ABSTRACT
Server Load Balancing (SLB) is a popular technique to build high-availability web services as offered from Google and Amazon for example. Credit based load balancing strategies have been proposed in the literature where the back end servers dynamically report a metric called Credit to the Load Balancer (LB) which reflects their current capacity. This enables the LB to adapt the load balancing strategy. The benefit of Credit based SLB has been shown by simulations, but up to now, it is not used in productive systems, since efficient implementations were missing.

This paper presents the evaluation of an implementation of Credit based SLB, the so-called Self-Adapting Load Balancing Network (salbnet). We evaluate salbnet for a cluster of web servers. The measurements are done with a representative workload based on a Wikipedia trace and confirm the benefit of the self-adapting load balancing approach.

KEY WORDS
Cluster Computing, Load Balancing, Performance evaluation

1 Introduction
Dispatcher based SLB as shown in Figure 1 is an efficient way to provide scalable, flexible and fault tolerant services.

Figure 1. Dispatcher based SLB scenario.

Most of well known SLB algorithms like Round Robin (RR) or Least Connection (LC) use weights to perform better in heterogeneous systems. These are the Weighted Round Robin (WRR) and Weighted Least Connection (WLC) algorithms. Determining weights correctly is considered to be a critical performance issue as shown in [5, 15].

Therefore, Credit based load balancing has been introduced in the literature [5]. The reported Credit values represent the actual remaining processing capabilities of a back end server. Additional weight information are not required anymore with this approach.

Simulations in [5] have already shown that the Credit based SLB approach can gain significant performance enhancements compared to the traditional SLB algorithms. But there was still a lack of a prototype implementation and validating measurements. This paper presents the implementation of salbnet and has the following goals:

Validation of Simulation Confirm the results of previous simulations with measurements. Therefore, evaluate salbnet against traditional algorithms like Round Robin.

Application Independence Proof that the Credit based approach can be implemented independent from specific back end server applications.

In the next section related work is presented. It follows a short introduction of the Credit based approach in Section 3 providing an overview of the concept and new attempts for improvements like the Highest Credit scheduling algorithm. Section 4 presents the architecture of the salbnet implementation, Section 5 presents its detailed evaluation with realistic workloads. Finally, we give a conclusion.

2 Related Work

In [4] a server state based dispatching approach is proposed, which suggests to select the Least Loaded (LL) back end server and that the back end server should report load metrics periodically to the dispatching LB. In [3] a similar method is proposed, using an agent and optionally Push the load information to the LB. Implementation issues were not discussed.

The LB scheduler presented in [1] selects the back end servers which still have processing capacities randomly. The processing capabilities are calculated in an adaptive manner and require statically defined so called
Performance Targets for each server. The idea is to increase the fraction of accepted new client requests whenever the smoothed performance measures of the web servers are below their targets, and to decrease it when above the target. The presented simulation results show that the concept performs better than static RR, but the authors give no comparison with WRR and only one of the proposed metrics is used: the server utilization based on the number of handled sessions.

More recently, the Dynamic Feedback Minimum Weight Equalization [8] combines several coarse grain load metrics like for example the CPU utilization, memory usage, I/O rates of the back end servers and the network utilization, to calculate the back end server weights. In opposition to this approach, we only use the currently observed communication accept behavior of the application. Measurements with a synthetic workload show the benefit of the Dynamic Feedback Minimum Weight Equalization algorithm over the WLC implementation of the Linux Virtual Server (LVS) [13].

3 Credit-based SLB

The Credits represent the maximum number of connections which a back end server is able to handle and are reported from the back end servers to the LB. They are calculated from information of the transport layer and hence are application independent.

The Load Balancer considers a back end server only if Credits are available for this server. Therefore, the back end servers limit the work they will get in a Self-Adapting manner.

3.1 Credit Metrics

For common TCP/IP based applications, the fill level of the TCP backlog queue was introduced as credit metric. There are the following two different Credit types introduced in [5]:

**Hard Credits** The Hard Credits represent the maximum number of connections which a back end server is currently able to handle. They are calculated as the difference between the maximum queue length and the current queue length:

\[
\text{Hard Credits} = (\text{SYN}_{\text{max}} + \text{ACK}_{\text{max}}) - (\text{SYN}_{\text{curr}} + \text{ACK}_{\text{curr}}) \quad (1)
\]

If more requests arrive at the server, the back end server starts to Drop incoming connections.

