Abstract

Most file systems are a thin layer of organization on top of a block device and cannot efficiently address data at large scale. This paper focuses on OneFS, a modern file system that meets the unique needs of Big Data. OneFS includes the Job Engine, a parallel scheduling and job management framework, which enables data protection and storage management tasks to be distributed across the cluster and run efficiently.

August 2019
# TABLE OF CONTENTS

**Introduction** ........................................................................................................................................................................ 4

**Intended Audience** ............................................................................................................................................................ 4

**Job Engine** ......................................................................................................................................................................... 4

  - Overview ............................................................................................................................................................................. 4
  - Job hierarchy ..................................................................................................................................................................... 5
  - Job types .......................................................................................................................................................................... 6
  - Inode (LIN) scan ............................................................................................................................................................... 6
  - Tree walk .......................................................................................................................................................................... 6
  - Drive scan ........................................................................................................................................................................ 6
  - Changelist .......................................................................................................................................................................... 6
  - Job engine jobs ................................................................................................................................................................. 6
  - Job impact policies .......................................................................................................................................................... 9
  - Job priority ....................................................................................................................................................................... 12

**Multiple job execution** .................................................................................................................................................... 13

**Job exclusion sets** .............................................................................................................................................................. 13

  - Restriping exclusion set .................................................................................................................................................... 13
  - Marking exclusion set ....................................................................................................................................................... 14
  - Non-exclusion jobs ........................................................................................................................................................... 15

**Job Engine management** .................................................................................................................................................. 16

  - Manual job execution .................................................................................................................................................... 16
  - Scheduled job execution .................................................................................................................................................. 17
  - Proactive job execution .................................................................................................................................................... 20
  - Reactive job execution .................................................................................................................................................... 20
  - Job control ........................................................................................................................................................................ 20

**Job Engine orchestration and execution** .......................................................................................................................... 21

  - Coordinator process ....................................................................................................................................................... 22
  - Director process ................................................................................................................................................................. 22
  - Manager process ............................................................................................................................................................... 22
  - Worker threads ................................................................................................................................................................. 23
  - Job Engine checkpoints .................................................................................................................................................. 23
  - Job Engine resource utilization monitoring ................................................................................................................... 23
  - Job Engine throttling and flow control ............................................................................................................................ 23
  - Job Engine low space behaviour .................................................................................................................................. 24
  - Job performance ............................................................................................................................................................... 24
File system protection and management ................................................................. 25
  FlexProtect .................................................................................................................. 25
  MultiScan ..................................................................................................................... 26
  AutoBalance ............................................................................................................... 26
  Collect .......................................................................................................................... 26
  MediaScan ................................................................................................................... 27
  IntegrityScan ............................................................................................................... 27
  Jobs and OneFS licensing .......................................................................................... 27

Job Engine monitoring and reporting ........................................................................ 28
  Job events ..................................................................................................................... 28
  Job reports .................................................................................................................. 29
  Active job details ....................................................................................................... 29
  Performance resource management .......................................................................... 30

Job Engine best practices ......................................................................................... 31

Job Engine considerations ......................................................................................... 31

Conclusion .................................................................................................................. 33

Version History .......................................................................................................... 33

TAKE THE NEXT STEP .............................................................................................. 34

Shop Dell EMC Isilon to compare features and get more information ....................... 34
Introduction

IT managers across all industries are facing unprecedented rates of data growth, driving up the cost and complexity of managing their file storage environments. Dell EMC Isilon provides a linearly scalable storage resource pool that is efficient, safe, simple to provision and operate, and which focuses on the efficient management of the data itself, rather than the storage infrastructure.

Intended Audience

This paper presents information for configuring and managing system jobs on a Dell EMC Isilon cluster. This paper does not intend to provide a comprehensive background to the Isilon OneFS architecture.

Please refer to the OneFS Technical Overview white paper for further details on the OneFS architecture.

The target audience for this white paper is anyone configuring and managing an Isilon clustered storage environment. It is assumed that the reader has an understanding and working knowledge of the OneFS components, architecture, commands and features.

More information on OneFS commands and feature configuration is available in the OneFS Administration Guide.

Job Engine

Overview

Dell EMC Isilon OneFS is a highly distributed network attached storage architecture that leverages the power and efficiency of automation via parallel process execution wherever possible. Examples of this include multiple nodes participating in data reconstruction in the event of a hard drive failure, multiple workers and data streams accelerating replication and backup tasks, etc. Such cluster tasks require a distributed scheduling, execution, and reporting framework to help manage them efficiently. This execution and scheduling framework is the OneFS Job Engine.

After duplicate blocks are discovered, SmartDedupe moves a single copy of those blocks to a special set of files known as shadow stores. During this process, duplicate blocks are removed from the actual files and replaced with pointers to the shadow stores.

With post-process deduplication, new data is first stored on the storage device and then a subsequent process analyzes the data looking for commonality. This means that initial file-write or modify performance is not impacted, since no additional computation is required in the write path.

Job Engine architecture

The OneFS Job Engine runs across the entire cluster and is responsible for dividing and conquering large storage management and protection tasks. To achieve this, it reduces a task into smaller work items and then allocates, or maps, these portions of the overall job to multiple worker threads on each node. Progress is tracked and reported on throughout job execution and a detailed report and status is presented upon completion or termination.

Job Engine includes a comprehensive check-pointing system which allows jobs to be paused and resumed, in addition to stopped and started. The Job Engine framework also includes an adaptive impact management system, drive-sensitive impact control, and the ability to run multiple jobs at once.

The Job Engine typically executes jobs as background tasks across the cluster, using spare or especially reserved capacity and resources. The jobs themselves can be categorized into three primary classes:

- **File System Maintenance Jobs**
  These jobs perform background file system maintenance, and typically require access to all nodes. These jobs are required to run in default configurations, and often in degraded cluster conditions. Examples include file system protection and drive rebuilds.

- **Feature Support Jobs**
  The feature support jobs perform work that facilitates some extended storage management function, and typically only run when the feature has been configured. Examples include deduplication and anti-virus scanning.

- **User Action Jobs**
These jobs are run directly by the storage administrator to accomplish some data management goal. Examples include parallel tree deletes and permissions maintenance.

Although the file system maintenance jobs are run by default, either on a schedule or in reaction to a particular file system event, any Job Engine job can be managed by configuring both its priority-level (in relation to other jobs) and its impact policy. These are covered in more detail below.

![Figure 1: OneFS Job Engine Primary Functions](image)

**Job hierarchy**

Job Engine jobs often comprise several phases, each of which are executed in a pre-defined sequence. These run the gamut from jobs like TreeDelete, which have just a single phase, to complex jobs like FlexProtect and MediaScan that have multiple distinct phases.

A job phase must be completed in entirety before the job can progress to the next phase. If any errors occur during execution, the job is marked “failed” at the end of that particular phase and the job is terminated.

Each job phase is composed of a number of work chunks, or Tasks. Tasks, which are comprised of multiple individual work items, are divided up and load balanced across the nodes within the cluster. Successful execution of a work item produces an item result, which might contain a count of the number of retries required to repair a file, plus any errors that occurred during processing.
Job types

When a Job Engine job needs to work on a large portion of the file system, there are three main access methods available to accomplish this:

- Inode (LIN) Scan
- Tree Walk
- Drive Scan

Inode (LIN) scan

The most straightforward access method is via metadata, using a Logical Inode (LIN) Scan. In addition to being simple to access in parallel, LINs also provide a useful way of accurately determining the amount of work required.

Tree walk

A directory tree walk is the traditional access method since it works similarly to common UNIX utilities, such as find – albeit in a far more distributed way. For parallel execution, the various job tasks are each assigned a separate subdirectory tree. Unlike LIN scans, tree walks may prove to be heavily unbalanced, due to varying sub-directory depths and file counts.

