Los Alamos National Laboratory Interviews

Technical Report UCSC-SSRC-11-06
September 2011

Christina R. Strong
crstrong@cs.ucsc.edu

Stephanie N. Jones
snjones@cs.ucsc.edu

Aleatha Parker-Wood
aleatha@cs.ucsc.edu

Alexandra Holloway
fire@cs.ucsc.edu

Darrell D.E. Long
darrell@cs.ucsc.edu

Storage Systems Research Center
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
http://www.ssrc.ucsc.edu/

This technical report contains the compilation of notes from a series of interviews conducted at Los Alamos National Laboratory (LANL). The interviews were focused on the scientists who used the supercomputers at LANL and the code developers who wrote code for use on the supercomputers. Interviewee data has been anonymized.

Los Alamos National Laboratory Interviews

Christina R. Strong, Stephanie N. Jones, Aleatha Parker-Wood, Alexandra Holloway, Darrell D.E. Long
University of California, Santa Cruz

Overview of LANL Computing

Los Alamos National Laboratory's (LANL) high performance computing is divided into three network partitions depending on the security level of the work: Turquoise for more open projects, Yellow for protected or sensitive projects, and Red. A Network File System (NFS) is used for home directories and project directories, run on NetApp servers. Panasas is used for the Parallel File System (PFS), which is shared among supercomputers and used for scratch space. The archival storage is mainly tapes; Turquoise uses a LANL developed GPFS solution while Yellow and Red use HPSS.

NFS is meant for "smaller files" that are serially written, while PFS is meant for large files and parallel writes. There are two ways checkpoint files are written to PFS: N-N and N-1. N-N means that every processor writes its own checkpoint file. N-1 means that all processors write to a single file (often in a strided manner, where all temperatures are written together, all pressures are written together, etc rather than segmented where each processor writes all of its information sequentially). A benefit of N-1 writes is that it allows a restart on any number of nodes (N-M restart); N-N requires the same number of nodes (N).

Panasas is composed of many object storage devices (OSDs) and metadata servers (MDS), the number of each depends on the number of 'shelves' in the system. At LANL each user is "assigned" to a MDS for use (a MDS can have multiple users), so when a user writes or reads occur everything goes through one MDS. When writing or reading N-N, this can be problematic for file operations (such as create), so there is a solution in the works called PLFS (Parallel Log File System). PLFS is an interposition layer that has two main benefits: it spreads your writes across multiple metadata servers, and it makes the checkpoint look to the user as if it is being written N-1 but actually writes it as N-N to the filesystem. This is beneficial for both developers and users: writing as N-N makes it much faster (utilizing more spindles) while maintaining the N-1 view that is preferred (and sometimes required) by the users.

When a user runs a piece of code, she/he gets a dedicated set of nodes to run on – no one else is allowed to use them. However, you are limited in the amount of time you are allowed. At LANL, the limit is usually between 8 and 24 hours.

Users

While most of the users we talked with were scientists, their jobs differed quite a bit. User A's job is to understand how the problem should be set up, input it to the computer, and keep an eye on it as it runs (most of his code has very long runtimes). In a typical simulation, the input file and code are very "vanilla" for easy transport. While the code is mostly written in Fortran, there is a movement toward C/C++. Along with checkpoint files (which can also be used to visualize the entire problem space), User A writes out small graphics files (HDF files) so that he can play the simulation as a movie. These are written at a defined time step and there is a movie for each variable. Every morning he goes in and plays the movie for each variable to check for anomalies. There is a script to clean up the directory when the job ends, that creates an HDF directory with subdirectories for each variable and then moves the files to the correct directories.

User A keeps everything in PFS on scratch, he doesn't use NFS. However, he doesn't really care where it lives (NFS or PFS or archive), he just wants to be able to see it and store it. He'd like to be able to get

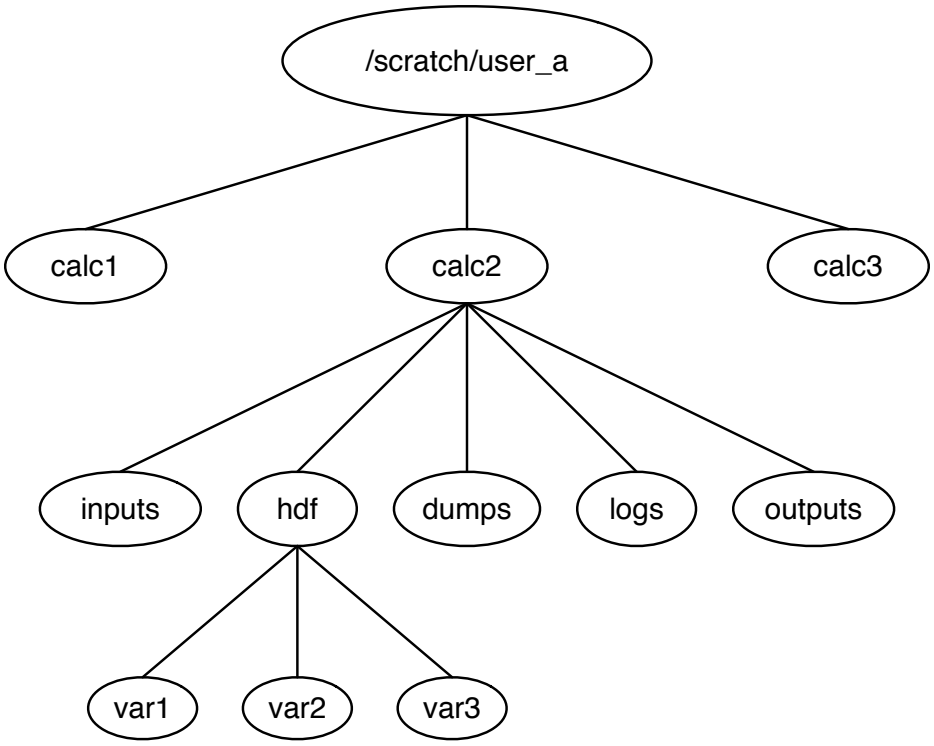

Figure 1: User A's scratch hierarchy.

