An Overview of the Reliable Server Pooling Architecture

Thomas Dreibholz (dreibh@exp-math.uni-essen.de)
University of Duisburg-Essen, Institute for Experimental Mathematics

26th August 2004

Abstract

The convergence of classical PSTN and IP networks requires the transport of SS7 signalling over IP. Since SS7 has very strict availability requirements to the signalling components, redundancy is mandatory. The goal of the IETF RSerPool working group is to define a lightweight, flexible and realtime redundancy concept to fulfil the availability requirements of SS7: reliable server pooling (RSerPool). RSerPool is currently under standardization, its functionality and improvement are subject of our research.

Our poster presents our RSerPool proof-of-concept implementation and a research oriented, discrete event based simulation model. We provide simulation results showing limitations of the server selection procedures defined in the standards and a way to solve them. These improvements are now going into standardization by the IETF. Furthermore, we present some of our results on efficient algorithms and data structures for pool management. The poster concludes with an outlook on our currently progressing evaluations of service reliablity in failure scenarios.

1 What is Reliable Server Pooling

The convergence of classical circuit-switched networks (i.e. PSTN/ISDN) and data networks (i.e. IP-based) is rapidly progressing. This implies that PSTN signalling via the SS7 protocol is transported over IP networks. Since SS7 signalling networks offer a very high degree of availability (e.g. at most 10 minutes downtime per year for any signalling relation between two signalling endpoints; for more information see [5]), all links and components of the network devices must be redundant. When transporting signalling over IP networks, such redundancy concepts also have to be applied to achieve the required availability. Link redundancy in IP networks is supported using the Stream Control Transmission Protocol (SCTP); redundancy of network device components is supported by the SGP/ASP (signalling gateway process/application server process) concept. However, this concept has some limitations: no support of dynamic addition and removal of components, limited ways of server selection, no specific failover procedures and inconsistent application to different SS7 adaptation layers.

To cope with the challenge of creating a unified, lightweight, realtime and flexible redundancy solution, the IETF Reliable Server Pooling Charter has been founded. An overview of their concept Reliable Server Pooling (RSerPool), which is currently in the standardization process and described by several Internet Drafts, is shown in figure 1. Redundant servers providing the same service belong to a so called *server pool*, identified by a unique ID called *pool handle* within the set of all server pools, the so called *namespace*. A server in a pool is called *pool element* (PE) of its pool. The namespace is managed by redundant *name servers* (NSs), they synchronize their view of the namespace using the Endpoint Name Resolution Protocol (ENRP). NSs announce themselves using broadcast/multicast, i.e. it is not necessary to configure any NS address to other components described in the following.

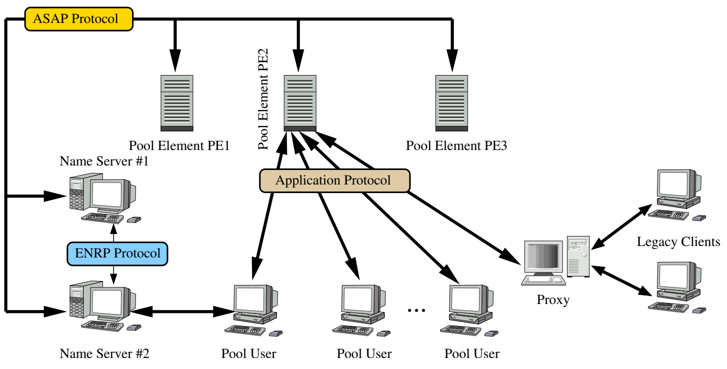

Figure 1: An Overview of the RSerPool Architecture

PEs can register to a pool of the namespace at an arbitrary NS using the Aggregate Server Access Protocol (ASAP). The NS chosen by the PE for registration monitors the PE using SCTP heartbeats and ASAP keep-alives; the frequency of monitoring messages depends on the provided service's availability requirements. When a PE becomes unavailable, it is immediately removed from the namespace. A PE can also intentionally deregister from the namespace by an ASAP deregistration. NS failures are handled by requiring PEs to re-register regularly (and therefore choosing a new NS when necessary) and by NSs monitoring their peer NSs and invoking an ENRP takeover procedure in case of failure.

