High level design of RPC Formation

By Anand Vidwansa <anand_vidwansa@xyratex.com>
Anup Barve <anup_barve@xyratex.com>
Date: 2011/04/08
Revision: 0.1

[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 a RPC formation module from RPC layer in Colibri C2 core. 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.

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.]

RPC formation component is part of RPC core component [0], which determines when an RPC can be created and what its contents will be. This helps in optimal use of network bandwidth and helps in maximizing the IO throughput.

RPC formation component acts as a state machine in the RPC core layer.

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.]

- fop, *file operation packet,* a description of file operation suitable for sending over network and storing on a storage device. File operation packet (FOP) identifies file operation type and operation parameters;
- rpc item is an entity containing FOP or other auxiliary data which is grouped into RPC object after formation.
- rpc, is a collection of rpc items.
- endpoint (not very good term, has connotations of lower levels) is a host on which service is being executed.
- session, corresponds to network connection between two services.
- max_rpcs_in_flight, is the number of RPC objects that can be in-flight (on-wire) per endpoint. This parameter will be handled as a resource by resource manager.
- max_message_size, is the largest possible size of RPC object.
- max_message_fragments, is the number of maximum disjoint buffer an RPC object can contain.
- urgent rpc item, is rpc item with zero deadline value.

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.rpc_formation.network_optimization]: Formation component should make optimum use of network bandwidth.
[r.rpc_formation.maximize_io_throughput]: Form RPCs in such a manner that will help maximize the IO throughput.
[r.rpc_formation.extensible]: RPC Formation component should be extensible enough to accommodate further changes so that a Lustre like algorithm could be plugged in [1].
[r.rpc_formation.send_policy]: Formation component will decide when to send an RPC to the output component in compliance with certain network parameters.

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.]

RPC formation component will incorporate a formation algorithm which will create RPC objects from RPC items taking into consideration various parameters.
The formation component will be implemented as a state machine with state transitions based on certain external events[1].

4. Functional specification

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

RPC formation component will employ a formation algorithm which will act on a cache of RPC items. The algorithm will decide the items to be selected from the cache and it will put them together in an RPC object. The formation algorithm will make sure that maximum size of an RPC object is limited by max_message_size so that it makes optimal use of network bandwidth[2]. The maximum number of disjoint buffers in an RPC object is limited by max_message_fragments. The max_message_fragments limit is enforced due to a limitation of RDMA to transfer only certain number of disjoint buffers in a request.

Multiple RPC items will be coalesced into one RPC item if intents are similar. This will be typically useful in case of vectored read/writes[3]. The individual rpc items being coalesced will be kept intact and an intermediate structure will be introduced to which all member rpc items will be tied. On receiving reply of the coalesced rpc item, callbacks to individual rpc items will be called which are part of the intermediate structure. The coalesced rpc item will put the constituent items in increasing order of file offset thus benefiting from sequential IO (reduced disk head seek). As far as possible, coalescing

[1][r.rpc_formation.extensible]

[2][r.rpc_formation.network_optimization]

[3][r.rpc_formation.maximize_io_throughput]

will be done within rpc groups. If not, coalescing can be done across groups. Coalescing is not done for items belonging to different update streams[0, 3].

The RPC item cache could contain bounded and unbounded items in the sense that they may or may not have session information embedded within them. The formation algorithm queries and retrieves the session information for unbounded items after formation is complete and before sending the RPC object on wire.[2, 3].

The RPC Formation algorithm will be triggered only if current rpcs in flight per endpoint are less than max_rpcs_in_flight[4]. It will also take care of distributed deadlocks.

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.]

