tr>
</tbody>
</table>

**Hardware Window**

The Supervisor GUI can be used to register agents for **Nodes** and **Connectors**, and to create or remove separate zones for racks or data centers or for any logical grouping set for assignment to RING Nodes and Connectors.

**RingSH Command Line Interface**

*RingSH* is a scriptable command line interface (CLI) that can be used to manage and monitor the RING and the Connectors that access it. This CLI provides a rich set of commands for managing the complete stack, as well as providing access to system statistics and health metrics.

The RingSH CLI can be used on the Supervisor host, on any Connector server, or on any storage server as a simple interactive shell with a command history, or it can be called non-interactively via regular UNIX scripts.
Data Routing

Large distributed systems rely on fast, efficient routing of requests among their member Nodes.

Protocols

Various protocols enable Scality RING to expedite the routing of objects and files in the system, including Chord, Distributed Hash Table (DHT), and the Universal Key Scheme (UKS).

Chord

The architecture of the Scality RING is based on MIT’s peer-to-peer Chord protocol, which was initially developed for the purpose of mapping stored objects onto a virtual, logically circular range of keys, or Keyspace. Scality has augmented and patented enhancements to the basic Chord protocol to enable high levels of data durability, high performance, self-healing and simplified management.

The Chord protocol’s value lies in its routing efficiency, since with Chord each Node in the RING requires only the knowledge of its own Keyspace and that of its predecessor and successor.

Consequently, the RING can provide complete coverage of all allotted Keyspaces. This makes it the perfect basis for a distributed storage system designed to hyper-scale to billions of objects, putting into practice Scality’s distributed, 100% parallel design principle.
The Chord protocol can route data dispatch requests for a given key quickly and efficiently — from any Node in the RING to the Node that owns the key. At most, any lookup will require \( \frac{1}{2} \log_2(N) \) operations (where \( N \) = the number of Nodes in the RING). This means that the number of hops scales sub-linearly and deterministically for RINGs with very large numbers of Nodes and massive storage capacity. For example, in a 1000 Node system, five (5) hops are the maximum required to locate a key on the RING.

<table>
<thead>
<tr>
<th>Number of Nodes in RING</th>
<th>RING Capacity*</th>
<th>Number of Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>5.7 PB</td>
<td>3</td>
</tr>
<tr>
<td>1,000</td>
<td>56 PB</td>
<td>5</td>
</tr>
<tr>
<td>10,000</td>
<td>560 PB</td>
<td>7</td>
</tr>
</tbody>
</table>

*Assumes six (6) Nodes per physical server, 56 HDDs per server, 6TB per drive

Another essential property of Chord is that the protocol is dynamic, able to adapt rapidly to changes in the Keyspace as a result of new Nodes joining or departing the RING. A rebalance mechanism allows the system to move the set of keys owned by a Node to the new Node(s) assigned to the affected addresses in the Keyspace. Also data that was owned by a departing Node is easily and automatically moved to its predecessor Node.

**Distributed Hash Table (DHT)**

Underpinning Scality’s storage technology lies an efficient decentralized Distributed Hash Table (DHT) mechanism, implemented over a structured peer-to-peer protocol that provides the routing mechanism for locating objects on the RING. With this lookup service, key-value pairs are stored in the DHT, and any participating Node can efficiently retrieve the value associated with a given key.

The DHT is distributed among the StoreNodes assigned to the Keyspace, and the instance on each Node only has knowledge of its own key range, its immediate predecessor and successor Nodes, and a few additional Nodes at well-known geometric “projections” across the RING. Importantly, the DHT does not represent a centrally shared “metadata store” of the Keyspace, it simply captures the local Node’s knowledge of a subset of the RING topology so that lookup operations can efficiently compute the next-best estimate of a key’s location on other Nodes until that key is found. Naturally, multiple hops may occur during key lookups, however the algorithm uses knowledge of predecessor and successor Nodes to deterministically, and with low-latency (tens of milliseconds), locate the right Node.

**Node lookups do not necessitate disk seek operations, but merely require navigating the DHT algorithm across a sequence of Nodes.**

The distribution of the DHT fractionally across the RING’s Nodes ensures that global-update or consistency of key maps is not necessary for every storage operation across all Nodes. This practice reduces broadcast requests and inter-Node communications, and scales the system to very high-levels in a much more efficient manner.

The DHT can dynamically adapt to changes in the RING topology caused by Nodes joining and leaving the RING, due either to normal operations (e.g., scaling) or to disk or server failure.
events. Normal operations can take place while any changes are occurring, with the system serving lookup and storage requests without any disruption.

Another important property of the DHT is that the small Keyspace modifications that result from Node departures or additions only affect a relatively small number of keys, and hence only a proportionally small number of keys require balancing.

**Universal Key Scheme (UKS)**

The Universal Key Scheme (UKS) defines the scheme to which the 160-bit keys for all objects stored on the RING must adhere.

