

Storage system evaluation criteria

A storage system at Fermilab must support two distinct types of workflows. Files, once their creation is complete, are considered immutable in either type of workflows.

- A. The first is independent processing of small numbers of input files to produce one or a few output files. This is typical of HEP work. The number of such processes is large (many hundred to a few thousand) but the average I/O rate of each is low.
- B. Correlated processing of input and output files - an HPC workload such as LQCD's, suited to parallel IO, but achievable with separate files per process. Sub cases:
 - B1. Single file parallel IO - many processes write a single file, at different offsets.
 - B2. Multifile IO - each process writes a separate file, and the files may or may not need to be concatenated afterward.

One system might not fit both. Will distinguish A or B where specific to the one application.

It is assumed all A requirements apply as well to B

Capacity:

- CAP_1. Must be able to grow indefinitely in data capacity by adding units.
- CAP_2. Disk subsystem imposes no limit on sizes of files. (The capacity of a single tape cartridge might still be a limit.)
- CAP_3. Storage capacity must have a scalable unit and it must be easy to add more, remove, and replace the scalable unit.

Data storage functionality, scalability and IO Performance:

- FSI_1. Aggregate maximum IO rate must scale when more scalable units are added
- FSI_2. The system must support at least 600 WAN and 6000 LAN transfers simultaneously, with a mixture of reads and writes.
- FSI_3. System must be able to sustain 5 GByte/s of aggregate IO today (assuming file sizes of 1 GByte), with increasing ceilings in the future.

- FSI_4. A single LAN client process must be able to have at least 100 files open for reading.
- FSI_5. The entire system must support having roughly 1/10 of all its files open for reading.
- FSI_6. The system must support at least a million files on-line, with this limit growing with time and scale.
- FSI_7. The aggregate system capacity must be at least 5 petabytes, with this limit growing with time and scale.
- FSI_8. The system should be architected such that it will scale when units are replaced by ones with advanced technology
- FSI_9. With smaller files, performance may degrade, but system must not collapse.
- FSI_10. Must be able to control number of WAN and LAN transfers independently, and/or set limits for each transfer protocol.
- FSI_11. The system must be able to queue the offered load before performance degradation, without dropping requests or deadlocking.
- FSI_12. Writes should be spread (in aggregate) across storage hardware and network infrastructure.
- FSI_13. Reads should be spread across storage hardware and network infrastructure, by replicating "hot" data or otherwise.
- FSI_14. Simultaneous reading of a single file by many clients. Each client would have an arbitrary file pointer and in general these pointers will be dispersed throughout the file. Large client counts (of the order of hundreds to a thousand) are likely. [XX]
- FSI_15. [B1] Simultaneous non-overlapping writes to a single file by many clients is a desirable capability. Alternatively ...
- FSI_16. [B2] Each client reads or writes a slice of a large data structure with each slice mapped to a separate file. Simultaneous non-overlapping writes to individual slices is then not required.
- FSI_17. [B] High throughput is not required for a single write operation, but aggregate throughput across the ensemble of files open by a family of clients (a set of MPI processes could be as high as 1024) with no large variations in client speed is required. (HPC processes may synchronize on such an IO operation.) - jitter]
- FSI_18. The system cannot suffer deadlocks, nor be significantly impaired by hung or deadlocked clients.
- FSI_19. The system should not suffer performance degradation due to fragmentation when run for long period.
- FSI_20. The system should not show degradation in performance or failures due to OS issues when run for a prolonged period.

Data integrity:

- IGT_1. CRC or hashes will be kept for all files. They must be accommodated in the metadata with fast access.
- IGT_2. Support for multiple CRC or hash algorithms is desirable.

IGT_3. The system must scan itself, or allow itself to be scanned, for file corruption without undue impact on performance. Any benchmarks are to be measured with integrity checks enabled.

IGT_4. Transfer protocols are expected to support end-to-end integrity verification.

Tape integration:

TAP_1. Must provide a means for transparent reading and writing from/to a tape HSM back-end.

TAP_2. Ability to use Enstore as that HSM is very desirable.

TAP_3. Must allow tape-backed data to be flushed from online storage without removing namespace entries and transparently staged in when accessed.

TAP_4. It must be possible to enroll our existing data and metadata.

Useability: Maintenance, Troubleshooting and problem isolation:

USE_1. It must be easy to add, remove and replace scalable units without affecting operations and existing storage, and without causing performance anomalies. For example, adding new, empty units of storage must not attract too many new writes into the new units.

USE_2. Must be possible to have some storage units offline and still have the system function for whatever files are available. If missing data is on tape, the system must stage it in to working units and tolerate the duplication if the offline units return to service.

USE_3. Faults must be localized to a scalable unit. For example, fixing a filesystem problem on one unit cannot require the entire system to be down.

USE_4. Running fsck or similar operation on a storage unit or laying a file system down on a node should take hours not days.

USE_5. Must be able to expand namespace capacity and performance within a reasonable downtime (better yet, none).

USE_6. Restarting a component of the system must have minimal and recoverable impact on operations in progress.

USE_7. Maintenance and management operations must be amenable to scripting from remote Unix systems. GUI-only management is not acceptable. An API is a plus.

Data accessibility:

ACC_1. Must support POSIX IO access to the data, possibly through a preload library. NFSv4 would be good.

ACC_2. Must accommodate a GridFTP/SRM interface for WAN and LAN access

Namespace and Namespace performance:

- NAM_1. There must be a reasonable approximation to a Unix filesystem to facilitate user browsing.
- NAM_2. The namespace must be mountable and browsable on ~2000 nodes.
- NAM_3. Must be able to do removes, renames, stat, directory listings, from many clients simultaneously without affecting primary data operations.
- NAM_4. It must be possible to analyze namespace usage patterns to identify abusive users.
- NAM_5. Namespace must be scalable to support millions of files with no degradation in system performance.
- NAM_6. Namespace must be scalable to support hundreds of client operations per second.
- NAM_7. It must be possible to make a backup or dump of the namespace and metadata without taking the system down.
- NAM_8. There must be some provision for restoring a namespace and/or metadata backup and bringing the system to a consistent state, given that the system had been active after the backup was made. Desirable - HA on headnode services

Platform and System:

- SYS_1. Storage must primarily use a general-purpose platform so existing hardware can be added to or subtracted from the system.
- SYS_2. Ethernet will be the primary access medium for capacity computing.

Bottom line:

- BOT_1. The total cost of acquiring and operating the system must be better than what we have now.

Security:

Unresolved: do we assume a trusted network interconnecting the components?
We require integrity and some availability, but not confidentiality.

Other:

ACLs, quotas, allocation, auditing.