"Takeover Suggestion" – A Registrar Redundancy Handling Optimization for Reliable Server Pooling Systems

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

***Abstract*—Reliable Server Pooling (RSerPool) is the IETF's new standard for a common server redundancy and session failover framework to support availability-critical applications. Server pools are maintained by redundant management components denoted as registrars. These 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 illustrate the implications of registrar redundancy on the performance of RSerPool systems. After that, we present an optimization approach for the server pool management, which improves the management performance in case of registrar problems like hardware failures or Denial of Service attacks. The performance of our approach is evaluated in real life using PLANETLAB measurements.[1] [2]

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

I. INTRODUCTION AND SCOPE

Reliable Server Pooling (RSerPool, see [1], [2]) is the IETF's new standard for a generic, application-independent server pool [3], [4] and session management [5] framework. While there have already been a number of publications on the performance of RSerPool for application load balancing [5]–[9], server failure handling [10]–[12] and the pool management data structures in general [3], [4], 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) and also due to Denial of Service (DoS) attacks on the RSerPool setup [13]–[15].

In this paper, we first analyse the implications of registrar redundancy on the server pool performance by simulations. Awareness of these implications is essential for achieving good system performance at reasonable overhead costs. Next, we present an optimization approach for the server pool management to enhance the system performance in case of registrar problems (which e.g. may be caused by hardware failures or DoS attacks). We evaluate our approach by using the RSerPool implementation RSPLIB [5], [13], [16] in a real-world PLANETLAB [17] setup.

[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 Wencai Du, wencai@hainu.edu.cn.

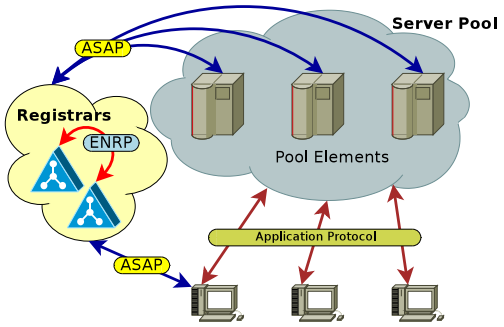

Figure 1. The RSerPool Architecture

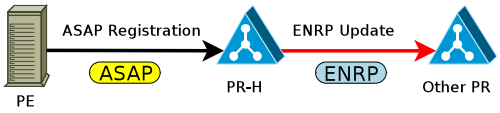

Figure 2. Pool Element Registration

II. THE RSERPOOL ARCHITECTURE

The three component types of the RSerPool architecture are depicted in figure 1: 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) (shortly denoted as *registrars*). Within the handlespace, each pool is identified by a unique *pool handle* (PH).

A. Components and Protocols

The PRs of an *operation scope* synchronize their view of the handlespace by using the Endpoint haNdlespace Redundancy Protocol (ENRP) [18], transported via SCTP [19]. Unlike Grid Computing [20], 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). This property results in a small management overhead [3], [4], 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 [16].

PEs choose an arbitrary PR of the operation scope to register into a pool by using the Aggregate Server Access

Protocol (ASAP) [21], 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 by 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 [18]. 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 by using ASAP again. 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 [6]. While adaptive strategies make their assignment decisions based on the current status of the processing elements (which of course requires up-to-date states), non-adaptive algorithms do not need such status data. A basic set of adaptive and non-adaptive pool policies is defined in [22]. 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 constitute a single point of failure – which RSerPool should avoid – there must be multiple PRs in an operation scope. 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 by using ENRP Presence messages, which are sent in an interval denoted as $\mathrm{PeerHeartbeatCycle}$ (default is 30s [18]). If there is no ENRP Presence within a timeout $\mathrm{MaxTimeLastHeard}$ (default is 61s [18]), the peer is assumed to be dead and a so-called *takeover procedure* [18] 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 by 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 [5]: 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*. Requests are queued and sequentially assigned.

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):

$$d_{\mathrm{Handling}} = d_{\mathrm{Queuing}} + d_{\mathrm{Startup}} + d_{\mathrm{Processing}}. \quad (1)$$

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 as % of the average PE capacity.