RING keys are divided into predefined parts, indicating the unique identifier for the object, its location, and the number of copies of the object that should exist. At the highest level, there are three parts of a key: Dispersion, Payload, and Class of Service.

![RING Key Scheme Diagram](image)

**Dispersion**

The initial bits of a RING key represent the high order bytes of an evenly distributed MD5 hash of the payload, used to ensure that adjacent objects are dispersed across all Storage Nodes.

**Payload**

The bits of a RING key that follow the Dispersion part uniquely identify an object. The payload is where the universal key scheme defines bit-fields to provide a common means for addressing objects in a RING, helping to ensure that different services and different objects do not clash.

**Class of Service (CoS)**

Class of Service, or CoS, resolves the problem of replicas clashing with keys of another object. Specified at the end of a key, CoS consists of the class itself (4 bits) and an indication of the replica number for a particular key (4 bits for Data Replication, 8 bits for ARC Erasure Coding).

Classes of Service are divided in Standard classes (CoS numbers 0-5) and Special classes (CoS numbers 6-8).

<table>
<thead>
<tr>
<th>CoS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 5</td>
<td>Specifies the number of replicas an object should have (i.e., for class 2 there are three total instances of the object, the original and two replicas)</td>
</tr>
<tr>
<td>6</td>
<td>Not currently in use</td>
</tr>
<tr>
<td>7</td>
<td>Employed internally for object storage in an ARC system; the full ARC schema requires use of 24 bits from a key’s Payload</td>
</tr>
<tr>
<td>8</td>
<td>A special class for geo-distributed solutions</td>
</tr>
</tbody>
</table>
Furthermore, a RING can store objects durably according to one or more Classes of Service, for flexibility in storing mixed size workloads.

**Keyspace Provisioning**

The Scality RING manages stored objects that are mapped into a logically circular, virtual Keyspace. These objects are identified by Scality's unique 160-bit keys, which are uniformly distributed to StoreNodes on the Keyspace.

Each of the StoreNodes in the RING also has a unique 160-bit key that serves as the upper end delimiter for the object keys they may contain. In addition, each StoreNode is responsible for all objects whose keys fall within the range defined by its delimiter key and that of its predecessor. Consequently, each StoreNode manages only a limited portion of the total Keyspace.

To guarantee data safety against common failure events, the simplest RING is comprised of a minimum of six (6) physical servers. To subdivide the Keyspace more effectively across physical capacity, each physical server is assigned a set of at least six (6) StoreNodes, which are arranged into the logically circular Keyspace according to their assigned Key value.

In this simplified illustration, with key delimiters from 0 to 350, Storage Node 50 is responsible for storing all object Keys in the range from 40 to 49. If Storage Node 50 departs the RING – either intentionally or due to a failure – its Keys are automatically reassigned to its successor, the Storage Node with Key 60, which will then assume responsibility for Keys in the range 40 to 59 (for information on rebalancing the RING, refer to Proxy and Rebalance on page 27).

**Disk Selection**

Nodes in the RING keep track of how many I/O operations are pending in their scope (refer to Storage Nodes and IO Daemons), how much space is available on each disk, and whether there are any status flags relevant to disk selection. The Nodes use this information to determine the order of preference for selecting a disk for input data.
Resource Failure Management

As with any distributed computer system, the RING is occasionally subject to failure at the disk level, the server level, and the network level. Such issues can be lessened through regular maintenance and monitoring, of course, however they cannot be fully eliminated. Scality RING is designed to effectively deal with such setbacks when they occur, though, and without the necessity for significant system downtime.

Disk

Disk failure does not impact RING function as replication allows the Connectors to continue serving the requested objects, using their replicas to do so.

By default, failed disks are not automatically removed from RING service. When a potential disk issue occurs, the disk is marked with a flag that by default prevents it from being used for 300 seconds before the disk is retried. During this period, all inbound writes complete to one of the other disks in the system.

A failed disk in the RING can be quickly replaced by another disk through use of the scaldiskreplace tool.

Server

As with disk failure, a storage server failure does not impact RING function as replication allows the Connectors to continue serving the requested objects using their replicas.

When a server is lost – perhaps due to issues with its OS disks or chassis – and its storage disks are intact, it can be rebuilt using a small number of Node configuration files that are typically backed up daily to a compressed TAR archive on the server. Assuming the availability of this TAR backup, reconstructing a lost server is simply a matter of (1) copying that file to the /etc directory on the replacement server, (2) decompressing the TAR to extract the failed server’s Node configuration files, and (3) distributing the configuration files to their proper locations on the replacement server.

Network

A RING that spans two or more data centers can split when the link between the data centers is lost, a condition referred to as “split brain” (refer to Split Network Detection on page 27). When this occurs, new writes to the RING are made with the same number of replicas as under normal conditions, but only to the Nodes in one of the data centers.