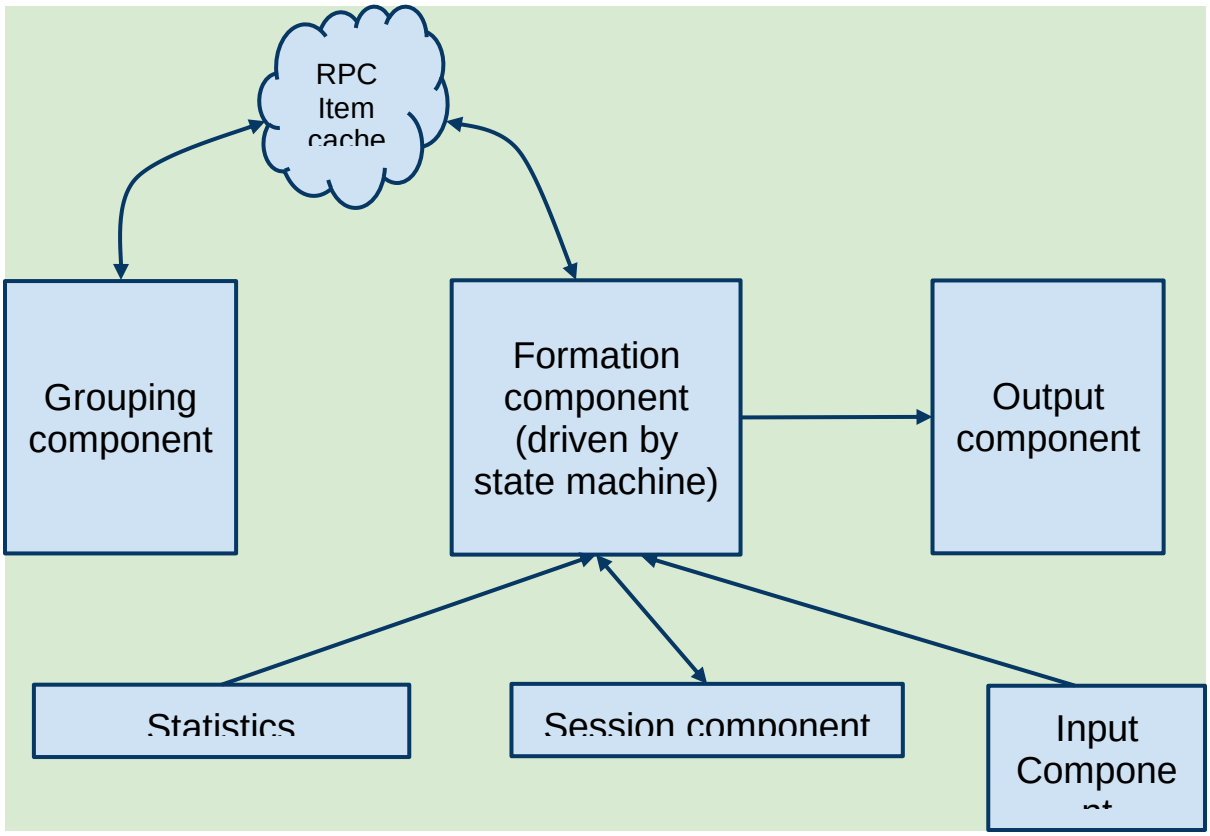

Fig. 1 - Interaction of RPC Formation with other components.

[4][r.rpc_formation.send_policy]

- Formation algorithm is implemented as cooperation of multiple policies like
    - network transport type policy (guided by max_message_size, max_message_fragments)
    - target service policy (guided by max_rpcs_in_flight)
    - item type policies: pack IO requests together in offset order, honoring rpc item deadlines, its priorities and sending rpc groups as one unit.
- The formation component is a state machine mostly driven by external events.
- When there are no events, the formation component is in idle state.
- Other components like RPC Grouping, Sessions and Input component post most of the events to the Formation component.
- The various external events are
    - Addition of item in rpc item cache.
    - Deletion of item from rpc item cache.
    - Change in parameter for an item from rpc item cache.
    - Reply received from network layer.
    - Deadline expired for item from rpc item cache.
- On triggering of events like mentioned above, corresponding state functions will be called which will carry out necessary actions.
- RPC Formation component uses an abstract data structure to gather information of rpc items from input cache. The details of this structure are part of Detailed Level Design.
- This abstract data structure gathers critical information like number of groups, number of rpc items in each group, possible candidates for coalescing, URGENT items &c.
- Formation algorithm will follow hints about rpc groups which are indications of more items following the current one and will not form a RPC object right away.
- The formation algorithm will consult this abstract data structure to make optimal rpc objects.
- On successful formation of an RPC object, it will be posted to the output component.
- Any deadline expiry event will trigger immediate formation of RPC object with whatever size RPC object that can be formed.
- If current rpcs in flight for an endpoint have already reached the number max_rpcs_in_flight for that endpoint, no more rpc formation is done until any replies from that endpoint are received and the current rpcs in flight number drop to less than max_rpcs_in_flight.
- The overpowering criteria is current rpcs in flight should be less than max_rpcs_in_filght. So, even if a timeout event is triggered for an rpc item but if current rpcs in flight have reached max_rpcs_in_flight, no formation will be done.