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

$$\mathrm{U} = \mathrm{NumPEs} * \mathrm{puToPERatio} * \frac{\frac{\mathrm{RequestSize}}{\mathrm{RequestInterval}}}{\mathrm{PoolCapacity}}. \quad (2)$$

Obviously, the primary provider-side performance metric is the system utilization, since only utilized servers gain revenue. In practise, a well-designed client/server system is dimensioned for a certain *target system utilization* of e.g. 50%. 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 also [5]). A provider's secondary performance metric is of course the handlespace management overhead [3], [4] (i.e. CPU resources and network bandwidth).

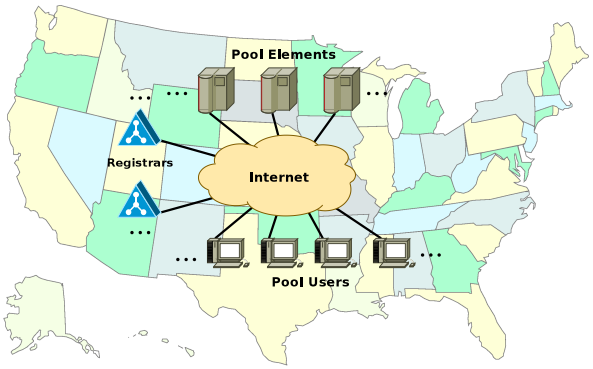

Figure 3. The System Setup

IV. System Setup

In order to evaluate the performance, we have used the OMNeT++-based RSerPool simulation model RSPSIM [6], [23], [24] as well as the implementation RSPLIB [5], [16] (which is also the IETF's reference implementation, see [1, chapter 5]) for measurements in

[3]An application-specific view of capacity may be mapped to this definition, e.g. CPU cycles.

a PLANETLAB setup. Both – simulation model and implementation – contain the protocols ASAP [21] and ENRP [18], a PR module and PE as well as PU modules for the request handling scenario defined in section III.

The PLANETLAB [17] setup distributes the components to different machines in the U.S.A.. This country provides a sufficient number of PLANETLAB nodes and a country-wide setup is also realistic for an RSerPool setup in a large company – for protecting a critical service against e.g. earthquakes, power failures or terrorist attacks. By `ping`-based tests, we have observed inter-node network delays of about 20ms to 30ms.

For our simulation and measurement setup, which is depicted in figure 3, we use the following parameter settings unless otherwise specified:

- The target system utilization is 50%. Request size and request interval are randomized using a negative exponential distribution (in order to provide a generic and application-independent analysis [5], [6]). There are 25 PEs; each one provides a capacity of 10^6 calculations/s.
- A PU:PE ratio of 3 is used (i.e. a non-critical setting as explained in [6]).
- We use a request size:PE capacity setting of 10; i.e. being processed exclusively, the average processing takes 10s – see also [6].
- There is only a single PR, since we do not examine PR failure scenarios here (see [6] for such scenarios). PEs re-register every 2s (re-registration interval) and on every load change of the adaptive LU policy.
- ASAP requests are transmitted once. If there is no reply within 5s, the PR is assumed to be dead and a random other PR is contacted.
- ENRP uses the default parameters of $\mathrm{PeerHeartbeatCycle}$=30s and $\mathrm{MaxTimeLastHeard}$=61s (as in RFC [18]).
- For the simulation, 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. The inter-component network delay is 25ms, which corresponds to our PLANETLAB setup.
- Each measurement run takes 15min; each run is repeated at least 19 times.

For statistical post-processing of the results, GNU R [23], [24] is used. Each resulting plot shows the average values and their 95% confidence intervals.

V. IMPLICATIONS OF REGISTRAR REDUNDANCY

Usually, an RSerPool setup contains multiple PRs to avoid a single point of failure. The existence of multiple PRs affects the performance. Knowledge of the resulting performance implications is therefore necessary to provision efficient systems. The important effects are demonstrated by the simulation results (based on our paper [25]) in figure 4 for increasing the number of PRs NumPRs for a varying number of PEs NumPEs and inter-component network delay d. The left-hand side presents the results for the LU policy: Obviously, increasing NumPRs leads to a lower handling speed, in particular for a high inter-component network delay d (here: 150ms). The latency leads to somewhat inaccurate load state information in the handlespace, which results in a reduced PE selection quality. Furthermore, for a large pool (here: NumPEs=100), the impact of a higher number of PRs is stronger: simply, the higher the corresponding number of PUs (here: the PU:PE ratio is 3, i.e. 300 PUs for 100 PEs), the higher the probability of nearly-simultaneous requests. This results in the usage of inaccurate load information, because PEs just having accepted new requests are selected again for more PUs – since their increased load states have not yet reached the selecting remote (i.e. non-PR-H) PRs. In view of RSerPool setups mostly covering a restricted geographical area[4], the delay d is usually low (e.g. 5ms to 15ms [16] within Europe or North America). Therefore, the resulting impact on the performance of the LU policy is usually quite small.

