The Performance of Reliable Server Pooling Systems in Different Server Capacity Scenarios

Thomas Dreibholz
University of Duisburg-Essen
Ellernstrasse 29
45326 Essen, Germany
Email: dreibh@exp-math.uni-essen.de
Telephone: +49 201 183-7637
Fax: +49 201 183-7373

Erwin P. Rathgeb
University of Duisburg-Essen
Ellernstrasse 29
45326 Essen, Germany
Email: rathgeb@exp-math.uni-essen.de
Telephone: +49 201 183-7670
Fax: +49 201 183-7373

***Abstract*— Reliable Server Pooling (RSerPool) is a protocol framework for server pool management and session failover, currently under standardization by the IETF RSerPool WG. While the basic ideas of RSerPool are not new, their combination into one architecture is. Some research into the performance of RSerPool for certain specific applications has been made, but a detailed, application-independent sensitivity analysis of the system parameters is still missing.**

The goal of this paper is to systematically investigate RSerPool's load distribution behaviour on changes of workload and system parameters, to determine basic guidelines on designing efficient RSerPool systems. In this paper, we focus particularly on scenarios of server pools consisting of servers with unequal capacities.

I. Introduction

The Reliable Server Pooling (RSerPool) architecture [1] currently under standardization by the IETF RSerPool WG is an overlay network framework to provide server replication and session failover capabilities to its applications. These functionalities themselves are not new, but their combination into one application-independent framework is.

While there has already been some research on the performance of RSerPool for applications like SCTP-based mobility [2], distributed computing [3]–[9], e-commerce [10] and battlefield networks [11], a generic application-independent performance analysis is still missing. The goal of our work is therefore an application-independent quantitative characterization of RSerPool systems and a generic load distribution sensitivity analysis on changes of workload and system parameters. In particular, we want to identify critical parameter spaces to provide guidelines for designing efficient RSerPool systems. In this paper we concentrate our analysis on failure-free scenarios, since servers are usually available in 99.9x% of their lifetime and therefore best performance in this case is most crucial to a system's cost benefit ratio.

II. The RSerPool Architecture

An illustration of the RSerPool architecture defined in [1] is shown in figure 1. It consists of three component classes: servers of a pool are called *pool elements* (PE); each pool is identified by a unique *pool handle* (PH) in the handlespace, i.e. the set of all pools; the handlespace is managed by *registrars* (PR). PRs of an operation scope (e.g. a company or an organization) synchronize their view of the handlespace using the Endpoint haNdlespace Redundancy Protocol (ENRP [12], [13]), transported via SCTP [14]–[16]. An operation scope

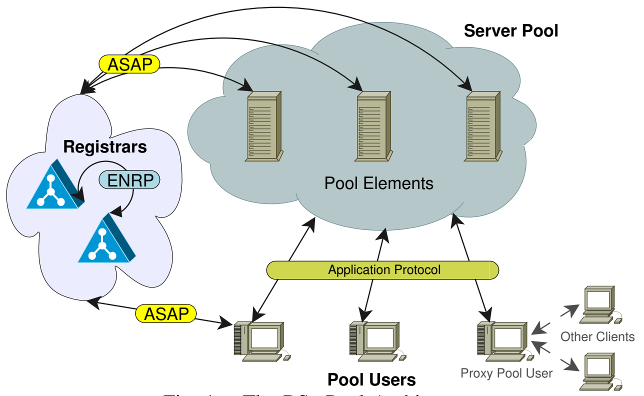

Fig. 1. The RSerPool Architecture

has a limited range, e.g. a company or organization; RSerPool does not intend to scale to the whole Internet. Nevertheless, it is assumed that PEs can be distributed worldwide, for their service to survive localized disasters (e.g. earthquakes or floodings).

A client is called *pool user* (PU) in RSerPool terminology. To use the service of a pool given by its PH, a PE has to be selected. The selection works in two stages: first, an arbitrary PR of the operation scope is asked for a *handle resolution* of the PH to a list of PE identities. This communication between PU and PR uses the Aggregate Server Access Protocol (ASAP [13], [17]). The PR selects the requested list of PE identities using a pool-specific selection rule, called *pool policy*. The PU writes this list into its local cache, denoted as PU-side cache, and selects again one PE for its communication. Subsequent handle resolutions may be directly satisfied from the cache, until its entries time out. This timeout is called *stale cache value*. Using a stale cache value of zero, every handle resolution must query a PR.