Once the network connection is restored between the two data centers, all of the data written to the original receiving data center over the duration of the split is replicated to the companion data center’s Nodes, and the replicas stored in the extra Nodes of the original receiving data center are deleted. Thereafter, when the RING receives requests for data entered prior to the split it will once again look for the replicas in both data centers.
Data Durability and Protection

Scality RING is able to anticipate and manage a wide range of component failures, including disks, servers, and networks, thereby ensuring that the data under its dominion remains available and consistent.

The RING provides data durability through a set of flexible protection methodologies and mechanisms optimized for distributed systems that allow applications to select the best data protection strategies for their data. Through Scality design principles, these mechanisms are able to address a wide spectrum (80%) of storage workloads and data sizes.

Methodologies

Scality’s data replication and ARC data protection methodologies provide extremely high-levels of data durability, with the ability to fine-tune performance and space characteristics for different data types.

Data Replication

As one of its data protection strategies, the RING employs automatic data replication, or the storage of multiple replicas of an object within the RING. To separate these replicas from common failures, the RING spreads them across multiple Storage Nodes and across multiple disk drives (assuming the availability of a sufficient number of servers and disks).

The RING supports several Class-of-Service levels for data replication (as described in detail in Universal Key Scheme (UKS)), indicating that the system can maintain between up to six (6) instances of an object (or put another way, up to five replicas of an object original). Thus, at its CoS threshold of 5 the RING can endure up to five simultaneous disk failures while still preserving access and storage of the original object.

Any failure will cause the RING to self-heal the lost replica, thus automatically bringing the object back up to the number of replicas required by the Class-of-Service as fast as possible.

While data replication is optimal for many use cases wherein the objects are small and access performance is critical, it does impose a high storage overhead penalty compared to the
original data. For example, a 100KB object stored with a CoS of 3 will use 400KB of physical capacity on the RING (4 x 100KB) in order to maintain the original object and its 3 replicas.

Such an overhead is acceptable in many cases for small objects, however at the megabyte/gigabyte level (e.g., video objects) it can quickly become a significant cost burden, requiring a more efficient data protection mechanism.

**ARC Erasure Coding**

Scality’s Advanced Resiliency Protection (ARC) provides an alternative data protection methodology to standard data replication, one that is optimized for large objects and files.

ARC erasure coding addresses the same problematic as RAID devices: how to increase failure tolerance at a storage cost lower than the classic replication of data on different disks. However, while ARC uses the same mathematical background calculations as a RAID system, it arranges data in a completely different way that is more suited to a Scality RING storage pool.

Basically, instead of dealing with multiple replicas of an original object, ARC breaks an object into multiple data “chunks” \( m \), and applies a mathematical encoding to produce an additional set of parity chunks \( k \). The resulting set of chunks \( m+k \) are distributed across the RING Nodes, providing the capability of accessing the original object as long as any \( k \) subset of \( m \) data and parity chunks is available. Put another way, ARC provides a means for storing an object with protection against \( k \) failures, with only \( k/m \) overhead in storage space.

<table>
<thead>
<tr>
<th>Benefits</th>
<th><strong>ARC Erasure Coding</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost/Space Efficiency:</strong></td>
<td>ARC massively reduces hardware overhead, providing considerable cost savings for the same data protection capabilities. For instance, for a system able to tolerate two Node failures, the overhead for an ARC deployment is 50% or less, compared to 200% for a typical data replication scheme.</td>
</tr>
<tr>
<td><strong>Flexibility:</strong></td>
<td>Any combination of ( k ) data chunks and ( m ) coding chunks can be employed for different Classes-of-Service and failure tolerance, and these can coexist on the same RING.</td>
</tr>
</tbody>
</table>
**ARC Erasure Coding**

<table>
<thead>
<tr>
<th>Potential Issues</th>
<th>More I/O Ops and Bandwidth Required: ARC rebuild operations require more I/O operations and use more network bandwidth than those of a typical data replication system.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Greater margins advised for expected failure events: In the event there are more failures than anticipated by the class of storage selection, more data is typically lost with higher probability compared to the data replication model.</td>
</tr>
</tbody>
</table>

With data replication, the class defining the number of object replicas is set in the penultimate 4-bit position of the object key, while for ARC data the penultimate position is occupied by the special class 7 to indicate that ARC is in use. The class for ARC data must take into account all the possible combinations of \( k \) and \( m \) (number of data and coding parts) as well as the ARC schema, thus the key needs more than the 4 bits used to define the class of replicated objects.

As an example, assume a 1.4 GB object is to be stored using an ARC\(4+4\) erasure coding schema. This implies that the original object will be divided into 14 equal chunks, each in this case sized at 100MB. The system then applies ARC encoding on these 14 chunks to produce four (4) additional parity chunks that are also 100MB in size. The resulting total of 18 chunks require 1.8GB of storage space (18 x 100MB), a 29% space overhead (0.4GB/1.0GB) that supplies protection against four simultaneous disk failures, which is significantly less than the 300% space overhead that would be required to store 4 instances of an object using data replication (4 x 1.4GB = 5.6GB).

