Evaluation and Optimization of the Registrar Redundancy Handling in Reliable Server Pooling Systems

Xing Zhou*, Thomas Dreibholz†, Fu Fa*, Wencai Du* and Erwin P. Rathgeb†
*Hainan University, College of Information Science and Technology
Renmin Avenue. 58, 570228 Haikou, Hainan, China
{zhouxing,fufa,wencai}@hainu.edu.cn
†University of Duisburg-Essen, Institute for Experimental Mathematics
Ellernstrasse 29, 45326 Essen, Germany
{dreibh,rathgeb}@iem.uni-due.de

***Abstract*—The Reliable Server Pooling (RSerPool) architecture is the IETF's new standard for a lightweight server redundancy and session failover framework to support availability-critical applications. RSerPool combines the ideas from different research areas into a single, resource-efficient and unified architecture. Server pools are maintained by redundant management components, which are called registrars. Registrars monitor the availability of servers in the pool and remove them in case of failure. Furthermore, they synchronize their view of the pool with other registrars to provide information redundancy.**

In this paper, we first analyse the implications of registrar redundancy on the server pool performance. Furthermore, we present an optimization approach for the server pool management, which improves the system performance in case of registrar problems by hardware failures or Denial of Service attacks.[1] [2]

Keywords: Reliable Server Pooling, Redundancy, Takeover, Handlespace Management, Performance Analysis

I. Introduction and Scope

Reliable Server Pooling (RSerPool, see [1]) denotes the IETF's new standard for a generic, application-independent server pool [2] and session management [3] framework. While there have already been a number of publications on the performance of RSerPool for application load balancing [3], [4], server failure handling [5] and the pool management data structures in general [2], [6], there has been very little research on the behaviour of the pool management in case of failures of the redundant management components which are denoted as *registrars*. Such failures may occur due to hardware problems (e.g. network or power failures) but also due to Denial of Service (DoS) attacks on the RSerPool setup [7]–[9].

The goal of this paper is to first analyse the implications of registrar redundancy on the server pool performance. Knowledge of these implications is important to provide RSerPool systems for achieving good system performance at small overhead costs. Furthermore, we present an optimization approach for the server pool management, which improves the system performance in case of registrar problems due to hardware failures or DoS attacks. This approach is evaluated using our RSerPool simulation model RSPSIM [4], [10].

[1]Funded by the State Administration of Foreign Experts Affairs, P. R. China (funding number 20084600036) and the German Research Foundation (Deutsche Forschungsgemeinschaft).

[2]Corresponding author is Thomas Dreibholz, dreibh@iem.uni-due.de.

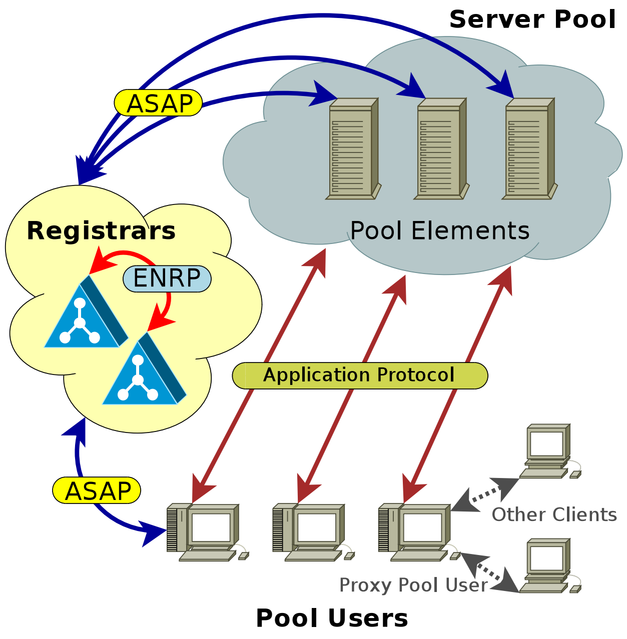


Figure 1. The RSerPool Architecture

II. The RSerPool Architecture

Figure 1 illustrates the RSerPool architecture [1], [3], [11] which consists of three types of components: servers of a pool are called *pool elements* (PE), a client is denoted as *pool user* (PU). The *handlespace* – which is the set of all pools – is managed by redundant *pool registrars* (PR). Within the handlespace, each pool is identified by a unique *pool handle* (PH).

A. Components and Protocols

PRs of an *operation scope* synchronize their view of the handlespace by using the Endpoint haNdlespace Redundancy Protocol (ENRP) [12], transported via SCTP [13].