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.]

[r.rpc_formation.network_optimization]: Formation component takes care of amortizing the message as compared to the overhead of sending the message.
[r.rpc_formation.maximize_io_throughput]: IO requests (read, write) are coalesced instead of sending multiple requests.
[r.rpc_formation.extensible]: Formation algorithm is a simple state machine which can accommodate

changes as required. The abstract data structures used are also extensible to make future additions.
[r.rpc_formation.send_policy]: Formation component throttles the transmission of RPC objects so that only certain number of RPC objects exist on the wire between 2 endpoints. The size of RPC object is also restricted to a limit.

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.]

1. RPC Grouping component: needed for populating the rpc items cache.
    1. Grouping will put the items in the cache in a sorted manner corresponding to timeouts. This will help to ensure timely formation of needy rpc items.
    2. Grouping component will make one list per endpoint and hence there will be multiple lists in the rpc items cache(one per endpoint).
    3. raising events like addition of rpc item to cache, deletion of rpc item from cache, rpc item parameter changed.
    4. rpc core/grouping component should take care of attaching timer objects with rpc items to take care of deadlines.
2. RPC Sessions component: needed for getting sessions information for unbounded items.
3. RPC Input component: needed to raise events for an incoming reply.
4. RPC Output component: needed to keep track of current rpcs in flight per endpoint.
5. RPC Statistics component: needed to get/set various parameters crucial to RPC formation.

5.3. Security model

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

None.

5.4. Refinement

[This sub-section enumerates design level requirements introduced 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.]

1. The rpc items need to be sorted based on some criteria (timeouts) by the grouping layer.
2. Session component provides an API to get sessions information for an unbounded rpc item.

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.]

The RPC formation component is implemented as a state machine mostly driven by triggering of external events.

6.1. States, events, transitions

[This sub-section enumerates state machine states, input and output events and state transitions incurred by the events with a table or diagram of possible state transitions. UML state diagrams can be used here.]

Original State --> EVENTS | V	**WAITING[n]**	**UPDATING**	**FORMING**
Slot ready to send next item*	UPDATING	UPDATING	UPDATING
Reply Received*	FORMING	FORMING	FORMING
RPC item deadline expired*	FORMING	FORMING	FORMING
Slot becomes idle*	FORMING	FORMING	FORMING
Freestanding item added to session*	UPDATING	UPDATING	UPDATING
Unsolicited item added*	UPDATING	UPDATING	UPDATING
State Succeeded#	WAITING[n]	FORMING	WAITING[n]
State Failed#	WAITING[n]	WAITING[n]	WAITING[n]

* - external events
# - internal events

Table 1 - State Transition table and events.

State	Purpose
WAITING[n]	Formation is waiting for any event to trigger. During this state, rpc objects could be in flight. If current rpcs in flight [n] is less than max_rpcs_in_flight, state transitions to UPDATING, else keeps waiting till n is less than max_rpcs_in_flight.
UPDATING	Formation is updating its internal data structure by taking necessary locks.
FORMING	Core of formation algorithm. This state scans the rpc items cache and internal data structure to form an RPC object by cooperation of multiple policies.

Table 2. Description of each state in formation state machine.

Table 1 describes the flow of states on triggering of all events. The rows describe the states and there is a column per event. The resultant state is mentioned at the intersection of row and column. Table 2 describes the intent of each state in the formation state machine.

6.2. State invariants

[This sub-section describes relations between parts of the state invariant through the state modifications.]

6.3. Concurrency control

[This sub-section describes what forms of concurrent access are possible and what forms on concurrency control (locking, queuing, *etc.*) are used to maintain consistency.]

The internal abstract data structure will have locks for concurrency control. The rpc item cache should have some mechanism to guarantee concurrency control.
Formation algorithm makes no assumptions about threads and is purely driven by events.

