Abstract

The Reliable Server Pooling effort (abbreviated "RSerPool"), provides an application-independent set of services and protocols for building fault tolerant and highly available client/server applications. This document provides an overview of the protocols and mechanisms in the reliable server pooling suite.
Table of Contents

1. Introduction ................................................. 3
2. ASAP Overview ............................................. 5
   2.1. Pool Initialization .................................... 5
   2.2. Pool Entity Registration ................................. 5
   2.3. Pool Entity Selection .................................. 5
   2.4. Endpoint Keepalive ..................................... 6
   2.5. Failover Services ....................................... 6
      2.5.1. Cookie Mechanism .................................. 6
      2.5.2. Business Card Mechanism .......................... 6
      2.5.3. Failover Callback Mechanism ....................... 7
3. ENRP ......................................................... 7
   3.1. Initialization .......................................... 7
   3.2. Server Discovery ....................................... 7
   3.3. Server Pool Maintenance ................................. 8
4. Example Scenarios ............................................ 8
   4.1. Example Scenario Using RSerPool Resolution Service .... 8
   4.2. Example Scenario Using RSerPool Session Services .... 9
5. Security Considerations ...................................... 10
6. IANA Considerations ......................................... 10
7. Acknowledgements ........................................... 11
8. Normative References ....................................... 11
Authors’ Addresses ............................................. 11
Intellectual Property and Copyright Statements .................. 13
1. Introduction

The requirements for the Reliable Server Pooling effort are defined in RFC3237 [2]. The central idea of this architecture is to provide client applications ("pool users") with the ability to select a server (a "pool element") from among a group of servers providing equivalent service (a "pool"). The pool is accessed via an identifier called a "pool handle", which is a location-independent name separate from the IP address of any pool server.

The RSerPool architecture supports high-availability and load balancing by enabling a pool user to identify the most appropriate server from the server pool at a given time. The architecture is defined to support a set of basic goals:

- application-independent protocol mechanisms
- separation of server naming from IP addressing
- use of the end-to-end principle to avoid dependencies on intermediate equipment
- separation of session availability/failover functionality from application itself
- facilitate different server selection policies
- facilitate a set of application-independent failover capabilities
- peer-to-peer structure

The basic components of the Rserpool architecture are shown in Figure 1 below:
A server pool is defined as a set of one or more servers providing the same application functionality. The servers are called Pool Elements (PEs). Multiple PEs in a server pool can be used to provide fault tolerance or load sharing, for example. The PEs register into and deregisters out of the pool using the Aggregate Server Access Protocol ASAP [3].

Each server pool is identified by a unique byte string called the pool handle. The pool handle allows a mapping from the pool to a specific Pool Element located by its IP address and port. The pool handle is what is specified by the Pool User (PU) when it attempts to access a server in the pool, again using ASAP. Both IPv4 and IPv6 PE addresses are supported.

To resolve the pool handle to the address necessary to access a Pool Element, the PU consults an entity called the Endpoint haNdlespace Redundancy Protocol (ENRP) server. This server may be a standalone server supporting many PUs or a part of the PU itself, however it is envisioned that ENRP servers provide a fully distributed and fault-tolerant registry service using ENRP [4] to maintain synchronization of data concerning the pool handle mapping space.

This document provides an overview of the RSerPool protocol suite, specifically the Aggregate Server Access Protocol ASAP [3] and the
In addition to the protocol specifications, there is a common parameter format specification COMMON [5] for both protocols, as well as a security threat analysis SEC [6].

2. ASAP Overview

ASAP is a straight-forward implementation of a set of mechanisms identified as necessary for support of the creation and maintenance of pools of redundant servers. These mechanisms include:

- registration of a new server for the server pool
- deregistration of an existing server in the pool
- resolution of a pool ‘handle’ to a server or list
- liveness detection for servers in the pool
- failover mechanisms for handling server failure

2.1. Pool Initialization

Pools come into existence when a PE registers the first instance of the pool name. They disappear when the last PE deregisters. In other words, the starting of the first PE on some machine causes the creation of the pool when the registration reaches the ENRP server.

2.2. Pool Entity Registration

A new server joins an existing pool by sending a Registration message in ASAP indicating the ‘handle’ of the pool that it wishes to join, a pool identifier for itself (chosen randomly) information about it’s lifetime in the pool, and what transport protocols and selection policies it supports. The Registration message is sent to its Home ENRP server.

