# High level design of an SNS client module for C2 T1

By Nikita Danilov <nikita.danilov@clusterstor.com>
Date: 2010/07/29
Revision: 1.0

---

[Text in square brackets and with a light-green background is a commentary explaining the structure of a design document. The rest is a fictional design document, used as a running example.]

This document presents a high level design (HLD) of an SNS client module from C2 T1. The main purposes of this document are: (i) to be inspected by C2 architects and peer designers to ascertain that high level design is aligned with C2 architecture and other designs, and contains no defects, (ii) to be a source of material for Active Reviews of Intermediate Design (ARID) and detailed level design (DLD) of the same component, (iii) to serve as a design reference document.

Inspection trail of this specification is here [4].

The intended audience of this document consists of C2 customers, architects, designers and developers.

## 0. Introduction

[This section succinctly introduces the subject matter of the design. 1--2 paragraphs.]

[The following color marking is used in this document: incomplete or todo item, possible design extension or future directions.]
SNS client component interacts with Linux kernel VFS and VM to translate user application IO operation requests into IO FOPs sent to the servers according to SNS layout.

See companion SNS server HLD [5] specification.

## 1. Definitions

[Definitions of terms and concepts used by the design go here. The definitions must be as precise as possible. References to the C2 Glossary are permitted and encouraged. Agreed upon terminology should be incorporated in the glossary.]

The following terms are used to discuss and describe SNS client:

- *IO operation* is a read or write call issued by a user space application (or any other entity invoking C2 client services, *e.g.*, loop block device driver or NFS server). Truncate is also, technically, IO operation, but truncates are beyond the scope of this document;
- *network transfer* is a process of transmitting operation code and operation parameters, potentially including data pages, to the corresponding data server, and waiting for server reply. Network transfer is implemented as a synchronous call to C2 rpc service;
- an IO operation consists of *(IO) updates*, each accessing or modifying file system state on a single server. Updates of the same IO operation are *siblings*.

Definitions from the Parity De-clustering Algorithm HLD [3] are incorporated by reference.

## 2. Requirements

[This section enumerates requirements collected and reviewed at the Requirements Analysis (RA) and Requirements Inspection (RI) phases of development. References to the appropriate RA and RI documents should go here. In addition this section lists architecture level requirements for the component from the Summary requirements table and appropriate architecture documentation.]

- [r.sns-client.nocache]: T1 client does not cache data because (i) this simplifies its structure and implementation, (ii) T1 exports block device interface and block device users, including file systems, do caching internally;

- [r.sns-client.norecovery]: recovery from node or network failure is beyond the scope of the present design;

The following requirements from the Summary Requirements Table [1] are relevant for SNS client:

- [R.C2.IO.DIRECT]: direct-IO is supported.

For completeness and to avoid confusion, below are listed non-requirements (i.e., things that are out of scope of this specification):

- resource management, including distributed cache coherency and locking;
- distributed transactions;
- recovery, replay, resend;
- advanced layout functionality:
  - layout formulae;
  - layouts on file extents;
  - layout revocation;
- meta-data operations, including file creation and deletion;
- security of IO;
- data integrity, fsck;
- availability, NBA.

## 3. Design highlights

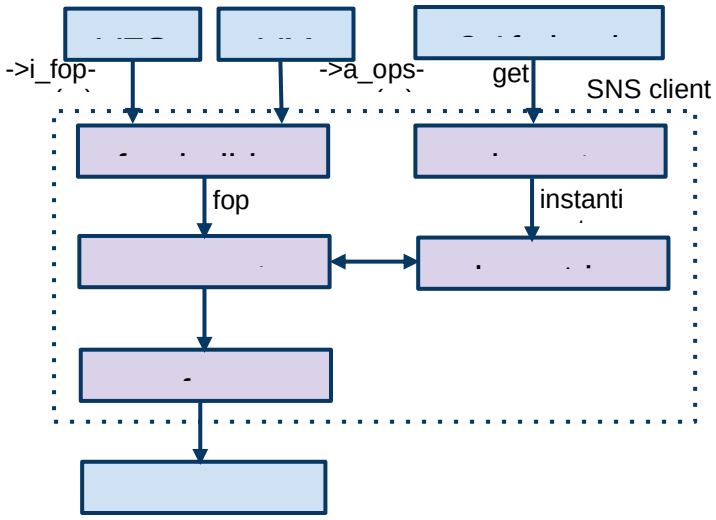
[This section briefly summarises key design decisions that are important for understanding of the functional and logical specifications and enumerates topics that a reader is advised to pay special attention to.]

The design of C2 client will eventually conform to the Conceptual Design [0]. As indicated in the Requirements section, only a small part of complete conceptual design is considered at the moment. Yet the structure of client code is chosen so that it can be extended in the future to conform to the conceptual design.