Adaptive and non-adaptive pool policies are defined in [18]; for this paper the relevant policies are non-adaptive Round Robin (RR) and Random (RAND), Weighted Round Robin (WRR) (integer weights specify how many times per round robin round a PE has to be selected), Weighted Random (WRAND) (weights specify the PEs' proportional selection probabilities) and the adaptive policy Least Used (LU). LU selects the least-used PE, according to up-to-date load information. Round robin selection is applied among multiple least-loaded PEs [19], [20]. The definition of *load* is application-specific and could e.g. be the current number of users, band-

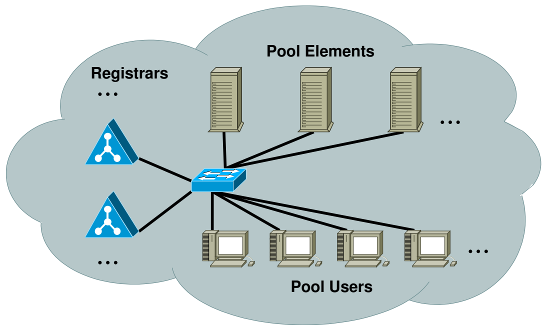

Fig. 2. Our Simulation Setup

width or CPU load.

For more detailed information on RSerPool, see [2]–[5], [9], [10], [19], [21], [22].

III. Quantifying an RSerPool System

The service provider side of an RSerPool system consists of a pool of PEs, using a certain server selection policy. Each PE has a request handling *capacity*, which we define in the abstract unit of calculations per second. Depending on the application, an arbitrary view of capacity can be mapped to this definition, e.g. CPU cycles, bandwidth or memory usage. Each request consumes a certain amount of calculations, we call this amount *request size*. A PE can handle multiple requests simultaneously, in a processor sharing mode as commonly used in multi-tasking operating systems.

On the service user side, there is a set of PUs. The amount of PUs can be given by the ratio between PUs and PEs (PU:PE ratio), which defines the parallelism of the request handling: the higher the PU:PE ratio, the more requests that have to be simultaneously handled by the PEs. Each PU generates a new request in an interval denoted as *request interval*. Requests are queued in the *request queue* and are sequentially assigned to PEs selected by the RSerPool mechanisms (see section II).

Using the definitions above, it is now possible to give a formula for the system's utilization:

$$\mathrm{sysUtil} = \mathrm{puToPERatio} * \frac{\frac{\mathrm{reqSize}}{\mathrm{reqInt}}}{\mathrm{peCapacity}} \qquad (1)$$

The load fraction generated by a single PU is given by the following formula:

$$\mathrm{puLoad} = \frac{\mathrm{reqSize}}{\mathrm{reqInt} * \mathrm{peCapacity}} \qquad (2)$$

In summary, the workload of a RSerPool system is given by the three dimensions (1) PU:PE ratio, (2) request interval and (3) request size. In a well-designed client/server system, the amount and capacities of servers are provisioned for a certain *target system utilization*, e.g. 80%. That is, by setting any two of the parameters, the value of the third one can be calculated using equation 1.

To evaluate the performance impacts of parameter variation, we define the system utilization given in equation 1 as performance metric: the better the system utilization, the more revenue is gained from the available server resources.

IV. Our Simulation Model

For our performance analysis, we have developed a simulation model [3]–[5] using OMNeT++ [23], containing full implementations of the protocols ASAP [17] and ENRP [12], a PR module and PE and PU modules modelling the request handling scenario defined in section III.

Network delay only becomes significant when the request sizes and intervals have the network delay's order of magnitude. This case is very unlikely for the load distribution scenarios of this paper. Therefore, we omit network delay here. The latency of the pool management by PRs is also negligible, as we show in our paper [19].

Since our goal is a generic parameter sensitivity analysis being independent of specific applications, we use negative exponential distribution for request intervals and request sizes and a target system utilization (see section III) of 80%. The PU-side handle resolution cache is turned off (*stale cache value* set to 0s). For the LU policy, we define *load* as the current amount of simultaneously handled requests; for the WRAND policy, we define *weight* as the PE's capacity. The average capacity of a PE is 10^6 calculations/s; unless otherwise specified, we use 10 PEs. The simulation runtime is 120 minutes; each simulation has been repeated 12 times with different seeds to ensure statistical accuracy.

