



# Developing Lustre™ in the HPC Community

---

January 2011

*Eric Barton and John K. Dawson*

**Contents**

**Executive Summary ..... 3**

**Community development challenges ..... 3**

**Community development strategies ..... 4**

**Maintain an architecture roadmap ..... 4**

**Manage technical debt ..... 4**

**Maintain quality ..... 5**

**Rigorous gatekeeping ..... 6**

**Conclusion ..... 6**

## Executive Summary

Whamcloud, Inc. is committed to the continued success of Lustre as a vendor and hardware platform neutral Open Source file system that meets the most demanding needs of the HPC community for performance, scalability and reliability. This paper describes some of the challenges facing the community in maintaining Lustre as a production-quality file system while enhancing it to meet the demands of exascale computing.

The difficulty of meeting the IO requirements of HPC are driven by three factors:

- Moore's law relentlessly drives performance and scalability.
- Instability in the file system loses customers and stops development progress.
- Providing very high performance, shared, distributed access to data is an inherently difficult problem and solving it requires an extremely sophisticated solution.

The strategies to solve these problems with Lustre are to:

- Create a long-term development roadmap to minimize detours and reverses.
- Develop and deliver incrementally with rigorous quality assurance processes to minimize planning uncertainty and avoid prolonged stabilization efforts.
- Foster collaboration across the community while minimizing the expense and overhead of gatekeeping with strict requirements for accepting code contributions.

## Community development challenges

Lustre is the leading distributed file system for the High Performance Computing (HPC) community. More than half the sites in the Top 500 list use Lustre to meet the most demanding needs in production. While Moore's Law drives relentless increases in hardware scalability and performance at these sites, it in turn drives demand for Lustre to scale and deliver the increased performance potential of the hardware.

All this must be achieved without sacrificing stability. Stability is obviously important for end users since the filesystem is a mission-critical resource, downtime can be extremely expensive and data loss or corruption is unacceptable. Less obviously, instability effectively halts development. The quality of new code contributions cannot be assessed if they are being landed on a development base that already fails

regression tests. New landings therefore have to be halted while the base is stabilized since stabilization cost and effort quickly escalates if new landings are allowed to proceed.

Lustre encompasses a client filesystem, networking, server and storage, which must operate resiliently in an environment where component failures are routine. It is therefore a significant amount (roughly 250,000 lines) of complex software. Rapidly enhancing a software product as large and complex as this while simultaneously maintaining stability presents a significant engineering challenge. This challenge is compounded when the development community includes people from many different organizations with potentially conflicting needs.

## **Community development strategies**

The strategies described below are intended to address the challenges mentioned above. They aim to share knowledge, manage complexity, maintain quality and foster collaboration to allow the HPC community to move Lustre forward as fast as possible while maintaining reliability.

### **Maintain an architecture roadmap**

Detours and reverses in the development path are prohibitively expensive. It is essential that current developments achieve progress towards future goals if these costs are to be minimized. This can be done by:

- Agreeing a set of long-term requirements for HPC IO and storage that is broadly accepted by the HPC community.
- Maintaining an architecture roadmap that prioritizes and meets these requirements.

The architectural roadmap is not a release roadmap. The goal is not to identify which feature will be available in which upcoming Lustre release, but to define how the architecture for Lustre will evolve to meet HPC IO requirements for the next decade. It also captures logical dependencies between architectural features to ensure coherent developments that build on each other and maximize the pace of development while minimizing costs.

### **Manage technical debt**

Technical debt describes the situation when faster software development today is gained at the expense of additional cost tomorrow. Just like a monetary debt, technical debt accrues interest the longer it remains outstanding. Implementation shortcuts and incomplete restructuring invite instability by introducing additional complexity to handle the special cases they leave behind and the problem is compounded the longer it remains unaddressed.

Lustre, with its sheer size and complexity, constant pressure for development and requirement for version interoperability is therefore vulnerable to technical debt. This can be managed, and the pace of development maintained if the community is willing to acknowledge the hidden costs of technical debt and resource its management.

## Maintain quality

The cost of finding and eliminating defects escalates as development progresses from design through implementation to deployment. Strategies for maintaining quality include:

- Design documentation enabled knowledge sharing and peer review of features and enhancements. Formal inspections ensure designs fit with the architecture roadmap and that defects are detected and eliminated as early as possible.
- Consistent and rigorous application of coding standards to promote clear, unambiguous, readable code and ensure code can be maintained by someone other than the original developer.
- Code inspection and unit test to promote early detection of implementation defects. Gerrit, a browser-based code review tool, offers well-tested support for code reviews and is tightly integrated with git. It is well suited to use in development projects with contributors in different organizations and geographies and can be used to automate and manage development workflows. It can be integrated with bug tracking systems and automated test systems and can serve as a searchable repository of posted patches and review comments.
- Incremental changesets. Maintaining a large development as a sequence of much smaller "single idea" developments makes inspection easier and less fallible. Also regressions are easier to find if a large development is split up into testable increments.
- Shared test infrastructure. A standard test automation environment will make it easier to exploit test resources wherever they may be and ensure testing is performed consistently. A standard test results database format will promote sharing of quality information, help monitor convergence on stability and be used to guide test effort.
- Root cause defect analysis feeding back all the way to architecture and design. It is essential to ensure that as defects are found they are fixed at the right level and not simply avoided or obscured.

## Rigorous gatekeeping

The stability of the main shared development branch is crucial to determining the quality of new code and therefore maintaining the pace of development. The quality of new code contributions must therefore be evaluated and verified before they are allowed to land permanently on the main development branch and risk destabilizing it. This requires human intervention in the form of a gatekeeper. The gatekeeper must be an acknowledged expert, trusted by the development community to control code landings on the main development branch competently and fairly. This potentially represents a vast investment in time and resources by the gatekeeper and to do his or her job properly the gatekeeper needs help.

- “No surprises.” New community contributions will take time to vet properly. Landing delays can be minimized if the landing process is anticipated from the start of the development process and feature design, implementation and test processes are well understood prior to landing.
- Landing collateral. This is information that helps the gatekeeper decide whether to allow landing. It includes all the information generated to assure the quality of the development including design documentation, inspection reports, test plans and a test history.

## Conclusion

Maintaining stability is the key to ensuring not only that Lustre is fit to deploy in production, but also that Lustre development can keep up with the relentless pace of advances in scaling and performance forced by Moore’s Law. This depends on strict adherence to quality assurance processes at all stages of development from architecture through to implementation and landing. Since detours and reverses in development incur unacceptable cost, a long-term architecture road map is needed to ensure that current developments are consistent with long-term development goals. As Lustre development becomes more of a community process, these same processes and goals must be agreed, standardized and shared.