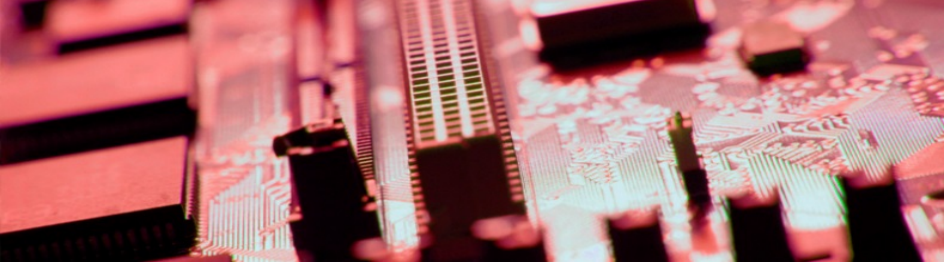


LCI workshop
on high performance clustered computing

Linux Clusters Institute: Spectrum Scale

Georgia Tech, August 15th – 18th 2017

J.D. Maloney | Storage Engineer
National Center for Supercomputing Applications (NCSA)
malone12@illinois.edu

NCSA

Spectrum Scale (GPFS) Overview

- Product of IBM, gone through many name changes
- Licensed file system, based on server socket count and client count
- One of the two “prominent” file systems in used today by the world’s largest supercomputers
- Generally considered easier to administer due to product maturity and Enterprise level features
- For most part expects to be run on top of reliable disks presented through redundant RAID controllers

IBM
Spectrum
Scale

Quick History of Spectrum Scale

- Began as the Tiger Shark File System in 1993 to handle multimedia applications
- Also was influenced by IBM's Vesta File System which was in development around the same time for use in different applications
- Was productized in 1994, and the name changed to IBM GPFS (General Parallel File System) around 1998
- Gone through many version changes and feature adds and in 2015 the name was changed to Spectrum Scale
  - Though we're all still getting used to that ☺

Quick History of Spectrum Scale

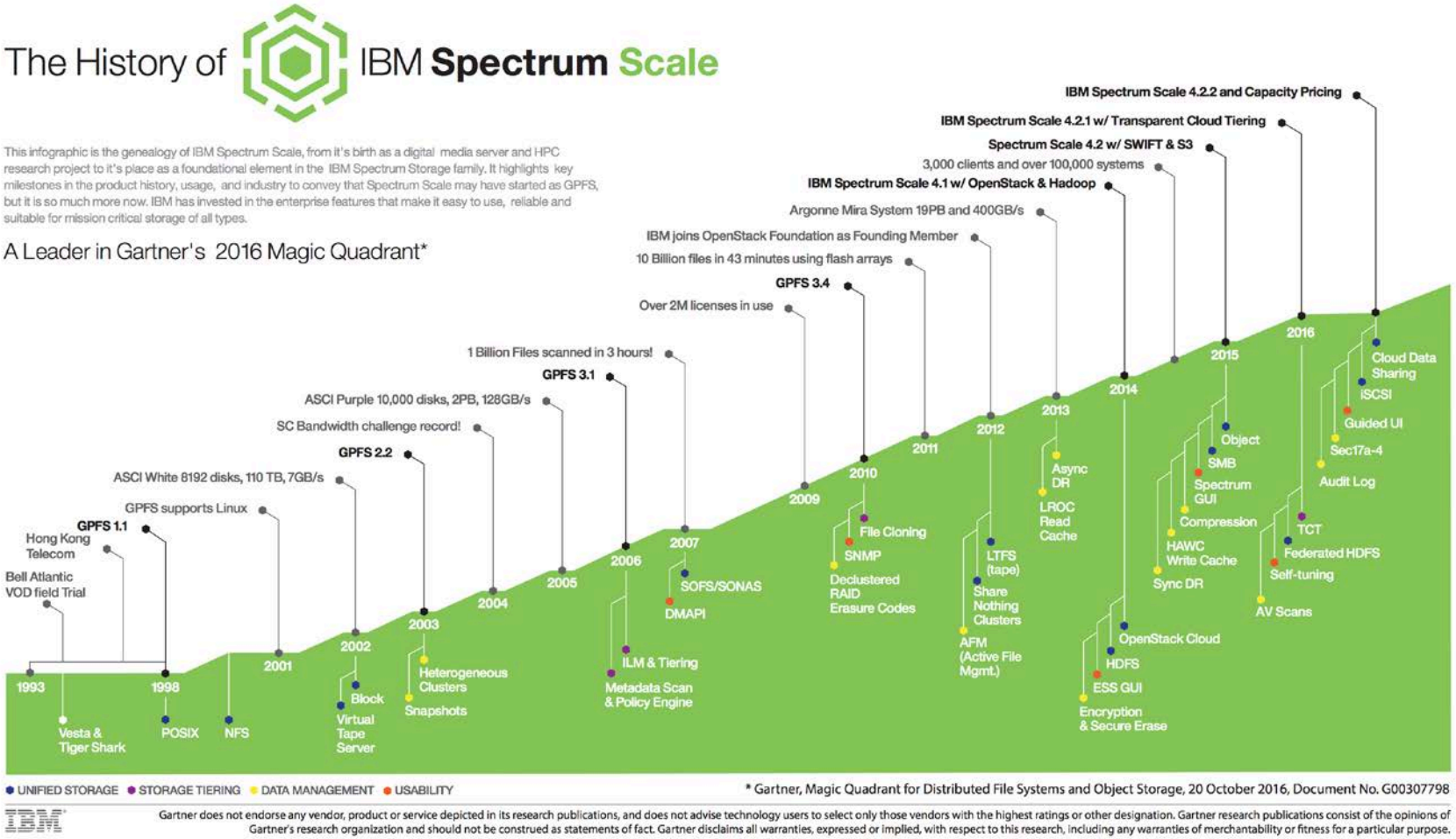


Image Credit: spectrumscale.org

Stand Out Spectrum Scale Features

- Distributed metadata servers, no real limit to number, could have it in all of them
- Allows data and metadata to be written inline with each other the same storage device, no separate devices needed
- Supports “Super inodes” where files less that ~3.8K actually fit inside the inode itself
  - Very handy when you have metadata pools that run on all flash devices
  - Leads to much improved small file performance

