OpenSFS Lustre Development

SCHEDULE OF ARTICLES
FOR
CONTRACT NO. SFS-DEV-001

ARTICLE 1 – SCOPE OF WORK

A.The Contractor shall perform the OpenSFS Lustre Development further described in the Statement of Work incorporated in this Contract.

B.The Contractor shall address the projects described herein as well as the milestone-tasks, which are the units of work, and the milestones and deliverables described in the *Milestones & Deliverables* section of this document.

C.The Contractor shall furnish all personnel, supervision, materials, supplies, equipment, tools, facilities, transportation, testing, and other incidental items and services necessary for performance of the work. The Contractor shall deliver the materials, products, supplies, reports and residuals, as specified.

D.Acceptance of the work under this Contract shall be based on the Contractor's performance and completion of the work in consonance with high professional standards and compliance with the delivery and reporting requirements specified herein.

Project 1: ID Mapping in the Lustre 2.X Filesystem

To properly implement the ID Mapping feature, it must be made available to various modules within the Lustre filesystem server software stack. The MDT must be able to properly map IDs in order to maintain metadata. The OST module must be able to map IDs in order to maintain quotas. The MGS is the most sensible point of management for the map. Rather than duplicate this functionality within each module, it would be simpler to add an ID Mapping kernel module to ensure consistency of the map across the cluster, using the ptlrpc system to communicate changes to the map from the MGS to the other modules.

Subproject 1.1: ID Mapping Kernel Module

Unlike Contractor’s current mapping model, the mapping model proposed here will consolidate nid ranges into clusters, which will represent client machines that share the same UID/GID space. Like Contractor’s current model, the map will consist of a forest of binary trees with cluster as the root of each tree. This phase will focus on the kernel module that maintains the NID to Cluster and Cluster to UID/GID maps on the MGS node and the OSS and MDS nodes. The maps should be versioned to ensure that only updates are transferred between the canonical map on the MGS node and the cached copies on the MDS and OSS nodes. The latest version of the map should be written to disk on the MGS. Changes to the maps should be made only on the MGS. The command line function lctl will be altered to allow administration of the canonical map on the MGS.

Subproject 1.1 will provide for the construction and management of the map and not the synchronization of that canonical map with the cached copies.

How we will minimize Technical Debt:

Operations on and storage of the UID/GID map will be performed by functions contained in a separate kernel module. This is done to reduce redundancy and complexity, as those functions will be required by multiple existing Lustre subsystems.

The addition of mapping management features to lctl should be sufficiently segregated from other functions so as to not cause any negative interactions.

The memory footprint of ID mapping in a tree structure is expected to be 64 bytes per ID, but may vary depending on implementation issues. Constant limits could be placed on the size of the map to prevent memory overrun without placing unreasonable limitations on the number of clients and users contained within the map.

Subproject 1.2: Map Synchronization

This work will keep the cached maps on he MDS and OSS nodes in sync with the canonical map on the MGS node. The design of this phase will borrow strongly from Whamcloud's Imperative Recovery Target Status Table model.

This phase of development will include the locking and callbacks required to ensure that the maps are properly loaded on the MGS and MDS/OSS nodes upon file system startup, read lock grants on the map tables, and callbacks when the maps are updated.

How we will minimize Technical Debt:

The bulk of the work in this phase will be to increase the base functionality provided by the ID Mapping kernel module created in Subproject 1.1. Enhancements to the kernel module will produce proper results from the load and callback, however additional work must be done on the MGS and the MGC to provide callback functions for handling changes to the map.

Changes will be segregated and the path to new code will be taken only if ID Mapping is enabled for the file system.

Subprojects 1.3: Map Synchronization

This work will ensure that if an ID Map exists on the MDS or OSS, the mdt and the osdfilter modules will properly map the nid to cluster when a client connects. That mapping will then determine the proper client ID to file system ID mapping table that should be used.

Additionally, operations that require UIDs and GIDs (such as ACLs and quotas) will be altered to return expected results for the mapped environment. Because these operations already work in the GSSAPI environment, altering ACLs to properly handle multiple remote UIDs mapping to a site-specific nobody user should be the primary challenge.

