# What's new in the SPECstorage Solution 2020 benchmark

By
Don Capps
Date: 9/08/2020
Document revision 1.11

## Table of Contents

The SPECstorage Solution 2020 benchmark ..... 2
Major release ..... 2
New workloads ..... 2
AI_Image workload in SPECstorage Solution 2020: ..... 2
Genomics workload in SPECstorage Solution 2020: ..... 3
New features in SPECstorage Solution 2020: ..... 4
Workloads from SPECsfs2014_SP2 that were removed from SPECstorage Solution 2020. ..... 8
The YAML strategy: ..... 8
Changes to the configuration files ..... 9
The sfs_rc file changes: ..... 9
Custom workload definition changes: ..... 9
The YAML configuration file: (Example with only one workload shown) ..... 10
New build requirements for building SPECstorage Solution 2020: ..... 15
Custom heterogeneous workloads with heterogeneous client types ..... 15
Custom homogeneous workloads with heterogeneous client types ..... 16

# The SPECstorage Solution 2020 benchmark

## Major release

The SPECstorage Solution 2020 benchmark is a new major release from SPEC. The term "Major release" means that this version of the benchmark is *not* performance neutral with respect to previous versions. No comparisons of results between this version of the benchmark and other versions will be permitted. Some workloads may have similar names, but the nature of the benchmark has changed in ways that alter the workload behaviors and resource consumption.

## New workloads

### AI_Image workload in SPECstorage Solution 2020:

The SPECstorage Solution 2020 release contains a new workload. AI_IMAGE. (AI image processing)

This workload is representative of AI Tensorflow image processing environments and is expected to be popular as this market continues to expand.

The traces used for the basis were collected from Nvidia DGX based systems running COCO, Resnet50, and CityScape processing.

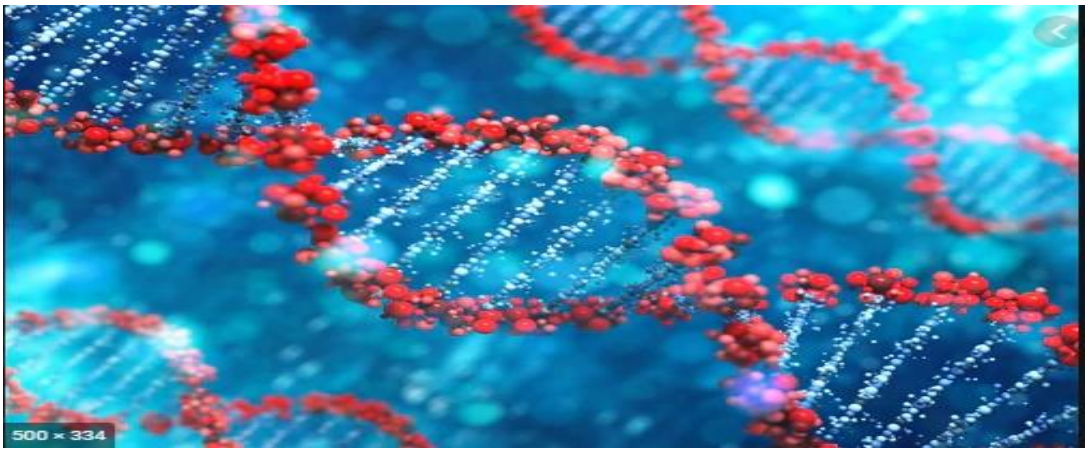
https://www.cityscapes-dataset.com/benchmarks/