Stand Out Spectrum Scale Features

- Robust Tiering architecture based on storage pools
- Build in Policy Engine that can be used to query the file system and/or drive data movement
  - Run things like automated purges based on parameters
  - Move data between storage pools based on certain criteria
- Built in rebalancing of data across NSDs (LUNs)
  - Handy when you grow your storage system over time or when you're doing big migrations or upgrades
- Filesets for isolating different datasets
- Block Sharding
  - Block size can be split /32 to increase disk use

Spectrum Scale Weaknesses

- License cost that scales with deployment
  - Not an open source FS
- Sequential I/O performance doesn't scale as well
  - Progress on this in development for the CORAL program
- Multiple fabric support is less robust
- Requires more robust/enterprise hardware to present reliable NSDs to servers
  - Not true of FPO, but that is not a common HPC deployment type
  - Right now locks design into traditional RAID LUNs or GNR (which only comes from two vendors)
- Client limitation of around 10,000 clients (per IBM)

Popular Spectrum Scale Appliances

- Can buy ready built appliances from <u>many</u> vendors, here are <u>some</u>:

Seagate

DDN

Lenovo

Spectrum Scale Hardware

- Many other vendors will sell you hardware pre-configured for Spectrum Scale file systems
- Find solution that hits your price point and you have confidence in your vendor to provide a solid product
- Needs to be built off of reliable storage appliance that can present LUNs through multiple controllers to multiple hosts
  - Can be tested on less robust hardware but not for production
- Some gains can be had from integrated appliances, but comes with trade off of limited flexibility/customization

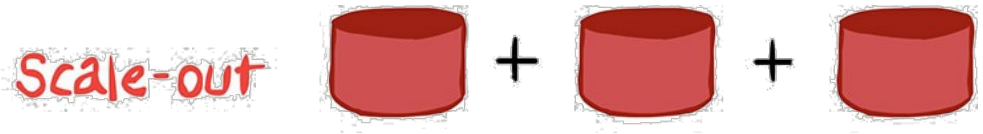
Spectrum Scale Concepts

Key Definitions

- NSD (Network Shared Disk) – LUN presented to a Spectrum scale server to be used for the file system(s)
- Cluster Manager – Spectrum Scale server that is elected to handle disk leases, detects and recovers node failures, distributes configuration information, etc.
- File System Manager – Spectrum Scale server that coordinates token management, disk space allocation, mount/unmount requests, etc. for a file system
- Quorum Node- Spectrum Scale server that helps the cluster maintain data integrity in case of node failure
- File System – A group of NSDs that are grouped together to form a mountable device on the client

Scaling Out

- Since Spectrum Scale servers can each deal with both data and metadata, scaling comes by just increasing the total number of NSD servers
- Many file systems can be run out of the same Spectrum Scale cluster (256 FS limit)
- What servers are cluster and file system managers is dynamic
  - Election held during startup of the mmfs daemon and managers can be moved around by admins to get them on desired nodes if there is a preference
  - Usually like to have even distribution as much as possible



Cluster vs Scatter

- Two different block allocation map types
- Parameter is chosen at file system create time, cannot be changed afterward
- Cluster allocates blocks in chunks (clusters) on NSDs
  - Better for clusters with smaller quantities of disks and or clients
- Scatter allocates blocks randomly across NSDs
  - Better for clusters with larger quantities of disks or clients
- The default setting is chosen based on the number of nodes and NSDs present in the cluster at the time of the create command
  - Threshold for switch from cluster to scatter is currently 8 nodes or disks

Spectrum Scale NSD Server

- Powerful Dual Socket CPU System
- More memory the better, used for page pool
  - Lowest you'd want to probably go is 32GB/64GB
  - We set our NSD server memory at 384GB currently
- Fast disks for metadata pool if possible
  - Great candidate for this is NVME
  - Disks now come in U.2 form factor for easier access
  - Metadata disks presented individually to Spectrum Scale
- Good network connectivity
  - Connectivity type partially depended on how you access your disk (IB SAN, SAS, Fiber Channel)
  - Cluster network type match your compute nodes (IB, OPA, Ethernet)
  - Balance the two as much as possible, leave some overhead for other tasks