The amount of PRs has been set to 1, since this parameter does not significantly affect the results; PR synchronization via ENRP only introduces the delay of the network. For this paper, we neglect congestion and failure scenarios of these connections, because we assume ENRP connections to be highly reliable, due to the usage of SCTP multi-homed associations [19]. Furthermore, they are established in controlled networks (e.g. of a company or an organization) where QoS mechanisms could be applied easily.

For the statistical post-processing of our results, we used R Project [24] for the computation of 95% confidence intervals and plotting.

V. Results

A. General Workload Parameter Effects

In our first simulation, we examine the performance impact of varying the three workload parameters: PU:PE ratio, request size (normalized by the PE capacity) and request interval on a system designed for a target utilization of 80% and using PEs of equal capacity (homogeneous scenario). For our simulation set, we varied the PU:PE ratio r from 1 to 20 for request size:PE capacity ratios s from 1 to 100. For each pair of both values, the request interval can be calculated based on equation 1 described in section III:

$$\mathrm{reqInt} = \frac{\mathrm{puToPERatio} * \mathrm{reqSize}}{\mathrm{targetSysUtil} * \mathrm{peCapacity}}$$

That is, our simulation covers all dimensions of the workload parameter space.

As shown, the PU:PE ratio r giving the degree of parallelism in request handling has a significant impact on the utilization: at $r = 1$, the utilization is at 53% for the RAND policy and at about 65% for RR. Using LU, it nearly reaches 80%. The utilization difference for the policies becomes significantly smaller when r increases: for $r = 5$, the difference is about 6% and for $r = 10$, it decreases to about 3%.

The reason for this behaviour is the amount of simultaneous requests processed by the PEs: at $r = 1$, there should be exactly one PE for every PU. That is, each PU expects to get a PE exclusively, which processes its requests during 80% (target utilization) of its runtime (see equation 2). Each time the "wrong" PE is selected for a request, one PE is idle while another one has to split up its capacity to handle two requests

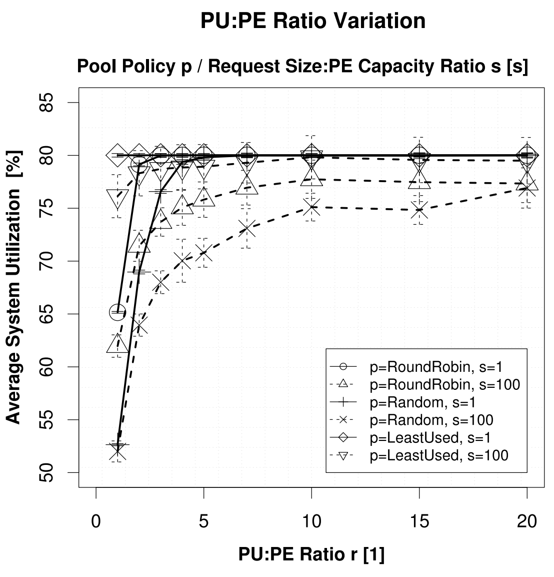

Fig. 3. Workload Parameter Variation

simultaneously. Obviously, this behaviour is most frequent when PEs are randomly chosen. For RR selection, the PE just selected should be chosen again only after having used every other PE before. This method already achieves a significant improvement over RAND. Finally, LU has the knowledge of the PEs' current load states; therefore - except for the rare cases of simultaneous selection - the least-loaded PE can be used. This is the reason for the good performance of LU.

Observing the utilization for a variation of the request size:PE capacity ratio s, only minor differences are shown. Even for a change of two orders of magnitude as presented in figure 3, the utilization between $s = 1$ (solid lines) and $s = 100$ (dotted lines) only decreases by 1%-2% for LU, about up to 4% for RR and up to about 7% for RAND. The reason for the small decrement is that longer requests increase the impact duration of the selection decision. That is, the longer the requests, the longer the impacts of a non-optimal selection decision. Clearly, the probability of a non-optimal assignment is highest for RAND and lowest for LU, explaining the performance differences between these policies.

For high PU:PE ratios and $s = 100$ (dotted lines), it can be observed that the utilization for RAND slightly exceeds the value of RR (at $r = 14$) and even LU (at $r = 18$): the frequency of PE load changes increases due to the rising amount of concurrently handled requests. Trying to assume which PE is a good choice to select - based on load information for LU or by list position for RR - becomes more and more inappropriate.

In summary, the PU:PE ratio r has been identified as the most influential workload parameter; small values of r are critical, since the per-PU workload is highest here. Larger values of r introduce more parallelism, decreasing the penalty on choosing the "wrong" PE. While LU provides superior utilization in case of low r, the performance of RAND and RR improves with rising r. A large job size:PE capacity ratio s extends the impact duration of inappropriate selection decisions, leading to a slightly decreased performance.