When a client requests a service a from a pool, it asks an arbitrary NS to translate the pool handle to a list of PE transport addresses selected by the pool's selection policy (*pool policy*), e.g round robin or least used (see [7] for additional standardized policies). Then, it selects again one PE and establishes a transport connection to this PE using the application's protocol. The client then becomes a so called *pool user* (PU) of the PE's pool. In case of PE failure, the procedure is repeated to establish a connection to a new PE. Optionally, a PU can report a PE failure to a NS, which may decide to remove this PE from the namespace.

RSerPool supports optional client-based state synchronization [1] for failover. That is, a PE can provide its current state as *state cookie* to the PU. When a failover to a new PE is made, the PU can transmit this state cookie to the new PE, which can then restore this state. RSerPool is not restricted to client-based state synchronization, any other application-specific failover procedure can be used as well by the application layer itself.

The lightweight, realtime and flexible architecture of RSerPool is not only usable for SS7-based telephony signalling. Other application scenarios include SIP, mobility management [3] and the management of distributed computing pools [4, 8].

2 Our Research Results

RSerPool is a completely new standard, so almost no research on the subject of this protocol framework has been made yet. While partial aspects of RSerPool like pools, availability monitoring and load

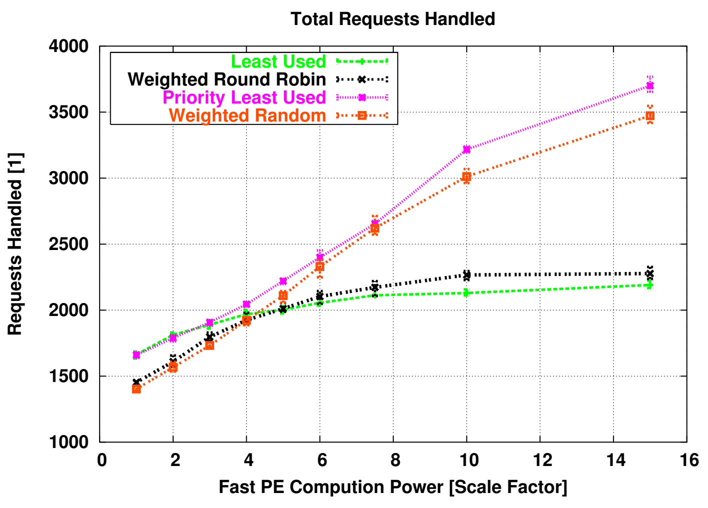

Figure 2: A Pool Policy Load Distribution Performance Result

distribution themselves are not completely new ideas, their combination into RSerPool is. The goal of our RSerPool activity is to examine and evaluate this new framework, develop improvements and finally bring them into the standardization process. One of our successful contributions to the standard is the concept of client-based state synchronization using state cookies, which we first described in [1] and which is now part of the ASAP draft.

The main focus of our RSerPool research activity is based on a discrete event simulation model. This model contains modules for the ASAP and ENRP protocols as well as modules of a SGP PU and an ASP PE. Currently, we focus our research on the load distribution performance of pool policies defined by the IETF RSerPool Internet Drafts.

One selected result is shown in figure 2: The standard document defines the policies Least Used (a dynamic one, requiring distribution of PE load information) and Weighted Round Robin (a static one, using a weight constant). While these standard policies are sufficient for homogeneous scenarios, they become inefficient in inhomogeneous ones: We examined their load distribution behaviour in a scenario having a pool of 18 equal-powered PEs, scaling the computation power of only three "fast" PEs. 36 PUs request service computation from the pool, using an average inter-request time of 10s (exponentially distributed). The average request handling effort is 10s (request size is exponentially distributed) on a normal PE when being processed exclusively. A PE can handle multiple requests simultaneously; in this case the computation power is shared between them (like different Unix processes).