http://cocodataset.org/#home

https://github.com/KaimingHe/deep-residual-networks

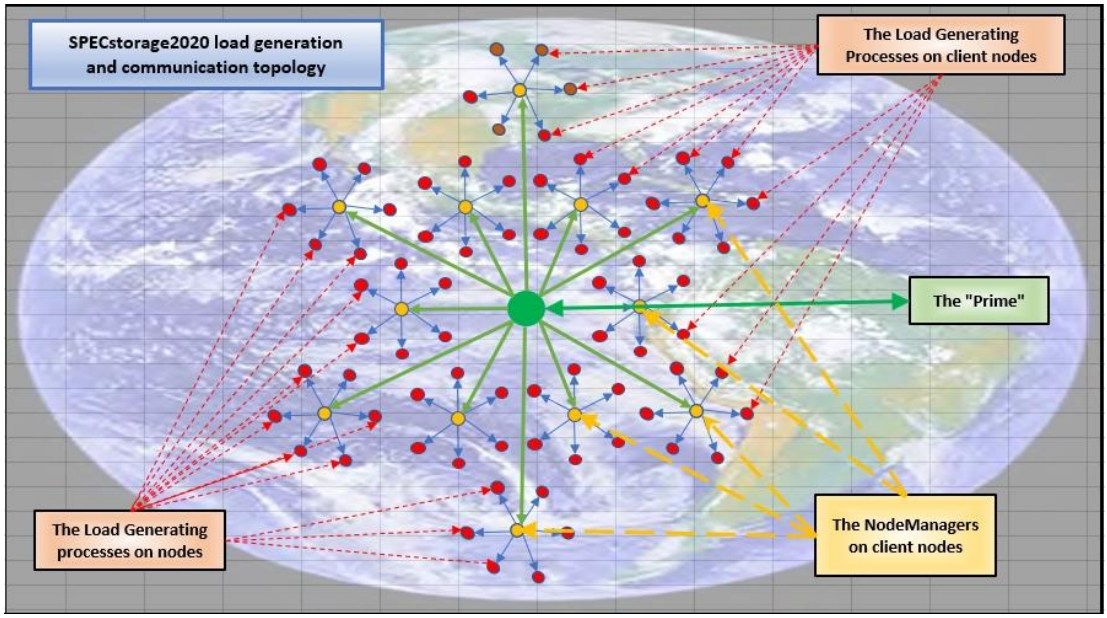
## Genomics workload in SPECstorage Solution 2020:

This workload is representative of the Genomics workflow processing. The traces used to construct this workload came from commercial and research facilities that perform genetic analysis. The I/O behavior was captured and is synthesized by the benchmark. The data has been sanitized so that it does not contain any of the original genome data.

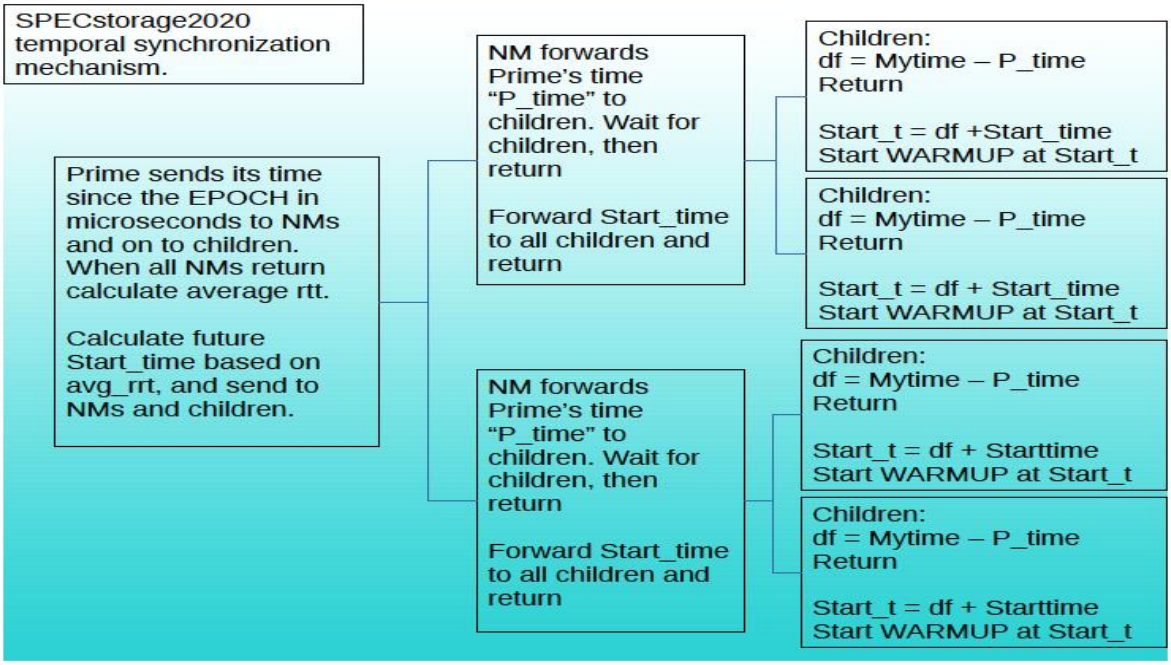


## New features in SPECstorage Solution 2020:

- SPECstorage Solution 2020 contains workloads that are similar as those found in SPECsfs2014_SP2. There are two new workloads AI_IMAGE, Genomics. Continuations of previous workloads include SWBUILD (Software Build), EDA_BLENDED (Electronic Design Automation), and VDA (Video Data Acquisition)
  The VDI and DATABASE workloads are being deprecated in SPECstorage Solution 2020, as the technology of these has changed and is continuing to evolve. These workloads could be re-introduced later, after the collection of new traces could be completed.
- Scaling in SPECsfs2014_SP2 was up to 60,000 load generating processes globally. Scaling in SPECstorage Storage 2020 is expected to increase this limit to somewhere around 4 Million load generating processes globally, where the load generators may be geographically distributed around the planet.