Drive scan

Disk drives provide excellent linear read access, so a drive scan can deliver orders of magnitude better performance than a directory tree walk or LIN scan for jobs that don’t require insight into file system structure. As such, drive scans are ideal for jobs like MediaScan, which linearly traverses each node’s disks looking for bad disk sectors.

Changelist

The fourth class of Job Engine jobs utilize a ‘changelist’, rather than LIN-based scanning. The changelist approach analyzes two snapshots to find the LINs which changed (delta) between the snapshots, and then dives in to determine the exact changes.

Job engine jobs

The table below provides a comprehensive list of the exposed jobs and operations that the OneFS Job Engine performs, and their file system access methods:
<table>
<thead>
<tr>
<th>Job Name</th>
<th>Job Description</th>
<th>Access Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoBalance</td>
<td>Balances free space in the cluster.</td>
<td>Drive + LIN</td>
</tr>
<tr>
<td>AutoBalanceLin</td>
<td>Balances free space in the cluster.</td>
<td>LIN</td>
</tr>
<tr>
<td>AVScan</td>
<td>Virus scanning job that ICAP server(s) run.</td>
<td>Tree</td>
</tr>
<tr>
<td>ChangelistCreate</td>
<td>Create a list of changes between two consecutive SyncIQ snapshots</td>
<td>Changelist</td>
</tr>
<tr>
<td>CloudPoolsLin</td>
<td>Archives data out to a cloud provider according to a file pool policy.</td>
<td>LIN</td>
</tr>
<tr>
<td>CloudPoolsTreewalk</td>
<td>Archives data out to a cloud provider according to a file pool policy.</td>
<td>Tree</td>
</tr>
<tr>
<td>Collect</td>
<td>Reclaims disk space that could not be freed due to a node or drive being unavailable while they suffer from various failure conditions.</td>
<td>Drive + LIN</td>
</tr>
<tr>
<td>ComplianceStoreDelete</td>
<td>SmartLock Compliance mode garbage collection job.</td>
<td>Tree</td>
</tr>
<tr>
<td>Dedupe</td>
<td>Deduplicates identical blocks in the file system.</td>
<td>Tree</td>
</tr>
<tr>
<td>DedupeAssessment</td>
<td>Dry run assessment of the benefits of deduplication.</td>
<td>Tree</td>
</tr>
<tr>
<td>DomainMark</td>
<td>Associates a path and its contents with a domain.</td>
<td>Tree</td>
</tr>
<tr>
<td>DomainTag</td>
<td>Associates a path and its contents with a domain.</td>
<td>Tree</td>
</tr>
<tr>
<td>EsrsMftDownload</td>
<td>ESRS managed file transfer job for license files.</td>
<td></td>
</tr>
<tr>
<td>FilePolicy</td>
<td>Efficient SmartPools file pool policy job.</td>
<td>Changelist</td>
</tr>
<tr>
<td>FlexProtect</td>
<td>Rebuilds and re-protects the file system to recover from a failure scenario.</td>
<td>Drive + LIN</td>
</tr>
<tr>
<td>FlexProtectLin</td>
<td>Reprotects the file system.</td>
<td>LIN</td>
</tr>
<tr>
<td>FSAnalyze</td>
<td>Gathers file system analytics data that is used in conjunction with InsightIQ.</td>
<td>Changelist</td>
</tr>
<tr>
<td>IndexUpdate</td>
<td>Creates and updates an efficient file system index for FilePolicy and FSAnalyze jobs,</td>
<td>Changelist</td>
</tr>
<tr>
<td>IntegrityScan</td>
<td>Performs online verification and correction of any file system inconsistencies.</td>
<td>LIN</td>
</tr>
<tr>
<td>LinCount</td>
<td>Scans and counts the file system logical inodes (LINs).</td>
<td>LIN</td>
</tr>
<tr>
<td>MediaScan</td>
<td>Scans drives for media-level errors.</td>
<td>Drive + LIN</td>
</tr>
<tr>
<td>MultiScan</td>
<td>Runs Collect and AutoBalance jobs concurrently.</td>
<td>LIN</td>
</tr>
<tr>
<td>PermissionRepair</td>
<td>Correct permissions of files and directories.</td>
<td>Tree</td>
</tr>
<tr>
<td>QuotaScan</td>
<td>Updates quota accounting for domains created on an existing directory path.</td>
<td>Tree</td>
</tr>
<tr>
<td>Job Name</td>
<td>Job Description</td>
<td>Access Method</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>SetProtectPlus</td>
<td>Applies the default file policy. This job is disabled if SmartPools is activated on the cluster.</td>
<td>LIN</td>
</tr>
<tr>
<td>ShadowStoreDelete</td>
<td>Frees space associated with a shadow store.</td>
<td>LIN</td>
</tr>
<tr>
<td>ShadowStoreProtect</td>
<td>Protect shadow stores which are referenced by a LIN with higher requested protection.</td>
<td>LIN</td>
</tr>
<tr>
<td>ShadowStoreRepair</td>
<td>Repair shadow stores.</td>
<td>LIN</td>
</tr>
<tr>
<td>SmartPools</td>
<td>Job that runs and moves data between the tiers of nodes within the same cluster. Also executes the CloudPools functionality if licensed and configured.</td>
<td>LIN</td>
</tr>
<tr>
<td>SmartPoolsTree</td>
<td>Enforce SmartPools file policies on a subtree.</td>
<td>Tree</td>
</tr>
<tr>
<td>SnapRevert</td>
<td>Reverts an entire snapshot back to head.</td>
<td>LIN</td>
</tr>
<tr>
<td>SnapshotDelete</td>
<td>Frees disk space that is associated with deleted snapshots.</td>
<td>LIN</td>
</tr>
<tr>
<td>TreeDelete</td>
<td>Deletes a path in the file system directly from the cluster itself.</td>
<td>Tree</td>
</tr>
<tr>
<td>Undedupe</td>
<td>Removes deduplication of identical blocks in the file system.</td>
<td>Tree</td>
</tr>
<tr>
<td>Upgrade</td>
<td>Upgrades cluster on a later OneFS release.</td>
<td>Tree</td>
</tr>
<tr>
<td>WormQueue</td>
<td>Scan the SmartLock LIN queue</td>
<td>LIN</td>
</tr>
</tbody>
</table>

Figure 3: OneFS Job Engine Job Descriptions
### Job impact policies

An impact policy can consist of one or many impact intervals, which are blocks of time within a given week. Each impact interval can be configured to use a single, pre-defined impact-level which specifies the amount of cluster resources to use for a particular cluster operation. The available impact-levels are:

- **Paused**
- **Low**
- **Medium**
- **High**
This degree of granularity allows impact intervals and levels to be configured per job, in order to ensure smooth cluster operation. And the resulting impact policies dictate when a job runs and the resources that a job can consume. The following default job engine impact policies are provided:

<table>
<thead>
<tr>
<th>Impact policy</th>
<th>Schedule</th>
<th>Impact Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW</td>
<td>Any time of day</td>
<td>Low</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>Any time of day</td>
<td>Medium</td>
</tr>
<tr>
<td>HIGH</td>
<td>Any time of day</td>
<td>High</td>
</tr>
<tr>
<td>OFF_HOURS</td>
<td>Outside of business hours (9AM to 5PM, Mon to Fri), paused during business hours</td>
<td>Low</td>
</tr>
</tbody>
</table>

Figure 5: OneFS Job Engine Default Impact Policies

These default impact policies cannot be modified or deleted.

However, new impact policies can be created, either via the “Add an Impact Policy” WebUI button, or by cloning a default policy and then modifying its settings as appropriate.

A mix of jobs with different impact levels will result in resource sharing. Each job cannot exceed the impact levels set for it, and the aggregate impact level cannot exceed the highest level of the individual jobs.