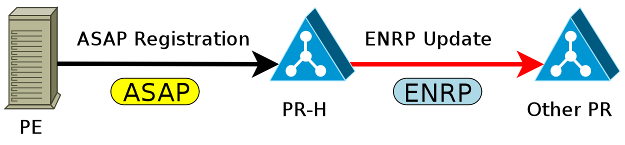


Figure 2. Pool Element Registration

Unlike Grid Computing [14], an operation scope is restricted to a single administrative domain. That is, all of its components are under the control of the same authority (e.g. a company or an organization). This property results in a small management overhead [2], which also allows for RSerPool usage on devices providing only limited memory and CPU resources (e.g. embedded systems like routers). Nevertheless, PEs may be distributed globally to continue their service even in case of localized disasters [15].

PEs choose an arbitrary PR of the operation scope to register into a pool by using the Aggregate Server Access Protocol (ASAP) [16], [17], again transported via SCTP and using TLS or IPSEC. Within its pool, a PE is characterized by its PE ID, which is a randomly chosen 32-bit number. Upon registration at a PR using an ASAP Registration message, the chosen PR becomes the Home-PR (PR-H) of the newly registered PE. A PR-H is responsible for monitoring its PEs' availability by ASAP Endpoint Keep-Alive messages (to be acknowledged by the PE within a given timeout) and propagates the information about its PEs to the other PRs of the operation scope via ENRP Handle Update messages [12]. PEs re-register regularly (again using an ASAP Registration message) in an interval denoted as *registration lifetime* as well as for information updates. Figure 2 illustrates the context of a PE's registration at its PR-H and the synchronization with another PR.

In order to access the service of a pool given by its PH, a PU requests a PE selection from an arbitrary PR of the operation scope, again using ASAP. The PR selects the requested list of PE identities by applying a pool-specific selection rule, called *pool policy*. RSerPool supports two classes of load distribution policies: non-adaptive and adaptive algorithms [4]. While adaptive strategies base their assignment decisions on the current status of the processing elements (which of course requires up-to-date states), non-adaptive algorithms do not need such data. A basic set of adaptive and non-adaptive pool policies is defined in [18]. Relevant for this paper are the non-adaptive policies Round Robin (RR) and Random (RAND) as well as the adaptive policy Least Used (LU).

B. Registrar Redundancy

Since a single PR would be a single point of failure – which RSerPool should avoid – there must be multiple PRs. Each PR in the operation scope is identified by a PR ID, which is – similar to the PE ID – a randomly chosen 32-bit number. PRs monitor the availability of each other PR using ENRP Presence messages, which are sent in an interval denoted as $\mathrm{PeerHeartbeatCycle}$ (default is 30s [12]). If there is no ENRP Presence within a timeout $\mathrm{MaxTimeLastHeard}$ (default is 61s [12]), the peer is assumed to be dead and a so called *takeover procedure* [12] initiated for the PEs managed by the dead PR: from all PRs having started this takeover procedure, the PR with the highest PR ID takes over the ownership of these PEs. The PEs are informed about their takeover by their new PR-H using an ASAP Endpoint Keep-Alive with Home-flag set.

As soon as PEs and PUs detect the failure of their PR (i.e. their request is not answered within a given timeout), they simply try another PR of the operation scope for their registration or handle resolution requests. Note, that the takeover procedure for PEs is intended as a double safeguarding: for the case the PE does not immediately detect its PR-H failure (in particular when using a long re-registration interval in case of non-adaptive policies).

III. Quantifying an RSerPool System

For our quantitative performance analysis, we use the application model from [3]: the service provider side of an RSerPool system consists of a pool of PEs. Each PE has a request handling *capacity*, which we define in the abstract unit of calculations per second[3]. Each request consumes a certain number of calculations; we call this number *request size*. A PE can handle multiple requests simultaneously – in a processor sharing mode as provided by multitasking operating systems.

On the service user side, there is a set of PUs. The number of PUs can be given by the ratio between PUs and PEs (*PU:PE ratio*), which defines the parallelism of the request handling. Each PU generates a new request in an interval denoted as *request interval*. The requests are queued and sequentially assigned to PEs.

The total delay for handling a request d_{Handling} is defined as the sum of queuing delay d_{Queuing}, startup delay d_{Startup} (dequeuing until reception of acceptance acknowledgement) and processing time $d_{\mathrm{Processing}}$ (acceptance until finish):

$$d_{\mathrm{Handling}} = d_{\mathrm{Queuing}} + d_{\mathrm{Startup}} + d_{\mathrm{Processing}}. \tag{1}$$

That is, d_{Handling} not only incorporates the time required for processing the request, but also the latencies of queuing, server selection and message transport. The user-side performance metric is the *handling speed*, which is defined as:

$$\mathrm{HandlingSpeed} = \frac{\mathrm{RequestSize}}{d_{\mathrm{Handling}}}.$$

For convenience, the handling speed (in calculations/s) is represented in % of the average PE capacity.