There is a sophisticated synchronization mechanism that keeps all of the geographically distributed load generating processes in sync, at a sub milli-second resolution. ( as depicted below)

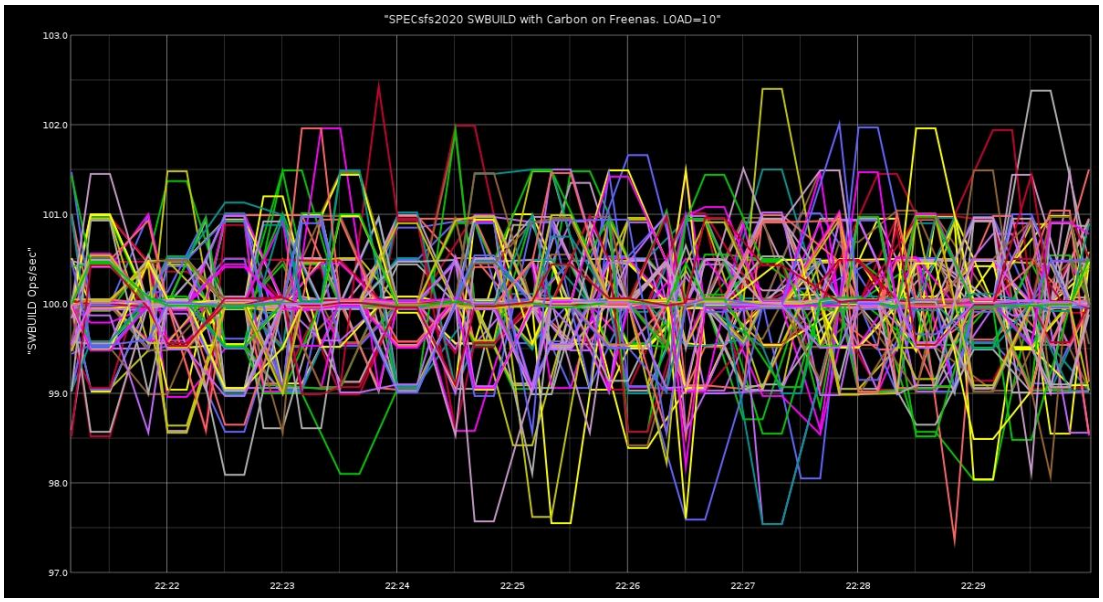



In the diagram above, the Prime sends its current time in microseconds since the EPOCH to the NodeManagers, and they forward this on to the child processes. The child processes calculate the difference between the Prime's timer, and their time since the EPOCH. Once all of the children have finished this calculation, they return to the NodeManager, and the NodeManagers return to the Prime. The prime keeps track of how long it took for all of the NodeManagers to complete this operation. Using this knowledge the Prime can predict how long in the future it would take to send a message to all of the NodeManagers and their children. The prime calculates this future time in microseconds since the EPOCH and sends this to the NodeManagers, who send the on to their children. The children now calculate the time relative to their time since the EPOCH to obtain the time when they should begin work. This is achieved by using the time difference they stored in the first exchange from the prime. The children wait until this relative time, and then begin work. Since all children have a synchronized relative time to start, they will being work within a sub-millisecond time synchronized barrier.
The WARMUP and RUN phases perform their duties for the prescribed duration of time. At the end of their duration they move on to the next phase. This 'end of phase' time is also synchronized using the same temporal synchronization mechanism.

- Because of the new internal IPC mechanisms used in SPECstorage Solution 2020, there is a significant reduction in the number of TCP ports used, as well as the number of DNS lookups. This improves scalability, reliability, startup time, termination and cleanup time, and robustness.
- SPECstorage Solution 2020 implements a new logging mechanism. This improves the content, and consistency of all logging throughout the benchmark. Consistent, and higher resolution timestamps are also included in the new logging functionality.