Spectrum Scale Architecture

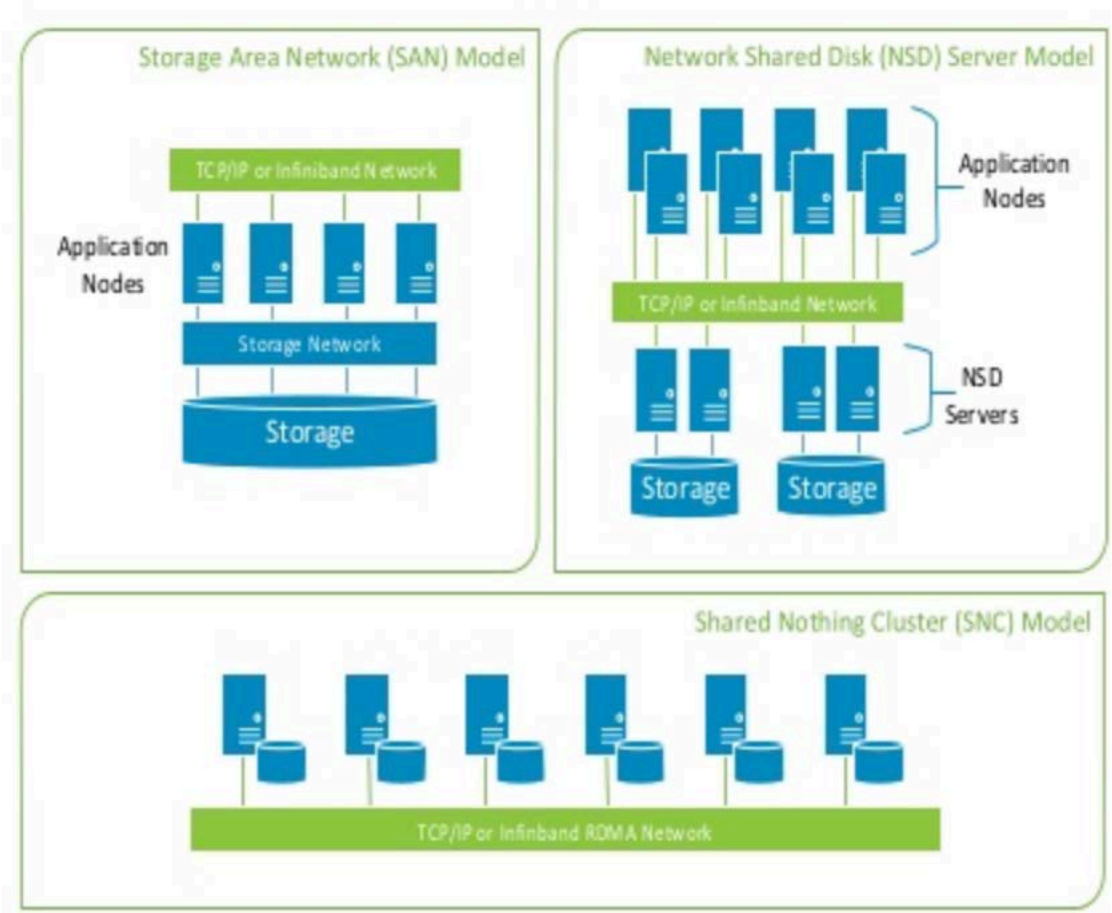


Image Credit: ibm.com

File Sets

- A way of breaking up a file system into different units that can all have different properties that all still use the same underlying NSDs
- Allows an admin to not have to sacrifice performance for the sake of logical separation
- Enables policy engine scans to run on individual file sets (if using independent inode spaces)
  - Speeds up the policy run
- Parameters that can each file set can have tuned separately:
  - Block Size
  - Inode Limits
  - Quotas

Spectrum Scale Node Classes

GPFS Node Classes

- A node class is simply a user defined logical grouping of nodes
- You can use a node class with any GPFS command that uses the ”-N” option to specify a list of nodes
- The systems in a group may perform the same type of functions
- The systems in a group may have the same characteristics, such as GPU processors, larger memory, faster CPUs, etc
- You may group servers together that have special GPFS configuration settings just for them

Creating a Node Class

- Can be handy to create a node class for your core Spectrum Scale servers (ones that host the disk)

```
# mmcrnodeclass
mmcrnodeclass: Missing arguments.
Usage:
  mmcrnodeclass ClassName -N {Node[,Node...] | NodeFile | NodeClass}

# mmcrnodeclass coreio -N ss-demo1.local,ss-demo2.local
mmcrnodeclass: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.
```

- Other potentially handy classes: CES nodes, login nodes, clients, GridFTP nodes

List of Node Classes

- Use the “mmlsnodeclass” command to view the current node classes on the system and what members are in them

```
# mmlsnodeclass
Node Class Name       Members
--------------------- -----------------------------------------------------------
coreio                ss-demo1.local,ss-demo2.local
#
```

Modifying a Node Class

- Use “mmchnodeclass” to modify a node list after it’s created (as more nodes come in, nodes leave, etc.)

```
# mmchnodeclass
mmchnodeclass: Missing arguments.
Usage:
  mmchnodeclass ClassName {add | delete | replace}
                -N {Node[,Node...] | NodeFile | NodeClass}

add - allows you to add additional nodes the the specified node class.

delete – deletes the specified nodes from the node class.

replace – replaces all nodes in the specified node class with the new list
provided.
```

Removing a Node Class

- Use the “mmdelnodeclass” command to remove a node class that is no longer necessary

```
# mmdelnodeclass coreio
mmdelnodeclass: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.
#
```

Spectrum Scale Tuning

Tuning Parameters – Where to start

- Start with the operating system and the attached disk systems. Make sure you have the optimal settings for your environment first before trying to tune Spectrum Scale.
- Run a baseline IOR and mdtest on the file system so you know what your initial performance numbers look like.
- Only make 1 change at time, running the IOR and mdtest after each change to verify if what you did hurt or helped the situation.