Using the definitions above, it is possible to delineate the average system utilization (for a pool of NumPEs servers and a total pool capacity of $\mathrm{PoolCapacity}$) as:

$$\mathrm{SystemUtil} = \mathrm{NumPEs} * \mathrm{puToPERatio} * \frac{\frac{\mathrm{RequestSize}}{\mathrm{RequestInterval}}}{\mathrm{PoolCapacity}}. \tag{2}$$

[3]An application-specific view of capacity may be mapped to this definition, e.g. CPU cycles, harddisk space, bandwidth share or memory usage.

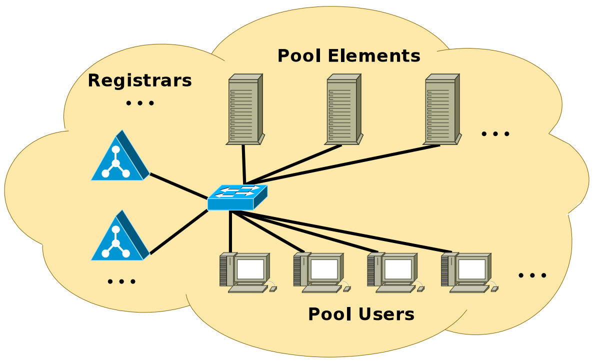


Figure 3. The Simulation Setup

Obviously, a provider-side performance metric is the system utilization, since only utilized servers gain revenue. Furthermore, the overhead of the pool management is important for the provider.

In practise, a well-designed client/server system is dimensioned for a certain *target system utilization* of e.g. 50%. That is, by setting any two of the parameters (PU:PE ratio, request interval and request size), the value of the third one can be calculated using equation 2 (see [3] for detailed examples).

IV. Setup Simulation Model

For our performance analysis, the RSerPool simulation model RSPSIM [3], [4], [10] has been used. This model is based on the OMNeT++ [19] simulation environment and contains the protocols ASAP [16], [17] and ENRP [12], a PR module and PE as well as PU modules for the request handling scenario defined in section III. Network latency is introduced by link delays only. Therefore, only the network delay is significant. The latency of the pool management by PRs is negligible [2].

Unless otherwise specified, the basic simulation setup – which is also presented in figure 3 – uses the following parameter settings:

- The average inter-component network delay is 10ms (which is realistic for components distributed within a region like Europe or North America, see [15]).
- The target system utilization is 80%. Request size and request interval are randomized using a negative exponential distribution (in order to provide a generic and application-independent analysis [3], [4]). There are 100 PEs; each one provides a capacity of 10^6 calculations/s.
- A PU:PE ratio of 10 is used (i.e. a non-critical setting as explained in [4]).
- We use request size:PE capacity setting of 10; i.e. being processed exclusively, the average processing takes 10s – see also [4].
- PEs re-register every 300s (registration lifetime) as well as on every load change of the adaptive LU policy.
- There are 5 PRs (we will examine the impact varying the number of PRs in section V).
- ASAP requests are transmitted once. If there is no reply within 5s, the PR is assumed to be dead and another PR is contacted (selected by random).
- ENRP uses the default parameters of $\mathrm{PeerHeartbeatCycle}$=30s and $\mathrm{MaxTimeLastHeard}$=61s (as in RFC [12]).
- The simulated real-time is 120min; each simulation run is repeated at least 24 times with a different seed in order to achieve statistical accuracy.

GNU R is used for the statistical post-processing of the results. Each resulting plot shows the average values and their 95% confidence intervals.