- Csv log files are a new addition in SPECstorage Solution 2020. This makes it much easier for external applications to interface and parse the information contained in the logs.
- The source control and build processes within SPEC for SPECstorage Solution 2020 are maintained within a Git repository. This improves access and collaboration of the SPEC Storage member companies.
- Added neg_stat_op(). Provides a stat() for non-existent files to the workload definitions. This measures the time it takes to do a stat() on a file that doesn't exist.
- Added posix_fadvise(). Provides the Posix_fadvise() to the workload definitions.
- Added posix_madvise(). Provides the Posix_madvise() to the workload definitions.
- Added support for user defined filename length to the workload definitions.
- Added more information in logs to help with any debug, or fault isolation.
- Added MD5 hashes to each workload to prevent tampering with official definitions for publications.
- Added CRC32 checksums to intra-node communication to protect the integrity of the messages from dirty/lossy networks, or tampering
- Enhanced the reliability and speed of Control-C for terminating the benchmark.
- Enhanced detection of client node oversubscription and logging of warnings.
- All of the workload definitions and attributes are now in a single YAML configuration file. This provides a single file with all of the workload information in a human readable format.

- A new statistical collection mechanism was enabled that facilitates the user extracting runtime counters and connecting them to a database for graphical representation. E.g. Graphite, Carbon, and Grafana.




This provides the user more insights as to the behavior of the system being tested as well as per operation type details. Configuration and usage of this new statistical functionality is covered in more detail in section 9.3 of the SPECstorage Solution 2020 User's guide.

- In the past every process within the benchmark performed a validation of all of the possible OP types that would be used by the assigned workload. This was inefficient. In SPECstorage Solution 2020 only one of the processes for any given workload will perform OP validation. This optimization takes into consideration the locality within a node, as well as the workload type and directory being used for testing and eliminates the redundant testing by processes on the same node, with the same workload and workdir, saving large quantities of lab resources and time.
- The SPECstorage Solution 2020 release introduces support for SKIP_INIT. This is a feature where one can re-use an initialized data set without even incurring the overhead of the stat() operations to check for the existence of files. This feature can save considerable quantities of lab time. This feature is not valid for publications however it is very convenient for regression testing in labs that are running the benchmark on a regular basis.
- Added support for testing the AFS filesystem, which uses user space RPCs to communicate with AFS storage, bypassing the client operating system's call stack for I/O.
- Added support for user definable shared libraries to plug-in to the benchmark at the system call boundary. This provides the capability of testing a non-POSIX storage system. Currently there is a plugin shared library for accessing an AFS storage service. Other non-POSIX storage systems may be added in the future to support object store, S3, Ceph, and more.
- Added support for file size distribution tables in the workload definitions.

- Added support for user definable depth of directory structure as well as adding chaff (unused files) at every level of the directory structure. This is convenient when testing deep and wide directory structures.
- Added stack back-trace of any fatal errors. E.g. SIGSEGV or SIGILL. Works on most platforms. This helps in providing details for any abnormal terminations of the benchmark and reduces time to solution for customer support.
- Improved precision of pattern layouts which provides support for dedup and compression.
- Added a thread based keep-alive mechanism that keeps track of the distributed components of the benchmark and informs the user if any of the components have failed and conducts a graceful shutdown of the benchmark so as to release all resources and log appropriate errors. This eliminates the silent hangs and failures that could occur in SPECsfs2014_SP2. This thread mechanism is also used for IPC between the load generating processes and the NodeManager, as well as between the NodeManager and the Prime.
- Added support for Doxygen. This enables creation of documentation from the source code in multiple formats. http, rich text, and pdf.

## Workloads from SPECsfs2014_SP2 that were removed from SPECstorage Solution 2020.

The VDI, and DATABASE workloads were deprecated. These workloads have become obsolete due to changing, and evolving implementations of these application stacks. For example, VDI implementations are evolving from non-clones to clones. Also, new virtualization mechanisms are appearing in the market. E.g. Dockers, Kubernetes, Containers, KVM, Azure. There are evolving technologies from Google, Amazon, Microsoft, RedHat, and Oracle.
Databases are also evolving, and their usage and behaviors are also undergoing change.
Thus, new traces from the commercial and research areas are needed so that these could form the basis of the next generation of VDI and DATABASE workloads.