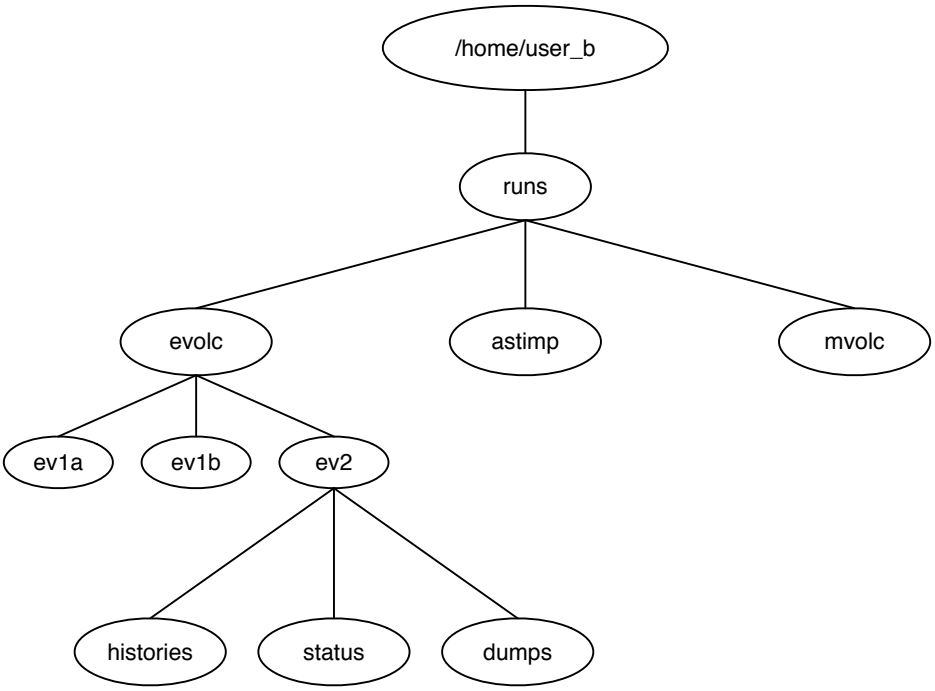

Figure 2: User B's home directory hierarchy.

around the wildcard limit on the number of files, since if they accidentally write too many visualization files, they have to move them by hand. Currently he stores everything, including his checkpoint files. Ideally he would like to be able to store only the "important files", such as the checkpoints in which a variable was changed. He would very strongly like to be able to just say "Give Alice permission to Experiment A" and have the permissions work out. There are many people who do not understand or are unfamiliar with UNIX permissions that he needs to share access with, which often results in over or under sharing.

User B is currently working at the National Metacenter for High Performance Computing in Norway (NOTUR). With respect to file systems in high performance computing, his opinion is "The short answer is that no one has done it right". The code that he runs generates massive amounts of data, causing the file system to fill up far too rapidly. The interim solution in Norway is to have a separate, offline storage center linked by a 10Gbit line.

User B runs mainly on two computers, one at the University of Oslo and the other at the University of Tromsø. The computer in Tromsø has limited scratch space, so while everything is run in the global scratch, he has to keep a careful eye on it to make sure the space doesn't fill up. Since unlike LANL, NOTUR puts no limit on the amount of time a job can run for, User B has a script that moves files to his home space. Like LANL, the scratch space is a parallel file system, while his home directory is on NFS. To analyze his files, he has to move them from the home space to his personal computer (a Mac). If he's interested in keeping it, he moves it to the National Archive. The computer in Oslo has a much larger scratch space, so he does not need to move files to his home space nor does he have to download them to his personal computer to do analysis.

To run experiments, User B actually starts by drawing a sketch of what the problem space looks like (e.g., explosive volcanism). He then determines what materials are in the problem space and which variables he

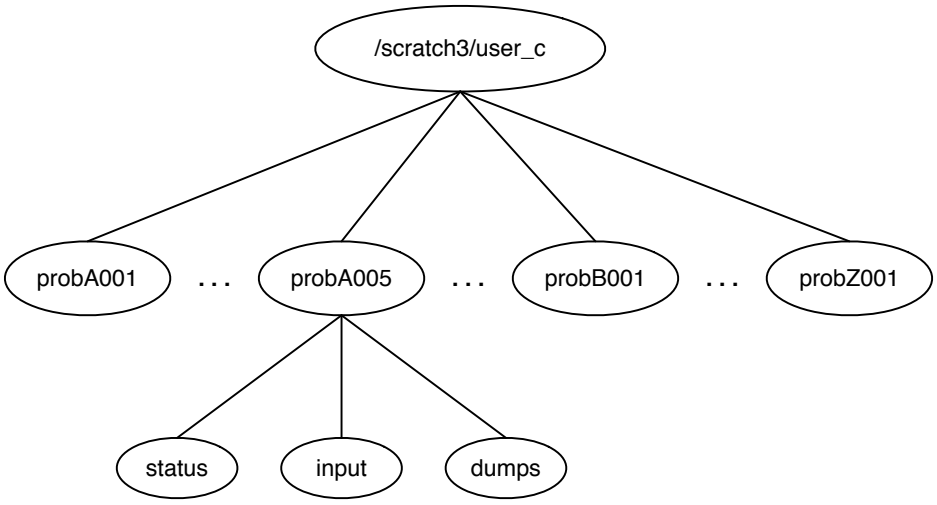

Figure 3: User C's scratch directory.

wants to keep track of and modify. This translates to a 2D table (which lives on his personal computer). His files and folders are then named with an abbreviation of the problem name and a parameter such as 1a; the explanation of what 1a is would be in the table. He has several scripts which move files to the correct directory, and has a very hierarchical setup in his home directory. The reason his personal computer is a Mac is because he can do a content search on it very quickly. One way he remembers runs from a long time ago is that he embeds a frame of the video into his spreadsheet of parameters for the run.

Ideally User B would like to be able to do in-situ analysis; currently the major roadblock for that is bandwidth and space issues. He would also like some automated way of collecting results and correlating it with his parameter spreadsheet. The embedded video frame is often used as a primary search mechanism, so it would be good to have the frame correlated as well. Currently naming is done by dividing into subdirectories, but he would like to be able to ask a question such as "find the run that had this particular scenario". Like User A, User B would like to be able to say "Give Alice access to Calculation A" and have the permissions work out correctly.