Similar procedures are applied to de-register itself from the server pool (or alternatively the server may simply let its previously state lifetime expire and be gracefully removed from the pool.

2.3. Pool Entity Selection

When an endpoint wishes to be connected to a server in the pool, this requires the resolution of a server ‘handle’ to the IP addresses of a server or list of servers in the pool. This process may involve a
number of policies for server selection, for which the RSerPool protocol suite supports a few simply defined policies and allows the use of external server selection input for more complex policies.

The endpoint generates a Handle Resolution message in ASAP and sends this to its home ENRP server to start the resolution process. The ENRP server resolves the handle based on its knowledge of pool servers and returns a Handle Resolution Response in ASAP.

2.4. Endpoint Keepalive

In order to maintain status information for members of the server pool, the ENRP server may audit the status of a particular pool element using an ASAP Keep Alive message. When received by the pool element, it responds by verifying its membership in the pool in an Ack message.

When a PE is found to be unreachable, for example, an endpoint conversing with the pool element finds that it can no longer be reached by its transport connection, the endpoint can also inform its home ENRP server by sending an Endpoint Unreachable message.

2.5. Failover Services

While maintaining application-independence, the RSerPool protocol suite provides some simple hooks for supporting failover of an individual session with a pool element. Generally, mechanisms for failover that rely on application state or transaction status cannot be supported without more specific knowledge of the application being supported. However, some simple mechanisms supported by RSerPool allow some level of failover that any application can use.

2.5.1. Cookie Mechanism

Cookies may optionally be generated by the ASAP layer and periodically sent from the PE to the PU. The PU only stores the last received cookie. In case of fail over the PU sends this last received cookie to the new PE. This method provides a simple way of state sharing between the PEs. Please note that the old PE should sign the cookie and the receiving PE should verify the signature. For the PU, the cookie has no structure and is only stored and transmitted to the new PE.

2.5.2. Business Card Mechanism

A PE can send a business card to its peer (PE or PU) containing its pool handle and guidance concerning which other PEs the peer should use for failover. This gives a PE a means of telling a PU what it
identifies as the "next best" PE to use in case of failure, which may be based on pool considerations, such as load balancing, or user considerations, such as PEs that have the most up-to-date state information.

2.5.3. Failover Callback Mechanism

TBD

3. ENRP

ENRP is used between ENRP servers in order to maintain a distributed, fault-tolerant real-time registry service. ENRP servers communicate with each other in order to exchange information such as pool membership changes, handlespace data synchronization, etc.

3.1. Initialization

Each ENRP server initially generates a 32-bit server ID that it uses in subsequent messaging and remains unchanged over the lifetime of the server. It then attempts to learn all of the other ENRP servers within the scope of the server pool, either using a pre-defined Mentor server or by sending out Presence messages on a well-known multicast channel to determine other ENRP servers from the responses and select one as Mentor.

It then requests the most current data about the pool handlespace from its Mentor server and unpacks received Handle Table Response messages into its local database.

It is then ready to provide ENRP services.

3.2. Server Discovery

PEs can now register their presence with the newly functioning ENRP server by using ASAP messages. They discover the new ENRP server after the server sends out an ASAP Server Announce message on the well-known ASAP multicast channel.

The PE may have a configured list of ENRP servers to talk to, in which case it will start to setup associations with some number of them and assign the first one that responds to it as its Home ENRP Server.

Alternatively it can listen on the multicast channel for a set period and when it hears an ENRP server, start an association. The first server it gets up can then become its Home ENRP Server.
3.3. Server Pool Maintenance

PE failure detection, keepalive, etc. TBD

4. Example Scenarios

4.1. Example Scenario Using RSerPool Resolution Service

RSerPool can be used in a ‘standalone’ manner, where the application uses RSerPool to determine the address of a primary server in the pool, and then interacts directly with that server without further use of RSerPool services. If the initial server fails, the application uses RSerPool again to find the next server in the pool.

For pool user ("client") applications, if an ASAP implementation is available on the client system, there are typically only three modifications required to the application source code:

1. Instead of specifying the hostnames of primary, secondary, tertiary servers, etc., the application user specifies a pool handle.

2. Instead of using a DNS based service (e.g. the Unix library function gethostbyname()) to translate from a hostname to an IP address, the application will invoke an RSerPool service primitive GETPRIMARYSERVER that takes as input a pool handle, and returns the IP address of the primary server. The application then uses that IP address just as it would have used the IP address returned by the DNS in the previous scenario.