## The YAML strategy:

In the previous versions of SPECsfs2014, the workload information was distributed across multiple files. The sfs_rc file contained some of the information, the benchmarks.xml had more, and if one was using a custom workload then the workloads.txt file also contained information.
This created confusion and was one of the top suggested areas for improvement. Thus, a new method has been introduced in SPECstorage Solution 2020 release. The workload information is now contained in a single YAML configuration file. The user/site specific information continues to be in the sfs_rc file and is generally the only file that the end user will ever need to modify.
**With this change any/all previous environments that were making modifications to the benchmarks.xml or the workloads.txt files will now need to be updated to use the storage2020.yml file.**

## Changes to the configuration files

### The sfs_rc file changes:

The sfs_rc file is now focused on the user and site specific variables that the user needs to modify as part of the setup for their environment.

Examples:

| | | |
|---|---|---|
| LOAD= | … | The initial LOAD value for the run. |
| INCR_LOAD= | … | The amount to increment LOAD for each load point in the run. |
| NUM_RUNS= | … | The number of load points in this run. |
| CLIENT_MOUNTPOINTS= | … | The client hostname:/mountpoints list, or a file with same info. |
| PASSWORD= | … | The user account's password for remote command execution. (Not needed for Unix or Windows systems that use Openssh and ssh-keys) |
| PRIME_MON_SCRIPT= | … | The path to a user defined monitoring script. |
| PRIME_MON_ARGS= | … | Variables to pass to the user defined script. |
| BENCHMARK= | … | Name of the benchmark to run. |
| EXEC_PATH= | … | Path to the executable on the Prime client. /usr/local/netmist |
| WINDOWS_EXEC_PATH | … | Path to the Win32 executable on remote clients. |
| UNIX_EXEC_PATH | … | Path to the Unix executable on remote clients. |
| USER= | … | User's name to use for this run. |
| IPV6_ENABLE= | … | Enable IPv6 if needed. |
| NETMIST_LOGS= | … | Path to directory for client logs. Local to each node. |
| UNIX_CLIENT_MOUNTPOINTS | … | client hostname:/mountpoints list for Unix clients in heterogeneous workloads with heterogeneous clients |
| WINDOWS_CLIENT_MOUNTPOINTS | | client hostname:\\server\\share for Windows in heterogeneous workloads with heterogeneous clients. |
| WINDOWS_CLIENT_LIST | … | Client hostnames for Windows systems in homogeneous workloads with heterogeneous clients. |
| UNIX_CLIENT_LIST | … | Client hostnames for Unix systems in homogeneous workloads with heterogeneous clients. |

The benchmarks.xml changes:

The benchmarks.xml file is no longer used in the benchmark. Any/all customizations made to this file will need to be ported to using the YAML configuration file. default = "storage2020.yml"

### Custom workload definition changes:

Custom workload definitions were done in the past by exporting, modifying, and importing user defined workloads. These import workload files are no longer supported. Custom modifications to the workload definitions are accomplished via the YAML configuration file.