The methods used to provide an ID Map to the osdfilter and mdt module currently implemented via the GSSAPI mechanism will be refined to reduce their memory footprint and increase their efficiency by replacing the individual ID Map linked lists that are kept per client with a shared map, implemented as a binary tree.

How we will minimize Technical Debt:

The idmap functionality that is used by GSSAPI will be refined to reduce its cost, while the interface will remain unaltered to ensure that GSSAPI will continue to work correctly, albeit in a more efficient manner.

Project 2: Shared Key Authentication and Encryption in Lustre 2.X

To allow secure Lustre connections without the use of a Kerberos infrastructure involves extending the existing GSSAPI security protocol that is in place in the Lustre 2.X code tree to allow an independent security model. GSSAPI is designed to be an extensible library that allows for various security models. Currently, in the GSSAPI implementation used within Lustre, only Kerberos is included. To allow for shared key authentication and encryption at the client node level, we need to extend GSSAPI to include shared key support, and add code to Lustre that uses the additional GSSAPI mechanisms by supporting additional “security flavors”.

Subproject 2.1: Additional GSSAPI Mechanisms

This subproject focuses on producing two additional GSSAPI mechanisms to provide authentication and encryption to GSSAPI enabled resources. The additional modules should include a NULL mechanism (or simple XOR mechanism) with which we can easily test the Lustre GSSAPI interface functionality.

The second should be a variation of the first that (rather than use a NULL or XOR algorithm) uses a strong shared key encryption algorithm.

How we will minimize Technical Debt:

Additional security mechanisms should be self-contained, and only the build configuration of the GSSAPI code should need to be updated to inform the objects about the new functions available.

Subproject 2.2: Additional Lustre Security Flavors

This subproject focuses on allowing Lustre to access the additional GSSAPI security mechanism through the existing procedure of selecting a security flavor. This should allow the administrator of the

Lustre filesystem to choose either authentication and/or encryption. Existing kernel implementations of the encryption algorithms will be used.

Shared keys should be available to Lustre via the Linux keychain mechanism.

How we will minimize Technical Debt:

Adding additional security mechanisms will allow the current Lustre code to simply use its current GSSAPI call out routines to access the additional GSSAPI security mechanisms. This addresses the issue of adding additional incompatible security code to Lustre proper, as it pushes these change out to the GSSAPI mechanisms.

Subproject 2.3: Key Management

Command line utilities for provisioning shared keys and making them accessible to Lustre for use will be developed. This functionality will be folded into lctl if appropriate.

How we will minimize Technical Debt:

The command line to to update keychain with shared keys should be an extension to lctl, but be sufficiently segregated from other functions to not cause any conflicts.

Project Approach

Technical Approach

Contractor shall use a rigorous development process, which focuses on the early processes in the software development lifecycle (SDLC) to properly define the requirements and solutions early in the process. By focusing early in the project on the Scope Clarification, Solution Architecture, and High-Level Designs (HLD), Contractor shall define proper solutions early, eliminate defects early, and keep OpenSFS better informed throughout the process.

Management Approach

This section identifies the standard processes used when developing any of the subprojects, and will be used for each of the subprojects listed above. If there is a planned variation to these steps, for example when a Solution Architecture document is not necessary, this is explicitly identified below.

Project Milestone Detailed Overview

1. **Scope Statement:** Contractor shall provide a brief (2-3 pages) summary of their understanding of the problem statement and resulting project scope to be reviewed by the OpenSFS Project

Approval Committee (PAC) and submitted for approval or rejection by the Contract Administrator (CA). This statement shall include:

a. Problem statement
b. Statement of high-level project requirements/goals to be satisfied
c. Statement of in-scope and out-of-scope work for project
d. Statement of project assumptions or constraints
e. Statement of key deliverables and milestones

2. **Solution Architecture:** Contractor shall provide a document that outlines requirements, use cases, and a solution framework in an effort to address items defined in the scope statement. The Solution Architecture shall be reviewed by the PAC and submitted for approval or rejection by the CA. This document shall include:
    a. Identification of requirements that address the subproject requirement
    b. Defined list of use cases for Test Plan
    c. Defined list of Acceptance Criteria-the acceptance criteria will be defined for the entire subproject (excluding the Delivery task) and are measured during the Demonstration Milestone.
    d. Proposed high-level solution that addresses the subproject requirements