The handling speed results for the RR and RAND policies at d=150ms are presented on the right-hand side of figure 4: while the RAND policy keeps unaffected by the latency, there is a small performance decrease for RR: because different PRs perform their round robin selection independently [6], [22], 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 – in contrast to the adaptive LU policy – no impact of 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 – regardless of the delay effect described above. For real-world RSerPool depolyments, e.g. for simulation processing pools using the SIMPROCTC toolchain [23], [24], the number of PRs is assumed to be in the range of about 2 to 5.

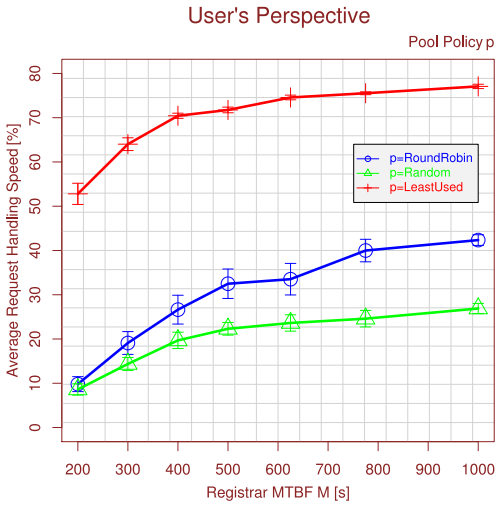

Figure 5. Registrar Failure Handling

Registrar redundancy is necessary to handle registrar failures. Handling speed simulation results showing the effectiveness of the RSerPool PR failure handling procedures (which are described in subsection II-B) are presented in figure 5. In this scenario, the average Mean Time Between Failures (MTBF) M of the 5 PRs has been varied from 200s to 1000s – using an average downtime of 100s (both parameters have negative exponential distribution). As intended, 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 case

[4]See [16] and [26] for mechanisms to handle high-latency scenarios.

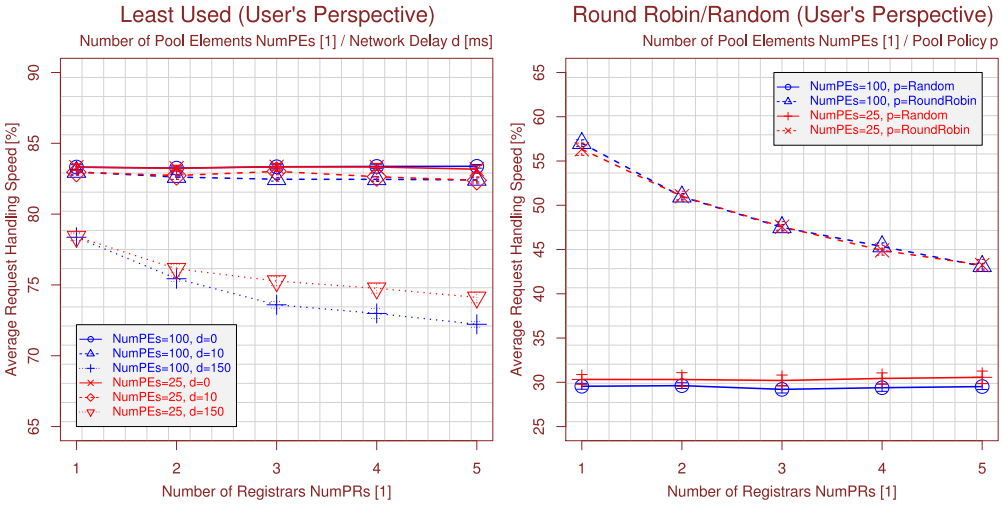

Figure 4. The Influence of Registrar Redundancy on the Handling Speed

of having no PR usable for a longer duration (here: for M <500s), the service performance significantly suffers. However, such small uptimes are very unrealistic for real setups, where an average PR MTBF can be assumed in the range of many weeks.

In summary, handling speed will not suffer as long as there is at least one PR available at any time. That is, from the perspective of the service user, everything is fine. However, it is also interesting to have a look at the service provider's perspective.