User C mainly uses scratch space for his calculations. Things that are not going to change are stored in netscratch, such as equations of state, opacities, geometries, and a separate directory for scripts. Within scratch, his problems are named things like ProbA123. He keeps a separate PowerPoint presentation as a notebook that is saved to his desktop to keep track of which run 123 stands for, as well as storing the information in a header in the input deck.

He uses PowerPoint "out of necessity", because he imports the visualization files to the presentation as well as keeping the text information there. It is an easy way for him to combine text and images. It's also easier for him to think of his experiments along the lines of "It's in the range of runs 123-127", and makes wildcard searches easier. He actually downloads the HDF files and the processed output files to his desktop, in order to use a tool for engineering analysis (igor). The processed output are text columns that were generated by scripts that ran on the HPC. Once a run is complete, he archives the entire directory from scratch (as well as the directory from netscratch, though that is archived on a campaign (set of runs) basis). After a month he goes back and deletes the checkpoint folder.

Ideally he would like to have user defined tags so that he can use them to help alleviate his dependency

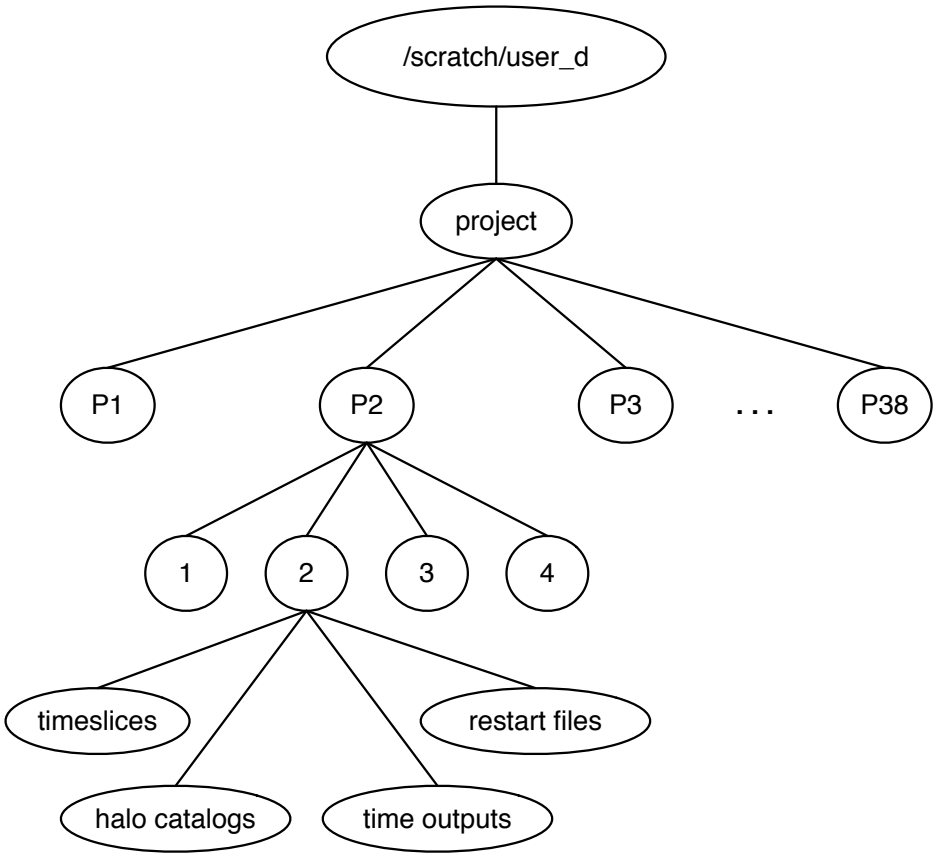

Figure 4: User D's scratch directory.

on the PowerPoint presentations. He would like to be able to do a search such as "Give me runs with these tags but not this tag". He also likes the idea of being able to say "Share experiment A with Alice" and having the permissions work correctly. Given the nature of his experiments, he would like to be able to do ranged queries over a set of runs.

We spoke with a team of scientists whose code generates very science-rich output, and they want to store all of it rather than just the information about the questions they are currently asking. They are working on putting it onto a server so that the cosmological community can also analyze the data output (specifically there is a large community that would greatly benefit from the data but do not have the resources to run the same kind of experiments that the team can). Even within the team, rather than one person doing one calculation and another person doing a different calculation, they tend to do one very large calculation and then they all analyze the output.

The metadata is encoded into the file names, such as the file type, the process number, and the time slice. Each directory has a file to identify what that run was. The team tends not to delete data, though the restart files are deleted once the runs are complete. The restart files capture the exact state of the problem, while a time slice is a paraphrased state of the problem. The time slices are used for visualization. The tool Paraview allows you to tweak parameters in the time slices and see realtime results of how it would effect the problem.

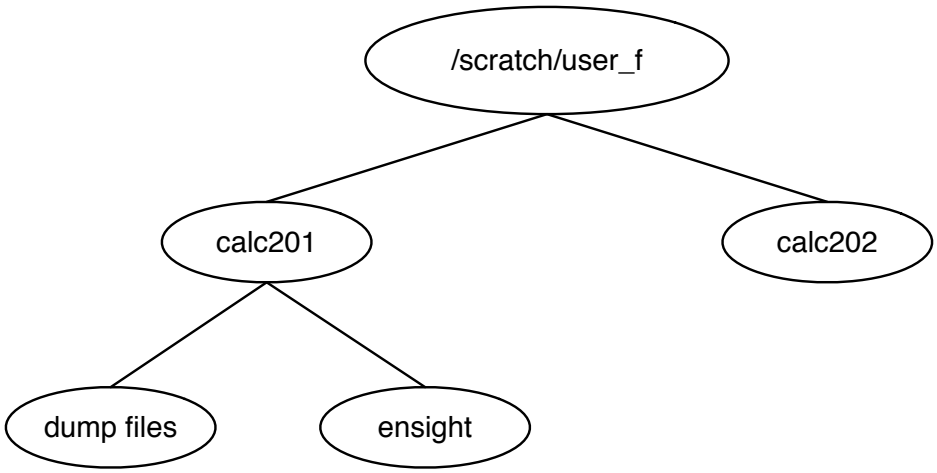

Figure 5: User F's scratch directory.

