Mods to pftool for the gpfs-object-campaign concept

Some background:

Pftool has

* an initialization where it reads up configuration information
* an organizer process that assigns work, starts up everything and stops everything, it uses number of files and amount of data as a way to load balance and built batches of work. It also does directory creation on the target because the target directory has to be there before work can be dispatched to move stuff into that directory
* a set of readdir tasks that do readdir and passes lists of names to the organizer
* a set of stat tasks that stat each file and passes the stat info in batches back to the organizer
* there are mover/comparer tasks that do the read from src, write to target or read from source read from target and compare. This process also updates the metadata on the target for all but multi-part files. For multi-part files it just moves the data. The completed work info is passed to the tracker process.
* There is a tracker process that gets info from the stat and mover process. It reports on the work, so in the case of a list run it lists the files, in the case of a move or compare it lists those actions. It also tracks the progress of multi-part file movement and determines when the entire mutli-part file has been moved and updates the metadata for those files.

So the src tree is walked in parallel, with multiple readdir tasks and multiple stat procs which do lists and also determine what needs to be moved to the dest by stating the dest. The movement work is done either lots of small files in a bunch or a part of a large file. The work is load balanced but it favors the mover tasks because that almost always takes longer. The mover tasks put back pressure on the staters and readdirs .

Restart is accomplished by using the mtime attribute on the source/dest. The last thing done to the dest file is to set the src mtime to dest mtime, so you can walk the tree over and over until all the files have their mtimes the same and you know you are done. This works fine for smaller files, but for very large multi-part files I don’t think there is a restart

You can compare metadata only, compare file data, list trees, move/sync trees with only files that have changed, or move/sync files with overwrite.

Modifications for the gpfs-object-campaign

One must figure out how to hide the gpfs metadata only mountpoint from the user, so it is possible that the pftool process family needs to do a private mount of the gpfs metadata name space using unshare or other mechanisms

Init: will need to read the config info/db for how to treat the gpfs object store

* place to write small (packed), medium (uni), and large (striped or multi) into the object store(s). it will
* optimal chunk size for writing to the object store(s)
* whether to do packed files on the object store(s)
* dirty % for the object store(s)
* security method for object store(s)
* access method/path (url) etc.
* future stuff like latency/retry/loadbal/failover/etc.

if this is a copy to a dest that is over quota, just kill the job right away, and you check the quota by looking at the bytes in the filesystem(part) that is provided by the periodic running of the lazy-fsinfo process.

Readdir: no change that I can think of, walking the gpfs metadata seems like it will be the same as for a normal file system src or target

Stat: not too much change, it would be wise to increase the record size for stat info that is passed around between stat/tracker/organizer to include the xattrs like actualwritten-chunksize, objectnamespace, objectname, object offset, if you are moving data or comparing, so the mover/comparer tasks don’t have to go back and get the attributes to know what to do, it would just be passed to them by the staters.

Organizer: much the same except if a multipart file is being overwritten moving the file to trash needs to be done here. If reading from gpfs-object try to use actual chunksize written to chunk up requests for read. If writing to gpfs-object try to use the desired chunksize for that dest for the requests. It is possible in the future that the organizer or other related process might want to sort read (src) workload in object system/object/offset order to make read of packed objects efficient by assigning them to the same mover process in reasonable order.

Movers/comparers: if src or dest is gpfs-object you would do reading and writing in a different way than today which is just pread/pwrite. You have to obey the “where to write based on size”, chunksize for that dest, pack or not, security/access method etc. and all the stuff read in by the init task above.

* If file exists and overwriting, you move that file to the trash (unless its multi-part in which case the the organizer already did that)
* dest
  + If packing, you move all tiny dest overwrite files to trash, then allocate an object to pack into and pack the files into the object. Then update the metadata for those files including xattrs, size, and mtime (using trunc) appropriately. Send appropriate info to tracker
  + If uni file, just move to trash if necessary, allocate object, write object, and update metadata including xattrs, size, and mtime. Send appropriate info to tracker
  + If multi-part, just write the correct data into from src into object, build appropriate record and send to tracker (tracker will update the metadata for multi-part file) (yes there could be trash we have to pick up later ☹ )
* Src
  + If packed or uni, get correct data for that file out of the object using xattrs
  + If striped get appropriate data out of object(s) (could be objects if you are writing a different chunk size than you read). You have to use xattr which will tell you that you need to open the gpfs file and get the right object(s) to fetch). There is probably some optimization that could be done, but that could come later. The scheduler should make this somewhat optimum for reading or writing to gpfs-object store. If reading to and writing from gpfs object store, probably should choose to optimize the write.

Tracker:

Updating metadata for mulit-part file would need to update the xattr, mtime, size, and add the object list or some formula or something to represent the objects that make up the file using the chunksize being written). Later to assist with partial large file restart, one could if all the chunks up to X in size are successfully written, you could update the gpfs file with the chunklist and the size, with an xattr flag that you are in the middle of updating this file. This way eventually one could not read/write that chunk again upon restart. This requires logic, in several places and even has risks of the xattr flag never being updated. Work to do clean multi-part restart needs to be thought out, as it has changes in the movers, the tracker, and maybe even the stater.