![Data Source](image)

**Data Source**

With Scality’s ARC, data chunks are stored in the clear without any encoding, which circumvents any potential performance penalty during normal read access (often seen with commercial storage solutions in which all chunks – including the original data – are encoded prior to being stored, and must subsequently be decoded for access operations). As such, ARC’d data can be accessed as fast as other data, except in the event that a data chunk is missing, which would require a parity chunk to be accessed and decoded.

**Data Replication and ARC may be combined, even on a single Connector, by configuring a Connector policy to store objects below a certain size threshold with a replication Class-of-Service, but still store files above the file size limit with a specific ARC schema. This system configuration allows**
client applications to simply store objects without concern for the optimal storage strategy per object (as that will be handled by the RING).

Mechanisms

The RING has several self-healing mechanisms that prevent single points of failure and help guarantee against data loss or corruption.

- Rebuild
- Repair
- Purge
- Relocation
- Removing Nodes and Servers
- Proxy and Rebalance
- Split Network Detection

Rebuild

Launched by the Nodes themselves and programmed to run automatically, the RING’s Rebuild mechanism cycles through the RING, replacing any lower versions of objects with the highest version available. The task will reconstruct objects with problems such as missing data, differing versions, and so on.

Each Node starts a Rebuild process independently of the other nodes, although the time between the automatic rebuilds for each Node is the same.

Repair

In contrast to the Rebuild task, the Repair mechanism is dedicated to mending missing or corrupted objects on the affected Node (provided the Node is not restarted). The task checks the predecessor of the given objects and attempts to reconstruct them in place. Following the formatting of a disk or a Node, however, a global rebuild is the only task that will re-create the lost objects.

The Repair task uses a mechanism similar to that of the Rebuild task to reconstruct missing or corrupted data. It regenerates the data from a predecessor replica, though, rather than a successor replica. The mechanism also starts directly from the missing or corrupted chunk rather than iterating through successor chunks to get the object and reconstruct the chunk.

Purge

The Purge mechanism physically removes data chunks marked as deleted, thus avoiding the creation of orphaned chunks and allowing for the reclamation of disk space. The task is handled directly by the Nodes, using a deleted queue on each node that lists all deleted objects on the fly and destroys them at a preconfigured expiration time.

Relocation

Occurring automatically according to set parameters, Relocation is a process that frees up space previously occupied by data that has been marked for deletion and then purged from
the StoreNode. The mechanism takes the containers with the most data to be erased and recreates new containers with the remaining data.

**Removing Nodes and Servers**

Scality’s peer-to-peer protocol makes use of consistent hashing to assign fixed keys to Nodes, thus ensuring that the adding or removing of Nodes does not disrupt the RING architecture, and limiting the amount of data to balance when the RING topology changes.

To retire a Node server, that server need only be shut down. To physically remove the server, however, it is best to first use the Supervisor GUI, selecting the Leave action (Operation tab | Nodes button) for each of the server’s Nodes and then the Stop action (Operation | Actions) for the server itself.

Nodes are automatically forced to leave the RING if all the disks to which they have access are marked as full or out of service. This allows the RING to isolate problem disks and redistribute objects and replicas to other disks that remain functional.

**Proxy and Rebalance**

A Node is responsible for all objects that have keys between its predecessor on the RING and itself. Each Node reports its range, from the lowest-ranked object hosted to the highest. By default, the Nodes themselves – rather than the Supervisor – manage temporary overlaps between all key ranges.

At any instant, a Node may contain keys that properly belong to another Node. For each such overlap, a Node can send proxy requests and the Supervisor can send rebalance requests. Both of these mechanisms can be independently enabled or disabled.

Proxy requests provide additional routes to the Nodes involved in an overlap. This facilitates Node additions by allowing data access before the data is moved to its final destination. Rebalance requests trigger the relocation of misplaced objects from an overlapping Node to its proper successor Node.

**Split Network Detection**

Each StoreNode includes a RING split detection algorithm that checks for range inconsistency, based on what a Node sees of the RING. If too many disparities are detected between ranges, the StoreNode will block all automated balancing and rebuilding tasks until manual intervention occurs or until the disparities disappear. Such RING splits are generally caused by network link level issues (e.g., a switch malfunction or a dead network card). During such an incident, all Nodes that remain available are capable of accepting data.

**Geo-Replication**

To enable site-level disaster recovery solutions, Scality offers two geo-distribution options, whereby the RING can be deployed across multiple sites (data centers) with a failure tolerance of one or more sites. The first – the Mirrored RING – involves independent RINGs, each within
its own data center, with asynchronous mirroring employed to eventually manage synchronization between the RINGS. The second option is the Stretched RING, which makes use of a single logical RING deployed across multiple data centers and thus requires no synchronization.