## The YAML configuration file: (Example with only one workload shown)

```
Globals:
     - Skip_init: 0
     - Do_op_validate: 1
     - Run_time: 300

Benchmarks:
     - Benchmark_name:
         - SWBUILD:
             # Global options for the Workload.
             Proc_oprate_threshold: 75
             Global_oprate_threshold: 95
             Proc_latency_threshold: 0
             Global_latency_threshold: 0
             Workload_variance: 0
             Dedicated_subdirectory: 0
             Business_metric: "BUILDS"
             Warmup_time: 300
         - Components:
             - SWBUILD:
                 # Component options
                 Op_rate: 100
                 Platform_type: "Unix"
                 Instances: 5
                 File_size: "16k"
                 Dir_count: 50
                 Files_per_dir: 100
                 Extra_dir_levels: 0
                 Chaff_count: 0
                 Shared_buckets: 1
                 System_call_distribution:
                     - Names:
                         # Op type name strings
                         Read_string: "Read"
                         Read_file_string: "Read_file"
                         Mmap_read_string: "Mmap_read"
                         Random_read_string: "Random_read"
                         Write_string: "Write"
                         Write_file_string: "Write_file"
                         Mmap_write_string: "Mmap_write"
                         Random_write_string: "Random_write"
                         Read_modify_write_string: "Read_modify_write"
                         Mkdir_string: "Mkdir"
                         Rmdir_string: "Rmdir"
                         Unlink_string: "Unlink"
                         Unlink2_string: "Unlink2"
                         Create_string: "Create"
                         Append_string: "Append"
                         Lock_string: "Lock"
                         Access_string: "Access"
                         Stat_string: "Stat"
```

```
            Neg_stat_string: "Neg_stat"
            Chmod_string: "Chmod"
            Readdir_string: "Readdir"
            Copyfile_string: "Copyfile"
            Rename_string: "Rename"
            Statfs_string: "Statfs"
            Pathconf_string: "Pathconf"
            Trunc_string: "Trunc"
            Custom1_string: "Custom1"
            Custom2_string: "Custom2"
        - Percentages:
            #
            # Percentages of each op type in the mix
            #
            Read_percent: 0
            Read_file_percent: 6
            Mmap_read_percent: 0
            Random_read_percent: 0
            Write_percent: 0
            Write_file_percent: 7
            Mmap_write_percent: 0
            Random_write_percent: 0
            Read_modify_write_percent: 0
            Mkdir_percent: 1
            Rmdir_percent: 0
            Unlink_percent: 2
            Unlink2_percent: 0
            Create_percent: 1
            Append_percent: 0
            Lock_percent: 0
            Access_percent: 6
            Stat_percent: 70
            Neg_stat_percent: 0
            Chmod_percent: 5
            Readdir_percent: 2
            Copyfile_percent: 0
            Rename_percent: 0
            Statfs_percent: 0
            Pathconf_percent: 0
            Trunc_percent: 0
            Custom1_percent: 0
            Custom2_percent: 0
    Transfer_sizes:
        - Read_xfer_elements:
            # Read transfer range slots. Min, Max, Percent
            Read_elem_0_xfer_min_size: 1
            Read_elem_0_xfer_max_size: 511
            Read_elem_0_xfer_percent: 1
            Read_elem_1_xfer_min_size: 512
            Read_elem_1_xfer_max_size: 1023
            Read_elem_1_xfer_percent: 5
            Read_elem_2_xfer_min_size: 1024
            Read_elem_2_xfer_max_size: 2047
```

```
    Read_elem_2_xfer_percent: 7
    Read_elem_3_xfer_min_size: 2048
    Read_elem_3_xfer_max_size: 4095
    Read_elem_3_xfer_percent: 7
    Read_elem_4_xfer_min_size: 4096
    Read_elem_4_xfer_max_size: 4096
    Read_elem_4_xfer_percent: 0
    Read_elem_5_xfer_min_size: 4096
    Read_elem_5_xfer_max_size: 8191
    Read_elem_5_xfer_percent: 45
    Read_elem_6_xfer_min_size: 8192
    Read_elem_6_xfer_max_size: 8192
    Read_elem_6_xfer_percent: 0
    Read_elem_7_xfer_min_size: 8192
    Read_elem_7_xfer_max_size: 16383
    Read_elem_7_xfer_percent: 13
    Read_elem_8_xfer_min_size: 16384
    Read_elem_8_xfer_max_size: 16384
    Read_elem_8_xfer_percent: 0
    Read_elem_9_xfer_min_size: 16384
    Read_elem_9_xfer_max_size: 32767
    Read_elem_9_xfer_percent: 3
    Read_elem_10_xfer_min_size: 32768
    Read_elem_10_xfer_max_size: 65535
    Read_elem_10_xfer_percent: 2
    Read_elem_11_xfer_min_size: 65536
    Read_elem_11_xfer_max_size: 65536
    Read_elem_11_xfer_percent: 0
    Read_elem_12_xfer_min_size: 98304
    Read_elem_12_xfer_max_size: 98304
    Read_elem_12_xfer_percent: 0
    Read_elem_13_xfer_min_size: 65536
    Read_elem_13_xfer_max_size: 131072
    Read_elem_13_xfer_percent: 17
    Read_elem_14_xfer_min_size: 262144
    Read_elem_14_xfer_max_size: 262144