Tuning Parameters – Where to start

- As of Spectrum Scale 4.2.3.2, there are over 700 parameters within Spectrum Scale
  - Take a look at mmdiag --config output
- We are going to just touch on a few of them because Spectrum Scale has gotten much smarter at its own configuration

Tuning Parameters

Page Pool

- Determines the size of the Spectrum Scale file data block cache
- Unlike local file systems that use the operating system page cache to cache file data, Spectrum Scale allocates its own cache called the pagepool
- The Spectrum Scale pagepool is used to cache user file data and file system metadata
- Can be set on a node class basis
- Allocated at the startup of the mmfs daemon
  - For large pagepool sizes you may see delay on daemon startup while this gets allocated (tail log file /var/adm/ras/mmfs.log.latest)

Tuning Parameters

maxMBpS

- Specifies an estimate of how much performance into or out of a single node can occur
  - Default is 2048MB/s
- Value is used in calculating the amount of I/O that can be done to effectively pre-fetch data for readers and write-behind data from writers
- You can lower this amount to limit I/O demand from a single node on a cluster
- You can also raise this amount to increase the I/O demand allowed from a single node

Tuning Parameters

maxFilesToCache

- Controls how many file descriptors (inodes) each node can cache. Each file cached requires memory for the inode and a token(lock).
- Tuning Guidelines
  - The increased value should be large enough to handle the number of concurrently open files plus allow caching of recently used files and metadata operations such as "ls" on large directories.
  - Increasing maxFilesToCache can improve the performance of user interactive operations like running "ls".
  - Don't increase the value of maxFilesToCache on all nodes in a large cluster without ensuring you have sufficient token manager memory to support the possible number of outstanding tokens.

Tuning Parameters

maxStatCache

- The maxStatCache parameter sets aside pageable memory to cache attributes of files that are not currently in the regular file cache
  - This can be useful to improve the performance of stat() calls for applications with a working set that does not fit in the regular file cache
  - The memory occupied by the stat cache can be calculated as: maxStatCache × 176 bytes
- Upcoming versions of Spectrum Scale will support a peer-to-peer access of this cache to improve performance of the file system by reducing load on the NSD servers hosting the metadata

Tuning Parameters

nsdMaxWorkerThreads

- Sets the maximum number of NSD threads on an NSD server that will be concurrently transferring data with NSD clients
  - The maximum value depends on the sum of worker1Threads + prefetchThreads + nsdMaxWorkerThreads < 8192 on 64bit architectures
  - The default is 64 (in 3.4) 512 (in 3.5) with a minimum of 8 and maximum of 8,192
  - In some cases it may help to increase nsdMaxWorkerThreads for large clusters.
  - Scale this with the number of LUNs, not the number of clients. You need this to manage flow control on the network between the clients and the servers.

Spectrum Scale Snapshots

What Is A Snapshot

- A snapshot can preserver the state of a file system at a given moment in time
  - Snapshots at File System level are known as Global snapshots
- The space a snapshot takes up is the amount of blocks that would have been deleted or changed since the snapshot was taken
- Snapshots of a file system are read-only; changes can only be made to the active (that is, normal, non-snapshot) files and directories

What Is A Snapshot

- Creates a consistent copy of the file system at a given moment in time while not interfering with backups or replications occurring on the file system
- Allows for the easy recovery of a file, while not a backup, can be used as one in certain scenarios:
  - User accidental file deletion
  - Recovery of older file state for comparison
  - Accidental overwrite of file

Snapshot Types

File System Snapshot

- Taken for the entire file system. Again, only the changed blocks are stored to reduce the snapshot size

File Set Snapshot

- You can also take a snapshot of any independent inode file set separate from a file system snapshot
- Instead of creating a global snapshot of an entire file system, a fileset snapshot can be created to preserve the contents of a single independent fileset plus all dependent filesets that share the same inode space.
  - If an independent fileset has dependent filesets that share its inode space, then a snapshot of the independent fileset will also include those dependent filesets. In other words, a fileset snapshot is a snapshot of the whole inode space.

Snapshot Storage

- Snapshots are stored in a special read-only directory named .snapshots by default
- This directory resides in the top level directory of the file system.
- The directory can be linked into all subdirectories with the mmsnapdir command

```
Place a link in all directories:
# mmsnapdir fs0 –a

Undo the link above:
# mmsnapdir fs0 -r
```

Snapshot Creation(File system)

Use the “mmcrsnapshot” command to run the snapshot

```
Usage:
   mmcrsnapshot Device [[Fileset]:]Snapshot[,[[Fileset]:]Snapshot]...
                       [-j FilesetName[,FilesetName...]]
```

```
For a file system snapshot:
# mmcrsnapshot fs0 fs0_20170718_0001
Flushing dirty data for snapshot :fs0_20170718_0001...
Quiescing all file system operations.
Snapshot :fs0_20170718_0001 created with id 1.
#
```

Listing Snapshots

Use the “mmlssnapshot” command to view all the snapshots currently stored on a given file system

```
# mmlssnapshot fs0
Snapshots in file system fs0:
Directory                SnapId    Status  Created
Fileset
fs0_20170718_0001        1         Valid   Mon Jul 18 11:08:13 2017
#
```

Snapshot Creation(Fileset)

Creating a snapshot of just the home fileset on a file system

```
# mmcrsnapshot fs0 home:fs0_home_20170724_0612 -j home
Flushing dirty data for snapshot home:fs0_home_20170724_0612...
Quiescing all file system operations.
Snapshot home:fs0_home_20170724_0612 created with id 2.
#
```

```
Listing the snapshots for fs0 now shows a snapshot of the home fileset.
# mmlssnapshot fs0
Snapshots in file system fs0:
Directory                SnapId    Status  Created                   Fileset
fs0_20170718_0001        1         Valid   Mon Jul 24 11:08:13 2017
fs0_home_20170724_0612   2         Valid   Mon Jul 24 11:12:20 2017  home
[root@ss-demo1 ~]#
```