**Mirrored RING**

- Multi-RING in one or multiple datacenters for site-level DR protection
- Object RING-to-RING mirroring
- SOFS mirroring supports mixed Replication/ARC on source & target

**Stretched RING across multiple sites**

- “Active/Active” - managed as a single RING
- Location-aware allocation: both Replication and ARC can spread data across disparate domains for site failure tolerance
- Synchronous operations: best for lower-latency WAN environment

**Mirrored RING**

A Mirrored RING bridges two geo-distributed data centers. This mode provides the highest degree of protection against site failures, by maintaining two full data RINGs. It requires that the system replicate all data and metadata operations from the primary data center RINGs to the secondary data center RINGs, and hence requires twice the storage capacity.

**Stretched RING**

A Stretched RING differs from a Mirrored RING in that it is a single data RING shared across geo-dispersed data centers, complete with site protection and full data consistency. In this mode, all RING Nodes participate in the standard protocols as if they were local to one site.

Deployed with ARC erasure coding, a Stretched RING provides many benefits, including full site-level failure protection, active/active access from both data centers, and dramatically reduced storage overhead compared to Mirrored RINGs. To illustrate, an ARC7+5 schema for a three-site stretched RING will provide protection against one complete site failure, or up to four disk/server failures per site plus one additional disk/server failure at another site, with approximately 70% space overhead.
Maintaining the RING

Naturally, due to its inherently technical composition, the job of maintaining the Scality RING is routinely assigned to a customer’s IT department, with guidance as needed by Scality Customer Service Engineers (CSEs). While the upkeep of such an advanced system can delve many layers deep, the majority of maintenance tasks faced by RING administrators involve managing the hardware at the foundation, the upgrading of RING software, and RING capacity expansion.

**Disk Management**

Hard disks lie at the bottom layer of any RING installation — physical devices on which data is stored that are constantly in motion, worked hard and hot, and which will eventually fail. This failure inevitability makes the ability to detect disk errors of paramount importance in ensuring the RING’s reliability, and in enabling easy and seamless disk replacement without breaking the RING.

The RING automatically handles disk error detection by way of two helper files, *Error Policy Helper* and *Disk Repair Helper*. The RING also provides scaldiskadd and scaldiskreplace, tools whose purpose is to assist in disk addition and replacement following error detection.

**Upgrading Software**

The RING has no single points of failure, and needs no downtime for version upgrades, capacity scaling, and hardware maintenance operations (planned or unplanned).

In general, upgrading the RING is as easy as upgrading the Scality packages from the updated repository. To maintain production traffic, however, it is necessary to upgrade servers one at a time, waiting for a server to be completely active again before upgrading the next server.

Although all components can work with each other within one major RING release difference — because this eases the process — Scality recommends upgrading the Supervisor server first, then the StoreNode servers, and then the Connector servers.
**Expanding RING Capacity**

The capacity of an existing RING can be seamlessly expanded, without service disruption or the need to “forklift” existing capacity (that is, the capacity can be scaled-out without replacing existing servers).

**Scality recommends that customers consider expansion once their RING installation has reached 60-65% total capacity, to ensure that the expansion commences prior to the RING reaching 80-85% total capacity.**

A RING can be scaled-up by adding one or more new storage servers, or even a full rack of servers into its topology. Over time, these servers will come to have different densities (number of disk drives per server, individual disk drive densities), as well as differing mixes of drive types (HDD, SSD). Less common are scenarios in which server types are mixed within a RING, with individual disk drives added into a storage server or an open disk bay.

Existing RINGs can be scaled up by a percentage of existing capacity (for example, one server added to a rack of 9 existing servers, representing 11% capacity growth, or one rack added to an existing rack, representing 100% capacity growth). A matching scaling in performance (bandwidth) is required, of course, which can be achieved by scaling-up the number of Connectors to match the increased capacity.

**In expanding a RING, Scality recommends adding Node processes in excess of the number of physical servers, in order to maintain the proper Keyspace balance without need for modifying existing Node/Key assignment and thus triggering increased object migration.**

The current capacity scaling process follows a sequence of discrete steps:

1. Prep new server(s) with a supported Linux distribution.
2. Format and prep the disk drives on the new servers.
3. Manually install Scality RING packages on each new server.
4. Generate a new Keyspace via sprov (a keyspace attribution utility that is employed via the Supervisor GUI Provisioning tab).
5. Join new servers to RING with the new Keyspace.
6. Monitor the performance impact of rebalancing the RING, and adjust background processes as required to maintain system performance.
Monitoring the RING

Scality provides an extensive monitoring capability with the RING, offering a comprehensive set of tools, with a variety of interfaces. These tools include the Supervisor GUI, the RingSH command line interface, and an SNMP-compliant MIB that can be leveraged by standard third-party SNMP monitoring solutions (e.g., Nagios).