V. The Impact of Registrar Redundancy

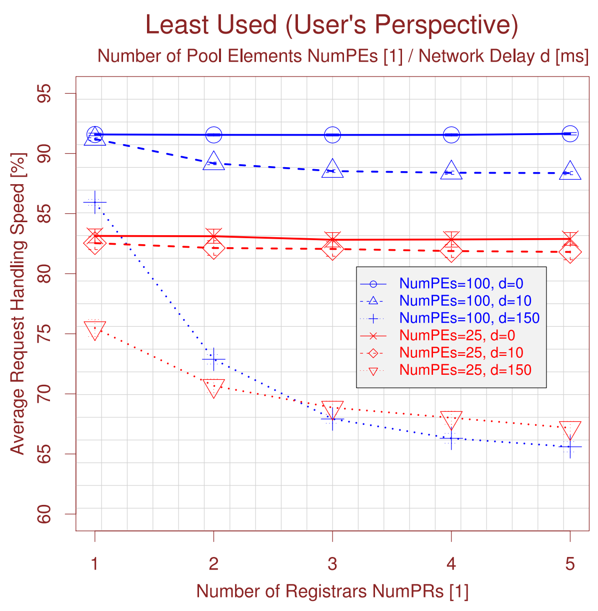
Clearly, since a single PR constitutes a single point of failure, there have to be multiple PRs in an RSerPool setup. In order to provide such a system, it is necessary to know the performance implications of PR redundancy. To show the essential effects, figure 4 provides the results of a simulation varying the number of PRs NumPRs for a varying number of PEs NumPEs and inter-component network delay d. The left-hand plot presents the results for the LU policy. Here, it can be observed that increasing NumPRs leads to a smaller handling speed, particularly for a high delay d – since the load state information becomes somewhat inaccurate due to the latency between its actual change and the time it is used for a PE selection at a PR. Furthermore, having a larger pool (here: NumPEs=100), the impact of a higher number of PRs is stronger: simply, the higher the corresponding number of PUs (here: a PU:PE ratio of 10, i.e. 1,000 PUs for 100 PEs), the higher the probability of nearly-simultaneous requests – which lead to the usage of inaccurate load information: PEs just having accepted new requests are selected for further PUs, since their updated load states have not yet reached the selecting remote (i.e. non-PR-H) PRs.

However, since RSerPool setups in most cases are not distributed globally[4], the delay d is usually quite small (e.g. 5ms to 15ms [15] within Europe or North America) so that the performance impact on LU is usually small.

The right-hand plot of figure 4 shows the curves for RR and RAND at d=150ms: while RAND keeps unaffected by the delay, there is a small performance decrease for RR: since different PRs perform their round robin selection independently [4], [18], the global view of RR selection differs from a selection in turn. The selection order is therefore less optimal. Since both policies are non-adaptive, there is – unlike for LU – no impact on the number of PEs NumPEs. Note, that the request handling speed of the LU policy, which is presented with a different y-axis scale in the left-hand plot, still has a significantly higher performance than RR and RAND – despite the delay effects.

For realistic RSerPool setups – e.g. for deploying simulation processing pools using SimProcTC [10] – the number of PRs is assumed to be in the range of about 2 to 5.

[4]See [15] and [20] for mechanisms to handle high-latency scenarios.