They would like to have where/how the file is stored abstracted away. They don't care whether it is written to PFS or NFS, and they would like to not have to worry about whether the system they are on prefers large files or multiple small files and how many they can have in a single folder. The team would use user defined tags if they had access to them, and would like to be able to search across different models or different realizations of a model. They also would like to have permissions fixed–User D has a folder named "User E" that User E has permissions to, and User E has a folder named "User D" that User D has permissions to. They would like to be able to retroactively push permissions down from the top and have it work correctly for files versus directories.

User F uses mainly netscratch and scratch; his home space is used for storing scripts. The project space is where information such as equations of state, executibles, and opacities are stored. He has a cron job that runs every night to back up netscratch and scratch to HPSS. He then deletes the checkpoint files as well as the ensight files from HPSS.

He uses the Mac Stickies application as an electronic notebook to keep track of what the calculations are (split out by theme), as well as maintaining a LaTeX file with the same information. Additionally, the input deck has comments explaining the details of what the calculation does. His standpoint is that documentation of the results and conclusions are far more important than having all the files that produced the results.

User F likes to have access and control of what is going on in the file system–abstracting away where things are stored, for example, is not something he would like. He would prefer to see a faster more reliable system than additional features. He agrees that sharing can be incredibly difficult, but would prefer to see more robust UNIX permissions, since he has run into group size limitations. He would also like to see the limit on the number of files removed.

User H is a climate scientist working on the Parallel Ocean Project (POP). They tend to have very long campaigns, that are usually distinct from each other. Things are kept in scratch until they can be archived. They generally compute anywhere they can, and do a combined computation with Oakridge National Laboratory (ORNL), since ORNL does atmospheric simulations.

User H also likes the query-able file system idea, though the climate community has a standard for metadata already. They have their information on the earthsystemgrid.org site, which is a distributed archive. Currently it's just a data archive, though they are working on putting analysis capabilities on the front end.

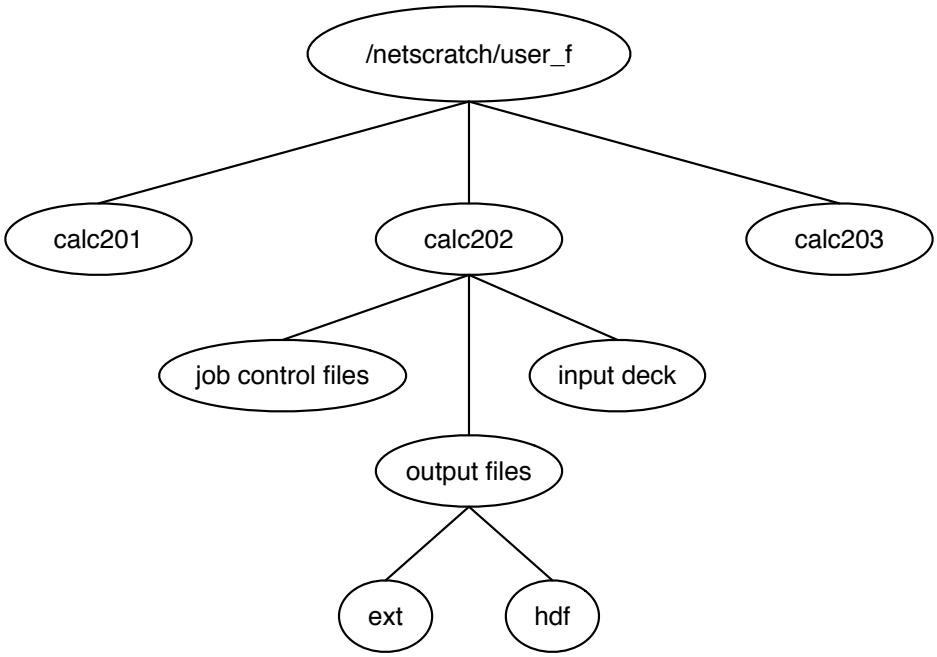

Figure 6: User F's netscratch directory.

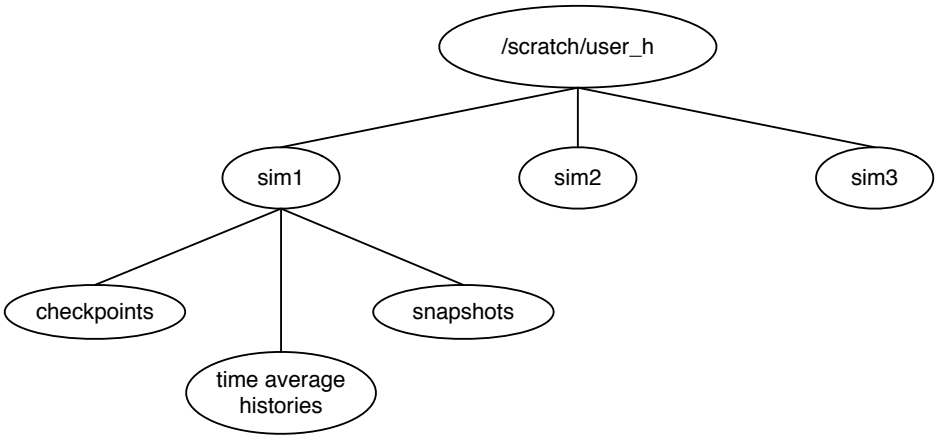

Figure 7: User H's /scratch directory.