**Soft Credits** The Soft Credits represent the recommended number of connections which the back end server currently wants to receive from the LB. The Soft Credits take, instead of the current value, the median of the history of the Incomplete Connection Queue (\(\text{SYN}_{\text{median}}\)) and the Completed Connection Queue (\(\text{ACK}_{\text{median}}\)) into account as shown in Equation 2.

The median is calculated regarding the last 16 values to handle short arising high request rates.

\[
\text{Soft Credits} = (\text{SYN}_{\text{max}} + \text{ACK}_{\text{max}}) - (\text{SYN}_{\text{median}} + \text{ACK}_{\text{median}}) \quad (2)
\]

The simulations in [5] show that Soft Credits avoid an early overloading of a server, since not all Credits are reported. Hard Credits are required to reduce the total number of Drops, because servers report their upper limits of available resources.

3.2 Credit Reporting Algorithms

The basic Credit collecting and reporting process is shown in Figure 2. A process on the back end server intercepts system calls from the application and collects the Credit metric. Depending on the chosen reporting algorithm, Credits are calculated and reported from the back end server to the LB.

In the simulation results presented in [5] the Dynamic Pressure Relieve (DPR) performs best:

**DPR** The DPR reporting algorithm utilizes different reporting marks: Soft Credits are dynamically reported in relation to the (amount of processed) Credit metric (data). Hard Credits are reported along with Soft Credits, but at least after a fixed defined minimum report mark value is reached.

The report marks are usually given in percent and calculated (dynamically) from the current (soft mark) and the maximum (hard mark) backlog length. The number of accept() system calls, of the back end server application, is compared against the report marks. If the soft mark is reached, a report of both Soft Credits and Hard Credits is issued. In addition, the frequency of the Hard Credits reports is constrained by the fixed hard mark. In effect, these additional Hard Credits are send every arbitrary minimum number of accept() system calls.

The reports containing both, Soft Credits and Hard Credits, are issued more often with fewer entries in the backlog and less often with a filled TCP backlog.
3.3 DPR-Quantize (DPR-Q)

Resulting from further development of the DPR algorithm a variation called DPR-Q is proposed.

**DPR-Q** The DPR-Q algorithm reports Credits after a minimum report mark value is reached. With each report the DPR-Q determines dynamically, in relation to the (amount of processed) Credit metric (data), if either both Soft Credits and Hard Credits or only Hard Credits are reported.

For example, with fewer entries in the backlog both Soft Credits and Hard Credits are reported and with a filled TCP backlog only Hard Credits are reported.

Compared to the DPR algorithm, the DPR-Q does (always) issue a Credit report after a fixed amount of processed Credit metric data. The Credit reports of the DPR are quantized using the minimum report mark value. Therefore, compared to the DPR algorithm, the DPR-Q algorithm reduces the number of Credit reports for low workloads, but increases them for higher load situations. The idea behind is to avoid peak situations where Credits may reported too rarely, despite the back end servers may have been Credits to report.

3.4 LB Scheduling Algorithms

In [5] the scheduling algorithm Next Credits was presented which choses the next available server in a Round-Robin manner. First, the set of back end servers with Soft Credits is checked. If no server with reported Soft Credits is available, the set of servers with reported Hard Credits is considered. If no server is found, the Load Balancer may Early Drop further incoming connections.

Additionally, we implemented a Highest Credit scheduling which choses the server with the highest number of reported credits.

4 Implementation Issues of salbnet

While simulations have shown that salbnet is a promising load balancing concept [5], an efficient implementation was missing. Since the Load Balancer has additionally to cope with collecting the Credit metric, there is the possibility to loose all the benefits of salbnet due to a poor implementation.

The following Sections describe the overall architecture as well as details of each component of the salbnet implementation.

4.1 Architecture Overview

We will present a salbnet implementation which is based on Linux as well as LVS [13] and consists of the following components:

- **salbd** The salbd daemon is the daemon for Credit metric collecting and reporting of the calculated Credits, which runs on the LB as well as on the back end servers.
- **LVS Scheduler** The LVS Scheduler is a kernel module which runs on the LB and implements the Credit based scheduling algorithms.
- **libnethook** The libnethook library is a library for hooking into (socket) system calls used on the back end servers.
- **libnetmsg** The libnetmsg library is a network abstraction library for sending messages over Ethernet and InfiniBand.