    Read_elem_14_xfer_percent: 0
    Read_elem_15_xfer_min_size: 524288
    Read_elem_15_xfer_max_size: 524288
    Read_elem_15_xfer_percent: 0
- Write_xfer_elements:
    # Write transfer range slots. Min, Max, Percent
    Write_elem_0_xfer_min_size: 1
    Write_elem_0_xfer_max_size: 511
    Write_elem_0_xfer_percent: 5
    Write_elem_1_xfer_min_size: 512
    Write_elem_1_xfer_max_size: 1023
    Write_elem_1_xfer_percent: 3
    Write_elem_2_xfer_min_size: 1024
    Write_elem_2_xfer_max_size: 2047
    Write_elem_2_xfer_percent: 10
    Write_elem_3_xfer_min_size: 2048
    Write_elem_3_xfer_max_size: 4095
```

```
    Write_elem_3_xfer_percent: 15
    Write_elem_4_xfer_min_size: 4096
    Write_elem_4_xfer_max_size: 4096
    Write_elem_4_xfer_percent: 0
    Write_elem_5_xfer_min_size: 4096
    Write_elem_5_xfer_max_size: 8191
    Write_elem_5_xfer_percent: 14
    Write_elem_6_xfer_min_size: 8192
    Write_elem_6_xfer_max_size: 8192
    Write_elem_6_xfer_percent: 0
    Write_elem_7_xfer_min_size: 8192
    Write_elem_7_xfer_max_size: 16383
    Write_elem_7_xfer_percent: 7
    Write_elem_8_xfer_min_size: 16384
    Write_elem_8_xfer_max_size: 16384
    Write_elem_8_xfer_percent: 0
    Write_elem_9_xfer_min_size: 16384
    Write_elem_9_xfer_max_size: 32767
    Write_elem_9_xfer_percent: 6
    Write_elem_10_xfer_min_size: 32768
    Write_elem_10_xfer_max_size: 65535
    Write_elem_10_xfer_percent: 4
    Write_elem_11_xfer_min_size: 65536
    Write_elem_11_xfer_max_size: 131072
    Write_elem_11_xfer_percent: 36
    Write_elem_12_xfer_min_size: 98304
    Write_elem_12_xfer_max_size: 98304
    Write_elem_12_xfer_percent: 0
    Write_elem_13_xfer_min_size: 131072
    Write_elem_13_xfer_max_size: 131072
    Write_elem_13_xfer_percent: 0
    Write_elem_14_xfer_min_size: 262144
    Write_elem_14_xfer_max_size: 262144
    Write_elem_14_xfer_percent: 0
    Write_elem_15_xfer_min_size: 524288
    Write_elem_15_xfer_max_size: 524288
    Write_elem_15_xfer_percent: 0
- File_size_elements:
    # File size range slots. Min, Max, Percent
    File_size_elem_0_min_size: 1
    File_size_elem_0_max_size: 511
    File_size_elem_0_percent: 0
    File_size_elem_1_min_size: 512
    File_size_elem_1_max_size: 1023
    File_size_elem_1_percent: 0
    File_size_elem_2_min_size: 1024
    File_size_elem_2_max_size: 2047
    File_size_elem_2_percent: 0
    File_size_elem_3_min_size: 2048
    File_size_elem_3_max_size: 4095
    File_size_elem_3_percent: 0
    File_size_elem_4_min_size: 4096
    File_size_elem_4_max_size: 4096
```

```
            File_size_elem_4_percent: 0
            File_size_elem_5_min_size: 4096
            File_size_elem_5_max_size: 8191
            File_size_elem_5_percent: 0
            File_size_elem_6_min_size: 8192
            File_size_elem_6_max_size: 8192
            File_size_elem_6_percent: 0
            File_size_elem_7_min_size: 8192
            File_size_elem_7_max_size: 16383
            File_size_elem_7_percent: 0
            File_size_elem_8_min_size: 16384
            File_size_elem_8_max_size: 16384
            File_size_elem_8_percent: 100
            File_size_elem_9_min_size: 16384
            File_size_elem_9_max_size: 32767
            File_size_elem_9_percent: 0
            File_size_elem_10_min_size: 32768
            File_size_elem_10_max_size: 65535
            File_size_elem_10_percent: 0
            File_size_elem_11_min_size: 65536
            File_size_elem_11_max_size: 65536
            File_size_elem_11_percent: 0
            File_size_elem_12_min_size: 98304
            File_size_elem_12_max_size: 98304
            File_size_elem_12_percent: 0
            File_size_elem_13_min_size: 65536
            File_size_elem_13_max_size: 131072
            File_size_elem_13_percent: 0
            File_size_elem_14_min_size: 262144
            File_size_elem_14_max_size: 262144
            File_size_elem_14_percent: 0
            File_size_elem_15_min_size: 524288
            File_size_elem_15_max_size: 524288
            File_size_elem_15_percent: 0
    Misc:
        # Sizing info for file names
        Min_pre_name_length: 3
        Max_pre_name_length: 8
        Min_post_name_length: 0
        Max_post_name_length: 5
        Percent_write_commit: 33
        Percent_direct: 0
        Percent_fadvise_seq: 0
        Percent_fadvise_rand: 0
        Percent_fadvise_dont_need: 0
        Percent_madvise_seq: 0
        Percent_madvise_rand: 0
        Percent_madvise_dont_need: 0
        Align: 0
        Percent_osync: 0
        Percent_geometric: 10
        Percent_compress: 80
        Percent_dedup: 0
```

```
Percent_dedup_within: 0
Percent_dedup_across: 0
Dedupe_group_count: 1
Percent_per_spot: 0
Min_acc_per_spot: 0
Acc_mult_spot: 5
Percent_affinity: 0
Spot_shape: 0
Dedup_Granule_size: 4096
Dedup_gran_rep_limit: 100
Unlink2_no_recreate: 0
Use_file_size_dist: 0
Comp_Granule_size: 8192
Background: 0
Sharemode: 0
Uniform_file_size_dist: 0
Rand_dist_behavior: 0
Cipher_behavior: 0
Notification_percent: 0
LRU: 0
Pattern_version: 2
Init_rate_enable: 1
Init_rate_speed: 0.0
Init_read_flag: 1
Release_version: 3
FS_type: "POSIX"
```