## 4. Functional specification

[This section defines a functional structure of the designed component: the decomposition showing *what* the component does to address the requirements.]

The following diagram illustrates interaction between SNS client and other components:




External SNS client interfaces are standard Linux `file_operations` and `address_space_operations` methods. VFS and VM entry points create fops describing the operation to be performed. The fop is forwarded to the request handler. At the present moment, request handler functionality on a client is trivial, in the future, request handler will be expanded with generic functionality, see Request Handler HLD [2]. A fop passes through a sequence of state transitions, guided by availability of resources (such as memory and, in the future, locks) and layout io engine, and staged in the fop cache until rpcs are formed. Current rpc formation logic is very simple due to no-cache requirement. rpcs are sent to their respective destinations. Once replies are received, fop is destroyed and results are returned to the caller.

## 5. Logical specification

[This section defines a logical structure of the designed component: the decomposition showing *how* the functional specification is met. Subcomponents and diagrams of their interrelations should go in this section.]

### 5.A. fop builder, NRS and request handler

A fop, representing IO operation is created at the VFS or VM entry point[1]. The fop is then passed to the dummy NRS[23], that immediately passes it down to the request handler. The request handler uses file meta-data to identify the layout and calls layout IO engine to proceed with IO operation.

[1] [u.FOP] ST

5.B. Layout formula

Layout formula generates a parity de-clustered file layout for a particular file, using file id (fid) as an identifier[2]. See Parity De-clustering Algorithm HLD [3] for details. At the moment, c2t1fs supports single file with fid supplied as a mount option.

### 5.C. Layout IO engine

Layout IO engine takes as an input layout and a fop (including operation parameters such as file offsets and user data pages). It creates sub-fops[3] for individual updates using layout[67] for data[4] , based on pool objects[5]. Sub-fops corresponding to the parity units reference temporarily allocated pages[6], released (under no-cache policy) when processing completes.

### 5.D. rpc

Trivial caching policy is used: fops are accumulated[7] in the staging area while IO operation is being processed by the request handler and IO layout engine. Once operation is processed, staged area is un-plugged[8] fops are converted into rpcs[9] and rpcs are transferred to their respective target servers. If IO operation extent is larger than parity group, multiple sibling updates on a given target server are batched together[10].

## 5.1. Conformance

[For every requirement in the Requirements section, this sub-section explicitly describes how the requirement is discharged by the design. This section is part of a requirements tracking mechanism, so it should be formatted in some way suitable for (semi-)automatic processing.]

---

[2][u.LAYOUT.PARAMETRIZED] ST

[3][u.FOP.SNS] ST

[4][u.LAYOUT.DATA] ST

[5][u.LAYOUT.POOLS] ST

[6][u.lib.allocate-page]

[7][u.fop.cache.add]

[8][u.fop.cache.unplug]

[9][u.fop.rpc.to]

[10][u.FOP.BATCHING] ST

- [r.sns-client.nocache]: holds per caching policy described in the rpc sub-section.

- [r.sns-client.norecovery]: holds obviously;

- [R.C2.IO.DIRECT]: no-caching and 0-copy for data together implement direct-io.

### 5.2. Dependencies

[This sub-section enumerates other system and external components the component depends on. For every dependency a type of the dependency (uses, generalizes, *etc.*) must be specified together with the particular properties (requirements, invariants) the design depends upon. This section is part of a requirements tracking mechanism.]

- layout:
  - [u.LAYOUT.SNS] ST: server network striping can be expressed as a layout
  - [u.LAYOUT.DATA] ST: layouts for data are supported
  - [u.LAYOUT.POOLS] ST: layouts use server and device pools for object allocation, location, and identification
  - [u.LAYOUT.PARAMETRIZED] ST: layouts have parameters that are substituted to perform actual mapping
- fop:
  - [u.fop] ST: C2 uses File Operation Packets (FOPs)
  - [u.fop.rpc.to]: a fop can be serialized into an rpc
  - [u.fop.nrs] ST: FOPs can be used by NRS
  - [u.fop.sns] ST: FOP supports SNS
  - [u.fop.batching] ST: FOPs can be batched in a batch-FOP
  - [u.fop.cache.put]: fops can be cached
  - [u.fop.cache.unplug]: fop cache can be de-staged forcibly
- NRS:
  - [u.nrs] ST: Network Request Scheduler optimizes processing order globally
- misc:
  - [u.io.sns] ST: server network striping is supported
  - [u.lib.allocate-page]: page allocation interface is present in the library

### 5.3. Security model

[The security model, if any, is described here.]

Security is outside of scope of the present design.

### 5.4. Refinement

[This sub-section enumerates design level requirements introduces by the design. These requirements are used as input requirements for the detailed level design of the component. This sub-section is part of a requirements tracking mechanism.]