Figure 3 shows all the salbnet components together. The gray components in Figure 3 are already available as part of the Linux operating system resp. the OpenFabrics Enterprise Distribution (OFED) [9] stack. In contrast the blue components in Figure 3 are part of the salbnet implementation and will be presented in detail in the following sections.

4.2 libnethook

The libnethook library is a lightweight library supposed to run on the back end servers and implements the (socket) system call interception required for the Credit reporting. Figure 4 shows the interaction of the libnethook library using the example of being utilized by the salbd to intercept the Apache HTTP Server [10] accept() system calls.

![Diagram of system call interception](image)

Figure 4. libnethook example system call interception.

The libnethook library is implemented as LD_PRELOAD library and utilizes shared memory as Inter-Process Communication (IPC) to keep the information of the system call interception from the application (threads and processes) in sync with the actual processing by the controlling program (salbd).

The LD_PRELOAD approach has the following advantages:

- The application does not need to be modified to be able to intercept (socket) system calls (as long as the application is not statically linked). 

251
The LD_PRELOAD approach takes place in user space. Hence, provides a better performance compared to similar solutions which require context switches into the kernel space.

4.3 libnetmsg

The libnetmsg library is a lightweight communication library supposed to run on both, the back end servers and on the LB. It is used for Credit reporting and hides the differences of the underlying networks.

The libnetmsg library is based on the OFED [9] and supports Remote DMA (RDMA), when used on InfiniBand, as well as IP Version 6 (IPv6), when used on Ethernet.

The implementation of the libnetmsg library is non-blocking and event based to scale as required. It uses epoll [6] on Linux.

An example for the high level of abstraction implemented by the libnetmsg is the accept () handling. The InfiniBand (IB) accept function includes several OFED [9] specific function calls. For example, the Protection Domain (PD) is allocated and the Completion Queue (CQ) is created, both on the Verbs layer. Afterwards, the IB RDMA memory region is registered and the RDMA Communication Manager (CM) creates the Queue Pair (QP) depending on the type. Finally, the RDMA CM function rdma_accept () is called and used to exchange the IB RDMA memory region (and rkey) information through the RDMA CM private data, while establishing the connection. The RDMA memory region (and rkey) information are appended to the RDMA CM private data (fields).

Once the client has received the RDMA memory region information (and rkey) the client can start sending messages, for example reporting Credits, to the server in an One-Sided Communication (OSC) manner.

4.4 salbd

The salbd is supposed to run on the back end servers (in client mode) and on the LB (in server mode) (see Figure 3). It calculates and reports resp. collects Credits. Both, the client and the server utilize the libnetmsg library (as described in Section 4.3).

salbd Client The salbd client implements the Credit reporting algorithm described in Section 3.2. Credit reports depend on the configured reporting algorithm and are caused by the interception of the application(s) system calls through libnethook (as described in Section 4.2).

The salbd client implementation uses some hard coded parameters (#define) for the Credit reporting as described in Table 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Mark</td>
<td>50%</td>
<td>Defines the report marks thresholds.</td>
</tr>
<tr>
<td>Report Mark Minimum</td>
<td>10%</td>
<td>Defines the lower limit for report marks.</td>
</tr>
<tr>
<td>Metric History Size</td>
<td>16</td>
<td>Defines how many Credit metric history values are kept for median calculations.</td>
</tr>
</tbody>
</table>

salbd Server The salbd server listens on two connections, one control connection to register new incoming back end servers dynamically and another data connection for the actual Credit reports. These two connections are required to be able to implement the IB specific RDMA
Credit reporting, while still being able to register back end servers dynamically.

The communication between the salbd server and the LVS Scheduler is implemented through ioctl() function calls on the character device /dev/salbnet. The salbd server registers the (RDMA) memory region which receives Credit reports from the back end servers at the LVS Scheduler through ioctl() function calls. After the initialization the salbd server loops and waits (listen()) for incoming back end server registrations. Once a back end server registers a segment of the (RDMA) memory region is assigned and used for the Credit reporting.

4.5 LVS Scheduler

The LVS Scheduler is a scheduling kernel module for the LVS [13] and supposed to run on the LB (see Figure 3). This salbnet scheduler module implements the actual Credit based scheduling algorithms (as described in Section 3.4).

Furthermore, the LVS Scheduler needs to provide an interface for the salbd for communication to be able to receive the reported Credits from user space. Therefore the salbnet LVS scheduler provides a memory region mapped to user space accessed by the salbd (as described in Section 4.4) and used for the back end server registration and the management of the reported Credits.