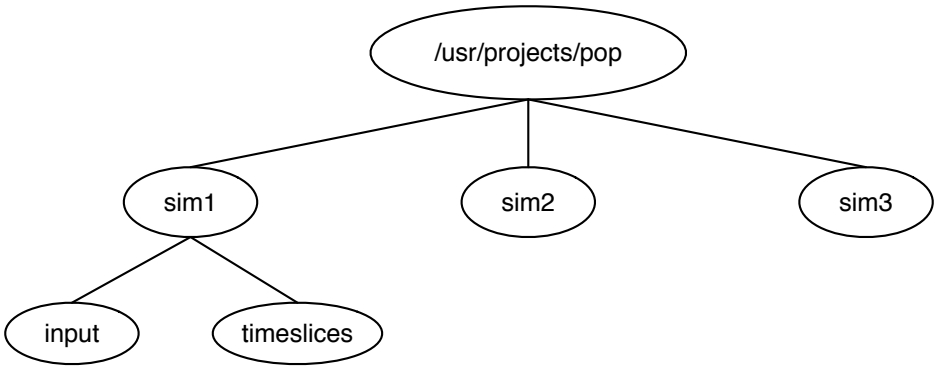

Figure 8: User H's /usr/projects/ directory.

Developers

Developer Z is a project lead on a massively parallel 3D Eulerian hydrocode (PAGOSA). In his code, every subdomain is the same on each processor, which means each processor is doing the same work with different data. The code is set up to create a set of ASCII text files so that users can keep up-to-date on the state of the problem they are running. These files are saved in NFS, while the checkpoint files (binary) are stored in PFS and keep track of every variable necessary for restart. A subset of these variables are represented in graphics files (such as HDF files).

There are five different ways the binary files can be written, providing the developers and users a large amount of flexibility. The default is *MPI-IO*, which writes a collective N-1. *All to 0* is an option in which all processors write to processor 0, which then writes to PFS. *Fixed machine size* is an N-N write option. *MPI-IOFMS* is a non-collective MPI-IO and *MPI-IOCTG* is a segmented MPI-IO. The file does not know how it was written, and thus can restart with a different number of processors. The code has a default for everything so that the user does not have to make specifications if he/she does not want to.

For this code, there are "ghost cells", one in each direction for the whole mesh, in order to handle boundary conditions. There are four pieces of metadata in the corners of the ghost cells, the simulation time, time step, cycle, and previous/next time step. This metadata information is used solely for restarts. Every piece of information is written out into an array, even if the material is not present (it is represented as a zero for that cell).

Restarts are done by simulation time (the frequency is chosen by the user), and it is done on a father/grandfather basis. The first checkpoint is saved as 1, the second as 2, then the third is stored as 1 again. At the end of the run, there is a TERM saved, so there are only three checkpoints total. The developers have a dump compare utility, which is used to check the difference between two checkpoints. This is to observe how fixing a bug is propagated through the variables.

Users are given a script which includes the line "setup current", which grabs the current version of PAGOSA. There is also a "setup alpha" and "setup beta" for the trial versions of the upcoming version, as well as "setup versionX" in order to rollback if necessary.

Developer Z works primarily in netscratch, and keeps the last 3-4 versions of the code available. Each directory is labeled very specifically to indicate what is being stored in that directory. He only keeps sample programs for one project at a time in scratch on PFS.

For code sharing, the CVS root is kept in the /usr/projects directory. It also contains the scripts and

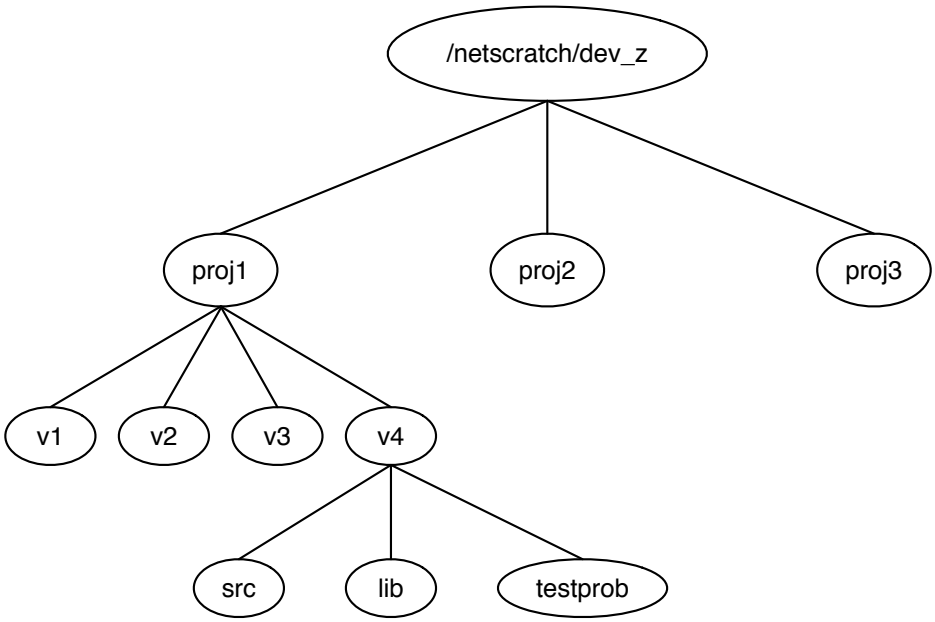

Figure 9: Developer Z's /netscratch/ hierarchy.

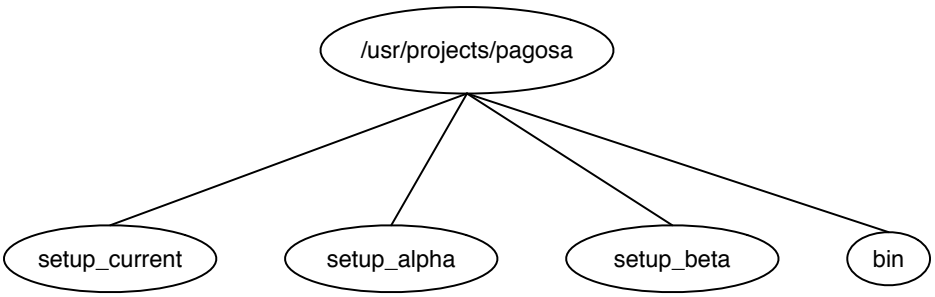

Figure 10: Developer Z's /usr/projects/ hierarchy for Pagosa.