For example:
Job A (HIGH), job B (LOW).
The impact level of job A is HIGH.
The impact level of job B is LOW.
The total impact level of the two jobs combined is **HIGH**.

Job A (MEDIUM), job B (LOW), job C (MEDIUM).
The impact level of job A is MEDIUM.
The impact level of job B is LOW.
The impact level of job C is MEDIUM.
The total impact level of the three jobs combined is **MEDIUM**.

Job A (LOW), job B (LOW), job C (LOW), job D (LOW).
The impact level of job A is LOW.
The impact level of job B is LOW.
The impact level of job C is LOW.
The impact level of job D is LOW.
The job that was most recently queued/paused, or has the highest job ID value will be paused.
The total impact level of the three running jobs, and one paused job, combined is **LOW**.

The following table shows the default impact policy and relative priority settings for the full range of Job Engine jobs. Typically, the elevated impact jobs will also be run at an increased priority. Isilon recommends keeping the default impact and priority settings, where possible, unless there’s a valid reason to change them.

<table>
<thead>
<tr>
<th>Job Name</th>
<th>Impact Policy</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoBalance</td>
<td>LOW</td>
<td>4</td>
</tr>
<tr>
<td>AutoBalanceLIN</td>
<td>LOW</td>
<td>4</td>
</tr>
<tr>
<td>AVScan</td>
<td>LOW</td>
<td>6</td>
</tr>
<tr>
<td>ChangelistCreate</td>
<td>LOW</td>
<td>5</td>
</tr>
<tr>
<td>Collect</td>
<td>LOW</td>
<td>10</td>
</tr>
<tr>
<td>ComplianceStoreDelete</td>
<td>LOW</td>
<td>6</td>
</tr>
<tr>
<td>Deduplication</td>
<td>LOW</td>
<td>4</td>
</tr>
<tr>
<td>DedupeAssessment</td>
<td>LOW</td>
<td>6</td>
</tr>
<tr>
<td>DomainMark</td>
<td>LOW</td>
<td>5</td>
</tr>
<tr>
<td>DomainTag</td>
<td>LOW</td>
<td>6</td>
</tr>
<tr>
<td>FilePolicy</td>
<td>LOW</td>
<td>6</td>
</tr>
<tr>
<td>FlexProtect</td>
<td>MEDIUM</td>
<td>1</td>
</tr>
</tbody>
</table>
The majority of Job Engine jobs are intended to run with “LOW” impact, and execute in the background. Notable exceptions are the FlexProtect jobs, which by default are set at “medium” impact. This allows FlexProtect to quickly and efficiently re-protect data, without critically impacting other user activities.

Isilon recommends keeping the default priority and impact settings for each job.

**Job priority**

Job Engine jobs are prioritized on a scale of one to ten, with a lower value signifying a higher priority. This is similar in concept to the UNIX scheduling utility, ‘nice’.

Higher priority jobs always cause lower-priority jobs to be paused, and, if a job is paused, it is returned to the back of the Job Engine priority queue. When the job reaches the front of the priority queue again, it resumes from where it left off. If the system schedules two jobs of the same type and priority level to run simultaneously, the job that was queued first is run first.

Priority takes effect when two or more queued jobs belong to the same exclusion set, or when, if exclusion sets are not a factor, four or more jobs are queued. The fourth queued job may be paused, if it has a lower priority than the three other running jobs.

In contrast to priority, job impact policy only comes into play once a job is running, and determines the amount of resources a job can utilize across the cluster. As such, a job’s priority and impact policy are orthogonal to one another.

The FlexProtect(LIN) and IntegrityScan jobs both have the highest job engine priority level of 1, by default. Of these, FlexProtect is the most important, because of its core role in re-protecting data.

All the Job Engine jobs’ priorities are configurable by the cluster administrator. The default priority settings are strongly recommended, particularly for the highest priority jobs mentioned above.
In OneFS 8.2, jobs will no longer be paused if there is only one temporarily unavailable device in each disk pool.

**Multiple job execution**

The OneFS Job Engine allows up to three jobs to be run simultaneously. This concurrent job execution is governed by the following criteria:

- Job Priority
- Exclusion Sets - jobs which cannot run together (ie, FlexProtect and AutoBalance)
- Cluster health - most jobs cannot run when the cluster is in a degraded state.

**Job exclusion sets**

In addition to the per-job impact controls described above, additional impact management is also provided by the notion of job exclusion sets. For multiple concurrent job execution, exclusion sets, or classes of similar jobs, determine which jobs can run simultaneously. A job is not required to be part of any exclusion set, and jobs may also belong to multiple exclusion sets. Currently, there are two exclusion sets that jobs can be part of; restripe and marking.

**Restriping exclusion set**

OneFS protects data by writing file blocks across multiple drives on different nodes. This process is known as ‘restriping’ in the OneFS lexicon. The Job Engine defines a restripe exclusion set that contains these jobs that involve file system management, protection and on-disk layout. The restripe exclusion set contains the following jobs:

- AutoBalance
- AutoBalanceLin
- FlexProtect
The restriping exclusion set is per-phase instead of per job. This helps to more efficiently parallelize restripe jobs when they don't need to lock down resources.

Restriping jobs only block each other when the current phase may perform restriping. This is most evident with MultiScan, whose final phase only sweeps rather than restripes. Similarly, MediaScan, which rarely ever restripes, is usually able to run to completion more without contending with other restriping jobs.

For example, below the two restripe jobs, MediaScan and AutoBalanceLin, are both running their respective first job phases. ShadowStoreProtect, also a restriping job, is in a 'waiting' state, blocked by AutoBalanceLin.

Running and queued jobs:

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>State</th>
<th>Impact</th>
<th>Pri</th>
<th>Phase</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>26850</td>
<td>AutoBalanceLin</td>
<td>Running</td>
<td>Low</td>
<td>4</td>
<td>1/3</td>
<td>20d 18h 19m</td>
</tr>
<tr>
<td>26910</td>
<td>ShadowStoreProtect</td>
<td>Waiting</td>
<td>Low</td>
<td>6</td>
<td>1/1</td>
<td>-</td>
</tr>
<tr>
<td>28133</td>
<td>MediaScan</td>
<td>Running</td>
<td>Low</td>
<td>8</td>
<td>1/8</td>
<td>1d 15h 37m</td>
</tr>
</tbody>
</table>

MediaScan restripes in phases 3 and 5 of the job, and only if there are disk errors (ECCs) which require data reprotection. If MediaScan reaches phase 3 with ECCs, it will pause until AutoBalanceLin is no longer running. If MediaScan's priority were in the range 1-3, it would cause AutoBalanceLin to pause instead.

If two jobs happen to reach their restriping phases simultaneously and the jobs have different priorities, the higher priority job (ie. priority value closer to “1”) will continue to run, and the other will pause. If the two jobs have the same priority, the one already in its restriping phase will continue to run, and the one newly entering its restriping phase will pause.

Marking exclusion set

OneFS marks blocks that are actually in use by the file system. IntegrityScan, for example, traverses the live file system, marking every block of every LIN in the cluster to proactively detect and resolve any issues with the structure of data in a cluster. The jobs that comprise the marking exclusion set are:

- Collect
- IntegrityScan
- MultiScan

Jobs may also belong to both exclusion sets. An example of this is MultiScan, since it includes both AutoBalance and Collect.

Multiple jobs from the same exclusion set will not run at the same time. For example, Collect and IntegrityScan cannot be executed simultaneously, as they are both members of the marking jobs exclusion set. Similarly, MediaScan and SetProtectPlus won’t run concurrently, as they are both part of the restripe exclusion set.
Non-exclusion jobs

The majority of the jobs do not belong to an exclusion set. These are typically the feature support jobs, as described above, and they can coexist and contend with any of the other jobs.