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.]

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

Scenario	usecase.rpc_formation.add_new_rpc_item_to_cache
Relevant quality attributes	usability
Stimulus	RPC Grouping raises an event after adding rpc item to the rpc item cache.
Stimulus source	RPC item grouped as per its destination endpoint.
Environment	RPC processing.
Artifact	Formation state machine transitions into UPDATING_INTERNAL_DATA state.
Response	1. Update internal data. 2. Execute formation algorithm. 3. If RPC object is created, post it to output component. 4. Return to Idle state.
Response measure	internal data updated and if possible RPC object is formed and posted.

Questions and issues	None.

Scenario	usecase.rpc_formation.delete_rpc_item_from_cache
Relevant quality attributes	usability
Stimulus	RPC Grouping raises an event before deleting rpc item from the rpc item cache.
Stimulus source	RPC item cache is full and new item needs a slot to be placed in the cache OR rpc item being deleted as a result of an abort/cancel action.
Environment	RPC processing.
Artifact	Formation state machine transitions into REMOVING state.
Response	REMOVING state will remove the item from rpc object if present. And it will remove the data of rpc item from internal data structure.
Response measure	internal data updated and RPC object does not contain the deleted rpc item.
Questions and issues	None.

Scenario	usecase.rpc_formation.update_rpc_item_parameter
Relevant quality attributes	usability
Stimulus	RPC Grouping raises an event before updating rpc item from the rpc item cache.
Stimulus source	User input changes the rpc item parameter like endpoint, grouping id, priority, timeout &c.
Environment	RPC processing.
Artifact	Formation state machine transitions into REMOVING state.
Response	REMOVING state will remove the item from rpc object if present. And it will change the data of rpc item from internal data structure.
Response measure	internal data updated with new parameters.
Questions and issues	None.

Scenario	usecase.rpc_formation.reply_received

Relevant quality attributes	usability
Stimulus	RPC Input component sends event about receiving a reply RPC item.
Stimulus source	RPC Input component receives RPC object which contains the RPC item reply.
Environment	RPC processing.
Artifact	Formation state machine transitions into UPDATING state.
Response	UPDATING state will check if current number of rpcs in flight are less than max_rpcs_in_flight for the given endpoint. If yes, it will invoke state transition to CHECKING state.
Response measure	None.
Questions and issues	None.

Scenario	usecase.rpc_formation.deadline_expired
Relevant quality attributes	usability
Stimulus	The timer object from rpc item triggers a timeout event.
Stimulus source	RPC item timeout expired.
Environment	RPC processing.
Artifact	Formation state machine transitions into CHECKING state.
Response	CHECKING state make sure that given rpc item is included in formation.
Response measure	New RPC object is created with given rpc item.
Questions and issues	None.

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.]

Failures are mostly handled by retries. No failures are permanent and are not propagated to other components.

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.]

1. Colibri components use "non-blocking server model" where
    1. scheduler overhead is reduced by using state machines instead of context switching amongst threads.
    2. keeping the number of threads limited and proportional to number of CPU cores in the system instead of thread per request.
2. Similarly, formation algorithm is a state machine which does not use any threads of its own and relies only on incoming threads. The formation algorithm always tries to form RPC bounded by a threshold which is max_message_size. This ensures the scalability in terms of change in number of incoming rpc items.

8.2. Other

[As applicable, this sub-section analyses other aspects of the design, *e.g.*, recoverability of a distributed state consistency, concurrency control issues.]

None.

8.2. Rationale

[This sub-section describes why particular design was selected; what alternatives (alternative designs and variations of the design) were considered and rejected.]

The formation component uses a pre-algorithm-execution state (UPDATING state in Fig.2) which collates information about various properties of rpc items in the cache. The algorithm itself depends on data generated by this state. This helps to reduce the frequent scanning of rpc items from the cache and gives a big picture view to form RPC objects efficiently.

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

9.1.2. Persistent storage

9.1.3. Core

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

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] HLD of rpc layer core
[1] Architecture Review of rpc layer
[2] Sessions HLD
[3] Sessions DLD

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.]

HLDIT (see "Discussions" tab)