3. **High-Level Design (HLD):** Contractor shall document a recommended solution that addresses the subproject requirements. This document shall describe how the solution will work including basic protocol structures as applicable. The proposed solution shall be documented with the elements below and reviewed by the PAC and submitted for approval or rejection by the CA:
    a. Description of the solution elements/implementation components and how they will work
    b. Explanation of why/how the proposed solution will address the subproject requirements
    c. Identify any risks or unknowns with the proposed solution
    d. Define API and protocol changes, if applicable
    e. Estimation of engineering and unit testing effort needed to complete the solution
    f. Prototype code for the solution, if complexity requires

4. **Implementation**: Contractor shall complete implementation and unit testing for the approved solution. Contractor shall regularly report feature development progress including progress metrics at project meetings and engineers shall share interim unit testing results as they are available. OpenSFS at its discretion may request a code review. Completion of the implementation phase shall occur when the agreed to solution has been completed up to and including unit testing and this functionality can be demonstrated on a test cluster. Code Reviews shall include:
    a. Discussion led by Contractor engineer providing an overview of Lustre source code changes

b. Review of any new unit test cases that were developed to test changes

5. **Demonstration** Upon functional completion of the feature, Contractor shall demonstrate the appropriate functionality of the subproject. This shall be done through execution of test cases designed to prove the acceptance criteria defined during the Solution Architecture. Demonstration specifics will be defined and mutually agreed to for each subproject in the scope and architecture phases.
    a. Functional Test Plan: Contractor shall develop and recommend a functional test plan, as defined by OpenSFS, designed to demonstrate the functional completeness of the feature. The results of functional testing with supporting documentation will be presented to OpenSFS for review.
    b. Performance Test Execution: Contractor shall define and recommend a set of performance tests as defined by OpenSFS to document the performance characteristics for performance related features. Contractor shall execute these tests and present results of these tests to OpenSFS for review. OpenSFS shall provide adequate test platforms when scale is necessary for performance testing as recommended by Contractor and defined by OpenSFS.

6. **Delivery**: Contractor shall develop against the canonical Lustre tree. All features and fixes from Contractor will be submitted to the community for inclusion in the canonical Lustre tree. All software development within the scope of this agreement shall be conducted in Lustre repository that is open to the community. Contractor's completion of this milestone shall be based on Contractor's integration of the project development into the Lustre tree as evidenced by the code's landing into the Master tree and eventual integration of these features into a standard supported release.

Acceptance by OpenSFS of Contractor's Delivery milestone shall be based on Contractor's integration of the project development branch into the canonical Lustre tree and integration of these features into a standard supported release. This release will then be tested within four (4) weeks or 20 business days (excluding the union of the Contractor's holiday schedule and that of the Contract Administrator), by OpenSFS (performance, stability, feature set compliance) for final acceptance or rejection. No written response from OpenSFS shall constitute implicit acceptance and approval. Lustre gatekeeper will land the project branch to the tree following internal landing practices and policies.

Project Approval Committee

OpenSFS shall designate a Project Approval Committee (PAC), comprised of community members that shall have final and documented authority for project-level decision making. The purpose of the PAC is to both facilitate clear decision-making throughout the project and provide an avenue for community collaboration during the project. Timely decision-making will support the advancement of projects and

ensure that OpenSFS is well informed about project milestone progress. The OpenSFS board-appointed Technical Representative shall serve as committee chair of the PAC and shall have final authority on all decisions of the PAC subject to the discretion of the OpenSFS board.

OpenSFS shall openly declare the membership of this committee and shall only change this membership during the course of the project with advance notification to Contractor.

Milestone-Task Approvals

A milestone-task is defined as the work or effort necessary to complete a specific project milestone, as described in *Project Milestone Detailed Overview* sub-section within this document, and is a sub-component or step along the way to completing a sub-project or project within the overall contract.

Approval will be required from the OpenSFS Contract Administrator in writing (email is acceptable) before Contractor starts work on a milestone-task.