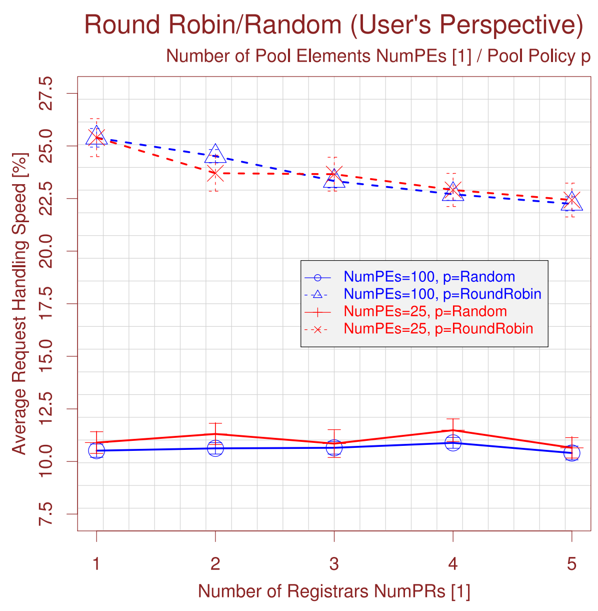


Figure 4. The Impact of Registrar Redundancy

VI. Handling Registrar Failures

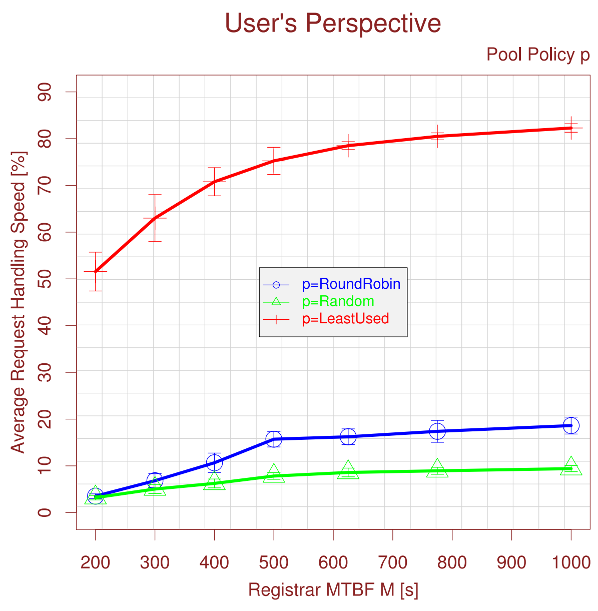


Figure 5. Handling Registrar Failures

In order to show the effectiveness of the RSerPool PR failure handling procedures described in subsection II-B, figure 5 presents the handling speed results for varying the average Mean Time Between Failures (MTBF) M of the 5 PRs from 200s to 1000s – with an average downtime of 100s (both parameters have negative exponential distribution). Obviously, the RSerPool system is able to cope with the PR failures: as long as there is at least one usable PR, the service performance is only slightly degraded. PUs and PEs use another PR when a request to their original PR fails. Also, the remaining PRs themselves perform takeovers for the PEs of failed PRs. Only in the case of having no PR usable for a longer duration (here: for M <500s) does the service performance suffer significantly. However, such small uptimes are very unrealistic for real setups, where an average PR MTBF can be assumed in the range of weeks or months.

As a result, having at least one working PR available at any time, the performance from the user's perspective will not suffer. But what about the provider's perspective?

VII. Avoiding Unbalanced Registrar Workloads

A. *Problem Identification*

To present the problem of the pool management overhead, we also use 5 PRs. PR #1 stays available all the time, only PR #2 to PR #5 have problems (e.g. a network problem or a DoS attack) and their MTBF is varied. Over the time, PR #2 to PR #5 will fail and their PEs and PUs have to use one of the remaining PRs. This is the intended behaviour and works (as illustrated in section VI), but eventually results in all components using PR #1 – which is always available. In particular, PR #1 will become PR-H of all PEs and be responsible for their monitoring (using ASAP Endpoint Keep-Alives) as well as propagating their PE entries to the other PRs (using ENRP Handle Updates). That is, all management tasks – which also have a certain computational overhead, as shown by [2] – are concentrated on a single PR. Even when the PR problems are solved and all PRs become online again, the situation is not changed quickly: PEs using PR #1 have no reason to rechange their PR-H, so they keep using PR #1. Only when the connection between PE and PR is broken, a PE decides to choose another PR.

B. *A Simple and Efficient Solution*

A method to distribute PEs among PRs is presented by [21]: the usage of the P2P algorithm Chord [22].

However, as explained in section V, the number of PRs is usually rather small in comparison to P2P nodes. Therefore, using a complex P2P algorithm like Chord seems to be an extreme overkill for the lightweight [2] RSerPool architecture. Instead, we propose a significantly simpler – but quite effective – approach: on registration of a PE i, a PR-H π_1 decides whether it is the most "appropriate" PR for this PE:

- If it is the most "appropriate" PR, there is nothing else to do.
- Otherwise, if there is a PR π_2 which is "better", this PR π_2 will be suggested to take over PE i.

This takeover suggestion is signalled within the ENRP Handle Update message by setting a bit which we call *Takeover Suggestion Flag*. We denote our approach therefore as *Takeover Suggestion*.

To find the most "appropriate" PR π_i^* for PE i, we simply apply an XOR metric: PR π_i^* is the PR of the operation scope where π_i^* XOR i is maximal. This approach only requires the identification of the PR π_i^* upon registration of PE i (from a PR list containing only very few entries). For the Takeover Suggestion Flag, a currently unused bit in the ENRP Handle Update message [3], [12] can be used, i.e. no new message types or additional bandwidth overhead are required.

C. Evaluation

To show the effectiveness of our approach, figure 6 presents the results of a simulation varying the MTBF M of PR #2 to PR #5, while PR #1 always stays available. The solid lines represent the results without using the Takeover Suggestion (τ=false, i.e. the current standard behaviour), while the dotted lines show the results for applying our Takeover Suggestion approach (τ=true).

First, the upper left-hand side plot of figure 6 presents the user's performance perspective: the handling speed. As shown, using the Takeover Suggestion has no impact on the user's performance – in particular, it does not decrease the handling speed for any of the three policies. For instance, it also does not affect the system utilization (which is not shown here, due to space limitations).