**Supervisor GUI Dashboard**

Visual, point-and-click style monitoring of RING software and its underlying physical platform layer is available via the Supervisor GUI. Specifically, the GUI provides a Dashboard with graphical RING views of the Servers, Disks, Connectors and Storage Nodes that comprise the RING, with the capability to drill down to the details of each component. The Supervisor GUI has multiple pages for operations and management of RING services, provisioning of the RING keyspace, and also representation of RING performance statistics, resource consumption, and health metrics through a rich set of graphs.

**Status Card**

The Status Card is available not only on the Supervisor GUI Dashboard, but persists throughout the GUI, in the tabs for Operation, Administration, and Provisioning.
The topmost section of the Status Card indicates the name of the RING and its current state, with the state also reflected in the color of the circle displayed on the card.

<table>
<thead>
<tr>
<th>Color</th>
<th>State Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>RUN state, no alerts noted</td>
</tr>
<tr>
<td>Orange</td>
<td>RING is balancing, or there are issues that require monitoring</td>
</tr>
<tr>
<td>Red</td>
<td>RING is incomplete or out of service</td>
</tr>
</tbody>
</table>

The Status Card also delivers basic RING metrics (hotlinked to correlating pages in the GUI), including the number of Nodes and Connectors associated with the RING, the number of Nodes with tasks running, the amount and type of any alerts present, the number and characteristics of objects stored, and the amount of disk space available.

**Graphical Display Views**

The Supervisor GUI Dashboard delivers a selection of graphical display views, accessible through a menu at the top of the tab that offers buttons labeled **Topology, Keyspace, Main, System, Ring, Connectors, Tiering, Disk Health, and Custom**.

**Topology**

Topology view displays a color "donut" chart that maps an operational view of RING Keyspaces.
are incomplete or out of service. Color gradients distinguish one Node from its neighbors, and one server from the other servers in the RING, and RING zones are displayed in varying shades of gray.

**Keyspace**

An analysis of the RING’s Keyspace is offered via **Keyspace** view, with the **Current configuration** section presenting information on both the classes of service supported and the ability to withstand various failure events within the context of the current Keyspace distribution.

![Keyspace of RING11_META](image)

A Classes of Service number followed by a + symbol represents the CoS level of objects on the RING when reading in consistent mode, consistency requiring that the majority of replicas remain readable after failure event occurrences.

The view also provides a list of theoretical and actual overhead percentages in both the nominal case (with no failure events), and in the ten worst cases for different types of failure events.

**Main, System, Storage, Ring, Tiering [Statistics Reporting]**

The Supervisor GUI Dashboard offers specific reporting views for RING operation and storage statistics, accessible via the **Main**, **System**, **Storage**, **Ring**, and **Tiering** buttons.

**Disk Health**

The Dashboard's **Disk Health** view offers a graphical representation of the GET, PUT, DELETE, and PHYSICAL DELETE transactions per second (TPS) for the disks of each physical Node server in a RING.

**Custom**

**Custom** view is the Dashboard’s monitoring jack-of-all-trades. In broad terms, statistics can be graphed and saved for:

- Specific Ring operations
- Operation system status
- Network connection processing
- Decentralized purge tasks
- RING tasks
- RING synchronization

**Saved Custom view graphs can be recalled, edited as necessary, and deleted.**
RingSH Command Line Interface

Monitoring of the RING can be accomplished through Scality’s RingSH command line interface, which provides various output commands for RING status and statistics, and for modifying RING module parameters.

SNMP-compliant MIB

Server statistics and alerts communication occurs via the Simple Network Management Protocol (SNMP), which the RING supports for Nodes, REST Connectors, and the Supervisor. In conjunction, the Scality-provided SNMP-compliant MIB (Management Information Base) enables popular data center tools (e.g., Nagios) to actively oversee the RING’s status and to receive alerts via SNMP traps. System health metrics, resource consumption, Connector and Storage Node performance statistics are all available for browsing via the MIB.
## Glossary

**ARC**
The Advanced Resilience Configuration (ARC) system is designed to increase failure tolerance at a storage cost lower than the classic replication of data to different disks. ARC uses Reed-Solomon erasure coding to gain space efficiency, with some added complexity for rebuild and GET operations where there is missing data. The advantages of using ARC in a Scality RING as compared to a data replication model are massively-reduced overhead and greater flexibility. With ARC, you can use any combination of data chunks and coding chunks for different classes of service and failure tolerance, and these can coexist on the same RING.

**Chord**
Scality RING uses the Chord protocol, initially developed at M.I.T., to map stored objects into a virtual keyspace. Chord’s value resides in its routing efficiency. Each Node just needs the knowledge of its successor and predecessor Nodes in the RING. Using the intelligence of the Chord protocol, the RING provides complete coverage of allotted keyspaces.

**Chunk**
A chunk is data that is stored on a Node. It has a key, and corresponding data, system metadata, and user metadata.