## New build requirements for building SPECstorage Solution 2020:

The SPECstorage Solution 2020 kit includes the libyaml source kit in the redistributable_sources sub-directory. The source kit for libyaml builds on Linux, FreeBSD, MacOS, Solaris, and Windows without modification. For other platforms one may need to download an appropriate "port" from the vendor's respective repositories.

See section 10 of the SPECstorage™ Solution 2020 User's guide for further details.

## Custom heterogeneous workloads with heterogeneous client types

In SPECstorage Solution 2020, custom workloads are supported. One can customize or create new workloads and then use these to test solutions. While such custom workloads would not be publishable, they can be quite useful in measuring a solution using alternative workloads that have been derived from other customer application behaviors.
It is also possible to create a workload that combines Unix and Windows load generators in the workload definition. This is for applications that use both Unix and Windows in their workflows. E.g. Windows systems collecting sensor data and storing it over SMB to a storage solution, while a Unix based compute farm is analyzing the data from this storage solution over NFS, Lustre, GPFS, or some other file system technology.

To create a heterogeneous workload, one will need to modify the storage2020.yml file and add the new workload definition, and within the subcomponents of the new workload definition one can specify the platform type as one of the parameters in the definition.

```
Benchmark_name
    - CUSTOM_BENCHMARK
              # Global options for the Workload.
              Proc_oprate_threshold: 75
              Global_oprate_threshold: 95
              Proc_latency_threshold: 0
              Global_latency_threshold: 0
              Workload_variance: 0
              Dedicated_subdirectory: 0
              Business_metric: "CUSTOM_METRIC"
              Warmup_time: 300
    - Components
        - UNIX_COMP
            # Component options
            Op_rate: 100
            Platform_type: "Unix"
            . . . . .
    - Components
        - WINDOWS_COMP
            # Component options
            Op_rate: 100
            Platform_type "Windows"
            . . . . .
```

In the sfs_rc file set your BENCHMARK_NAME=CUSTOM_BENCHMARK_NAME
The UNIX_CLIENT_MOUNTPOINTS list will contain the Unix load generators. The WINDOWS_CLIENT_MOUNTPOINT will be the list of Windows load generators.
Each list must contain the same number of mount points as the other mount point list.

## Custom homogeneous workloads with heterogeneous client types

In some cases, one may wish to use one of the standard workload definitions, however the goal is to use both Unix and Windows load generators at the same time. This enables the ability to stress the storage solution under simultaneous multi-protocol activity. E.g. NFS & SMB, or GPFS & NFS & SMB.

To achieve this multi-client type configuration, one must provide additional information about the clients so that the SM2020 python script can identify which mount points belong to which client type, as each needs to receive information that is appropriate for that client type.

Example:

UNIX_PASSWORD=myUnixPassword

WINDOWS_PASSWORD=myWindowsPassword

CLIENT_MOUNTPOINTS= Windows_name1:\\servername\share Unix_name1:/mnt/workdir
Windows_name2:\\servername\share Unix_name2:/mnt/workdir

WINDOWS_CLIENT_LIST= Windows_name1 Windows_name2

UNIX_CLIENT_LIST= Unix_name1 Unix_name2

The SM2020 python script will place the next workload object on the next client specified in the CLIENT_MOUNTPOINTS list and wrap at the end of the list. The selections are associated with the correct client type from the client type lists and arguments are sent in the appropriate format with the appropriate content. E.g. Windows_name1 and Windows_name2 will be sent the WINDOWS_PASSWORD while Unix_name1 and Unix_name2 will be sent the UNIX_PASSWORD

In this configuration the Prime client will use ssh to start remote instances of the benchmark across both Windows and Unix clients. This requires that the clients have ssh properly configured such that digital keys are exchanged, and no password challenge is required.