B. Deterministic Heterogeneous Scenarios

After having examined the basic system behaviour on workload parameter changes in scenarios of equal server capacities, we now have a look at heterogeneous scenarios. In all scenarios, we vary the composition of server capacities in the system but keep the overall system capacity constant. That is, the capacities are always *normalized*; in case of "good" load distribution, we expect the utilization not to decay when the capacity distribution differs more and more from the homogeneous case.

We ran simulations for a PU:PE ratio r ranging from 1 to 10 and a request size:average PE capacity ratio s ranging from 1 to 100. Our utilization curves show a carefully chosen subset of our results that presents the essential effects.

a) Linear Capacity Distribution: In our first capacity distribution scenario, we linearly vary the PE capacities using the factor γ. This factor γ defines the capacity ratio between the PE of least capacity and the PE of maximum capacity, i.e. for $\gamma = 3$ the fastest server has three times more capacity than the slowest one. The capacities of the other servers are linearly distributed between the minimum and maximum capacity. Such linear capacity scenarios are likely when different generations of servers are used in a pool.

Figure 4 presents the utilization results for a PU:PE ratio r of 1 (left part), 3 (middle part) and 10 (right part). The solid lines represent $s = 1$, the dotted lines $s = 100$. As expected, the utilization results for LU and WRAND are almost unaffected by changes of γ. Furthermore, only the small gap between the curves for $s = 1$ and $s = 100$ as already explained in section V-A can be observed. The utilization of RR and RAND decreases with γ, since for these policies the selection decision is not based on the PEs' capabilities (i.e. capacity or current load).

For rising r, the difference between the four policies becomes significantly smaller: as explained in section V-A, the penalty on "bad" selection decisions becomes smaller with r. At $r = 10$, the utilization difference between the four policies is less than 10% until $\gamma = 5$; for higher values of γ it increases for RR and RAND.

Clearly, a linear capacity scenario is simple, due to the systematic distribution of capacities. A PU being served by a slow PE will be served by a fast one soon; in average, the difference to a homogeneous scenario becomes small. While LU clearly achieves the best utilization, the difference to the other policies becomes small for higher r; the performance of RR and RAND decays for increasing γ, due to their lack of PE capability knowledge. Nevertheless, for sufficiently small γ RR provides a better performance than WRAND.

b) Fast Servers: A more irregular scenario than linear capacity distribution is to have one or more designated fast servers. Such a scenario is likely when there are a few high-capacity servers to do the usual work and a set of older ones to provide failure protection by redundancy. The utilization results for using one powerful PE (1 of 10) is presented in figure 5 for the PU:PE ratio $r = 1$ (left part), $r = 3$ (middle part) and $r = 10$ (right part); the utilization of one third of the servers (3 of 9) being powerful ones is shown in figure 6. The solid lines represent $s = 1$, the dotted lines $s = 100$. We varied the capacity ratio κ between fast and slow PEs, e.g. $\kappa = 5$ means a fast PE has five times the capacity of a slow one. Note, that the capacities are normalized to keep the system capacity constant.