5 Measurements and Evaluation

To evaluate the Self-Adapting Load Balancing Network, a dispatcher based SLB scenario is setup similar to the one shown in Figure 1. This setup is two armed, NAT based and uses route path as introduced in [2]. LVS is used as Load Balancer with the salbnet reporting algorithms DPR and DPR-Q and the salbnet scheduling algorithms Next Credits and Highest Credits.

We present measurement results to validate the simulation results from [5] with our prototype to show that a self-adapting approach based on the presented Credit metric performs better than the state of the art balancing strategies. Further, we want to evaluate whether the new presented DPR-Q reporting algorithm and the new Highest Credits scheduling algorithm perform better. For the measurements we use the servload benchmark [14] and a Wikipedia trace as workload.

5.1 Outcomes and Metrics

The service of the load balanced web server cluster system is to answer HTTP requests received from the web client. One possible outcome is the successful processing of the request including the successful transfer of a complete response. Another possible outcome could be failures in processing of the request for example due to an Overloaded Server, resulting in aborted requests and in wrong, incomplete or aborted responses. Finally, failures in the network connection result in aborted or incomplete requests and responses as well.

The metrics used as criteria to compare the performance of the different algorithms a are the median (First) Response Time \( r_{a,i} \) and the (Request) Errors \( e_{a,i} \). Each experiment is repeated \( n \)-times and the average of the median (First) Response Times \( \bar{r}_a \) and the average (Request) Errors \( \bar{e}_a \) are calculated. To compare these metrics for every algorithm \( a \) out of the set of tested algorithms \( A \) they are normalized relative to the worst algorithm per metric (as described in Equation 3 and 4). Both normalized metrics are multiplied resulting in the SLB Internet Service Provider (ISP) Penalty \( P_{ISP} (a) \) from Equation 6 (as similar introduced in [15]). This results in a maximal Penalty of 1 if an algorithm is worst for both metrics.

Further, we introduce the SLB Full Penalty which considers also the duration of a measurement. Given the mean duration \( \bar{t}_a \), we calculate the normalized duration \( \hat{t}_a \) (see Equation 5). The SLB Full Penalty \( P_{Full} (a) \) is defined as the product of the normalized Duration and the SLB ISP Penalty (see Equation 7). Again the maximal penalty of 1 is achieved if an algorithm is the worst of all compared in every of the three metrics ((First) Response Times, (Request) Errors, and Duration).

\[
\begin{align*}
\bar{r}_a &= \frac{1}{n} \sum_{i=1}^{n} r_{a,i} \\
\bar{e}_a &= \frac{1}{n} \sum_{i=1}^{n} e_{a,i} \\
\bar{t}_a &= \frac{1}{n} \sum_{i=1}^{n} t_{a,i}
\end{align*}
\]

\[
\begin{align*}
\hat{r}_a &= \frac{\bar{r}_a}{\max\{\bar{r}_a \forall x \in A\}} \quad (3) \\
\hat{e}_a &= \frac{\bar{e}_a}{\max\{\bar{e}_a \forall x \in A\}} \quad (4) \\
\hat{t}_a &= \frac{\bar{t}_a}{\max\{\bar{t}_a \forall x \in A\}} \quad (5)
\end{align*}
\]

\[
P_{ISP} (a) = \hat{r}_a \times \hat{e}_a \quad (6)
\]

\[
P_{Full} (a) = \hat{t}_a \times P_{ISP} (a) \quad (7)
\]

Furthermore, the CPU usage, the Load Average and the number of the established TCP connections are collected through the SNMP. The values are collected once a minute through SNMP v.1 from a shell script running on the LB. The impact of this monitoring is considered to be negligible.


Within this setup an instance of the English Wikipedia [12] based on a dump from 2008 is deployed. Both, the Wikipedia dump used on the back end servers, as well as the access traces used for replay, are available from [7]. Moreover, the traces are evaluated in [11].

The input for the preparation of the Wikipedia trace for the measurements is the one from 12. November 2007 (wiki.1194899823.gz). This trace is reduced to the
requests from the first 10 minutes of the log. Also a filter is applied to restrict the log to requests for common (upload) content (for example images) and requests for the English instance of Wikipedia. Finally, the trace is converted to the Common Log Format which is an input format of servload [14]. These preparations are done with the wikipedia2common.sh shell script which is part of the servload distribution release. The final sequence consists of 1,584,996 requests in total.