VI. Avoiding Unbalanced Registrar Workloads

For the service provider, it is important to control the handlespace management overhead. In order to accurately reflect the reality, we now go from simulation to real-life PlanetLab measurements to demonstrate the effects.

A. Management Challenge

In order to depict the management challenge, we again consider a scenario containing 5 PRs. But at this time, PR #1 stays available and only PR #2 to PR #5 discover e.g. network problems and their MTBF is varied. Over time, PR #2 to PR #5 will fail and their PEs and PUs have to choose and contact one of the remaining PRs. This strategy works well (as described in section V), but eventually results in all components using PR #1 – since this PR is always available. In particular, PR #1 will become PR-H of all PEs and be responsible for their monitoring (by using ASAP Endpoint Keep-Alives) as well as propagating their PE entries to the other PRs (by using ENRP Handle Updates). That is, all management tasks – which also have a certain computational overhead, as shown by [3], [4] – are concentrated on a single PR. Even when the PR problems are solved and all PRs become available again, the situation is not changed quickly: PEs using PR #1 have no reason to choose another PR-H. Therefore, they keep using PR #1. Only a broken connection between PE and PR leads to a PR-H change.

B. Our Optimization Approach

One possible approach to distribute PEs among PRs, which is suggested by [27], is the usage of the P2P algorithm Chord [28]. But the number of PRs is usually rather small in comparison to P2P nodes due to the restricted operation scope (as explained in section V). That is, using a complex P2P algorithm like Chord seems to be a quite inappropriate approach for the lightweight [3] RSerPool architecture. Our suggested approach is significantly simpler, but also very effective: 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 "best" PR for PE i, nothing needs to be done.
- Otherwise (i.e. there is a PR π_2 which is "better"), this PR π_2 is suggested to take over PE i.

The suggestion to take over a PE is signalled within the ENRP Handle Update message by setting a bit which we denote as *Takeover Suggestion Flag*. Therefore, we call our approach *Takeover Suggestion*. For our flag bit, a currently unused bit in the ENRP Handle Update message [5], [18] can be used, i.e. no new message types or additional bandwidth overhead are required.

We now have to specify a metric for which PR π_i^* is the most "appropriate" PR-H for PE i. Similar to some P2P approaches, we simply apply an XOR metric for this task: 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). That is, the computational overhead of our approach is very small.

C. Experimental Evaluation

In order to evaluate our approach, the MTBF M of PR #2 to PR #5 has been varied in a PlanetLab setup, while PR #1 remains available. The measurement results are presented in figure 6 for the handling speed (upper left-hand plot), number of ASAP Endpoint Keep-Alives (upper right-hand plot), number of ASAP Registrations (lower left-hand plot) and number of ENRP Updates (lower right-hand plot). Solid lines represent the results without using the Takeover Suggestion (τ=false, i.e. the current standard behaviour); dotted lines show the results for applying our Takeover Suggestion optimization (i.e. τ=true).

The user's performance perspective – which is the handling speed (shown in the upper left-hand plot of figure 6) – is not affected by the Takeover Suggestion