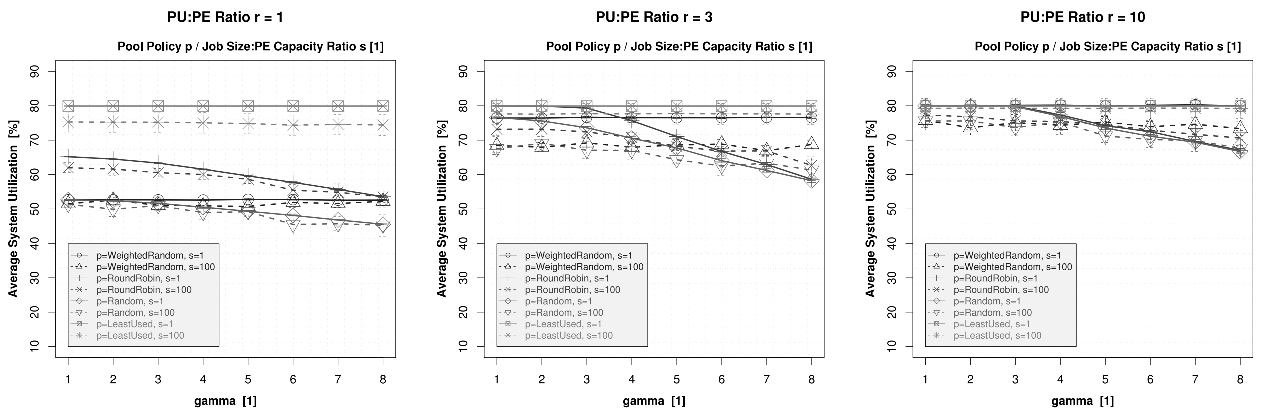

Fig. 4. Linear Capacity Distribution Scenario

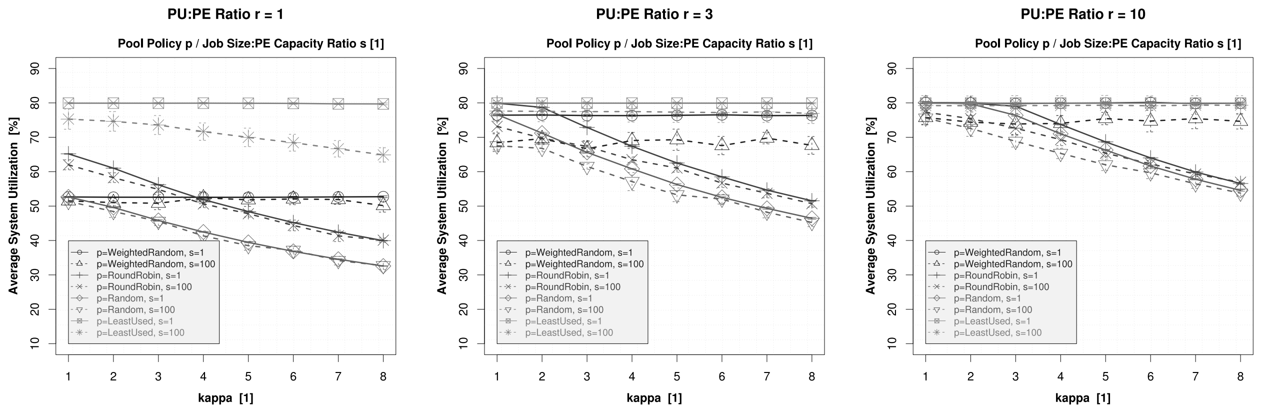

Fig. 5. One Fast Server Scenario

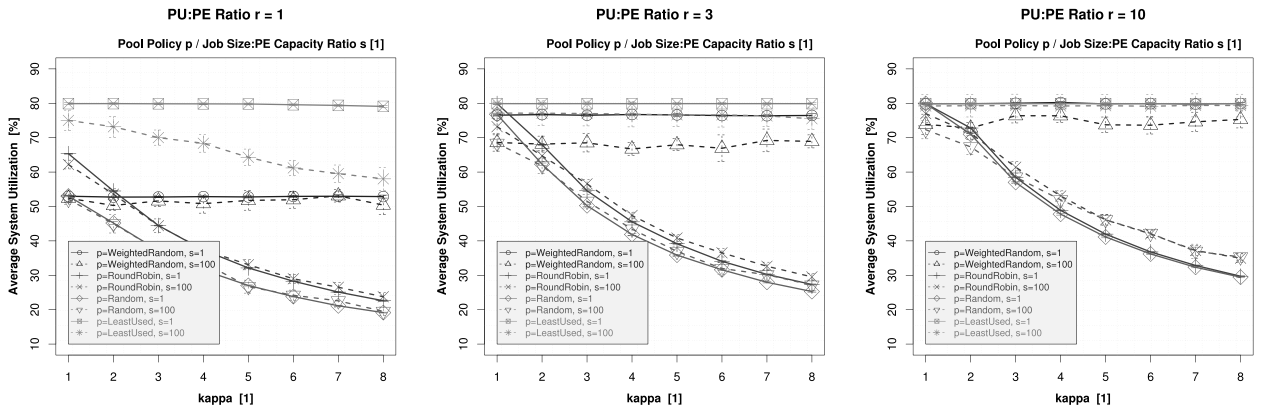

Fig. 6. One Third Fast Servers Scenario

Analysing the results for $r = 1$, the utilization for LU at $s = 1$ stays constant for rising κ for both, one fast PE and one third fast PEs. But for $s = 100$, a significant decay can be observed: LU bases its selection decision on PE load rather than capacity. That is, a slow but lightly-loaded PE is preferred over a faster but higher-loaded one. If a long-lasting request (i.e. s large) is mapped to a slow PE, this PE "holds" the PU, resulting in a lower system utilization. Clearly, the utilization decay is stronger in the one third fast servers scenario: due to capacity normalization, more capacity is incorporated in the set of fast PEs. That is, the negative effect of mapping a PU to a slow PE becomes more visual.