The total number of requests from the prepared Wikipedia trace, results in too much workload for the three back end web servers in the testbed. Therefore, the number of requests is reduced and three traces are generated which include every 8th, 16th resp. 32th request. The resulting numbers or requests are shown in Table 2. All three workloads represent Overload Situation where some of the requests will result in (Request) Errors.

Table 2. Number of requests from the first 10 minutes of the Wikipedia trace.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Number of Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>8th</td>
<td>198,125 requests</td>
</tr>
<tr>
<td>16th</td>
<td>99,063 requests</td>
</tr>
<tr>
<td>32th</td>
<td>49,532 requests</td>
</tr>
</tbody>
</table>

5.3 Measurement Environment

The corresponding measurement environment consists of five machines. Three Apache HTTP servers [10] in version 2.2.3 are used in the back end. Another machine is used for the LVS [13] LB with ipvsadm in version 1.24 (compiled with IPV6 in version 1.2.0). The LB and the three web server nodes as shown in Table 3 run CentOS Linux in version 5.7 with kernel 2.6.18-274.12.1.el5 and GCC in version 4.1.2.

Table 3. Hardware specifications of the web server cluster.

<table>
<thead>
<tr>
<th>Hostname</th>
<th>LB and Web Server 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Dual 1.8 GHz AMD Opteron 244</td>
</tr>
<tr>
<td>Memory</td>
<td>4 GB</td>
</tr>
<tr>
<td>IB HCA</td>
<td>2-Port Mellanox PCI-X MT23108 Infini-Host HCA</td>
</tr>
<tr>
<td>Hostname</td>
<td>Web Server 2</td>
</tr>
<tr>
<td>CPU</td>
<td>2.8 GHz Intel Pentium 4</td>
</tr>
<tr>
<td>Memory</td>
<td>4 GB</td>
</tr>
<tr>
<td>IB HCA</td>
<td>2-Port Mellanox PCIe MT25208 Infini-Host HCA</td>
</tr>
<tr>
<td>Hostname</td>
<td>Web Server 3</td>
</tr>
<tr>
<td>CPU</td>
<td>1.86 GHz Dual Core Intel Xeon 3040</td>
</tr>
<tr>
<td>Memory</td>
<td>4 GB</td>
</tr>
<tr>
<td>IB HCA</td>
<td>2-Port Mellanox PCIe MT25208 Infini-Host HCA</td>
</tr>
</tbody>
</table>

Further, we use a client machine running servload [14] in version 0.5. The client machine is a Dual 1.8 GHz AMD Opteron 244 with 4 GByte RAM, running Debian Linux in version 5.0.10 with kernel 2.6.26-2-amd64 and GCC version 4.3.2 installed and connected with a Flextronics 8-Port SDR (Part F-X430066) IB Switch.

5.4 Algorithms and Scenarios

For the given workload and the given test environment, the weights triple (39, 21, 55) performed best for WRR. Further, WRR performed even better than WLC [15]. Hence WRR is used as reference setup and compared against the salbnet reporting algorithms DPR and DPR-Q and the salbnet scheduling algorithms Next Credits and Highest Credits (as described in Section 3.2 and Section 3.4). The Table 4 summarizes the salbnet algorithms and configurations for the measurement runs.

Table 4. Algorithms and configurations for the measurement runs.

<table>
<thead>
<tr>
<th>SLB Algorithm</th>
<th>SLB Scheduler</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRR</td>
<td>LVS WRR</td>
<td>Weights triple (39, 21, 55)</td>
</tr>
<tr>
<td>DPR</td>
<td>LVS salbnet</td>
<td>Next Credits</td>
</tr>
<tr>
<td>DPR-Q</td>
<td>LVS salbnet</td>
<td>Highest Credits</td>
</tr>
</tbody>
</table>

5.5 Measurement Results

5.5.1 SLB ISP Penalty \( P_{ISP}(a) \)

The Figure 5 shows the results from the measurements done with servload. The SLB system setup was measured under the three different workloads as described in Table 2.
in Section 5.2. Each SLB algorithm was setup as described in Table 4. The results of the benchmark runs are labeled with the used SLB algorithm (for example WRR) and the used configuration.

![Diagram](image)

Figure 5. On top, the normalized (First) Response Time \( \hat{r}_a \) results are shown, next the normalized (Request) Errors \( \hat{e}_a \), and at bottom the SLB ISP Penalty \( P_{ISP}(a) \). The results from WRR always shown above and the DPR configurations below.