Snapshot Deletion(Filesystem)

Deleting the snapshot taken at the file system level using the "mmdelsnapshot" command

```
# mmdelsnapshot
mmdelsnapshot: Missing arguments.
Usage:
   mmdelsnapshot Device [[Fileset]:]Snapshot[,[[Fileset]:]Snapshot]...
               [-j FilesetName[,FilesetName...]] [--qos QosClass]
      [-N {all | mount | Node[,Node...] | NodeFile | NodeClass}]

# mmdelsnapshot fs0 fs0_20170718_0001
Invalidating snapshot files in :fs0_20170718_0001...
Deleting files in snapshot :fs0_20170718_0001...
 100.00 % complete on Mon Jul 24 11:17:52 2017  (    502784 inodes with total
1 MB data processed)
Invalidating snapshot files in :fs0_20170718_0001/F/...
Delete snapshot :fs0_20170718_0001 successful.
```

Snapshot Deletion(Fileset)

Deleting the snapshot taken at the file set level using the "mmdelsnapshot" command

```
# mmdelsnapshot fs0 home:fs0_home_20170724_0612 -j home
Invalidating snapshot files in home:fs0_home_20170724_0612...
Deleting files in snapshot home:fs0_home_20170724_0612...
 100.00 % complete on Mon Jul 24 11:25:56 2017  (    100096 inodes with total
0 MB data processed)
Invalidating snapshot files in home:fs0_home_20170724_0612/F/...
Delete snapshot home:fs0_home_20170724_0612 successful.
#

# mmlssnapshot fs0
No snapshots in file system fs0
#
```

File Level Restore from Snapshot

- In order to restore a file, you can traverse the directories in the .snapshots directory
- The directories have the name given to the snapshot when the mmcrsnapshot command was executed
- You can search for the file you want to restore and then use rsync or cp to copy the file wherever you would like, outside of the .snapshot directory

Directory Level Restore from Snapshot

- Restoring a directory is the same as restoring a file except that you must recursively copy things
- At NCSA, we use an in-house, publicly available script called mmsnaprest to handle the restores of files and directories
  - The source can be found at: https://github.com/ckerner/mmsnaprest.git

mmsnaprest : Snapshot Restore Utility

```
# mmsnaprest -h
   GPFS Restore From Snapshot

   Please note: This utility uses rsync style processing for directories.  If
                you are unsure of how that matching works, you may want to play
                with it in a test area.  There are examples in the EXAMPLES
                section of this help screen.

   Usage: mmsnaprest [-D|--debug]  [-u|--usage]  [-v|--verbose]  [-h|--help]
                     [--dry-run] [-ls SOURCE]   [-s SOURCE  -t TARGET]

   Option          Description
   -ls SOURCE      Just list all of the snapshot versions of a file/directory.

   -s SOURCE       Specify the source file/directory for the restore.  You will be
                   prompted for the version you wish to restore from.

   -t TARGET       Specify the target location for the restore.  If not specified,
                   you will be prompted for it.

                   TARGET is unique.  If you are restoring a file, you can specify
                   a directory to restore the file to, keeping its original name.
                   Or you can restore to a file name, either creating or overwriting
                   the original file.

   --dry-run       Generate a log of what would have been done, but don't do it.

   -v|--verbose    Show the rsync restore output on the terminal as well as logging it.

   -D|--debug      Turn on debugging.  This is very verbose.

   -h|--help       Print this help screen.
```

snappy: Snapshot Automation

- You can automate the creation of snapshots with a shell script, or even call the mmcrsnapshot command straight from cron if you like
- At NCSA, we use an in-house tool called snappy
  - Same utility for both file system and fileset snapshots
  - Written in python
  - Utilizes a simple windows ini style configuration file
  - Allows for a very customized approach to snapshots:
    - Hourly
    - Daily
    - Weekly
    - Monthly
    - Quarterly
    - Yearly
  - Available at: https://github.com/ckerner/snappy.git

snappy: Snapshot Automation

```
# snappy --help
usage: snappy [-h] [--cron] [--hourly] [--daily] [--dow] [--weekly] [--monthly] [--quarterly] [--yearly] [-v] [-d] [-t] [-n]
Spectrum Scale Snapshot Wrapper
optional arguments:
  -h, --help     show this help message and exit
  --cron         Generate the crontab entries to run all of the snapshots.
  --hourly       Generate HOURLY snapshots.
  --daily        Generate DAILY snapshots.
  --dow          Generate DAY OF WEEK snapshots.
  --weekly       Generate WEEKLY snapshots.
  --monthly      Generate MONTHLY snapshots.
  --quarterly    Generate QUARTERLY snapshots.
  --yearly       Generate YEARLY snapshots.
  -v, --verbose  Toggle Verbose Mode. DEFAULT: False
  -d, --debug    Toggle Debug Mode. DEFAULT: False
  -t, --trace    Toggle Debug Mode. DEFAULT: False
  -n             Do not actually run, but log what would be done. Implies debug option. DEFAULT: False

This requires GPFS to be installed in the default location.
#
```

Snapshot Configuration File: .snapcfg

The configuration file always must reside in the root top level directory of the file system.

```
# cat .snapcfg
[DEFAULT]
Active=False
SnapType=Fileset
Versions=30
Frequency=daily

[home]
Active=True

[projects]
Active=True
Versions=7

[software]
Active=True
Frequency=weekly
Versions=14
#
```

From this example, the default is for fileset snapshots, running daily, keeping 30 versions. The default action is to NOT take snapshots. So, if you want a snapshot, you must turn it on for each fileset individually.