The expected result should be that the pool is able to handle the more service requests the more powerful the fast PEs become. But as it is shown, the gradient of total requests handled is small. For higher scale factors than 7, even no significant gain can be observed. The reason is, that Least Used does not incorporate the fact that a fast PE may be able to process a new request faster than a lower-loaded normal one. For higher scale factors, the results for the dynamic policy are even worse than for the static Weighted Round Robin.

Our goal was to improve the load distribution behaviour by adding two new policies. The first one is Priority Least Used, which is Least Used with an information on how much a PE's load increases when answering a new request. This additional information is provided by the PE itself. As figure 2 shows, the total amount of requests handled by the pool almost linearly increases now. For every scale factor, our Priority Least Used is at least as good as the original Least Used, but providing much better utilization of the pool's resources the more inhomogeneously the scenario becomes. Our second new policy is Weighted Random, a static policy using a weight constant for a random distribution proportional to the PEs' weights. Again, this policy shows an almost linearly increasing request handling throughput for scaling the fast PE's computation power.

Our recommendations and improvements for the RSerPool standards have just been summarized into an Internet Draft [7], which has been presented and discussed at the 60th IETF meeting. Now, our results are going into standardization.

An additional result of our pool policy research is the development and evaluation of suitable algorithms and data structures for efficient namespace management [2]: We express pool policies in form of sorting orders and selection procedures and then store the namespace using multiple leaf-linked balanced binary trees. While our management concept allows easy addition of new policies, our performance evaluations also have shown that it is very runtime-efficient and scalable.

In cooperation with Siemens AG, we have realized a prototype implementation of the RSerPool framework [4, 6]. The main purpose for creating and maintaining this prototype is to make proof of concept tests and evaluations of theoretical results gained from our simulation model. By proving results to be useful in theory and also in practice, this supports our goal to bring our improvements into the standards process. Our prototype has already been used for a proof of concept on the usability of RSerPool for mobility management [3] and in a student project to examine the usability of RSerPool for the management of distributed computing pools [8]. Our poster will present the prototype and some example applications.

Our next research goal is to examine the load distribution behaviour of the policies under failure conditions, i.e. when links, PEs or NSs go out of service. Since RSerPool has been created to provide a service continuation by failover within strict time constrains, this is the main aspect of our ongoing research.

References

[1] T. Dreibholz. An efficient approach for state sharing in server pools. In *Proceedings of the 27th Local Computer Networks Conference*, Tampa, Florida/U.S.A., Oct 2002.

[2] T. Dreibholz. Policy Management in the Reliable Server Pooling Architecture. In *Proceedings of the Multi-Service Networks Conference 2004*, Abingdon, Oxfordshire/United Kingdom, Jul 2004.

[3] T. Dreibholz, A. Jungmaier, and M. Tüxen. A new Scheme for IP-based Internet Mobility. In *Proceedings of the 28th Local Computer Networks Conference*, Königswinter/Germany, Nov 2003.

[4] T. Dreibolz and M. Tüxen. High availability using reliable server pooling. In *Proceedings of the Linux Conference Australia 2003*, Perth/Australia, Jan 2003.

[5] K. D. Gradischnig and M. Tüxen. Signaling transport over ip-based networks using ietf standards. In *Proceedings of the 3rd International Workshop on the design of Reliable Communication Networks*, pages 168–174, Budapest, Hungary, 2001.

[6] Thomas Dreibholz's RSerPool Page. http://tdrwww.exp-math.uni-essen.de/dreibholz/rserpool.

[7] M. Tüxen and T. Dreibholz. Reliable Server Pooling Policies. Internet-Draft Version 00, IETF, RSerPool WG, Jul 2004. draft-tuexen-rserpool-policies-00.txt, work in progress.

[8] Y. Zhang. Distributed Computing mit Reliable Server Pooling. Masters thesis, Universität Essen, Institut für Experimentelle Mathematik, Apr 2004.