Detailed level design specification should address the following:

- concurrency control and liveness rules for fops and layouts;
- data structures for mapping between layout and target objects in the pool;
- instantiation of a layout formula;
- relationship between fop and its sub-fops: concurrency control, liveness, ownership.

## 6. State

[This section describes the additions or modifications to the system state (persistent, volatile) introduced by the component. As much of component behavior from the logical specification should be described as state machines as possible. The following sub-sections are repeated for every state machine.]

State diagrams are part of the detailed level design specification.

## 7. Use cases

[This section describes how the component interacts with rest of the system and with the outside world.]

### 7.1. Scenarios

[This sub-section enumerates important use cases (to be later used as seed scenarios for ARID) and describes them in terms of logical specification.]

| Scenario | [usecase.sns-client-read] |
|---|---|
| Relevant quality attributes | usability |
| Stimulus | an incoming read operation request from a user space operation |
| Stimulus source | a user space application, potentially meditated by a loop-back device driver |
| Environment | normal client operation |
| Artifact | call to VFS ->read() entry point |
| Response | a fop is created, network transmission of operation parameters to all involved data servers is started as specified by the file layout, servers place retrieved data directly in user buffers, once transmission completes, the fop is destroyed |
| Response measure | no data copying in the process |
| Questions and issues | |

| Scenario | [usecase.sns-client-write] |
|---|---|

| | |
|---|---|
| Relevant quality attributes | usability |
| Stimulus | an incoming write operation request from a user space operation |
| Stimulus source | a user space application, potentially meditated by a loop-back device driver |
| Environment | normal client operation |
| Artifact | call to VFS ->write() entry point |
| Response | a fop is created, network transmission of user supplied data is started as specified by the file layout, once transmission completes, the fop is destroyed |
| Response measure | no data copying in the process |
| Questions and issues | |

[UML use case diagram can be used to describe a use case.]

### 7.2. Failures

[This sub-section defines relevant failures and reaction to them. Invariants maintained across the failures must be clearly stated. Reaction to Byzantine failures (*i.e.*, failures where a compromised component acts to invalidate system integrity) is described here.]

Under [r.sns-client.norecovery] requirement failure handling is very simple: any error (memory allocation failure, error returned from a call to hosting kernel services, network trasmission error, error returned by one of the target servers, *etc.*) aborts operation, releases all resources and returns the error code to the caller. No attempts to maintain consistency between target servers are made.

## 8. Analysis

### 8.1. Scalability

[This sub-section describes how the component reacts to the variation in input and configuration parameters: number of nodes, threads, requests, locks, utilization of resources (processor cycles, network and storage bandwidth, caches), *etc.* Configuration and work-load parameters affecting component behavior must be specified here.]

No scalability issues are expected in this component. Relatively little resources (processor cycles, memory) are consumed per byte of processed data.

With large number of concurrent IO operation requests, scalability of layout, pool and fop data structures might become a bottleneck (in the case of small file IO initially).

## 9. Deployment

### 9.1. Compatibility

[Backward and forward compatibility issues are discussed here. Changes in system invariants (event

ordering, failure modes, *etc.*)]

#### 9.1.1. Network

No issues at this point.

#### 9.1.2. Persistent storage

The design is not concerned with persistent storage manipulation.

#### 9.1.3. Core

[Interface changes. Changes to shared in-core data structures.]

No issues at this point. No additional external interfaces are introduced.

### 9.2. Installation

[How the component is delivered and installed.]

The SNS client module is a part of c2t1fs.ko kernel module and requires no additional installation. System testing scripts in c2t1fs/st must be updated.

## 10. References

[References to all external documents (specifications, architecture and requirements documents, *etc.*) are placed here. The rest of the document cites references from this section. Use Google Docs bookmarks to link to the references from the main text.]


[0] Outline of C2 core conceptual design
[1] Summary requirements table
[2] Request Handler HLD
[3] Parity De-clustering Algorithm HLD
[4] High level design inspection trail of SNS client
[5] SNS server HLD


## 11. Inspection process data

[The tables below are filled in by design inspectors and reviewers. Measurements and defects are transferred to the appropriate project data-bases as necessary.]

### 11.1. Logt

| | Task | Phase | Part | Date | Planned time (min.) | Actual time (min.) | Comments |
|---|---|---|---|---|---|---|---|
| Jay | sns-client | HLDINSP | 1 | | 120 | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Huang Hua | sns-client | HLDINSP | 1 | 2010-Aut.-03 | 120 | 120 | good. done |

## 11.2. Logd

| No. | Task | Summary | Reported by | Date reported | Comments |
|---|---|---|---|---|---|
| 1 | sns-client | | | | |
| 2 | sns-client | | | | |
| 3 | sns-client | | | | |
| 4 | sns-client | | | | |
| 5 | sns-client | | | | |
| 6 | sns-client | | | | |
| 7 | sns-client | | | | |
| 8 | sns-client | | | | |
| 9 | sns-client | | | | |