These jobs include:

- AVScan
- ChangelistCreate
- CloudPoolsLin/Treewalk
- ComplianceStoreDelete
- Dedupe
- DedupeDryRun
- DomainMark/Tag
- FilePolicy
- FSAnalyze
- IndexUpdate
- LinCount
- PermissionRepair
- QuotaScan
- ShadowStoreDelete
- SmartPools/Tree
- SnapshotDelete
- SnapRevert
- TreeDelete
- Undedupe
- WormQueue

Exclusion sets do not change the scope of the individual jobs themselves, so any runtime improvements via parallel job execution are the result of job management and impact control. The Job Engine monitors node CPU load and drive I/O activity per worker thread every twenty seconds to ensure that maintenance jobs do not cause cluster performance problems.

If a job affects overall system performance, Job Engine reduces the activity of maintenance jobs and yields resources to clients. Impact policies limit the system resources that a job can consume and when a job can run. You can associate jobs with impact policies, ensuring that certain vital jobs always have access to system resources.
Job engine exclusion sets are pre-defined and cannot be modified or reconfigured.

Job Engine management

Manual job execution

The majority of the Job Engine’s jobs have no default schedule and can be manually started by a cluster administrator. This can be managed either via the CLI or the OneFS WebUI.
Scheduled job execution

Other jobs such as FSAnalyze, MediaScan, ShadowStoreDelete, and SmartPools, are normally started via a schedule. The default job execution schedule is shown in the table below.

<table>
<thead>
<tr>
<th>Job Name</th>
<th>Default Job Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoBalance</td>
<td>Manual</td>
</tr>
<tr>
<td>AutoBalanceLIN</td>
<td>Manual</td>
</tr>
<tr>
<td>AVScan</td>
<td>Manual</td>
</tr>
<tr>
<td>ChangelistCreate</td>
<td>Manual</td>
</tr>
<tr>
<td>CloudPoolsLin/Treewalk</td>
<td>Manual</td>
</tr>
<tr>
<td>Collect</td>
<td>Manual</td>
</tr>
<tr>
<td>ComplianceStoreDelete</td>
<td>The 2nd Saturday of every month at 12am</td>
</tr>
<tr>
<td>Dedupe</td>
<td>Manual</td>
</tr>
<tr>
<td>DedupeAssessment</td>
<td>Manual</td>
</tr>
<tr>
<td>DomainMark/Tag</td>
<td>Manual</td>
</tr>
<tr>
<td>FilePolicy</td>
<td>Every day at 22:00</td>
</tr>
<tr>
<td>FlexProtect</td>
<td>Manual</td>
</tr>
<tr>
<td>FlexProtectLIN</td>
<td>Manual</td>
</tr>
<tr>
<td>FSAnalyze</td>
<td>Every day at 22:00</td>
</tr>
<tr>
<td>IndexUpdate</td>
<td>Manual</td>
</tr>
<tr>
<td>IntegrityScan</td>
<td>Manual</td>
</tr>
<tr>
<td>LinCount</td>
<td>Manual</td>
</tr>
<tr>
<td>MediaScan</td>
<td>The 1st Saturday of every month at 12am</td>
</tr>
<tr>
<td>Job Name</td>
<td>Default Job Schedule</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>MultiScan</td>
<td>Manual</td>
</tr>
<tr>
<td>PermissionRepair</td>
<td>Manual</td>
</tr>
<tr>
<td>QuotaScan</td>
<td>Manual</td>
</tr>
<tr>
<td>SetProtectPlus</td>
<td>Manual</td>
</tr>
<tr>
<td>ShadowStoreDelete</td>
<td>Every Sunday at 12:00am</td>
</tr>
<tr>
<td>SmartPools</td>
<td>Every day at 22:00</td>
</tr>
<tr>
<td>SmartPoolsTree</td>
<td>Manual</td>
</tr>
<tr>
<td>SnapRevert</td>
<td>Manual</td>
</tr>
<tr>
<td>SnapshotDelete</td>
<td>Manual</td>
</tr>
<tr>
<td>TreeDelete</td>
<td>Manual</td>
</tr>
<tr>
<td>WormQueue</td>
<td>Every day at 02:00</td>
</tr>
</tbody>
</table>

Figure 11: OneFS Job Engine Default Job Schedules

The full list of jobs and schedules can be viewed via the CLI command "isi job types list --verbose", or via the WebUI, by navigating to Cluster Management > Job Operations > Job Types.

To create or edit a job’s schedule, click on the “View / Edit” button for the desired job, located in the “Actions” column of the “Job Types” WebUI tab above. From here, check the “Scheduled” radio button, and select between a Daily, Weekly, Monthly, or Yearly schedule, as appropriate. For each of these time period options, it’s possible to schedule the job to run either once or multiple times on each specified day.
The Job Engine schedule for certain feature supporting jobs can be configured directly from the feature’s WebUI area, as well as from the Job Engine WebUI management pages. An example of this is Antivirus and the AVScan job.
Proactive job execution

The Job Engine can also initiate certain jobs on its own. For example, if the SnapshotIQ process detects that a snapshot has been marked for deletion, it will automatically queue a SnapshotDelete job.

Reactive job execution

The Job Engine will also execute jobs in response to certain system event triggers. In the case of a cluster group change, for example the addition or subtraction of a node or drive, OneFS automatically informs the job engine, which responds by starting a FlexProtect job. The coordinator notices that the group change includes a newly-smart-failed device and then initiates a FlexProtect job in response.

Job control

Job administration and execution can be controlled via the WebUI, the command line interface (CLI), or the OneFS RESTful platform API. For each of these control methods, additional administrative security can be configured using roles based access control (RBAC). By restricting access via the ISI_PRIV_JOB_ENGINE privilege, it is possible to allow only a sub-set of cluster administrators to configure, manage and execute job engine functionality, as desirable for the security requirements of a particular environment.
When a job is started by any of the methods described above, in addition to starting and stopping, the job can also be paused.

**Job Operations**

<table>
<thead>
<tr>
<th>Status</th>
<th>ID</th>
<th>Type</th>
<th>Priority</th>
<th>Impact Policy</th>
<th>Elapsed</th>
<th>Phase</th>
<th>Progress</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>28</td>
<td>IntegrityScan</td>
<td>1</td>
<td>Main (Medium)</td>
<td>1.20 w</td>
<td>1 of 2</td>
<td>Processed 6530109501</td>
<td>View/Editor.More</td>
</tr>
<tr>
<td>Waiting</td>
<td>28</td>
<td>MediaScan</td>
<td>8</td>
<td>Main (Medium)</td>
<td>0 s</td>
<td>1 of 7</td>
<td>n/a</td>
<td>View/Editor.More</td>
</tr>
<tr>
<td>Waiting</td>
<td>28</td>
<td>SmartPools</td>
<td>6</td>
<td>Main (Medium)</td>
<td>0 s</td>
<td>1 of 2</td>
<td>n/a</td>
<td>View/Editor.More</td>
</tr>
<tr>
<td>Waiting</td>
<td>28</td>
<td>FSAnalyse</td>
<td>6</td>
<td>Main (Medium)</td>
<td>0 s</td>
<td>1 of 2</td>
<td>n/a</td>
<td>View/Editor.More</td>
</tr>
</tbody>
</table>

**Figure 14: Pausing and Cancelling Jobs from the WebUI**

Once paused, the job can also easily be resumed, and execution will continue from where the job left off when it became paused. This is managed by utilizing the Job Engines’ check-pointing system, described below.

**Job Engine orchestration and execution**

The job engine is based on a delegation hierarchy made up of coordinator, director, manager, and worker processes.

**Figure 15: OneFS Job Engine Distributed Work Allocation Model**
There are other threads which are not illustrated in the graphic, which relate to internal functions, such as communication between the various JE daemons, and collection of statistics. Also, with three jobs running simultaneously, each node would have three manager processes, each with its own number of worker threads.