The .snapcfg section name must be the same as the fileset name. Each section will inherit the DEFAULT section and then override it with the local values. Here is the breakdown for this file:

- [home] get daily snapshots with 30 versions saved.
- [projects] gets daily snapshots with 7 versions.
- [software] gets a weekly snapshot with 14 versions.

Spectrum Scale Cluster Export Services

CES – Cluster Export Services

- Provides highly available file and object services to a Spectrum Scale cluster such as NFS, SMB, Object, and Block

High availability

- With Spectrum Scale, you can configure a subset of nodes in the cluster to provide a highly available solution for exporting Spectrum Scale file systems usings NFS, SMB, Object, and Block.
  - Nodes are designated as Cluster Export Services (CES) nodes or protocol nodes. The set of CES nodes is frequently referred to as the CES cluster.
- A set of IP addresses, the CES address pool, is defined and distributed among the CES nodes
  - If a node enters or exits the CES Cluster, IP Addresses are dynamically reassigned
  - Clients use these floating IP Address to access the CES services

CES – Cluster Export Services

Monitoring

- CES monitors the state of the protocol services itself
  - Checks not just for host availability, but also the health of the services
  - If a failure is detected CES will migrate IP Address away from a node and mark it as offline for CES services

Protocol support

- CES supports the following export protocols: NFS, SMB, object, and iSCSI (block)
  - Protocols can be enabled individually
  - If a protocol is enabled, all CES nodes will serve that protocol
- The following are examples of enabling and disabling protocol services by using the **mmces** command:
  - mmces service enable nfs Enables the NFS protocol in the CES cluster.
  - mmces service disable obj Disables the Object protocol in the CES cluster.

CES Commands

- mmblock - Manages the BLOCK configuration operations
- mmces - Manages the CES address pool and other CES cluster configuration options
- mmnfs - Manages NFS exports and sets the NFS configuration attributes
- mmobj - Manages the Object configuration operations
- mmsmb - Manages SMB exports and sets the SMB configuration attributes
- mmuserauth - Configures the authentication methods that are used by the protocols

Spectrum Scale Policy Engine

Policy Engine

- The GPFS policy engine allows you to run SQL-like queries against the file system and get reports based on those queries
- The policy engine can also be used to invoke actions, such as compression, file movement, etc
- Customized scripts can also be invoked, letting you have full control over anything that is being done
- There are many parameters that can be specified. For a list of them, check out the Spectrum Scale Administration and Programming Reference

Example Policy Run

- Here is a simple sample policy that will just list all of the files in /fs0/projects along with the file's allocation, its actual size, owner and fileset name. It also displays the inode number and fully qualified path name.

```
# cat rules.txt
RULE 'listall' list 'all-files'
SHOW( varchar(kb_allocated) || '  ' || varchar(file_size) || ' ' ||
varchar(user_id) || ' ' || fileset_name  )
WHERE PATH_NAME LIKE '/fs0/projects/%'
```

Example Policy Run

Sample output from a policy run:

```
# mmapplypolicy fs0 -f /fs0/tmp/ -P rules.txt -I defer
[I] GPFS Current Data Pool Utilization in KB and %
Pool_Name                    KB_Occupied        KB_Total  Percent_Occupied
archive                           131072        41934848      0.312561047%
data                              192512        41934848      0.459074038%
system                                 0               0      0.000000000% (no user
data)
[I] 4422 of 502784 inodes used: 0.879503%.
[W] Attention: In RULE 'listall' LIST name 'all-files' appears but there is no
corresponding "EXTERNAL LIST 'all-files' EXEC ... OPTS ..." rule to specify a
program to process the matching files.
[I] Loaded policy rules from rules.txt.
Evaluating policy rules with CURRENT_TIMESTAMP = 2017-07-25@15:34:38 UTC
Parsed 1 policy rules.
RULE 'listall' list 'all-files'
SHOW( varchar(kb_allocated) || '  ' || varchar(file_size) || ' ' ||
varchar(user_id) || ' ' || fileset_name  )
WHERE PATH_NAME LIKE '/fs0/projects/%'
[I] 2017-07-25@15:34:39.041 Directory entries scanned: 385.
[I] Directories scan: 362 files, 23 directories, 0 other objects, 0 'skipped' files
and/or errors.
[I] 2017-07-25@15:34:39.043 Sorting 385 file list records.
[I] Inodes scan: 362 files, 23 directories, 0 other objects, 0 'skipped' files
and/or errors.
```

Example Policy Run

Sample output from a policy run (continued):

```
[I] 2017-07-25@15:34:40.954 Policy evaluation. 385 files scanned.
[I] 2017-07-25@15:34:40.956 Sorting 360 candidate file list records.
[I] 2017-07-25@15:34:41.024 Choosing candidate files. 360 records scanned.
[I] Summary of Rule Applicability and File Choices:
 Rule#      Hit_Cnt          KB_Hit          Chosen       KB_Chosen          KB_Ill
Rule
     0          360           61184             360           61184               0
RULE 'listall' LIST 'all-files' SHOW(.) WHERE(.)

[I] Filesystem objects with no applicable rules: 25.

[I] GPFS Policy Decisions and File Choice Totals:
 Chose to list 61184KB: 360 of 360 candidates;
Predicted Data Pool Utilization in KB and %:
Pool_Name                    KB_Occupied        KB_Total  Percent_Occupied
archive                           131072        41934848      0.312561047%
data                              192512        41934848      0.459074038%
system                                 0               0      0.000000000% (no user
data)
[I] 2017-07-25@15:34:41.027 Policy execution. 0 files dispatched.
[I] A total of 0 files have been migrated, deleted or processed by an EXTERNAL
EXEC/script;
        0 'skipped' files and/or errors.
#
```