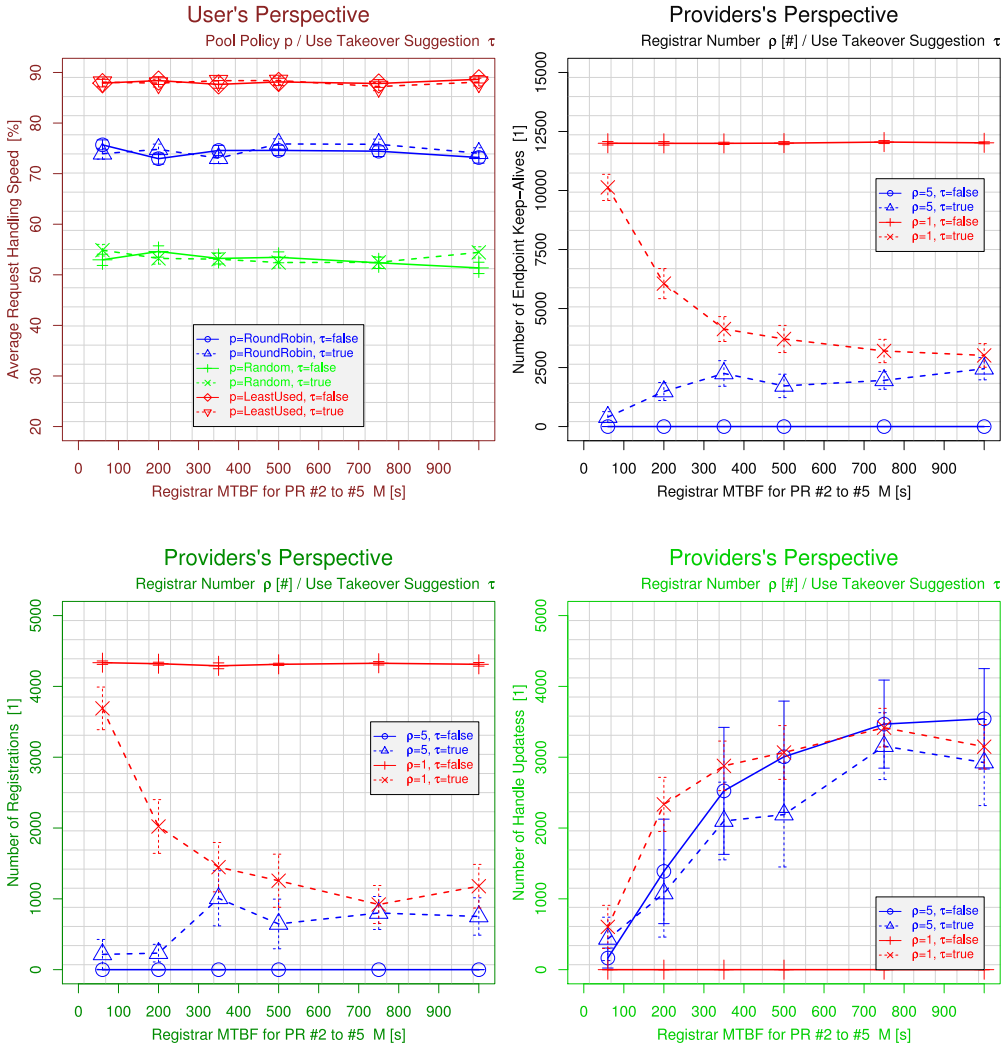

Figure 6. Avoiding Unbalanced Registrar Workloads

usage. In particular, it does not decrease the handling speed for any of the three policies. Also, it does not affect the system utilization (which is omitted here, due to space limitations).

The number of ASAP Endpoint Keep-Alives (see upper right-hand plot of figure 6) is independent of the pool policy. Obviously, for τ=false (i.e. no Takeover Suggestion), PR #1 (i.e. ρ=1) is responsible for monitoring almost all PEs. Therefore, within the 15min of measurement run time, it has to send about 12,000 messages – while the other PRs remain idle (represented here by ρ=5 for PR #5; the results of PRs #2 to #4 are omitted since they behave similarly). Note that the number of Endpoint Keep-Alives depends on pool size and keep-alive interval. The number would be significantly higher for larger pools with high intervals (which is useful for certain applications [5], [10]). Using our Takeover Suggestion optimization (i.e. τ=true), the desired behaviour is achieved: the other PRs take over some monitoring workload as long as they are available. In the example simulation at M=1000s, the workload of PR #1 decreases from about 12,000 messages to only about 2,500.

When applying an adaptive policy, there is the need for a PE to re-register (at its PR-H) as well as for handlespace synchronization to other PRs upon each policy information update (e.g. changed load state in case of LU). That is, the number of ASAP Registration and ENRP Handle Update messages depends on the policy type. Therefore, the plots for the number of processed ASAP Registrations (see the lower left-hand plot of figure 6) and the number of handled ENRP Handle Updates (lower right-hand plot) only show the results for the LU policy – i.e. the most labour-intensive case. Obviously, there is a significant registration workload on PE #1 (i.e. ρ=1) when our Takeover Suggestion optimization is turned off (i.e. τ=false), since PR #1 becomes the PR-H of almost all PEs: about 4,300 registrations within 15min of measurement run time for M=200s. 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. Measurements in [3] show that the registration operation is relatively expensive: it not only consists of the handlespace management itself, but also means maintaining an SCTP association with each owned PE. By applying Takeover Suggestion (i.e. τ=true), the registration workload keeps reasonably balanced: at $M \geq$750, there is no significant difference between PR #1 and the other PRs.