Once the work is initially allocated, the job engine uses a shared work distribution model in order to execute the work, and each job is identified by a unique Job ID. When a job is launched, whether it’s scheduled, started manually, or responding to a cluster event, the Job Engine spawns a child process from the isi_job_d daemon running on each node. This job engine daemon is also known as the parent process.

**Coordinator process**

The entire job engine’s orchestration is handled by the coordinator, which is a process that runs on one of the nodes in a cluster. Any node can act as the coordinator, and the principle responsibilities include:

- Monitoring work load and the constituent nodes’ status
- Controlling the number of worker threads per-node and cluster-wide
- Managing and enforcing job synchronization and checkpoints

While the actual work item allocation is managed by the individual nodes, the coordinator node takes control, divides up the job, and evenly distributes the resulting tasks across the nodes in the cluster. For example, if the coordinator needs to communicate with a manager process running on node five, it first sends a message to node five’s director, which then passes it on down to the appropriate manager process under its control. The coordinator also periodically sends messages, via the director processes, instructing the managers to increment or decrement the number of worker threads.

The coordinator is also responsible for starting and stopping jobs, and also for processing work results as they are returned during the execution of a job. Should the coordinator process die for any reason, the coordinator responsibility automatically moves to another node.

The coordinator node can be identified via the following CLI command:

```
# isi job status --verbose | grep Coordinator
```

**Director process**

Each node in the cluster has a job engine director process, which runs continuously and independently in the background. The director process is responsible for monitoring, governing and overseeing all job engine activity on a particular node, constantly waiting for instruction from the coordinator to start a new job. The director process serves as a central point of contact for all the manager processes running on a node, and as a liaison with the coordinator process across nodes. These responsibilities include:

- Manager process creation
- Delegating to and requesting work from other peers
- Sending and receiving status messages

**Manager process**

The manager process is responsible for arranging the flow of tasks and task results throughout the duration of a job. The manager processes request and exchange work with each other, and supervise the worker threads assigned to them. At any point in time, each node in a cluster can have up to three manager processes, one for each job currently running. These managers are responsible for overseeing the flow of tasks and task results.

Each manager controls and assigns work items to multiple worker threads working on items for the designated job. Under direction from the coordinator and director, a manager process maintains the appropriate number of active threads for a configured impact level, and for the node’s current activity level. Once a job has completed, the manager processes associated with that job, across all the nodes, are terminated. And new managers are automatically spawned when the next job is moved into execution.

The manager processes on each node regularly send updates to their respective node’s director, which, in turn, informs the coordinator process of the status of the various worker tasks.
Worker threads

Each worker thread is given a task, if available, which it processes item-by-item until the task is complete or the manager un-assigns the task. The status of the nodes' workers can be queried by running the CLI command “isi job statistics view”. In addition to the number of current worker threads per node, a sleep to work (STW) ratio average is also provided, giving an indication of the worker thread activity level on the node.

Towards the end of a job phase, the number of active threads decreases as workers finish up their allotted work and become idle. Nodes which have completed their work items just remain idle, waiting for the last remaining node to finish its work allocation. When all tasks are done, the job phase is considered to be complete and the worker threads are terminated.

Job Engine checkpoints

As jobs are processed, the coordinator consolidates the task status from the constituent nodes and periodically writes the results to checkpoint files. These checkpoint files allow jobs to be paused and resumed, either proactively, or in the event of a cluster outage. For example, if the node on which the Job Engine coordinator was running went offline for any reason, a new coordinator would be automatically started on another node. This new coordinator would read the last consistency checkpoint file, job control and task processing would resume across the cluster from where it left off, and no work would be lost.

Job engine checkpoint files are stored in ‘results’ and ‘tasks’ subdirectories under the path /ifs/.ifsvar/modules/jobengine/cp/\{job_id\}/ for a given job. On large clusters and/or with a job running at high impact, there can be many checkpoint files accessed from all nodes, which may result in contention. In OneFS 8.2, checkpoints are now split into sixteen subdirectories under both tasks and results to alleviate this bottleneck.

Job Engine resource utilization monitoring

The Job Engine resource monitoring and execution framework allows jobs to be throttled based on both CPU and disk I/O metrics. The granularity of the resource utilization monitoring data provides the coordinator process with visibility into exactly what is generating IOPS on any particular drive across the cluster. This level of insight allows the coordinator to make very precise determinations about exactly where and how impact control is best applied. As we will see, the coordinator itself does not communicate directly with the worker threads, but rather with the director process, which in turn instructs a node’s manager process for a particular job to cut back threads.

For example, if the job engine is running a low-impact job and CPU utilization drops below the threshold, the worker thread count is gradually increased up to the maximum defined by the ‘low’ impact policy threshold. If client load on the cluster suddenly spikes for some reason, then the number of worker threads is gracefully decreased. The same principal applies to disk I/O, where the job engine will throttle back in relation to both IOPS as well as the number of I/O operations waiting to be processed in any drive’s queue. Once client load has decreased again, the number of worker threads is correspondingly increased to the maximum ‘low’ impact threshold.

In summary, detailed resource utilization telemetry allows the job engine to automatically tune its resource consumption to the desired impact level and customer workflow activity.

Job Engine throttling and flow control

Certain jobs, if left unchecked, could consume vast quantities of a cluster’s resources, contending with and impacting client I/O. To counteract this, the Job Engine employs a comprehensive work throttling mechanism which is able to limit the rate at which individual jobs can run. Throttling is employed at a per-manager process level, so job impact can be managed both granularly and gracefully.

Every twenty seconds, the coordinator process gathers cluster CPU and individual disk I/O load data from all the nodes across the cluster. The coordinator uses this information, in combination with the job impact configuration, to decide how many threads may run on each cluster node to service each running job. This can be a fractional number, and fractional thread counts are achieved by having a thread sleep for a given percentage of each second.

Using this CPU and disk I/O load data, every sixty seconds the coordinator evaluates how busy the various nodes are and makes a job throttling decision, instructing the various job engine processes as to the action they need to take. This enables throttling to be sensitive to workloads in which CPU and disk I/O load metrics yield different results. Additionally, there are separate load thresholds tailored to the different classes of drives utilized in Isilon clusters, including high speed SAS drives, lower performance SATA disks and flash-based solid state drives (SSDs).
The Job engine allocates a specific number of threads to each node by default, thereby controlling the impact of a workload on the cluster. If little client activity is occurring, more worker threads are spun up to allow more work, up to a predefined worker limit. For example, the worker limit for a low-impact job might allow one or two threads per node to be allocated, a medium-impact job from four to six threads, and a high-impact job a dozen or more. When this worker limit is reached (or before, if client load triggers impact management thresholds first), worker threads are throttled back or terminated.

For example, a node has four active threads, and the coordinator instructs it to cut back to three. The fourth thread is allowed to finish the individual work item it is currently processing, but then quietly exit, even though the task as a whole might not be finished. A restart checkpoint is taken for the exiting worker thread’s remaining work, and this task is returned to a pool of tasks requiring completion. This unassigned task is then allocated to the next worker thread that requests a work assignment, and processing continues from the restart check-point. This same mechanism applies in the event that multiple jobs are running simultaneously on a cluster.

### Job Engine low space behaviour

In situations where the job engine sees the available capacity on one or more disk pools fall below a low space threshold, it engages low space mode. This enables space-saving jobs to run and reclaim space before the job engine or even the cluster become unusable. When the job engine is in low-space mode new jobs will not be started, and any jobs that are not space-saving will be paused. Once free space returns above the low-space threshold, jobs that have been paused for space are resumed.

The space-saving jobs are:

- **AutoBalance**(LIN)
- Collect
- MultiScan
- ShadowStoreDelete
- SnapshotDelete
- TreeDelete

Once the cluster is no longer space constrained, any paused jobs are automatically resumed.