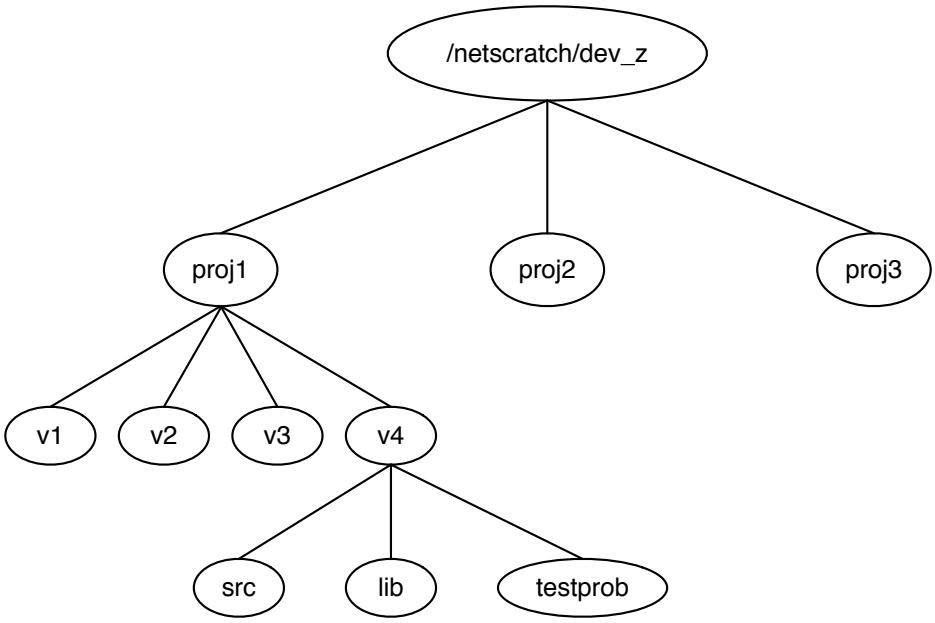

Figure 11: Developer Z's /scratch/ hierarchy.

executibles for the users. From his point of view, the ASCII file tells a user everything they need to know about the problem – the time, the size, the path to the output on scratch. Thus any question they might have they should be able to find the answer to by using grep on the ASCII files. In addition, the developers have created "parallel grep" to see how a variable has changed across different versions. His main concern is that it needs to be optimized or it will run incredibly slowly.

Developer Y keeps everything until he is "forced" to throw it away due to space limitations. He keeps various builds annotated with date and the local domain size in his home directory, and each build folder has things such as the physics, numerics, and initial conditions. However, his home directory is not large enough to hold his runs, so he uses his Mac to store things. He only uses /usr/projects/ if it's okay if everyone sees it, because the permissions are much looser. Therefore most of his development work is done in his home space and he uses that code in scratch. The directories in his scratch space are labeled hierarchically based on problem type, then global mesh size, then temperature.

He archives to both the LANL archive (GPFS, since he is on Turquoise), and to his Mac. In his team, they care first and foremost that it works – they can make it work for them regardless as long as it is robust. It would be nice to have a more efficient search, such as something that searches only over the header of the restart file where the metadata is or be able to tell the search tool to ignore binary files and only look at text files.

We had a lunch meeting with Developers X and W, and they wouldn't mind having the abstraction of where the files are stored. It is annoying for them when a scratch space goes down and they move to a different scratch space, and then when the first scratch comes back up you now have two scratch spaces for the same project.

Developer W has a permanent checkout from CVS in his home directory. The projects have distinct directories, and he tries to keep the size of his subdirectories small while not having too many. In scratch, his files are organized by the type of bug and then by project. All the files that are needed for testing the

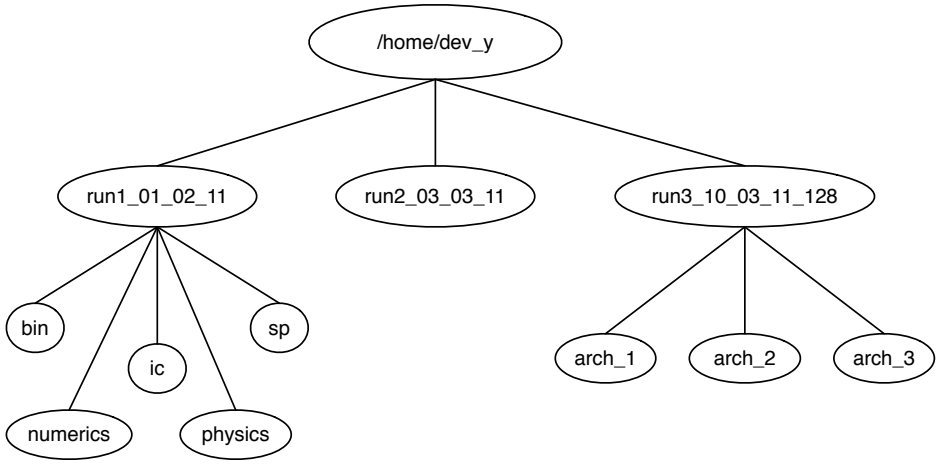

Figure 12: Developer Y's home directory.

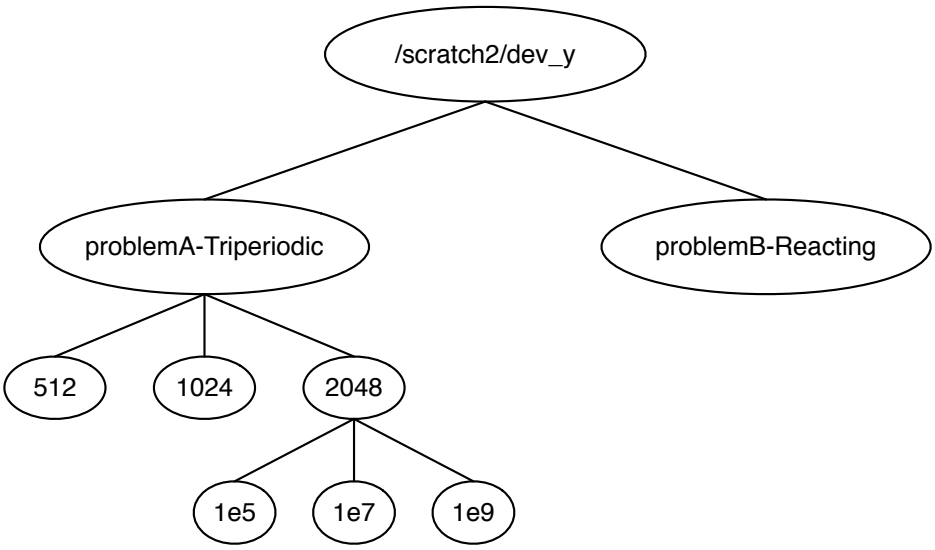

Figure 13: Developer Y's scratch directory.

