GOALS

The goal for this effort is to achieve a well accepted by industry POSIX I/O API extension, or set of extensions to make the POSIX I/O API more friendly to HPC, clustering, parallelism, and high concurrency applications. It is likely that this extension effort will need to be done in phases, due to the fact that some enhancements to the POSIX I/O API to help these applications are well understood, like high concurrence extensions, while other proposed enhancing ideas are not well fleshed out yet, like active storage concepts.

This initial meeting will cover:

- The initial mechanics for how the POSIX API is extended
- Ideas for extensions
    - Short term
    - Longer term
- Formation of a plan of attack, organization, and mapping of next steps.

Below here are extension ideas:

POSIX I/O API ISSUES

The POSIX API is "unnatural" for high-end computing applications. Opportunity abounds to make the POSIX I/O API friendlier to HPC, clustering, parallelism, and high concurrency applications. The entire set of operations should be combed over and carefully, consistently, enhanced for high-end computing, clustering, and high concurrency needs, while maintaining complete compatibility for legacy applications. Possible areas in which the POSIX I/O API could be extended are listed below.

Ordering

One of the prime reasons the POSIX I/O API is awkward for HPC stems from the "stream of bytes" paradigm in which POSIX I/O is based. POSIX I/O was developed to provide an interface from a single machine with a single memory space to a streaming device with some simple random access capabilities. HPC/parallel I/O applications are based on distributed memories being mapped to many storage devices. A re-interpretation towards the concept of a "vector of bytes" would be more appropriate for HPC applications versus a "stream of bytes" model. This would entail a careful reexamination of the POSIX I/O-related interfaces to eliminate or relax stream-oriented semantics.

Coherence

Probably the worst offense for really high bandwidth I/O is the fundamental read and write calls. They have two glaring problems. The first is coherency. The last-writer-wins semantic of the write call without cache-snooping, looking at the write cache of all participating nodes, is difficult, perhaps impossible, to achieve in a scalable fashion. Similarly, again without cache-snooping, the overhead of cache invalidation is enormous for reads, especially if attempted for regions for which an application might never have interest. Additionally, block boundaries can present coherence issues for the application. A standard way for applications to assume all responsibility for coherency is needed,

which implies that application control of cache flushing/invalidation at some level is also needed. Additionally dealing with block boundaries and alignment needs to be dealt with in the API in a consistent manner which takes alignment/block boundary issues away from the application. This problem is particularly bad in the implementation of O_Direct today.

Extension issues

The POSIX I/O API is organized as a set of mandatory functions and sets of extensions to that set of mandatory functions. Current extensions of the API are awkward for use in HPC. For instance, the real-time extensions for list-based I/O (listio) are useful; however they are awkward in that they have the restriction that the memory regions of interest must coincide exactly with a region in the file. This is not always, often, the case. Instead, two separate vectors, one of memory regions and one of file regions, could be passed and the two lists reconciled by the implementation. Such a concept allows scatter/gather-gather/scatter semantics.

Missing capabilities

The POSIX I/O API is also not as complete as one would like. For instance, there is a call to position and write a buffer of data (pwrite) but no call to position and write a vector of described memory regions, like a pwritev.

Metadata

The classic “wish” in this area is for the support of “lazy” attributes. These are results to the stat call, where some values may not be maintained at the same granularity normally expected. The most obvious fields are those that record timestamps for last update and modify. Many file systems implement these “lazy” attributes no two in the same way. A standard, portable set of semantics would be useful. Explorations into a more descriptive API for metadata management and query to allow applications to deal with the needed information could be helpful in this area. For years, the backup/archive industry has needed a portable bulk metadata interface to the metadata of file systems. Additionally alternative metadata access methods to the standard tree based access using readdir() and stat() operations should be considered for inclusion in a POSIX extension.