The impact on the number of sent ASAP Endpoint Keep-Alives for each PR ρ is presented in the upper right-hand plot of figure 6. Their number is independent of the pool policy. Clearly, for τ=false (i.e. no Takeover Suggestion), PR #1 (ρ=1) becomes responsible for the monitoring of almost all PEs. Therefore, within the 120min of simulation time, it has to send about 23,000 messages – while the other PRs almost have none to send (represented here by ρ=5 for PR #5; PRs #2 to #4 behave similarly; their curves have been omitted for enhancing readability). Note, that the number of Endpoint Keep-Alives depends on pool size and keep-alive interval. For large pools with high intervals – which is useful for certain applications [5] – their number would be significantly higher. Using Takeover Suggestion (τ=true), the desired behaviour is achieved: the other PRs – when online – take over some monitoring workload. In the example simulation at M=1000s, the workload of PR #1 already sinks from about 15,000 messages to about 5,000.

The number of ASAP Registration and ENRP Handle Update messages depends on the policy: using an adaptive policy, there is the need for a PE to re-register at its PR-H – as well as handlespace synchronization to other PRs – upon each policy information update (e.g. changed load state in case of LU). Therefore, the plots for the number of processed ASAP Registrations (lower left-hand side of figure 6) and the number of handled ENRP Handle Updates (lower right-hand side) present the results for LU – i.e. the most difficult case – only. Since PR #1 takes ownership of almost all PEs without using Takeover Suggestion (i.e. τ=false), there will be a significant registration workload on PE #1 (i.e. ρ=1): more than 115,000 registrations within 120min of simulation time for M=200s – and still about 80,000 registrations for M=1000s. At the same time, the registration workload of the other PRs (represented by PR #5, i.e. ρ=5; PRs #2 to #4 behave similarly) is quite small. As shown by [2], the registration operation is relatively expensive: it not only consists of the handlespace management itself, but also requires maintaining an SCTP association to each owned PE. Using Takeover Suggestion (i.e. τ=true), the registration workload keeps reasonably balanced: at $M > 700$, there remains only a small difference between PR #1 and the other PRs.

The results for the number of ENRP Handle Updates processed by each PR (lower right-hand side of figure 6) corresponds to the observations for the ASAP Registrations: using Takeover Suggestion (i.e. τ=true), the effort is reasonably balanced – while for τ=false PR #5 (as well as PR #2 to PR #4, which are not shown here) mostly synchronizes with PR #1 (i.e. many Handle Updates) and PR #1 mostly handles ASAP Registrations (i.e. it sends out ENRP Handle Updates, but does not have to handle incoming Handle Updates from other PRs).

In summary, our Takeover Suggestion approach leads to a significantly improved PR workload (i.e. monitoring by keep-alives, ASAP registrations and ENRP update) balancing, while not influencing the performance of the RSerPool applications. Furthermore, it is – unlike the P2P approach of [21] – very efficiently realizable.

VIII. Conclusions

In this paper, we have examined the PR redundancy of RSerPool systems and identified the problem of unbalanced PR workload as a result of the PR failure handling. Our Takeover Suggestion approach solves this problem, without affecting the RSerPool application performance. Furthermore, it is very simple and efficiently attained.

As a next step, we are going to realize our approach in our RSerPool implementation RSPLIB, which is also the IETF's reference implementation [1, chapter 5]. Using this implementation, we intend to perform real-world experiments [23] on the PlanetLab. The results of our RSerPool research are contributed as an Internet Draft [24] into the IETF's standardization process, which has just reached an important milestone in bringing RSerPool