While the utilization for WRAND is hardly affected by increasing κ, RR and RAND show a significant decrement. Unlike linear distribution, the PU's penalty on using a slow PE becomes more significant: in the fast server scenarios the chance for getting mapped to a fast PE the next time is much lower. Due to the fact that more capacity is clustered in the set of fast servers of the one third fast PEs scenario than in the one fast server scenario, the utilization decay becomes more visible in a scenario of multiple fast PEs.

As expected for rising PU:PE ratio r, the utilization difference between the policies becomes smaller. But unlike linear capacity distribution, the utilization improvement for RR and RAND remains small: due to the highly uneven capacity distribution of the fast servers scenarios, the penalty on making inappropriate selection decisions remains high. Only for very small settings of κ RR provides a slightly better utilization than WRAND.

In summary, only LU and WRAND are suitable in scenarios having designated fast servers. RR only achieves a performance gain when κ is sufficiently small, i.e. the scenario is nearly homogeneous.

C. Randomized Heterogeneous Scenarios

After having examined deterministic heterogeneous capacity scenarios, we now have a look at two randomized ones. In our first randomized capacity distribution scenario, the server capacities are uniformly randomized in an interval described by the factor ϑ. This factor ϑ gives the ratio between the minimum and maximum capacity, e.g. a setting of $\vartheta = 2$ results in a uniform capacity selection

$$c \in_R [\text{minCapacity}, \vartheta * \text{minCapacity}] \subset \mathbb{N}.$$

The resulting capacities are normalized by using an appropriate minimum capacity, i.e. the average system capacity remains constant for all settings of ϑ.

In our second randomized capacity distribution scenario, we use PE capacities randomly chosen using a normal distribution for a given average capacity (10^6 calculations/s) and standard deviation given by $\eta * \text{average capacity}$ (i.e. $\eta * 10^6$). Obviously, capacities cannot be negative, therefore we truncated the capacity selection by enforcing a lower limit (10^4 calculations/s). Finally, the PE capacities have been normalized to keep the system capacity constant.

The utilization results for the uniform distribution scenario are shown in figure 7 for a PU:PE ratio r of 1 (left side) and 3 (right side). The solid lines represent $s = 1$, the dotted lines $s = 100$. Figure 8 shows the corresponding results for the truncated normal capacity distribution scenario.

Mainly, the results reflect the observations of the deterministic distribution scenarios of section V-B: the utilizations of RR and RAND decay while WRAND is hardly affected by changing the distribution parameters ϑ or η. For scenarios sufficiently near to homogeneous distribution (i.e. the distribution parameter is small) RR achieves a slightly better performance than WRAND. However for LU, a stronger variance of the utilization can be observed in the form of larger confidence intervals for $r = 1$. The reason is that although the average system capacity remains constant due to normalization, it can differ from this average due to randomization. LU may use low-capacity PEs when their load value is low and $r = 1$ is the most critical setting due to highest per-PU load. Therefore, the observed effect is strongest here.

In summary, randomized capacities do not induce significant changes to the system's behaviour in comparison to deterministic scenarios.

D. Weighted Round Robin Policy

The RSerPool policies draft [18] defines the Weighted Round Robin (WRR) policy. Since RR provides a performance gain over RAND, one might expect that WRR results in a performance gain over WRAND in heterogeneous scenarios. Unfortunately, this assumption is wrong for the following reasons. The first problem is the systematic selection. Consider 3 PEs with weights $w_{\text{PE1}} = 2$, $w_{\text{PE2}} = 3$ and $w_{\text{PE3}} = 10$. In this case, the selection order of a round robin round could be as follows:

$$\underbrace{\text{PE}_1 \Rightarrow \text{PE}_2 \Rightarrow \text{PE}_3}_{2\,\text{times}} \underbrace{\Rightarrow \text{PE}_2 \Rightarrow \text{PE}_3}_{\text{once}} \underbrace{\Rightarrow \text{PE}_3}_{7\,\text{times}}$$

Obviously, after PE_1 and PE_2 have been selected the given number of times, WRR hammers PE_3. That is, when a PR selects PE_3 for 8 different PUs, 8 PUs simultaneously use PE_3 while other PEs may remain idle.