As expected, the number of ENRP Handle Updates processed by each PR (see the lower right-hand plot of

figure 6) corresponds to the observations for the ASAP Registrations: using our Takeover Suggestion optimization (i.e. τ=true), the effort gets balanced. On the other hand, when turning it off (i.e. τ=false), PR #5 (as well as PR #2 to PR #4, which behave similarly and are therefore omitted) 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 it does not have to handle incoming Handle Updates from other PRs).

In summary, our Takeover Suggestion optimization leads to a significantly better PR workload balancing (for ASAP Registrations and ENRP Handle Updates as well as monitoring by ASAP Endpoint Keep-Alives), while not influencing the performance of the RSerPool applications. In particular, unlike for the P2P approach suggested by [27], it is furthermore very efficiently realizable.

VII. Conclusions

In this paper, we have examined the PR redundancy of RSerPool systems. Using the standard PR failure handling procedure will lead to unbalanced management workload on the PRs. To overcome this problem, we have proposed our optimization "Takeover Suggestion". By using PlanetLab measurements, we have shown that it has efficiently solved the problem. Furthermore, our optimization is simple and can be realized with small CPU resources and without the need for additional network bandwidth.

The results of our RSerPool research are contributed as an Internet Draft [29] into the IETF's standardization process, which has just reached an important milestone of bringing RSerPool 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, "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.

[3] ——, "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.

[4] ——, "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.

[5] 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.

[6] 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.

[7] X. Zhou, T. Dreibholz, and E. P. Rathgeb, "A New Approach of Performance Improvement for Server Selection in Reliable Server Pooling Systems," in *Proceedings of the 15th IEEE International Conference on Advanced Computing and Communication (ADCOM)*, Guwahati/India, Dec. 2007, pp. 117–121, ISBN 0-7695-3059-1.

[8] ——, "Improving the Load Balancing Performance of Reliable Server Pooling in Heterogeneous Capacity Environments," in *Proceedings of the 3rd Asian Internet Engineering Conference (AINTEC)*, ser. Lecture Notes in Computer Science, vol. 4866. Springer, Nov. 2007, pp. 125–140, ISBN 978-3-540-76808-1.

[9] T. Dreibholz, X. Zhou, and E. P. Rathgeb, "A Performance Evaluation of RSerPool Server Selection Policies in Varying Heterogeneous Capacity Scenarios," in *Proceedings of the 33rd IEEE EuroMirco Conference on Software Engineering and Advanced Applications*, Lübeck/Germany, Aug. 2007, pp. 157–164, ISBN 0-7695-2977-1.

[10] T. Dreibholz and E. P. Rathgeb, "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.

[11] Ü. Uyar, J. Zheng, M. A. Fecko, S. Samtani, and P. Conrad, "Evaluation of Architectures for Reliable Server Pooling in Wired and Wireless Environments," *IEEE JSAC Special Issue on Recent Advances in Service Overlay Networks*, vol. 22, no. 1, pp. 164–175, 2004.

[12] M. Bozinovski, L. Gavrilovska, R. Prasad, and H.-P. Schwefel, "Evaluation of a Fault-tolerant Call Control System," *Facta Universitatis Series: Electronics and Energetics*, vol. 17, no. 1, pp. 33–44, 2004.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] L. Peterson and T. Roscoe, "The Design Principles of PlanetLab," *Operating Systems Review*, vol. 40, no. 1, pp. 11–16, Jan. 2006.

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

[19] R. Stewart, "Stream Control Transmission Protocol," IETF, Standards Track RFC 4960, Sept. 2007.

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

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

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

[23] T. Dreibholz, X. Zhou, and E. P. Rathgeb, "SimProcTC – The Design and Realization of a Powerful Tool-Chain for OMNeT++ Simulations," in *Proceedings of the 2nd ACM/ICST OMNeT++ Workshop*, Rome/Italy, Mar. 2009, ISBN 978-963-9799-45-5.

[24] 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.

[25] X. Zhou, T. Dreibholz, F. Fa, W. Du, and E. P. Rathgeb, "Evaluation and Optimization of the Registrar Redundancy Handling in Reliable Server Pooling Systems," in *Proceedings of the IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA)*, Bradford/United Kingdom, May 2009.

[26] 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.

[27] 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.

[28] 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.

[29] 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.