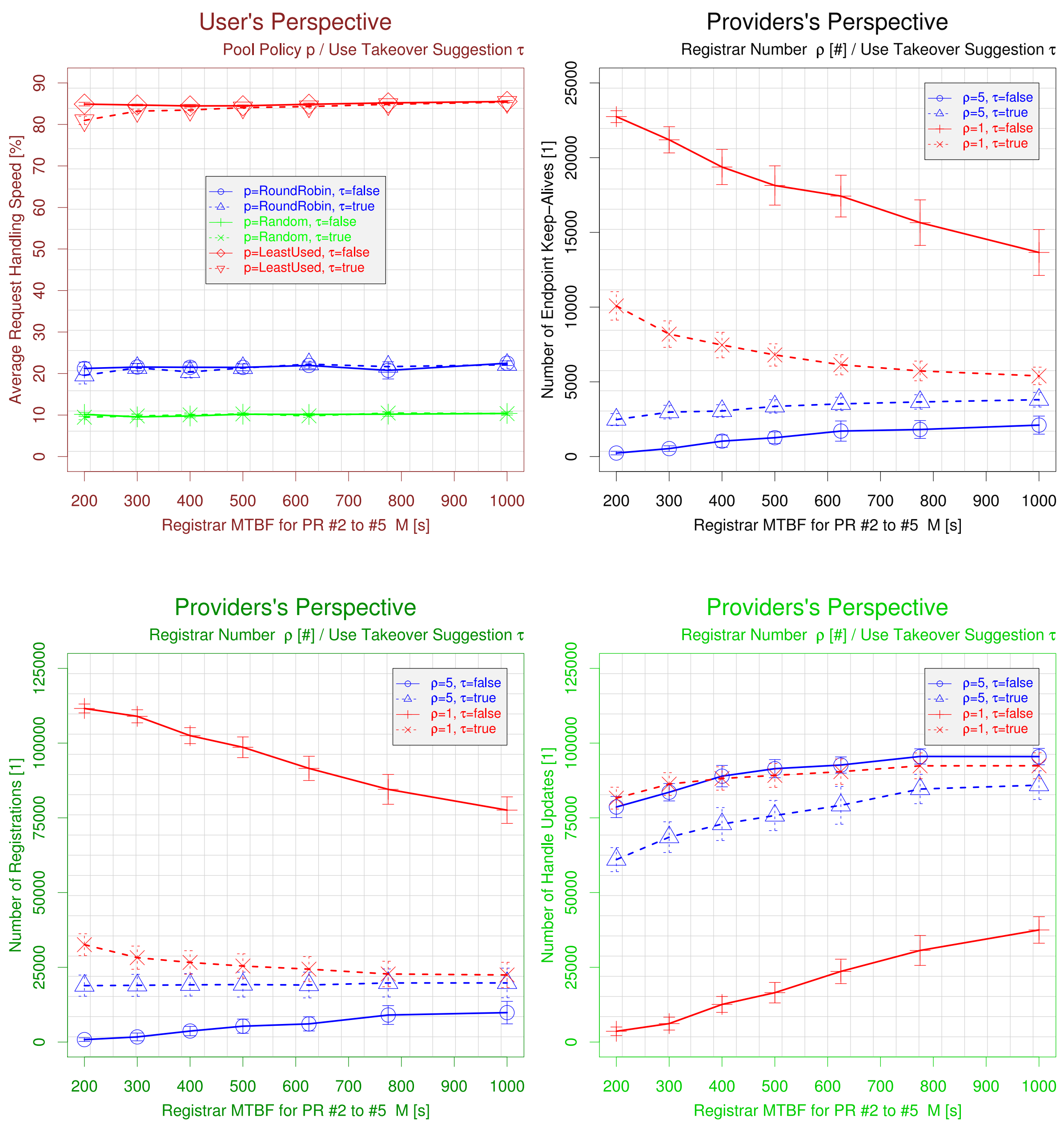


Figure 6. Avoiding Unbalanced Registrar Workloads

research to application by publication of its basic protocol documents as RFCs. Our goal is to provide configuration and optimization guidelines for application developers and users of the IETF's new RSerPool standard.

References

[1] P. Lei, L. Ong, M. Tüxen, and T. Dreibholz, "An Overview of Reliable Server Pooling Protocols," IETF, Informational RFC 5351, Sept. 2008.

[2] T. Dreibholz and E. P. Rathgeb, "An Evaluation of the Pool Maintenance Overhead in Reliable Server Pooling Systems," *SERSC International Journal on Hybrid Information Technology (IJHIT)*, vol. 1, no. 2, pp. 17–32, Apr. 2008.

[3] T. Dreibholz, "Reliable Server Pooling – Evaluation, Optimization and Extension of a Novel IETF Architecture," Ph.D. dissertation, University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems, Mar. 2007.

[4] T. Dreibholz and E. P. Rathgeb, "On the Performance of Reliable Server Pooling Systems," in *Proceedings of the IEEE Conference on Local Computer Networks (LCN) 30th Anniversary*, Sydney/Australia, Nov. 2005, pp. 200–208, ISBN 0-7695-2421-4.

[5] ——, "Reliable Server Pooling – A Novel IETF Architecture for Availability-Sensitive Services," in *Proceedings of the 2nd IEEE International Conference on Digital Society (ICDS)*, Sainte Luce/Martinique, Feb. 2008, pp. 150–156, ISBN 978-0-7695-3087-1.

[6] ——, "An Evaluation of the Pool Maintenance Overhead in Reliable Server Pooling Systems," in *Proceedings of the IEEE International Conference on Future Generation Communication and Networking (FGCN)*, vol. 1, Jeju Island/South Korea, Dec. 2007, pp. 136–143, ISBN 0-7695-3048-6.

