Network Working Group
Request for Comments: 5351
Category: Informational

P. Lei
Cisco Systems, Inc.
L. Ong
Ciena Corporation
M. Tuexen
Muenster Univ. of Applied Sciences
T. Dreibholz
University of Duisburg-Essen

September 2008

An Overview of Reliable Server Pooling Protocols

Status of This Memo

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

The Reliable Server Pooling (RSerPool) protocol suite is designed 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 protocols are currently targeted for Experimental Track.

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 the application itself
- facilitation of different server selection policies
- facilitation of a set of application-independent failover capabilities
- peer-to-peer structure

The basic components of the RSerPool architecture are shown in Figure 1 below:

```
                                           .......................
         ______          ______            .      +-------+      .
        / ENRP \        / ENRP \           .      |       |      .
        |Server| <----> |Server|<----------.----->|  PE 1 |      .
        \______/  ENRP  \______/  ASAP(1)  .      |       |      .
                           ^               .      +-------+      .
                           |               .                     .
                           | ASAP(2)       .     Server Pool     .
                           V               .                     .
                      +-------+            .      +-------+      .
                      |       |            .      |       |      .
                      |  PU   |<---------->.      |  PE 2 |      .
                      |       |  PU to PE  .      |       |      .
                      +-------+            .      +-------+      .
                                           .                     .
                                           .      +-------+      .
                                           .      |       |      .
                                           .      |  PE 3 |      .
                                           .      |       |      .
                                           .      +-------+      .
                                           .......................
```

Figure 1

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 de-register out of the pool at an entity called the Endpoint haNdlespace Redundancy Protocol (ENRP) server, using the Aggregate Server Access Protocol (ASAP) [RFC5352] (this association is labeled ASAP(1) in the figure).

Each server pool is identified by a unique byte string called the pool handle (PH). The pool handle allows a mapping from the pool to a specific PE located by its IP address (both IPv4 and IPv6 PE addresses are supported) and port number. The pool handle is what is specified by the Pool User (PU) when it attempts to access a server in the pool. To resolve the pool handle to the address necessary to access a PE, the PU consults an ENRP server using ASAP (this association is labeled ASAP(2) in the figure). The space of pool handles is assumed to be a flat space with limited operational scope (see RFC 3237 [RFC3237]). Administration of pool handles is not addressed by the RSerPool protocol documents at this time. The protocols used between the PU and PE are application-specific. It is assumed that the PU and PE are configured to support a common set of protocols for application layer communication, independent of the RSerPool mechanisms.

RSerPool provides a number of tools to aid client migration between servers on server failure: it allows the client to identify alternative servers, either on initial discovery or in real time; it also allows the original server to provide a state cookie to the client that can be forwarded to an alternative server to provide application-specific state information. This information is exchanged between the PE and PU directly, over the association labeled PU to PE in the figure.

It is envisioned that ENRP servers provide a fully distributed and fault-tolerant registry service. They use ENRP [RFC5353] to maintain synchronization of data concerning the pool handle mapping space. For PUs and PEs, the ENRP servers are functionally equal. Due to the synchronization provided by ENRP, they can contact an arbitrary one for registration/de-registration (PE) or PH resolution (PU). An illustration containing 3 ENRP servers is provided in Figure 2 below:

```
                          _______         ______
            ...          / ENRP \        / ENRP \          ...
          PEs/PUs  <---->|Server| <----> |Server|<---->  PEs/PUs
            ...     ASAP \______/  ENRP  \______/ ASAP     ...
                           ^                  ^
                           |                  |
                           |     / ENRP \     |
                           +---->|Server|<----+
                            ENRP \______/ ENRP
                                    ^
                                    | ASAP
                                    v
                                   ...
                                 PEs/PUs
                                   ...
```

Figure 2

The requirements for the Reliable Server Pooling framework are defined in RFC 3237 [RFC3237]. It is worth noting that the requirements on RSerPool in the area of load balancing partially overlap with grid computing/high-performance computing. However, the scope of both areas is completely different: grid and high-performance computing also cover topics like managing different administrative domains, data locking and synchronization, inter-session communication, and resource accounting for powerful computation services, but the intention of RSerPool is simply a lightweight realization of load distribution and session management. In particular, these functionalities are intended to be used on systems with small memory and CPU resources only. Any further functionality is not in the scope of RSerPool and can -- if necessary -- be provided by the application itself.