### Job performance

Not all OneFS Job Engine jobs run equally fast. For example, a job which is based on a file system tree walk will run slower on a cluster with a very large number of small files than on a cluster with a low number of large files. Jobs which compare data across nodes, such as Dedupe, will run more slowly where there are many more comparisons to be made. Many factors play into this, and true linear scaling is not always possible. If a job is running slowly the first step is to discover what the specific context of the job is.

There are three main methods for jobs, and their associated processes, to interact with the file system:

- **Via metadata**, using a LIN scan. An example of this is IntegrityScan, when performing an on-line file system verification.
- **Traversing the directory structure directly via a tree walk.** For example, QuotaScan, when performing quota domain accounting.
- **Directly accessing the underlying cylinder groups and disk blocks**, via a linear drive scan. For example, MediaScan, when looking for bad disk sectors.

Each of these approaches has its pros and cons and will suit particular jobs. The specific access method influences the run time of a job. For instance, some jobs are unaffected by cluster size, others slow down or accelerate with the more nodes a cluster has, and some are highly influenced by file counts and directory depths.

For a number of jobs, particularly the LIN-based ones, the job engine will provide an estimated percentage completion of the job during runtime (see figure 20 below).
With LIN scans, even though the metadata is of variable size, the job engine can fairly accurately predict how much effort will be required to scan all LINs. The data, however, can be of widely-variable size, and so estimates of how long it will take to process each task will be a best reasonable guess.

For example, the job engine might know that the highest LIN is 1:0009:0000. Assuming the job will start with a single thread on each of three nodes, the coordinator evenly divides the LINs into nine ranges: 1:0000:0000-1:0000:ffff, 1:0001:0000-1:0001:ffff, etc., through 1:0008:0000-1:0009:0000. These nine tasks would then be divided between the three nodes. However there is no guaranty that each range will take the same time to process. For example, the first range may have fewer actual LINs, as a result of old LINs having been deleted, so complete unexpectedly fast. Perhaps the third range contains a disproportional number of large files and so takes longer to process. And maybe the seventh range has heavy contention with client activity, also resulting in an increased execution time. Despite such variances, the splitting and redistribution of tasks across the node manager processes alleviates this issue, mitigating the need for perfectly-fair divisions at the onset.

Priorities play a large role in job initiation and it is possible for a high priority job to significantly impact the running of other jobs. This is by design, since FlexProtect should be able to run with a greater level of urgency than SmartPools, for example. However, sometimes this can be an inconvenience, which is why the storage administrator has the ability to manually control the impact level and relative priority of jobs.

Certain jobs like FlexProtect have a corresponding job provided with a name suffixed by ‘Lin’, for example FlexProtectLin. This indicates that the job will automatically, where available, use an SSD-based copy of metadata to scan the LIN tree, rather than the drives themselves. Depending on the workflow, this will often significantly improve job runtime performance.

On large clusters with multiple jobs running at high impact, the job coordinator can become bombarded by the volume of task results being sent directly from the worker threads. In OneFS 8.2, this is mitigated by certain jobs performing intermediate merging of results on individual nodes and batching delivery of their results to the coordinator. The jobs that support results merging include:

- AutoBalance(Lin)
- AVScan
- CloudPoolsLin
- CloudPoolsTreewalk
- Collect
- FlexProtect(Lin)
- LinCount
- MultiScan
- PermissionRepair
- QuotaScan
- SnapRevert
- SnapshotDelete
- TreeDelete
- Upgrade

**File system protection and management**

The fundamental responsibility of the jobs within the Restripe exclusion set is to ensure that the data on /ifs is:

- Protected at the desired level
- Balanced across nodes
- Properly accounted for

It does this by running various file system maintenance jobs either manually, via a predefined schedule, or based on a cluster event, like a group change. These jobs include:

**FlexProtect**

FlexProtect is responsible for maintaining the appropriate protection level of data across the cluster. For example, it ensures that a file which is supposed to be protected at +2, is actually protected at that level.

Run automatically after a drive or node removal or failure, FlexProtect locates any unprotected files on the cluster, and repairs them as quickly as possible. The FlexProtect job includes the following distinct phases:
• **Drive Scan.** FlexProtect scans the cluster’s drives, looking for files and inodes in need of repair. When one is found, the job opens the LIN and repairs it and the corresponding data blocks using the restripe process.

• **LIN Verification.** Once the drive scan is complete, the LIN verification phase scans the inode (LIN) tree and verifies, re-verifies and resolves any outstanding re-protection tasks.

• **Device Removal.** In this final phase, FlexProtect removes the successfully repaired drives(s) or node(s) from the cluster.

In addition to FlexProtect, there is also a FlexProtectLin job. FlexProtectLin is run by default when there is a copy of file system metadata available on solid state drive (SSD) storage. FlexProtectLin typically offers significant runtime improvements over its conventional disk-based counterpart.

Unlike previous releases, in OneFS 8.2 FlexProtect does not pause when there is only one temporarily unavailable device in a disk pool, when a device is smartfailed, or for dead devices.

**MultiScan**

The MultiScan job, which combines the functionality of AutoBalance and Collect, is automatically run after a group change which adds a device to the cluster. AutoBalance(Lin) and/or Collect are only run manually if MultiScan has been disabled.

OneFS 8.2 scalability enhancements mean that fewer group change notifications are received. This results in MultiScan being triggered less frequently. To compensate for this in OneFS 8.2, MultiScan is now started when:

- Data is unbalanced within one or more disk pools, which triggers MultiScan to start the AutoBalance phase only.
- When drives have been unavailable for long enough to warrant a Collect job, which triggers MultiScan to start both its AutoBalance and Collect phases.

**AutoBalance**

The goal of the AutoBalance job is to ensure that each node has the same amount of data on it, in order to balance data evenly across the cluster. AutoBalance, along with the Collect job, is run after any cluster group change, unless there are any storage nodes in a “down” state.

Upon visiting each file, AutoBalance performs the following two operations:

- File level rebalancing
- Full array rebalancing

For file level rebalancing, AutoBalance evenly spreads data across the cluster’s nodes in order to achieve balance within a particular file. And with full array rebalancing, AutoBalance moves data between nodes to achieve an overall cluster balance within a 5% delta across nodes.

There is also an AutoBalanceLin job available, which is automatically run in place of AutoBalance when the cluster has a metadata copy available on SSD. AutoBalanceLin provides an expedited job runtime.

**Collect**

The Collect job is responsible for locating unused inodes and data blocks across the file system. Collect runs by default after a cluster group change, in conjunction with AutoBalance, as part of the MultiScan job.

In its first phase, Collect performs a marking job, scanning all the inodes (LINs) and identifying their associated blocks. Collect marks all the blocks which are currently allocated and in use, and any unmarked blocks are identified as candidates to be freed for reuse, so that the disk space they occupy can be reclaimed and re-allocated. All metadata must be read in this phase in order to mark every reference, and must be done completely, to avoid sweeping in-use blocks and introducing allocation corruption.

Collect’s second phase scans all the cluster’s drives and performs the freeing up, or sweeping, of any unmarked blocks so that they can be reused.
MediaScan

MediaScan’s role within the file system protection framework is to periodically check for and resolve drive bit errors across the cluster. This proactive data integrity approach helps guard against a phenomenon known as 'bit rot', and the resulting specter of hardware induced silent data corruption.

MediaScan is run as a low-impact, low-priority background process, based on a predefined schedule (monthly, by default).

First, MediaScan’s search and repair phase checks the disk sectors across all the drives in a cluster and, where necessary, utilizes OneFS’ dynamic sector repair (DSR) process to resolve any ECC sector errors that it encounters. For any ECC errors which can't immediately be repaired, MediaScan will first try to read the disk sector again several times in the hopes that the issue is transient, and the drive can recover. Failing that, MediaScan will attempt to restripe files away from irreparable ECCs. Finally, the MediaScan summary phase generates a report of the ECC errors found and corrected.