Example Policy Run

Sample output from a policy run:

```
]# wc -l /fs0/tmp/list.all-files
360 /fs0/tmp/list.all-files

# head -n 10 /fs0/tmp/list.all-files
402432 374745509 0  3584  1741146 0 projects -- /fs0/projects/dar-2.4.1.tar.gz
402434 229033036 0  0  1217 1000 projects -- /fs0/projects/dar-2.4.1/README
402435 825781038 0  256  43668 1000 projects -- /fs0/projects/dar-2.4.1/config.guess
402436 1733958940 0  256  18343 1000 projects -- /fs0/projects/dar-2.4.1/config.rpath
402437 37654404 0  0  371 1000 projects -- /fs0/projects/dar-2.4.1/INSTALL
402438 1471382967 0  0  435 1000 projects -- /fs0/projects/dar-2.4.1/TODO
402440 398210967 0  0  376 1000 projects -- /fs0/projects/dar-2.4.1/misc/batch_cygwin
402441 292549403 0  0  738 1000 projects -- /fs0/projects/dar-2.4.1/misc/README
402442 1788675584 0  256  3996 1000 projects -- /fs0/projects/dar-2.4.1/misc/dar_ea.rpm.proto
402443 637382920 0  256  4025 1000 projects -- /fs0/projects/dar-2.4.1/misc/dar64_ea.rpm.proto
#
```

Spectrum Scale Storage Pools & File Placement

Storage Pools

- Physically, a storage pool is a collection of disks or RAID arrays
  - Allow you to group multiple storage systems within a file system.
- Using storage pools, you can create tiers of storage by grouping storage devices based on performance, locality, or reliability characteristics
  - One pool could be an All Flash Array (AFA) with high-performance SSDs
  - Another pool might consist of numerous disk controllers that host a large set of economical SAS/SATA drives

Example Storage Pool Configuration

Block Size

- A system pool with a 1M block size for metadata
- A data pool with a block size that best meets the storage requirements of the users

Hardware

- System pool backed by NVME flash
- Data pool backed by SATA/SAS based storage appliance (DDN, NetApp, etc)

Information Lifecycle Management in Spectrum Scale

- Spectrum Scale includes the ILM toolkit that allows you to manage your data via the built in policy engine
- No matter the directory structure, Spectrum Scale can automatically manage what storage pools host the data, and for how long
  - Throughout the life of the data Spectrum scale can track and migrate data from your policy driven rules
- You can match the data and its needs to hardware, allowing for cost savings
- Great method for spanning infrastructure investments
  - New hardware is for more important/more used data
  - Older hardware becomes the slower storage pool

Information Lifecycle Management in Spectrum Scale

- There are three types of storage pools in Spectrum Scale:
  - A required system pool that you create
  - Optional user storage pools that you create
  - Optional external storage pools that you define with policy rules and manage through an external application (eg. Spectrum Protect/Tivoli)
- Create filesets to provide a way to partition the file system namespace to allow administrative operations to work at a narrower level than the entire file system
- Create policy rules based on data attributes to determine the initial file placement and manage data placement throughout the life of the file

Tiered Storage

- Tiered storage is the assignment of different categories of data to various storage media to reduce the total storage cost and/or meet certain business requirements
- Tiers are determined by performance and business criteria, cost of the media
- Data is ranked and stored based on performance and/or user requirements

Tiered Storage – What does it look like?

- Your Spectrum Scale cluster can have differing types of storage media connected by differing technologies as well, including but not limited to:
  - NVME – Non-Volatile Memory Express
  - SSD -- Solid State Disk
  - SAS attached disk (NetApp, Dell, etc)
  - Fiber attached disk
  - Infiniband attached disk(DDN, etc)
  - Self-Encrypting drives
  - High Speed SCSI drives
  - You may have differing drives sizes: 2T, 4T, 8T, 10T, etc
  - SMR – Shingled Magnetic Recording drives
  - And the list goes on and on

Tiered Storage – How do I use it?

- Lets say you have the following devices on your system:
  - /dev/nvme01 /dev/nvme02
  - /dev/sas01 /dev/sas02
  - /dev/smr01 /dev/smr02
- There are many different ways that you can configure a Spectrum Scale file system. To make it interesting, lets have the following business rules that we need to satisfy:
  - Very fast file creates and lookups, including a mirrored copy.
  - A decent storage area for data files
  - An additional area for files over a week old that are not going to be updated, just read consistently
  - What would this configuration look like?

Tiered Storage: Configuration Example

```
%pool:
   pool=data
   blockSize=4M
   usage=dataOnly
   layoutMap=scatter
%nsd:
   nsd=sas01
   usage=dataOnly
   pool=data
%nsd:
   nsd=sas02
   usage=dataOnly
   pool=data
```

```
%pool:
   pool=archive
   blockSize=4M
   usage=dataOnly
   layoutMap=scatter
%nsd:
   nsd=smr01
   usage=dataOnly
   pool=archive
%nsd:
   nsd=smr02
   usage=dataOnly
   pool=archive
```

```
%nsd:
   nsd=nvme01
   usage=metadataOnly
   pool=system
%nsd:
   nsd=nvme02
   usage=metadataOnly
   pool=system
```

File Placement Policies

- If you are utilizing multiple storage pools within Spectrum Scale, you must specify a default storage policy at a minimum.
- File placement policies are used to control what data is written to which storage pool.
- A default policy rule can be quite simple. For example, if you have a ’data’ pool and want to write all files there, create a file called policy with a single line containing the following rule:

```
# cat policy
rule 'default' set pool 'data'
```