This document provides an overview of the RSerPool protocol suite, specifically the Aggregate Server Access Protocol (ASAP) [RFC5352] and the Endpoint Handlespace Redundancy Protocol (ENRP) [RFC5353]. In addition to the protocol specifications, there is a common parameter format specification [RFC5354] for both protocols, a definition of server selection rules (pool policies) [RFC5356], as well as a security threat analysis [RFC5355].

2. Aggregate Server Access Protocol (ASAP) Overview

ASAP defines a straightforward set of mechanisms necessary to support the creation and maintenance of pools of redundant servers. These mechanisms include:

- o registration of a new server into a server pool
- o de-registration of an existing server from a pool
- o resolution of a pool handle to a server or list of servers
- o liveness detection for servers in a pool
- o failover mechanisms for handling a server failure

2.1. Pool Initialization

Pools come into existence when a PE registers the first instance of the pool name with an ENRP server. They disappear when the last PE de-registers. 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.

It is assumed that information needed for RSerPool, such as the address of an ENRP server to contact, is configured into the PE beforehand. Methods of automating this configuration process are not addressed at this time.

2.2. Pool Entity Registration

A new server joins an existing pool by sending a Registration message via ASAP to an ENRP server, indicating the pool handle of the pool that it wishes to join, a PE identifier for itself (chosen randomly), information about its lifetime in the pool, and what transport protocols and selection policy it supports. The ENRP server that it first contacts is called its Home ENRP server, and maintains a list of subscriptions by the PE as well as performs periodic audits to confirm that the PE is still responsive.

Similar procedures are applied to de-register itself from the server pool (or, alternatively, the server may simply let the lifetime that it previously registered with expire, after which it is gracefully removed from the pool).

2.3. Pool Entity Selection

When an endpoint wishes to be connected to a server in the pool, it generates an ASAP Handle Resolution message and sends this to its Home ENRP server. The ENRP server resolves the handle based on its knowledge of pool servers and returns a Handle Resolution Response message via ASAP. The response contains a list of the IP addresses of one or more servers in the pool that can be contacted. The process by which the list of servers is created may involve a number of policies for server selection. The RSerPool protocol suite defines a few basic policies and allows the use of external server selection input for more complex policies.

2.4. Endpoint Keep-Alive

ENRP servers monitor the status of pool elements using the ASAP Endpoint Keep-Alive message. A PE responds to the ASAP Keep-Alive message with an Endpoint Keep-Alive Ack response.

In addition, a PU can notify its Home ENRP server that the PE it used has become unresponsive by sending an ASAP Endpoint Unreachable message to the ENRP server.

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 defined 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 failover, 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 that 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.

3. Endpoint Handlespace Redundancy Protocol (ENRP) Overview

A set of server pools, which is denoted as a handlespace, is managed by ENRP servers. Pools are not valid in the whole Internet but only in smaller domains, called the operational scope. The ENRP servers use the ENRP protocol in order to maintain a distributed, fault-tolerant, real-time registry service. ENRP servers communicate with each other for information exchange, 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 by using a pre-defined Mentor server or by sending out Presence messages on a well-known multicast channel in order to determine other ENRP servers from the responses and select one as Mentor. A Mentor can be any peer ENRP server. The most current handlespace data is requested by Handle Table Requests from the Mentor. The received answer in the form of Handle Table Response messages is unpacked into the local database. After that, the ENRP server is ready to provide ENRP services.

3.2. Server Discovery and Home Server Selection

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. PEs only have to register with one ENRP server, as other ENRP servers supporting the pool will synchronize their knowledge about pool elements using the ENRP protocol.

The PE may have a configured list of ENRP servers to talk to, in the form of a list of IP addresses, in which case it will start to set up 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. Failure Detection, Handlespace Audit, and Synchronization

ENRP servers send ENRP Presence messages to all of their peers in order to show their liveness. These Presence messages also include a checksum computed over all PE identities for which the ENRP server is in the role of a Home ENRP server. Each ENRP server maintains an up-to-date list of its peers and can also compute the checksum expected from a certain peer, according to its local handlespace database. By comparing the expected sum and the sum reported by a peer (denoted as handlespace audit), an inconsistency can be detected. In such a case, the handlespace -- restricted to the PEs owned by that peer -- can be requested for synchronization, analogously to Section 3.2.