Written approval will be required from the Open SFS Contract Administrator at the end of each milestone-task, for Contractor to invoice for the milestone. Email approval is acceptable.

For all requested approvals except the Delivery milestone, the Contract Administrator, will have 2 weeks (10 business days excluding the union of the Contractor's holiday schedule and that of the Contract Administrator) from work submission or decision request to provide feedback and render a decision. No response from OpenSFS shall constitute implicit acceptance and approval. If at a later time there is a dispute by OpenSFS, a separate resolution meeting will be held between OpenSFS PAC and Contractor's Senior Leadership.

Approval of the Delivery milestone will follow the procedure described in the Project Milestone Detailed Overview.

Subproject Termination

Either party reserves the right to terminate any subproject at any time with thirty (30) days written notice. In the event of termination of a subproject the Contractor shall terminate work and prorated payment for the percentage of work completed by date of termination shall be made, if applicable, in accordance with the Terms and Conditions. Milestone will be considered complete once completion is recommended in writing (email is acceptable) by the PAC and approved by the Contract Administrator.

Project Planning

Projects contained within this SOW are comprised of Research and Development engineering effort and as such the solutions are not necessarily straightforward or easy to determine. Contractor shall make best efforts to define appropriate target delivery dates based on information available.

With the completion of each milestone, Contractor shall deliver updated estimates for both interim milestone delivery and effort estimates using data and information acquired during the process of the project to refine estimates.

If, during the course of a Research and Development project, information is discovered that was previously unknown and significantly complicates the solution, Contractor shall present this new evidence to OpenSFS with a list of proposed alternate solutions. Some solutions may become so large as to not be able to be completed under the fixed price of the contract. In this event, Contractor shall offer alternate options from which OpenSFS may select that can be completed within the bounds of the contract.

Reporting

Contractor shall submit a monthly progress report in a format as provided by Contractor and reasonably acceptable by OpenSFS no less than two business days after the last day of each month highlighting progress made on each deliverable. Contractor shall provide documentation of completion as defined in Project Milestones Overview for each deliverable.

Project Meetings

Weekly project calls shall be held with participation from both Contractor and OpenSFS project team members. The purpose of this meeting is for project team members to gather to discuss progress against plan. OpenSFS attendance shall include at least one PAC member. Contractor attendance to include Senior Engineer assigned to project, Project Manager assigned to project, and any relevant development engineers depending on topic(s) of discussion. Contractor shall provide a standard project review template as recommended by Contractor and defined by OpenSFS to include progress since the last meeting, current activities, risks and issues. Meeting minutes shall be recorded at all project meetings and stored as project artifacts.

Software Licensing and Source Code

All software developed under this contract shall be submitted under the terms of the OpenSFS contribution agreement.

Lustre Community Development

Contractor shall submit all GPL Lustre software changes (including test cases) to the community canonical Lustre code base. It is at community's discretion to incorporate Contractor's changes into the canonical Lustre tree. Contractor shall maintain a separate code repository and make this available to the broader Lustre community.

OpenSFS Lustre Development

Lustre Software Enhancements

All changes to Lustre software shall be made against a production release of Lustre 2.x for this proposal. Changes made to Lustre as part of this contract may break interoperation between Lustre 2.x servers and Lustre 1.8.x clients. While code enhancements moving forward will not be backported to earlier versions, code enhancements will interoperate with the release one minor revision behind though previous versions may not be able to utilize the features developed. No development or changes will be made against the Lustre 1.8.x or 1.6.x code base. Contractor shall independently test and compile all modifications against a production release of Lustre (Production Lustre Release + Contractor patches).

Vendor Neutrality

All software shall be designed and developed to be broadly applicable to the entire Lustre community. Contractor shall not design or implement to benefit or exploit a particular hardware configuration. The Lustre software has always been designed to work on a wide variety of vendor server and storage platforms; Contractor shall not change this fundamental Lustre design principle.

OpenSFS Furnished Access and Hardware

OpenSFS or its constituent members shall provide access to a test cluster (configuration to be recommended by Contractor and defined by OpenSFS) that will be available throughout the duration of Projects 1 & 2. Contractor shall be notified of any changes to this test cluster throughout the duration of Projects 1 & 2.