The second problem of WRR is that the weights have to be integers: let PE_4 have 3.5 times the capacity of PE_5 and PE_6 1.75 times the capacity of PE_4. In this case, the resuling weights could be $w_{\text{PE4}} = 4$, $w_{\text{PE5}} = 7$ and $w_{\text{PE6}} = 14$. In the end of the selection round, PE_6 would be consecutively selected 8 times. Clearly, the higher the factor to extend the weights to integers, the more worse becomes the selection problem described above.

Simulations for the heterogeneous scenarios described in section V-B and section V-C have shown that WRR's performance is much worse than the results for RR and even RAND (except for the homogeneous case where WRR is equal to RR). Since WRR is completely useless in the described scenarios, we omitted curves for this policy.

VI. Conclusions and Future Work

In this paper, we have first quantified the basic workload parameters of RSerPool systems (PU:PE ratio, request size and interval) and defined system utilization as the performance metric for our analysis. We then provided a simulative analysis of the general effects on workload parameter changes for a scenario of equal server capacities. Here, the PU:PE ratio describing the parallelism in request handling has been identified as the most critical parameter. Low parallelism implies high per-PU load, therefore the penalty for "wrong" PE selection decisions becomes high, leading to decreased system utilization.

After examining the general system behaviour, we have analysed the load distribution performance of multiple deterministic and randomized server capacity distribution scenarios and provided fundamental insights into the properties of the LU, RR, WRAND and RAND selection policies. Finally, we reasoned why the WRR policy is unusable for realistic scenarios.

The next step of our work is to verify our simulation results in real-life network scenarios. Based on our prototype implementation of RSerPool [6], [9], [25], we are going to build a lab test scenario and finally also want to analyse large-scale scenarios using the PlanetLab [26]. Our goal is to transfer the theoretical insights of our simulations to reality, providing guidelines for designing and tuning RSerPool systems and promoting standardization and deployment of RSerPool.

References

[1] M. Tüxen, Q. Xie, R. Stewart, M. Shore, J. Loughney, and A. Silverton, "Architecture for Reliable Server Pooling," IETF, RSerPool WG, Internet-Draft Version 10, Jul 2005, draft-ietf-rserpool-arch-10.txt, work in progress.

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

[3] T. Dreibholz and E. P. Rathgeb, "On the Performance of Reliable Server Pooling Systems," in *Proceedings of the 30th IEEE Local Computer Networks Conference*, Sydney/Australia, Nov 2005.