Installing File Placement Policies

```
# Usage: mmchpolicy Device PolicyFilename
       [-t DescriptiveName] [-I {yes|test}]
```

Test the policy before installing it is good practice!

```
# mmchpolicy fs0 policy -I test
Validated policy 'policy': Parsed 1 policy rules.
```

No errors on the policy, so lets install it:

```
# mmchpolicy fs0 policy
Validated policy 'policy': Parsed 1 policy rules.
Policy `policy' installed and broadcast to all nodes.
```

Viewing Installed Policies

```
# Usage: mmlspolicy Device

List the file placement policies:
# mmlspolicy fs0
Policy for file system '/dev/fs0':
   Installed by root@ss-demo1.os.ncsa.edu on Fri Jul 21 06:26:10 2017.
   First line of policy 'policy' is:
rule 'default' set pool 'data'
#
```

Viewing Storage Pool Information

- You can monitor the usage of a pool with the mmlspool command
  - Will show how much space is allocated and used within each storage pool of a file system

```
# mmlspool fs0
Storage pools in file system at '/fs0':
Name                    Id   BlkSize Data Meta Total Data in (KB)   Free Data in (KB)   Total Meta in (KB)    Free Meta in (KB)
system                   0   1024 KB   no  yes              0             0 (  0%)       20963328       16295936 ( 78%)
data                 65537      4 MB  yes   no       41934848       41795584 (100%)             0             0 (  0%)
archive              65538      4 MB  yes   no       41934848       41803776 (100%)             0             0 (  0%)
#
```

Spectrum Scale Monitoring

Monitoring with mmpmon

- Built in tool to report counters that each of the mmfs daemons keep
- Can output results in either machine parseable or human readable formats
- Some of the statistics it monitors on a per host basis:
  - Bytes Read
  - Bytes Written
  - File Open Requests
  - File Close Requests
  - Per NSD Read/Write
- The machine parseable output is easy to use for scripted data gathering

Monitoring with mmpmon

Sample output from mmpmon (human readable)

```
[                ~]# mmpmon -i infile
mmpmon node 1            name f          nlist add
initial status 0
name count 3
timestamp 1501092824/752635
node name          , OK (name used:          , IP address          )
node name          , OK (name used:          , IP address          )
node name          , status 22
final status 0
node names processed 2
current node list count 2
mmpmon node          name          fs_io_s OK
cluster:
filesystem:     testfs
disks:                6
timestamp:      1501092824/757325
bytes read:           0
bytes written:        0
opens:               10
closes:              10
reads:                0
writes:               0
readdir:             10
inode updates:        9
```

Monitoring with mmpmon

Sample output from mmpmon (machine parseable)

```
[root@fstest00 ~]# mmpmon -i infile -p
_nlist_ _n_ [redacted] _nn_ fstest00 _req_ add _rc_ 0 _t_ 1501092488 _tu_ 204061 _c_ 3
_nlist_ _n_ [redacted] _nn_ fstest00 _req_ add _rc_ 0 _t_ 1501092488 _tu_ 204061 _ni_ fstest00 _nx_ fstest00 _nxip_ [redacted]
_nlist_ _n_ [redacted] _nn_ fstest00 _req_ add _rc_ 0 _t_ 1501092488 _tu_ 204061 _ni_ fstest01 _nx_ fstest01 _nxip_
_nlist_ _n_ [redacted] _nn_ fstest00 _req_ add _rc_ 22 _t_ 1501092488 _tu_ 204061 _ni_ fstest02 _nx_ ? _nxip_ ?
_nlist_ _n_ [redacted] _nn_ fstest00 _req_ add _rc_ 0 _t_ 1501092488 _tu_ 204061 _did_ 2 _nlc_ 2
_fs_io_s_ _n_ [redacted] _nn_ fstest00 _rc_ 0 _t_ 1501092488 _tu_ 212557 _cl_ [redacted] _fs_ testfs _d_ 6 _br_ 0 _bw_ 0 _oc_ 70337 _cc_ 70332 _rdc_ 3 _wc_ 0 _dir_ 70323 _iu_ 57513
_fs_io_s_ _n_ [redacted] _nn_ fstest01 _rc_ 0 _t_ 1501092510 _tu_ 962031 _cl_ [redacted] _fs_ testfs _d_ 6 _br_ 0 _bw_ 0 _oc_ 62621 _cc_ 62619 _rdc_ 1 _wc_ 0 _dir_ 62615 _iu_ 254
_reset_ _n_ [redacted] _nn_ fstest00 _rc_ 0 _t_ 1501092488 _tu_ 213235
_reset_ _n_ [redacted] _nn_ fstest01 _rc_ 0 _t_ 1501092510 _tu_ 962573
```

Can be used to make useful graphs

Other Spectrum Scale Monitoring

- Using the built in ZiMon sensors with mmperfmon
- Spectrum Scale GUI now has the ability to have performance monitoring with graphs
- Spectrum Scale Grafana Bridge
  - Python standalone application that puts Spectrum Scale performance data into openTSDB which Grafana can understand
  - Data that is pushed “across” the bridge is gathered by the ZiMon Monitoring Tool

Resources

- https://www.ibm.com/support/knowledgecenter/en/STXKQY_4.2.0/ibmspectrumscale42_content.html
- http://www.spectrumscale.org/join/

Acknowledgements

- Members of the SET group at NCSA for slide creation and review
- Members of the steering committee for slide review

Questions