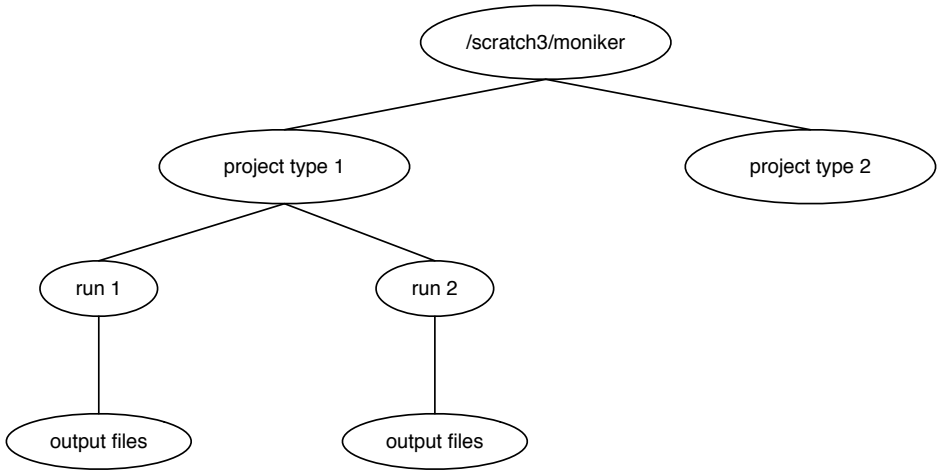

Figure 14: Developer X's /scratch directory.

bug are kept in the project directory. There is a readme file kept in each project directory that maintains the status of the run. He does periodic backups of his work to HPSS, but would like for metadata to return from the archive faster. Developer X runs in scratch, where he has folders for each project with subdirectories for each run. A post-processing script moves the important output files to a backed up NFS folder.

There are nightly regression tests that are run, but due to size restrictions they have to be run in different directories depending on the machine it is being run on. The results are stored in /usr/projects, but they have to be copied there since there is not enough space to run them directly from /usr/projects. Developer X prefers UNIX controls over access control lists (ACLs), but would like to be able to have more than 16 groups in UNIX.

Since HPSS compresses smaller files into one folder when it stores them, it would be useful if a run was stored in the same way. Any sort of file system needs to be self-diagnostic–when the secure machines break, it's a huge pain to try to fix them. Thus any file system we design needs to have trace and replay capabilities. It would be incredibly useful to have the ability to add metadata comments to files, as well as user defined tags. They would like to be able to do searches such as find all files associated to the tag problem1, and they would also like to be able to search over the tags themselves for specific things. They would like to in a sense have a single directory which is visible everywhere (addressing the multiple scratch space issue), and have UNIX commands that work. Most importantly they want a file system that is fast, reliable, and does not have any parallel bottlenecks.

Developer V spoke with us on behalf of User G, they work together looking at plasma physics (VPIC). Their code is unique in that it is funded by many different projects, so it does not live in any one specific project. It is "basic science code", but is very memory and disk intensive. Therefore their code determines exactly how much memory is available to it, and then uses every piece of it.

Since they can't write out and analyze every single particle (due to size restraints), they take low-order statistical moments of the particles and save those. They keep two checkpoint files, the most recent and second most recent (same idea as the father/grandfather practice of Developer Z). VPIC does N-N writes, but they actually throttle their write speed. They implemented "turnstiles" in MPI I/O which allows you to specify how many parallel writes are allowed at a time. They've found that 256 to 512 is a happy medium for their code.

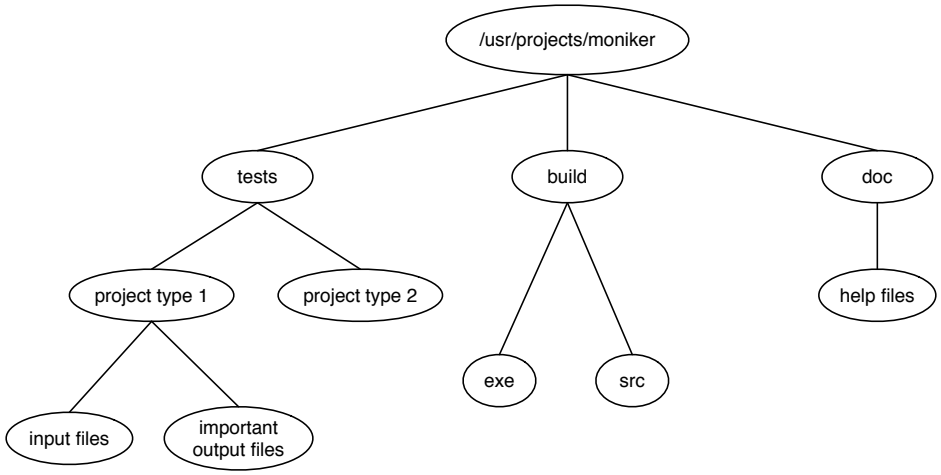

Figure 15: Developer X's /usr/projects/ directory.

Calculations are becoming larger, but they would rather re-run their calculations several times in order to refine the physics rather than store everything and analyze it later. Since they've optimized the VPIC code, it runs 2-3 times faster than any competitor's code, so they can run it three times and get much better results in the same amount of time. VPIC is the most adaptive and flexible code at LANL. Their input deck is not text like other input decks, but actually a list of C++ stubs which you can compile. This allows them to swap physics libraries in and out in the middle of their input deck, as well as read traditional input decks with ease.

