High level design of an SNS server server for C2 T1

By Nikita Danilov <nikita.danilov@clusterstor.com>
Date: 2010/07/30
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 server 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 [0].

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 server

See companion SNS client high level design specification [4].

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.]

Definitions from the Parity De-clustering Algorithm HLD [3] and SNS client HLD [4] 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-server.0-copy]: read and write IO operations with sufficiently large and properly aligned buffers incur no data copying in the complete IO path;

- [r.sns-server.conceptual-design]: sns server code is designed to be upward compatible with the C2 core conceptual design [5].

Refer to the SNS client HLD [4] for the list of topics currently omitted from SNS.

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.]

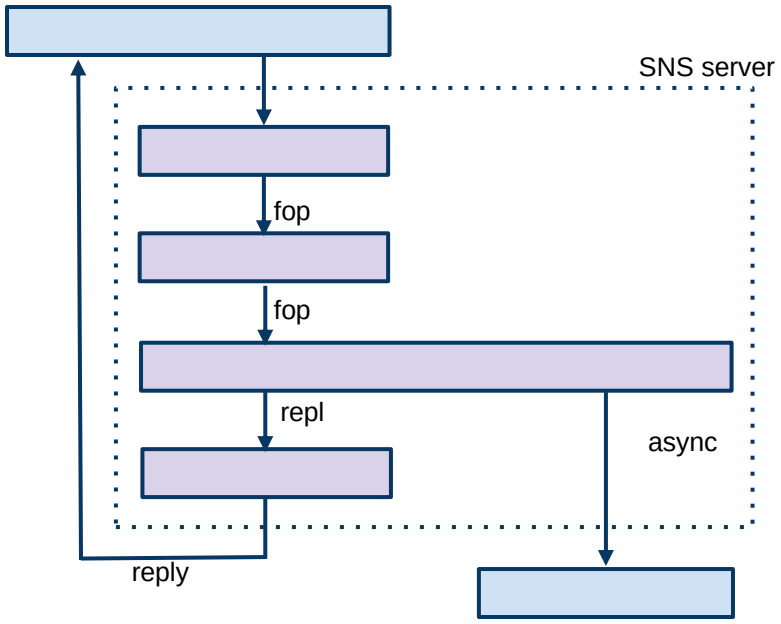
The design of C2 server will eventually conform to the Conceptual Design [0]. As indicated in the Requirements section, only a small part of complete conceptual design is considered at the moment. Yet the structure of client code is chosen so that it can be extended in the future to conform to the conceptual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

An important feature of proposed design is a concurrency model based on few threads and asynchronous request execution.

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 server and other components:



Server SNS component externally interacts with rpc sub-system and storage objects. A fop is constructed by fop builder from incoming rpc data. The fop is queued to the network request scheduler. A trivial scheduling policy is currently implemented: a queued fop is immediately forwarded to the request handler.

Request handler issues asynchronous storage transfer request. On completion, reply fop is constructed and queued. In the current design, replies are immediately sent back to client.

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 and NRS

A fop[1], representing IO operation is de-serialized from an incoming RPC[2]. The fop[3] is then passed to the dummy NRS[4] , 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.

5.B. request handler

Request handler uses a small pool of threads (*e.g.*, a thread per processor) to handle incoming fops. A fop is not bound to a thread. Instead, fop executes on an available thread until it has to wait (*e.g.*, to block waiting for storage transfer to complete). At that moment, fop places itself into a request handler supplied wait queue and *de-schedules* itself, freeing request handler thread for processing of another fop.

When event for which the fop is waiting happens, request handler retrieves the fop from the waiting queue and schedules it on some of its threads again.

Roughly speaking, the request handler implements a logic of a simple thread scheduler, using a small number of threads as processors to run fop execution on.

5.C. rpc

Rpc uses rdma to transfer data pages. Trivial reply caching policy is used: a reply fop is immediately serialized into rpc[5] and sent to the client.

5.1. Conformance

[For every requirement in the Requirements section, this sub-section explicitly describes how the

[1][u.FOP] ST

[2][u.fop.rpc]

[3][u.fop.nrs] st

[4][u.nrs] ST

[5][u.fop.rpc.to]

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.]

- [r.sns-server.0-copy]: use of rdma by rpc layer plus direct-io used by stob adieu provide 0-copy data path;

- [r.sns-server.conceptual-design]: the code structure outlines in the functional and logical specifications above can be easily extended to include other conceptual design components: resource framework, FOL, memory pressure handler, *etc.*

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.]

- fop:
  - [u.fop] ST: C2 uses File Operation Packets (FOPs)
  - [u.fop.rpc.from]: a fop can be buil
  - [u.fop.nrs] ST: FOPs can be used by NRS
  - [u.fop.sns] ST: FOP supports SNS
- NRS:
  - [u.nrs] ST: Network Request Scheduler optimizes processing order globally

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.]

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-server-write]
Relevant quality attributes	usability
Stimulus	an incoming write rpc
Stimulus source	an SNS client
Environment	normal file system operation
Artifact	call to registered read rpc call-back
Response	a fop is created, including write operation parameters, such as storage object id, offset and buffer size. User data are placed directly into rpc allocated buffers by RDMA. The fop is passed to the request handler through nrs. Asynchronous storage transfer (write) is started. On transfer completion, reply fop is formed, serialized to rpc and sent back to the originating client.
Response measure	no data copying in the process
Questions and issues	

Scenario	[usecase.sns-server-read]
Relevant quality attributes	usability
Stimulus	an incoming read rpc
Stimulus source	an SNS client
Environment	normal client operation
Artifact	call to registered write rpc call-back
Response	a fop is created, including read operation parameters. The fop os passed to the request handler through nrs. Pages to hold data are allocated and asynchronous read is started. On read completion, reply fop is allocated, referencing the pages with data. The reply is serialized in an rpc. The rpc is sent to the originating client, using RDMA to transfer data pages.
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.]

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.]

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

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.]

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

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-server	HLDINSP	1		120		
Huang Hua	sns-server	HLDINSP	1		120		

11.2. Logd

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