3.4. Server Takeover

If the unresponsiveness of an ENRP server is detected, the remaining ENRP servers negotiate which other server takes over the Home ENRP role for the PEs of the failed peer. After reaching a consensus on the takeover, the ENRP server taking over these PEs sends a notification to its peers (via ENRP) as well as to the PEs taken over (via ASAP).

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 getaddrinfo()) to translate from a hostname to an IP address, the application will invoke an RSerPool service primitive provisionally named GETPRIMARYSERVER that takes a pool handle as input, 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 a service primitive provisionally named 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.).

Note: at the time of this document, a full API for use with RSerPool protocols has not yet been defined.

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 (or policies).

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 [RFC5352]). Finally, when failures occur, these are reported to the pool via signaling present in ASAP [RFC5352] and ENRP [RFC5353]; 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 is 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, Stream Control Transmission Protocol (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 (so long as the application has saved enough state in a state cookie):

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 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. Reference Implementation

A reference implementation of RSerPool is available at [RSerPoolPage] and described in [Dre2006].

6. Security Considerations

This document does not identify security requirements beyond those already documented in the ENRP and ASAP protocol specifications. A security threat analysis of RSerPool is provided in THREATS [RFC5355].

7. IANA Considerations

This document does not require additional IANA actions beyond those already identified in the ENRP [RFC5353] and ASAP [RFC5352] protocol specifications.

8. Acknowledgements

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

9. References

9.1. Normative References

[RFC3237] Tuexen, M., Xie, Q., Stewart, R., Shore, M., Ong, L., Loughney, J., and M. Stillman, "Requirements for Reliable Server Pooling", RFC 3237, January 2002.

[RFC5352] Stewart, R., Xie, Q., Stillman, M., and M. Tuexen, "Aggregate Server Access Protocol (ASAP)", RFC 5352, September 2008.

[RFC5353] Xie, Q., Stewart, R., Stillman, M., Tuexen, M., and A. Silverton, "Endpoint Handlespace Redundancy Protocol (ENRP)", RFC 5353, September 2008.

[RFC5354] Stewart, R., Xie, Q., Stillman, M., and M. Tuexen, "Aggregate Server Access Protocol (ASAP) and Endpoint Handlespace Redundancy Protocol (ENRP) Parameters", RFC 5354, September 2008.

[RFC5355] Stillman, M., Ed., Gopal, R., Guttman, E., Holdrege, M., and S. Sengodan, "Threats Introduced by Reliable Server Pooling (RSerPool) and Requirements for Security in Response to Threats", RFC 5355, September 2008.

[RFC5356] Dreibholz, T. and M. Tuexen, "Reliable Server Pooling Policies", RFC 5356, September 2008.

9.2. Informative References

[RSerPoolPage] Dreibholz, T., "Thomas Dreibholz’s RSerPool Page", <http://tdrwww.iem.uni-due.de/dreibholz/rserpool/>.

[Dre2006] Dreibholz, T., "Reliable Server Pooling -- Evaluation, Optimization and Extension of a Novel IETF Architecture", Ph.D. Thesis University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems, March 2007, <http://duepublico.uni-duisburg-essen.de/servlets/DerivateServlet/Derivate-16326/Dre2006-final.pdf>.

Authors’ Addresses

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

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

Lyndon Ong
Ciena Corporation
PO Box 308
Cupertino, CA 95015
US

EMail: Lyong@Ciena.com

Michael Tuexen
Muenster Univ. of Applied Sciences
Stegerwaldstr. 39
48565 Steinfurt
Germany

EMail: tuexen@fh-muenster.de

Thomas Dreibholz
University of Duisburg-Essen, Institute for Experimental Mathematics
Ellernstrasse 29
45326 Essen, Nordrhein-Westfalen
Germany

Phone: +49 201 183-7637
Fax: +49 201 183-7673
EMail: dreibh@iem.uni-due.de
URI: http://www.iem.uni-due.de/˜dreibh/