IntegrityScan

The IntegrityScan job is responsible for examining the entire live file system for inconsistencies. It does this by systematically reading every block and verifying its associated checksum. Unlike traditional 'fsck' style file system integrity checking tools, IntegrityScan is designed to run while the cluster is fully operational, thereby removing the need for any downtime. In the event that IntegrityScan detects a checksum mismatch, it generates an alert, logs the error to the IDI logs and provides a full report upon job completion.

IntegrityScan is typically run manually if the integrity of the file system is ever in doubt. Although the job itself may take several days or more to complete, the file system is online and completely available during this time. Additionally, like all phases of the OneFS job engine, IntegrityScan can be prioritized, paused or stopped, depending on the impact to cluster operations.

Jobs and OneFS licensing

It was mentioned earlier that the Job Engine includes a number of feature support jobs. These jobs are related to and supporting of the operation of OneFS data and storage management modules, including SmartQuotas, SnapshotIQ, SmartPools, SmartDedupe, etc. Each of these modules requires a cluster-wide license to run. In the event that a feature has not been licensed, attempts to start the associated supporting job will fail with the following warning.

![Figure 16: OneFS Job Engine Unlicensed Job Warning](image-url)
If the SmartPools data tiering product is unlicensed on a cluster, the SetProtectPlus job will run instead, to apply the default file policy. SetProtectPlus is then automatically disabled if SmartPools is activated on the cluster.

Another principle consumer of the SmartPools job and filepool policies is OneFS Small File Storage Efficiency (SFSE). This feature maximizes the space utilization of a cluster by decreasing the amount of physical storage required to house the small files that comprise a typical medical dataset. Efficiency is achieved by scanning the on-disk data for small files, which are protected by full copy mirrors, and packing them in shadow stores. These shadow stores are then parity protected, rather than mirrored, and typically provide storage efficiency of 80% or greater.

If both SmartPools and CloudPools are licensed and both have policies configured, the scheduled SmartPools job will also trigger a CloudPools job when it’s executed. Only the SmartPools job will be visible from the Job Engine WebUI, but the following command can be used to view and control the associated CloudPools jobs:

```
# isi cloud job <action> <subcommand>
```

In addition to the standard CloudPools archive, recall, and restore jobs, there are typically four CloudPools jobs involved with cache management and garbage collection, of which the first three are continuously running:

- Cache-writeback
- Cache-invalidation
- Cloud-garbage-collection
- Local-garbage-collection

Similarly, the SmartDedupe data efficiency product has two jobs associated with it. The first, DedupeAssessment, is an unlicensed job that can be run to determine the space savings available across a dataset. And secondly, the SmartDedupe job, which actually performs the data deduplication, and which requires a valid product license key in order to run.

Licenses for the full range of OneFS products can be purchased through your Isilon account team. The license keys can be easily added via the “Activate License” section of the OneFS WebUI, which is accessed by navigating via Clusters > Licensing.

**Job Engine monitoring and reporting**

The OneFS Job Engine provides detailed monitoring and statistics gathering, with insight into jobs and Job Engine. A variety of Job Engine specific metrics are available via the OneFS CLI, including per job disk usage, etc. For example, worker statistics and job level resource usage can be viewed with CLI command ‘isi job statistics list’. Additionally, the status of the Job Engine workers is available via the OneFS CLI using the ‘isi job statistics view’ command.

**Job events**

Job events, including pause/resume, waiting, phase completion, job success, failure, etc, are reported under the ‘Job Events’ tab of the WebUI. Additional information for each event is available via the “View Details” button for the appropriate job events entry in the WebUI. These are accessed by navigating to Clusters > Job Operations > Job Events.
Job reports

A comprehensive job report is also provided for each phase of a job. This report contains detailed information on runtime, CPU, drive and memory utilization, the number of data and metadata objects scanned, and other work details or errors specific to the job type.

Active job details

While a job is running, an Active Job Details report is also available. This provides contextual information, including elapsed time, current job phase, job progress status, etc.
For inode (LIN) based jobs, progress as an estimated percentage completion is also displayed, based on processed LIN counts.

Figure 20: OneFS Job Engine Active Job Details

Detailed and granular job performance information and statistics are now available in a job’s report. These new statistics include per job phase CPU and memory utilization (including minimum, minimum, and average), and total read and write IOPS and throughput.

Performance resource management

OneFS performance resource management provides statistics for the resources used by jobs - both cluster-wide and per-node. This information is available via the isi statistics workload CLI command. Available in a ‘top’ format, this command displays the top jobs and processes, and periodically updates the information.

For example, the following syntax shows, and indefinitely refreshes, the top five processes on a cluster:

```
# isi statistics workload --limit=5 --format=top
```

```
last update: 2016-09-14T16:45:25 (s)ort: default

<table>
<thead>
<tr>
<th>CPU</th>
<th>Reads</th>
<th>Writes</th>
<th>L2</th>
<th>L3</th>
<th>Node</th>
<th>SystemName</th>
<th>JobType</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4s</td>
<td>9.1k</td>
<td>0.0</td>
<td>3.5k</td>
<td>497.0</td>
<td>2</td>
<td>Job: 237</td>
<td>IntegrityScan[0]</td>
</tr>
<tr>
<td>1.2s</td>
<td>85.7</td>
<td>714.7</td>
<td>4.9k</td>
<td>0.0</td>
<td>1</td>
<td>Job: 238</td>
<td>Dedupe[0]</td>
</tr>
<tr>
<td>1.2s</td>
<td>9.5k</td>
<td>0.0</td>
<td>3.5k</td>
<td>48.5</td>
<td>1</td>
<td>Job: 237</td>
<td>IntegrityScan[0]</td>
</tr>
<tr>
<td>1.2s</td>
<td>7.4k</td>
<td>541.3</td>
<td>4.9k</td>
<td>0.0</td>
<td>3</td>
<td>Job: 238</td>
<td>Dedupe[0]</td>
</tr>
<tr>
<td>1.1s</td>
<td>7.9k</td>
<td>0.0</td>
<td>3.5k</td>
<td>41.6</td>
<td>2</td>
<td>Job: 237</td>
<td>IntegrityScan[0]</td>
</tr>
</tbody>
</table>
```
The resource statistics tracked per job, per job phase, and per node include CPU, reads, writes, and L2 & L3 cache hits. Unlike the output from the ‘top’ command, this makes it easier to diagnose individual job resource issues, etc.

Job Engine best practices

For optimal cluster performance, Isilon recommends observing the following OneFS Job Engine best practices:

- Schedule jobs to run during the cluster’s low usage hours – overnight, weekends, etc.
- Where possible, use the default priority, impact and scheduling settings for each job.
- To complement the four default impact profiles, create additional profiles such as “daytime_medium”, “after_hours_medium”, “weekend_medium”, etc., to fit specific environment needs.
- Ensure the cluster, including any individual node pools, is less than 90% full, so performance is not impacted and that there is always sufficient space to re-protect data in the event of drive failures. Also enable virtual hot spare (VHS) to reserve space in case you need to smartfail devices.
- If SmartPools is licensed, ensure that spillover is enabled (default setting).
- Configure and pay attention to alerts. Set up event notification rules so that you will be notified when the cluster begins to reach capacity thresholds, etc. Make sure to enter a current email address to ensure you receive the notifications.
- It is recommended not to disable the snapshot delete job. In addition to preventing unused disk space from being freed, disabling the snapshot delete job can cause performance degradation.
- Delete snapshots in order, beginning with the oldest. Do not delete snapshots from the middle of a time range. Newer snapshots are mostly pointers to older snapshots, and they look larger than they really are.
- If you need to delete snapshots and there are down or smartfailed devices on the cluster, or the cluster is in an otherwise “degraded protection” state, contact Isilon Technical Support for assistance.
- Only run the FSAnalyze job if you are using InsightIQ. FSAnalyze creates data for InsightIQ’s file system analytics tools, providing details about data properties and space usage within /ifs. Unlike SmartQuotas, FSAnalyze only updates its views when the FSAnalyze job runs. Since FSAnalyze is a fairly low-priority job, it can sometimes be preempted by higher-priority jobs and therefore take a long time to gather all of the data.
- Schedule deduplication jobs to run every 10 days or so, depending on the size of the dataset.
- In a heterogeneous cluster, tune job priorities and impact policies to the level of the lowest performance tier.
- For OneFS versions prior to OneFS 7.1, pause any active or scheduled SnapshotDelete jobs before adding new nodes to the cluster to allow MultiScan (default lower priority) to complete without interruption.
- Before running a major (non-rolling) OneFS upgrade, allow active jobs to complete, where possible, and cancel out any outstanding running jobs.
- OneFS 8.1 and earlier, before running TreeDelete, ensure there are no quotas policies set on any directories under the root level of the data for deletion. In OneFS 8.2, TreeDelete can delete a directory to which a quota has been applied using the ‘--delete-quota’ flag. For example: #isi job start TreeDelete --paths=/ifs/quota -delete quota
- If FlexProtect is running, allow it to finish completely before powering down any node(s), or the entire cluster. While shutting down the cluster during restripe won’t hurt anything directly, it does increases the risk of a second device failure before FlexProtect finishes re-protecting data.

Job Engine considerations

For optimal cluster performance, bear in mind the following OneFS Job Engine considerations:

- When reconfiguring the default priority, schedule and impact profile for a job, consider the following questions:
• What resources am I impacting?

• What would I be gaining or losing if I re-prioritized this job?

• What are my impact options and their respective benefits and drawbacks?

• How long will the job run and what other jobs will it contend with?

• SyncIQ, the OneFS replication product, does not use job engine. However, it has both influenced, and been strongly influenced by, the job engine’s design. SyncIQ also terms its operations “jobs”, and its processes and terminology bear some similarity to job engine. The job engine impact management framework is aware of the resources consumed by SyncIQ, in addition to client load, and will throttle jobs accordingly.

• A job with a name suffixed by ‘Lin’, for example FlexProtectLin, indicates that this job will scan an SSD-based copy of the LIN tree metadata, rather than access the hard drives themselves. This can significantly improve job performance, depending on the specific workflow.

• When more than three jobs with the same priority level and no exclusion set restrictions are scheduled to run simultaneously, the three jobs with the lowest job ID value will run, and the remainder will be paused.

• There is only one ‘mark cookie’ available for jobs within the marking exclusion set. So, if marking job A is interrupted by another marking job B, job A will automatically be cancelled when it resumes after the completion of B.

• For mixed node (heterogeneous) clusters that do not have a license for the OneFS SmartPools data tiering module, the SetProtectPlus job will run instead, and apply the default file policy. SetProtectPlus will be automatically disabled in the event that a valid SmartPools license key is added to the cluster.

• By default, FlexProtect is the only job allowed to run if a cluster is in degraded mode. Other jobs will automatically be paused and will not resume until FlexProtect has completed and the cluster is healthy again.

• Restriping jobs only block each other when the current phase may perform restriping. This is most evident with MultiScan, whose final phase only sweeps rather than restripes. Similarly, MediaScan, which rarely ever restripes, is usually able to run to completion more without contending with other restriping jobs.

• MediaScan restripes in phases 3 and 5 of the job, and only if there are disk errors (ECCs) which require data reprotection. If MediaScan reaches phase 3 with ECCs, it will pause until AutoBalanceLin is no longer running. If MediaScan’s priority were in the range 1-3, it would cause AutoBalanceLin to pause instead.

• If two jobs happen to reach their restriping phases simultaneously and the jobs have different priorities, the higher priority job (i.e. priority value closer to “1”) will continue to run, and the other will pause. If the two jobs have the same priority, the one already in its restriping phase will continue to run, and the one newly entering its restriping phase will pause.

• During MediaScan’s verify and repair phases, attempts to re-read bad sectors can occasionally cause drives to stall briefly while trying to correct the error. This is typically only a very brief and limited interruption.

• In OneFS 8.2, when a cluster is low on free space, new jobs will not be started and any jobs that are not space-saving will be paused. Once the cluster is no longer space constrained, any paused jobs are automatically resumed.

• In OneFS 8.2, MultiScan is started when data is unbalanced within one or more disk pools, or when drives have been unavailable for long enough to warrant a Collect job run.

• When enabling and using the FilePolicy job, the recommendation is to continue running the SmartPools job at a reduced frequency.

• In OneFS 8.2, the FilePolicy and FSAnalyze jobs automatically share the same snapshots and index, created and managed by the IndexUpdate job.
• When a cluster running FSAnalyze is upgraded to OneFS 8.2, the legacy FSAnalyze index and snapshots are removed and replaced by new snapshots the first time that IndexUpdate is run. The new index stores considerably more file and snapshot attributes than the old FSA index. Until the IndexUpdate job effects this change, FSA keeps running on the old index and snapshots.

**Conclusion**

To date, traditional clustered storage implementations have typically been expensive, limited in scale and administratively complex. The OneFS Job Engine powers the industry’s leading Scale-Out NAS architecture, enabling Isilon to deliver on the promise of simple data efficiency at scale, without sacrificing simplicity, performance or data availability and protection.

With its parallel job execution framework and refined impact management controls, the Job Engine is easy to configure and manage, allowing Isilon clusters to seamlessly expand from terabytes to multiple petabytes. Linear scaling of performance and capacity, plus the ability to consolidate applications on a single storage pool, while retaining older capacity in the same system, means strong investment protection. Integration with OneFS storage management functions eliminates data risks and gives the user control over what system resources are allocated to cluster management.

**Version History**

*Author: Nick Trimbee*

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>November 2013</td>
<td>First version for OneFS 7.1</td>
</tr>
<tr>
<td>2.0</td>
<td>June 2014</td>
<td>Updated for OneFS 7.1.1</td>
</tr>
<tr>
<td>3.0</td>
<td>November 2014</td>
<td>Updated for OneFS 7.2</td>
</tr>
<tr>
<td>4.0</td>
<td>June 2015</td>
<td>Updated for OneFS 7.2.1</td>
</tr>
<tr>
<td>5.0</td>
<td>November 2015</td>
<td>Updated for OneFS 8.0</td>
</tr>
<tr>
<td>6.0</td>
<td>September 2016</td>
<td>Updated for OneFS 8.0.1</td>
</tr>
<tr>
<td>7.0</td>
<td>April 2017</td>
<td>Updated for OneFS 8.1</td>
</tr>
<tr>
<td>8.0</td>
<td>November 2017</td>
<td>Updated for OneFS 8.1.1</td>
</tr>
<tr>
<td>9.0</td>
<td>February 2019</td>
<td>Updated for OneFS 8.1.3</td>
</tr>
<tr>
<td>10.0</td>
<td>April 2019</td>
<td>Updated for OneFS 8.2</td>
</tr>
<tr>
<td>11.0</td>
<td>August 2019</td>
<td>Updated for OneFS 8.2.1</td>
</tr>
</tbody>
</table>
TAKE THE NEXT STEP

Contact your Dell EMC sales representative or authorized reseller to learn more about how Isilon scale-out NAS storage solutions can benefit your organization.

Shop Dell EMC Isilon to compare features and get more information.

© 2019 Dell Inc. or its subsidiaries. All Rights Reserved. Dell, EMC and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be trademarks of their respective owners. Reference Number: H12570.8