Developer V tends to work on only a few problems at once; User G will be working on 20 to 30 problems at once. User G keeps a separate file to keep track of what is being run where and what the status is. She has very long descriptive directory names, because she tars the entire folder and then uses the long descriptive name to help her find things in archival storage.

Developer V likes the idea of a highly query-able file system, he would like to be able to do a command such as "store everything in this tree except checkpoint files". He would also like to be able to say "give me calculations like this old one I did"; User G will go through a binder of old calculations to try to find the one she is thinking of. The idea of abstracting sharing by saying "share experiment A with Alice" is very interesting to him; he hates UNIX permissions.

Developer U keeps a very hierarchical file organization, with her files and folders named based on what she's doing. Often these file names become incredibly long and complex (e.g. `sage_512proc_plfs_N_Nwrites_rr`), which often leads to errors such as mistakenly switching parameters in the name or leaving them out entirely. In Developer U's ideal file system, she would never have to *du* or *find*. She could say "show me all files related to experiment A" and the result would be a list of all files related to Experiment A regardless of whether they were on PFS, NFS, or archive. Perhaps it would tell you which files were in which file system, but it would pull up a list of all the files.

Something Developer T would like to see is the ability to find subsets of files, not just a single file. In many projects, a file is really just coordinates and the characteristics of those coordinates, so he would like to be able to do a search of "find coordinates where the temperature is greater than X". He wants the file system to become simpler, and more functionality to move to the middleware and the compute nodes.

Users do not like N-N writes, as it is a poor abstraction for the way they think about their file. They

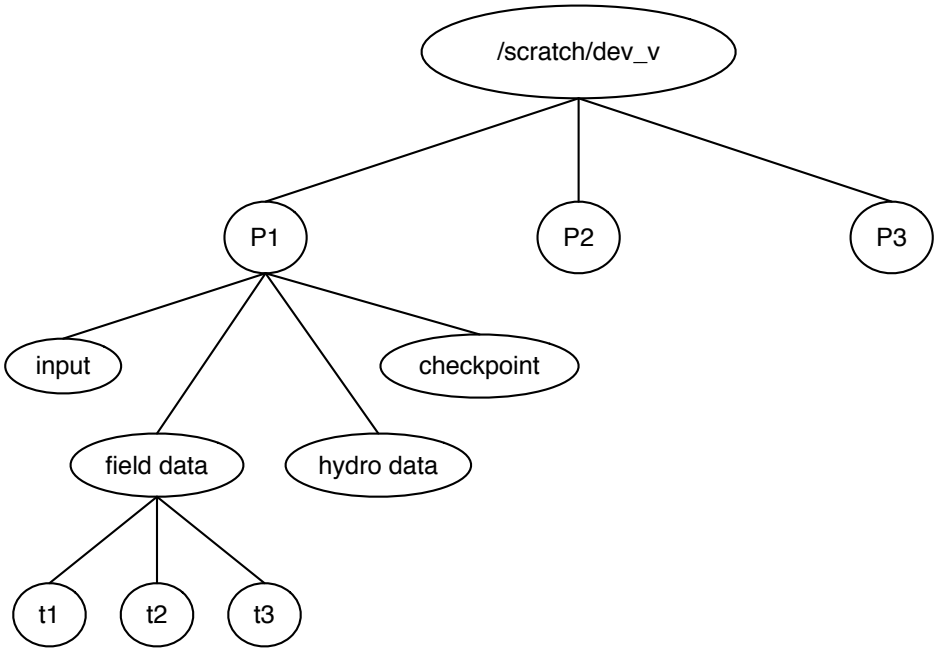

Figure 16: Developer V's /scratch directory.

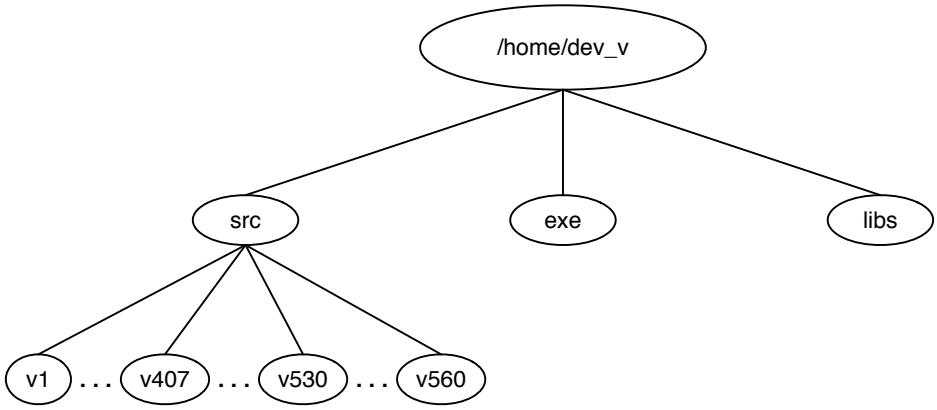

Figure 17: Developer V's home directory.

prefer N-1 files for analysis purposes. Developer T would really like to see something like HDF5, where the user can say "here is my data structure" and the system will store the data structure. The users then can talk in reasonable (to them) ways to the file system. One user currently has his code written such that it will check to see what machine it is running on, and then has specific code to optimize for that machine.

Other Conversations

We learned about the Alexandria Project, an ambitious provenance project that was started several years ago but was not continued (possibly due to catastrophic failure). It basically keeps track of the flow of information, attempting to find the lineage from a computer science build point of view. That is, it was looking at information such as "this file was built using this operating system, this computer, and these libraries". The most we currently know about it is at `http://www.taborcommunications.com/hpcwire/hpcwireWWW/04/1217/108996.html`.

While abstractions sound nice, they are hindered by POSIX standards–the way attributes work is not clean and permission structures are not conducive to this sort of thing. Existing workarounds are usable but clunky. Before the switch to UNIX there was a laboratory specific file system called Common File System (CFS). The archival storage system associated with CFS was called DataTree. CFS actually did a lot of the things we are trying to do right now, such as extended attributes and inheritance. The sharing mechanisms were more like NTFS, where sharing was done at the tree level. There was a lot of metadata that was kept, because it couldn't be stored in the file. Unfortunately, CFS was written and maintained by the Department of Energy, so it was extremely expensive.