3. Without the use of additional RSerPool services, failure detection and failover procedures must be designed into each application. However, when failure is detected on the primary server, instead of invoking DNS translation again on the hostname of a secondary server, the application invokes the service primitive GETNEXTSERVER, which performs two functions in a single operation.

   1. First it indicates to the RSerPool layer the failure of the server returned by a previous GETPRIMARYSERVER or GETNEXTSERVER call.

   2. Second, it provides the IP address of the next server that should be contacted, according to the best information available to the RSerPool layer at the present time (e.g. set of available pool elements, pool element policy in effect for the pool, etc.).
For pool element ("server") applications where an ASAP implementation is available, two changes are required to the application source code:

1. The server should invoke the REGISTER service primitive upon startup to add itself into the server pool using an appropriate pool handle. This also includes the address(es) protocol or mapping id, port (if required by the mapping), and pooling policy(s).

2. The server should invoke the DEREGISTER service primitive to remove itself from the server pool when shutting down.

When using these RSerPool services, RSerPool provides benefits that are limited (as compared to utilizing all services), but nevertheless quite useful as compared to not using RSerPool at all. First, the client user need only supply a single string, i.e. the pool handle, rather than a list of servers. Second, the decision as to which server is to be used can be determined dynamically by the server selection mechanism (i.e. a "pool policy" performed by ASAP; see ASAP [3]). Finally, when failures occur, these are reported to the pool via signaling present in ASAP [3] and ENRP [4], other clients will eventually know (once this failure is confirmed by other elements of the RSerPool architecture) that this server has failed.

4.2. Example Scenario Using RSerPool Session Services

When the full suite of RSerPool services are used, all communication between the pool user and the pool element is mediated by the RSerPool framework, including session establishment and teardown, and the sending and receiving of data. Accordingly, it is necessary to modify the application to use the service primitives (i.e. the API) provided by RSerPool, rather than the transport layer primitives provided by TCP, SCTP, or whatever transport protocol is being used.

As in the previous case, sessions (rather than connections or associations) are established, and the destination endpoint is specified as a pool handle rather than as a list of IP addresses with a port number. However, failover from one pool element to another is fully automatic, and can be transparent to the application:

The RSerPool framework control channel provides maintenance functions to keep pool element lists, policies, etc. current.

Since the application data (e.g. data channel) is managed by the RSerPool framework, unsent data (data not yet submitted by RSerPool to the underlying transport protocol) is automatically redirected to the newly selected pool element upon failover. If
the underlying transport layer supports retrieval of unsent data (as in SCTP), retrieved unsent data can also be automatically re-sent to the newly selected pool element.

An application server (pool element) can provide a state cookie (described in Section 2.5.1) that is automatically passed on to another pool element (by the ASAP layer at the pool user) in the event of a failover. This state cookie can be used to assist the application at the new pool element in recreating whatever state is needed to continue a session or transaction that was interrupted by a failure in the communication between a pool user and the original pool element.

The application client (pool user) can provide a callback function (described in Section 2.5.2) that is invoked on the pool user side in the case of a failover. This callback function can execute any application specific failover code, such as generating a special message (or sequence of messages) that helps the new pool element construct any state needed to continue an in-process session.

Suppose in a particular peer-to-peer application, PU A is communicating with PE B, and it so happens that PU A is also a PE in pool X. PU A can pass a "business card" to PE B identifying it as a member of pool X. In the event of a failure at A, or a failure in the communication link between A and B, PE B can use the information in the business card to contact an equivalent PE to PU A from pool X.

Additionally, if the application at PU A is aware of some particular PEs of pool X that would be preferred for B to contact in the event that A becomes unreachable from B, PU A can provide that list to the ASAP layer, and it will be included in A’s business card. (See Section 2.5.2).

5. Security Considerations

This document does not identify security requirements beyond those already documented in the ENRP and ASAP protocol specifications.

6. IANA Considerations

This document does not require additional IANA actions beyond those already identified in the ENRP and ASAP protocol specifications.
7. Acknowledgements

The authors wish to thank Maureen Stillman, Qiaobing Xie, Randall Stewart, and many others for their invaluable comments.

8. Normative References


Authors’ Addresses

Peter Lei
Cisco Systems, Inc.
955 Happfield Dr.
Arlington Heights, IL  60004
US

Phone: +1 773 695-8201
Email: peterlei@cisco.com