**Connectors**
Connectors are the interfaces between outside applications and the RING, receiving requests and dispatching data to different Storage Nodes. They implement a client version of the RING peer-to-peer protocol, and must be configured with the addresses of a small number of Storage Nodes which enable the Connectors to automatically discover all available Storage Nodes.

Connectors are designed to be stateless, allowing incoming requests to be load-balanced to any number of Connectors. They implement different protocols to access the underlying storage cloud, including:

- Scale Out File System (SOFS) Connectors that provide a view of the storage system to FUSE, CIFS/SMB, NFS, AFP, and CDMI applications.
- Dewpoint Connectors use the open standard CDMI protocol to provide a global name space for all files and objects stored in the RING.
- RS2 Connectors (which implement an Amazon S3 compatible API) and SRWS Connectors (for a simple HTTP-based REST storage service).
- NASDK Connectors, such as sproxd and scloned, which are also REST-based, and can be used in conjunction with other Connectors to provide specific functionality that may be especially useful in geo-distributed architectures. Also part of the NASDK are the `srebuildd` and `sindexd` components. The `srebuildd` component is used to rebuild missing or damaged objects in ARC.
erasure-coded deployments, while the sindexd component is used to facilitate indexing for RS2 Connector deployments.

Typically, Scality store deployments include a layer of Connector servers. However, when deployed with the Zimbra e-mail application, the Connectors run on each Zimbra server and do not require any additional hardware. Connectors can also be installed directly on Node servers.

**CoS**

The class of service (CoS) specifies the number of additional instances an object should have, i.e., for an object with a CoS of 2, there should be three total instances stored. The CoS takes up the final 8 bits of the UKS scheme for an object key, 4 bits for the class itself, and 4 bits to specify the replica number of a particular key. An object with a key ending with 20 will have two replicas, and those two replicas will have keys that end with 21 and 22 respectively.

Class numbers from 6 to 15 are reserved for special classes. For example, a key with a CoS of 7 specifies an object stored on an ARC system.

**inode**

An *inode* is a data structure that represents a filesystem object on Linux and other Unix-like operating systems. It is a number that is not computed, but which is picked at random when a file is created and stored in the directories. It stores all of the information on a file within the directories, save for its name and its actual data.

**Key range**

The key range of a StoreNode is the part of the RING for which the StoreNode is responsible. A RING goes from 0 to $(2^{160}) - 1$ and wraps on the $2^{160}$ value (like an analog wall clock that goes from 0 to 12, with the 0 and 12 having the same value). Each StoreNode is assigned a key on this RING.

The range of a StoreNode is defined by the key of the predecessor (i.e., the key of the nearest Node on the RING going counter-clockwise) and its own key. The key range is constructed to represent all the keys for which the StoreNode is responsible. For a given key, there is one (and only one) StoreNode that can be contacted to store or retrieve the data associated with the key.

The key range of a StoreNode may change if its direct predecessor is removed from the RING. The remaining StoreNode then becomes responsible for both key ranges merged into a single range: its original key range and the key range of the removed predecessor.

**Mesa**

The Mesa database is an object store, incorporating a B-tree architecture, with data entries in the key=value format. It performs order-of-relationship. The Mesa database offers online transaction processing of data, complete with commits and rollbacks, and it accepts typical storage queries for PUT, GET, and Delete operations.

**Node**

A Node is a virtual server that stores a range of objects. A minimum configuration of six Nodes per physical server as well as a minimum of six physical servers is recommended for each production-ready RING. Also known by the terms ‘StoreNode’ and ‘Storage Node’.

**Payload**

The major portion (128 bits) of a 160-bit object UKS key, where bit-fields provide a common way to address objects in a RING, helping to make sure that different services and different objects do not clash. The payload bit-field allocation is typically composed of 64 bits for the object ID, inode or message ID and revision; 32 bits for the volume ID, namespace or message box identifier; 8 bits for the application or service number; and 24 bits for an application-specific token or block number.
### Predecessor key
The key of the nearest Node on the RING going counter-clockwise. If the direct predecessor of a given Node is removed from the RING, the given Node takes responsibility for the key range of its removed predecessor, as well as for its own original key range.

### Purge task
The Purge task ensures that data marked as deleted is no longer retrievable and allows the I/O processes to reclaim the disk space occupied by the deleted chunks through the relocation process.

### Rebuild task
The Rebuild process re-creates missing replicas or updates out-of-date versions of an object.

### Relocation task
The Relocation task frees up space previously occupied by data that has been deleted by the end user and "physdeleted" (after a purge task) on the Bizstore node.

### Repair task
The Repair task reconstructs missing or corrupted data, regenerating the data from a predecessor replica, in contrast to the rebuild task, which reconstructs objects from a successor replica.

### RingSH
RingSH is a command line tool to monitor and manage a storage server RING and the Connectors that access the RING. It can be used as a simple interactive shell with a command history, or it can be called non-interactively from regular UNIX scripts.