[7] T. Dreibholz, E. P. Rathgeb, and X. Zhou, "On Robustness and Countermeasures of Reliable Server Pooling Systems against Denial of Service Attacks," in *Proceedings of the IFIP Networking*, Singapore, May 2008, pp. 586–598, ISBN 978-3-540-79548-3.

[8] P. Schöttle, T. Dreibholz, and E. P. Rathgeb, "On the Application of Anomaly Detection in Reliable Server Pooling Systems for Improved Robustness against Denial of Service Attacks," in

Proceedings of the 33rd IEEE Conference on Local Computer Networks (LCN), Montreal/Canada, Oct. 2008, pp. 207–214, ISBN 978-1-4244-2413-9.

[9] X. Zhou, T. Dreibholz, W. Du, and E. P. Rathgeb, "Evaluation of Attack Countermeasures to Improve the DoS Robustness of RSerPool Systems by Simulations and Measurements," in *Proceedings of the 16. ITG/GI Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, Kassel/Germany, Mar. 2009.

[10] T. Dreibholz and E. P. Rathgeb, "A Powerful Tool-Chain for Setup, Distributed Processing, Analysis and Debugging of OMNeT++ Simulations," in *Proceedings of the 1st ACM/ICST OMNeT++ Workshop*, Marseille/France, Mar. 2008, ISBN 978-963-9799-20-2.

[11] ——, "Towards the Future Internet – An Overview of Challenges and Solutions in Research and Standardization," in *Proceedings of the 2nd GI/ITG KuVS Workshop on the Future Internet*, Karlsruhe/Germany, Nov. 2008.

[12] Q. Xie, R. Stewart, M. Stillman, M. Tüxen, and A. Silverton, "Endpoint Handlespace Redundancy Protocol (ENRP)," IETF, RFC 5353, Sept. 2008.

[13] C. Hohendorf, E. P. Rathgeb, E. Unurkhaan, and M. Tüxen, "Secure End-to-End Transport Over SCTP," *Journal of Computers*, vol. 2, no. 4, pp. 31–40, June 2007.

[14] I. Foster, "What is the Grid? A Three Point Checklist," *GRID Today*, July 2002.

[15] T. Dreibholz and E. P. Rathgeb, "On Improving the Performance of Reliable Server Pooling Systems for Distance-Sensitive Distributed Applications," in *Proceedings of the 15. ITG/GI Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, Bern/Switzerland, Feb. 2007, pp. 39–50, ISBN 978-3-540-69962-0.

[16] R. Stewart, Q. Xie, M. Stillman, and M. Tüxen, "Aggregate Server Access Protcol (ASAP)," IETF, RFC 5352, Sept. 2008.

[17] T. Dreibholz, "Handle Resolution Option for ASAP," IETF, Individual Submission, Internet-Draft Version 04, Jan. 2009, draft-dreibholz-rserpool-asap-hropt-04.txt, work in progress.

[18] T. Dreibholz and M. Tüxen, "Reliable Server Pooling Policies," IETF, RFC 5356, Sept. 2008.

[19] A. Varga, *OMNeT++ Discrete Event Simulation System User Manual - Version 3.2*, Technical University of Budapest/Hungary, Mar. 2005.

[20] X. Zhou, T. Dreibholz, and E. P. Rathgeb, "A New Server Selection Strategy for Reliable Server Pooling in Widely Distributed Environments," in *Proceedings of the 2nd IEEE International Conference on Digital Society (ICDS)*, Sainte Luce/Martinique, Feb. 2008, pp. 171–177, ISBN 978-0-7695-3087-1.

[21] C. S. Chandrashekaran, W. L. Johnson, and A. Lele, "Method using Modified Chord Algorithm to Balance Pool Element Ownership among Registrars in a Reliable Server Pooling Architecture," in *Proceedings of the 2nd International Conference on Communication Systems Software and Middleware (COMSWARE)*, Bangalore/India, Jan. 2007, pp. 1–7, ISBN 1-4244-0614-5.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001, pp. 149–160.

[23] X. Zhou, T. Dreibholz, E. P. Rathgeb, and W. Du, "Takeover Suggestion – A Registrar Redundancy Handling Optimization for Reliable Server Pooling Systems," in *Proceedings of the 10th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2009)*, Daegu, South Korea, May 2009.

[24] T. Dreibholz and X. Zhou, "Takeover Suggestion Flag for the ENRP Handle Update Message," IETF, Individual Submission, Internet-Draft Version 01, Jan. 2009, draft-dreibholz-rserpool-enrp-takeover-01.txt, work in progress.