Locking schemes

Locking schemes that support cooperating groups of processes is also necessary. The current POSIX semantics assume only a single process would ever want exclusive write access. In the HPC/parallel world, groups of processes may want to gain exclusive access to a file. Perhaps a mandatory version of the fcntl and flock is needed. It is necessary that more options for how locks can be requested and revoked be provided. Legacy codes must continue to work as expected, so current locking semantics must be maintained.

Shared file descriptors

Shared file descriptors between nodes in a cluster would be of great value, not just between processes on a node. The component count for supercomputers is going up in the next generation of supercomputers. Full lookups in the name space are going to have to become a thing of the past for parallel codes. Some mechanism decreasing the need for

mass name-space traversal is desperately needed, even if it requires new semantics to accomplish it. It is possible to implement this via higher level function in the I/O stack. Implementation of shared file descriptors at the file system level might be difficult but none the less would be quite useful, if achievable with a reasonable amount of R&D investment. As mentioned in the metadata section above, an alternate API for name space traversal as well as alternate file organization (something other than a tree) might also be a way to assist in this area.

Portability of hinting for layouts and other information

There is a need for proper hinting calls. Things like stripe, stride, depth, raid layout options, etc. need to be accomplished in some portable way. Additionally, there needs to be mechanisms for adding standard hinting without major new standardization efforts. Perhaps the MPI-IO Info approach for hinting can serve as a prototype, particularly in terms of the semantics, like ignoring of unknown hints and the mechanism for getting the values to use.

POSIX extensions for archive applications

Many users often state the desire to access the archive via the POSIX API. This is a natural and understandable request, as the POSIX interface is so well known with an enormous list of applications that utilize this interface for file management. Familiar UNIX application utilities like ls, cd, mkdir, rm, tar, dd, etc. are all handy applications and if the archive could be manipulated via these commands, users would have a more flexible way to manage their archive activity. The HPSS has provided this function via an NFS interface to the archive, but this interface is less than satisfactory for performance and manageability. The fundamental problem with providing the POSIX interface to the archive is the assumption by most UNIX utilities that use the interface that all data is readily accessible on disk devices. In an archive often, data is not readily available; it is frequently on a tape device not mounted in a tape transport. Commands like grep which read the contents of every file in a directory issues to an archive could cause hundreds or thousands of tape mounts. Additionally commands like move and tar which move deal with entire trees of files could wreak havoc on the archive. There is a need to control the rate at which requests are made of the archive. Also, UNIX utilities that use the POSIX interface assume very little latency for most POSIX operations. Often, archive requests have much higher latencies. The Data Migration API is provided in some file systems which allows for HSM functionality to be implemented making a file system appear to be infinite by transparently migrating data from the file system to an archive and back as needed. This interface would be a nice way to implement a POSIX interface to an archive, and the HPSS supports this interface, but unfortunately few file systems have implemented this interface, industry does not seem to be supporting the need for this interface to be maintained, and it suffers from the same issue mentioned above, the difficulty in controlling the rate at which requests come to the archive. It might be possible to introduce new POSIX semantics that alleviate the issues that arise from trying to deploy a POSIX interface to archives.

The lack of a good way to deploy a POSIX interface to the archive has led to the proliferation of interfaces to the archive. If a manageable POSIX interface were attainable, a reduction in archive interfaces could be possible.

Necessary determinism in file systems

Additionally, all operations done on the basis of time are awkward to deal with on supercomputers with light weight operating systems due to the inability to respond vi asynchronous signaling to call back mechanisms. Supercomputing applications need more deterministic behavior and more control over the hardware throughout the entire computation and I/O hardware stacks. Operations with the ability to be driven from clients that can't listen for call backs is vital. It is quite likely that some variants of supercomputers with hundreds of thousands of processors simply won't be able to be bothered with call back mechanisms at all.

Active disk concepts support

As active disk concepts are proven through research and testing, an API to enable this new function should emerge. In order for this new paradigm to be widely used, supporting this API as a POSIX extension should be considered.