### RS2
The Scality REST Storage Service Connector (RS2) processes HTTP requests and returns HTTP responses to applications based on the RFC 2616 protocol. RS2 Connectors may include authentication, billing, and access control list functionality. They can be set up to run on untrusted public networks, enabling easy and universal accessibility from the internet, while at the same time providing a secure layer of protection to the trusted private networks hosting the RING and backend storage systems.

RS2 Connectors are recommended for Storage-as-a-Service (STaaS) applications. You can configure RS2 to upload, retrieve, and manage data on an Amazon S3 (Simple Storage Service) backend.

### RS2 light
RS2 light is the RS2 Connector in light mode, i.e., without indexing, authentication, billing, or access control.

### sagentd
The sagentd component provides an overview of each server on which it is installed. It sends information and statistics from all server Nodes and Connectors to the Supervisor (on which an instance of sagentd must also be installed). With this component, a single connection between the Supervisor and a Node server replaces the multiple connections required with a Node-centric view, thereby enabling significant performance and reporting benefits for RING components.

### scloned
The scloned Connector provides a means to synchronize data and metadata between a local RING and a remote RING. The scloned Connector is installed on the local RING, which may be configured for replication or erasure coding. The Connector uses an srest driver to send data to an sproxyd Connector running on a remote RING, which may be configured very differently from the local RING.
<table>
<thead>
<tr>
<th>Glossary</th>
<th>What is the RING?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sfused</strong></td>
<td>sfused is the daemon used by SOFS connectors to provide a regular file system interface for Scality RING. Each object (directory or file) stored on the RING can be accessed through a regular UNIX POSIX mountpoint in a single global namespace. Since this model can handle $2^{64}$ indexed Nodes (inodes) per volume, the number of objects it can process is virtually unlimited. The sfused Connector is recommended for Dovecot and other POSIX-based file system applications.</td>
</tr>
<tr>
<td><strong>SOFS</strong></td>
<td>SOFS provides a scale-out filesystem interface to the RING, offering the possibility of unlimited space for content storage, and unlimited root directories within the same volume. It is useful in the context of POSIX-based mail systems, mapping each file to a single blob in a one-to-one relationship. The SOFS family of Connectors include Connectors for FUSE, NFS, SMB and CDMI file systems.</td>
</tr>
<tr>
<td><strong>sprov</strong></td>
<td>The sprov utility is used to assure a balanced distribution of generated keys — and existing keys, if any — to all Nodes (running processes) in the RING. Storage node-key balancing is carried out in two phases. In phase 1, RING Nodes are swapped randomly while the utility searches for an optimal balance in the event that some of the physical servers fail. In phase 2, a constraint is applied to ensure that each replica of an object will be on a distinct physical server.</td>
</tr>
<tr>
<td><strong>sproxyd</strong></td>
<td>sproxyd is typically installed on Node servers and used as an intermediary Connector. Although a Connector itself, it is normally accessed through another Connector (e.g., sfused) installed at the client location. In this architecture, only one data copy needs to be sent or received via HTTP between the client location and sproxyd. Since sproxyd handles the data replication at a RING location, the required bandwidth between the client application and the RING is significantly reduced.</td>
</tr>
<tr>
<td><strong>Storage Node</strong></td>
<td>A Storage Node is a virtual server that stores a range of objects. A minimum configuration of six Nodes per physical server as well as a minimum of six physical servers is recommended for each production-ready RING. Also known by the terms ‘Node’ and ‘StoreNode’.</td>
</tr>
<tr>
<td><strong>StoreNode</strong></td>
<td>A StoreNode is a virtual server that stores a range of objects. A minimum configuration of six Nodes per physical server as well as a minimum of six physical servers is recommended for each production-ready RING. Also known by the terms ‘Node’ and ‘Storage Node’.</td>
</tr>
<tr>
<td><strong>Supervisor</strong></td>
<td>The Supervisor is a central management console that lets you configure and monitor the entire storage system. The Supervisor is a passive component. It can be shut down or unplugged at any time without impacting production traffic. Typically there is one supervisor per data center although each supervisor can be configured to monitor and manage multiple data centers.</td>
</tr>
<tr>
<td><strong>UKS</strong></td>
<td>The Universal Key Scheme (UKS) defines the scheme to which the keys of all objects stored on the RING must adhere. It is 20 bytes (160 bits) long: the first 24 bits for the MD5 hash of the payload, the next 128 bits for the payload itself, and the last 8 bits for the CoS and replicas.</td>
</tr>
<tr>
<td><strong>Volumes</strong></td>
<td>A RING volume is a set of RING Connectors of the same family (e.g., SOFS). Each Connector in a volume shares a set of parameters specific to the volume, although it is possible to override these settings for a specific Connector.</td>
</tr>
</tbody>
</table>