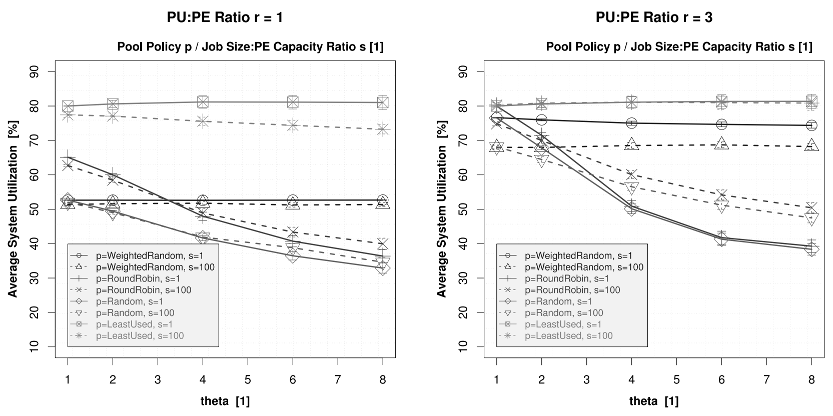

Fig. 7. Uniform Randomized Capacity Distribution Scenario

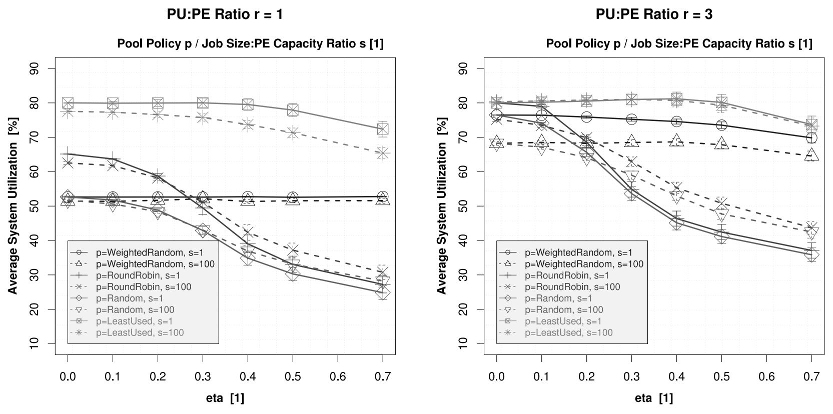

Fig. 8. Truncated Normal Randomized Capacity Distribution Scenario

[4] ——, “RSerPool - Providing Highly Available Services using Unreliable Servers,” in *Proceedings of the 31st IEEE EuroMirco Conference on Software Engineering and Advanced Applications 2005*, Porto/Portugal, Aug 2005.

[5] T. Dreibholz, E. P. Rathgeb, and M. Tüxen, “Load Distribution Performance of the Reliable Server Pooling Framework,” in *Proceedings of the 4th IEEE International Conference on Networking 2005*, Saint Gilles Les Bains/Reunion Island, Apr 2005.

[6] T. Dreibholz and E. P. Rathgeb, “An Application Demonstration of the Reliable Server Pooling Framework,” in *Proceedings of the 24th IEEE Infocom 2005*, Miami, Florida/U.S.A., Mar 2005.

[7] T. Dreibholz, “Applicability of Reliable Server Pooling for Real-Time Distributed Computing,” University of Duisburg-Essen, Internet-Draft Version 00, Jul 2005, draft-dreibholz-rserpool-applic-distcomp-00.txt, work in progress.

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

[9] “Thomas Dreibholz’s RSerPool Page,” http://tdrwww.exp-math.uni-essen.de/dreibholz/rserpool.

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

[11] U. 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] Q. Xie, R. Stewart, M. Stillman, M. Tüxen, and A. Silverton, “Endpoint Name Resolution Protcol (ENRP),” IETF, RSerPool WG, Internet-Draft Version 12, Jul 2005, draft-ietf-rserpool-enrp-12.txt, work in progress.

[13] R. Stewart, Q. Xie, M. Stillman, and M. Tüxen, “Aggregate Server Access Protocol (ASAP) and Endpoint Handlespace Resolution Protocol (ENRP) Parameters,” IETF, RSerPool WG, Internet-Draft Version 09, Jul 2005, draft-ietf-rserpool-common-param-09.txt, work in progress.

[14] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, “Stream Control Transmission Protocol,” IETF, Standards Track RFC 2960, Oct 2000.

[15] A. Jungmaier, E. Rathgeb, and M. Tüxen, “On the Use of SCTP in Failover-Scenarios,” in *Proceedings of the SCI 2002, Volume X, Mobile/Wireless Computing and Communication Systems II*, vol. X, Orlando/U.S.A., Jul 2002.

[16] A. Jungmaier, M. Schopp, and M. Tüxen, “Performance Evaluation of the Stream Control Transmission Protocol,” in *Proceedings of the IEEE Conference on High Performance Switching and Routing*, Heidelberg/Germany, June 2000.

[17] R. Stewart, Q. Xie, M. Stillman, and M. Tüxen, “Aggregate Server Access Protcol (ASAP),” IETF, RSerPool WG, Internet-Draft Version 12, Jul 2005, draft-ietf-rserpool-asap-12.txt, work in progress.

[18] M. Tüxen and T. Dreibholz, “Reliable Server Pooling Policies,” IETF, RSerPool WG, Internet-Draft Version 01, Jun 2005, draft-ietf-rserpool-policies-01.txt, work in progress.

[19] T. Dreibholz and E. P. Rathgeb, “Implementing of the Reliable Server Pooling Framework,” in *Proceedings of the 8th IEEE International Conference on Telecommunications 2005*, Zagreb/Croatia, Jun 2005.

[20] 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.

[21] ——, “Das rsplib–Projekt – Hochverfügbarkeit mit Reliable Server Pooling,” in *Proceedings of the LinuxTag 2005*, Karlsruhe/Germany, Jun 2005.

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

[23] “OMNeT++ Discrete Event Simulation System,” http://www.omnetpp.org.

[24] R Development Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2005, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

[25] T. Dreibholz, “An Overview of the Reliable Server Pooling Architecture,” in *Proceedings of the 12th IEEE International Conference on Network Protocols 2004*, Berlin/Germany, Oct 2004.

[26] “PlanetLab: Home,” http://www.planet-lab.org.