The SLB ISP Penalty \( P_{ISP}(a) \) is a lower-is-better metric. As shown in Figure 5, the measurements confirm prior simulation results. WRR is outperformed by salmnet using the Next Credits scheduling. The benefit of the Self-Adapting Load Balancing Network increases under higher load. WRR has a higher normalized (First) Response Time \( \hat{r}_a \), resulting in a higher SLB ISP Penalty.

The Next Credits DPR has the best performance and the lowest SLB ISP Penalty for the lower workloads, while the variant DPR-Q has a slightly better performance for the highest workload. This shows the influence of the (number of) Credit reports (resulting in Credit metric collecting) and the advantage of the quantization as described in Section 3.3.

The normalized (Request) Errors \( \hat{e}_a \) (as shown in the middle of Figure 5) are higher for the Next Credits DPR and DPR-Q than for the WRR measurements, which leads to the conclusion that early Drops happen as expected. As result, with increasing workloads, both the Next Credits DPR and its variant DPR-Q have a considerably lower normalized (First) Response Time \( \hat{r}_a \) and consequently a lower SLB ISP Penalty, therefore outperform the traditional LVS WRR algorithm.

Another outcome is that the DPR with the Highest Credits configuration performs worst in these measurements. This behavior will be investigated in more detail in the following.

### 5.5.2 Load Averages

Figure 6 shows the load average comparison of the DPR algorithm once configured with the Next Credits scheduling algorithm and once configured with the Highest Credits scheduling algorithm. The graphs show that the load average on the back end servers during 11 passes of the measurements is differing. Especially, Web Server 2 and Web Server 3 receive more workload under the Highest Credits scheduling, compared to the Next Credits scheduling. Further, all 11 runs have a much longer Duration compared to the original 10 minutes trace since the measurement environment with the three back end servers is less powerful compared to the original Wikipedia environment.

![Diagram](image)

Figure 6. Load average on the back end servers during the 11 IB measurement runs of the one-eighth workload comparing DPR with the Next Credits and the Highest Credits scheduling algorithm.

Furthermore, the 11 passes are finished earlier with the Highest Credits scheduling algorithm. This behavior is not reflected in the SLB ISP Penalty. Using the Next Credits scheduling algorithm the back end server are less loaded and therefore still have low (First) Response Times and less (Request) Errors which may keep ISP customers satisfied. This shows that in the experiment the Next Credits scheduling algorithm is better suited to handle overload situations compared to Highest Credits scheduling algorithm. Since the SLB ISP Penalty \( P_{ISP}(a) \) is created with the requirements of an ISP in mind it does not take the Duration of the measurement runs into account like the SLB Full Penalty \( P_{Full}(a) \).
5.5.3 SLB Full Penalty $P_{\text{full}}(a)$

Figure 7 shows that the included normalized Duration $i_a$ adjusted the SLB Full Penalty $P_{\text{full}}(a)$ values in favor of the Highest Credits scheduling algorithm. But the Next Credits scheduling algorithm still performs better than the Highest Credits scheduling algorithm. The results are similar for the DPR-Q and do not invert the previous results.

5.5.4 Established Connections

Figure 8 shows the TCP established connections comparison of the DPR algorithm once configured with the Next Credits scheduling algorithm and once configured with the Highest Credits scheduling algorithm. The graphs show that the number of established connections on the back end servers during 11 passes of the measurements is differing. Especially, Web Server 2 and Web Server 3 receive more workload under the Highest Credits scheduling.

Figure 8. Number of TCP established connections on the back end servers during the 11 IB measurement runs of the one-eighth workload comparing DPR with the Next Credits and the Highest Credits scheduling algorithm.

The maximum number of connections are limited on all web servers due to memory constraints. As shown in Figure 8, with the Highest Credits these limits are reached for all web servers and especially Web Server 2 and Web Server 3 have worse response times which influences both the SLB ISP Penalty and the SLB Full Penalty and is the reason for the worse results of the Highest Credits scheduling above.

6 Conclusion and Future Work

While the benefits of server load balancing strategies which adapt to changing load situations have been reported in the literature, they still are rarely supported by available load balancers. The main reason may be the challenges of an efficient implementation.

This paper presents an implementation called salbnet which uses a Credit based metric. The Credits represent the maximum number of connections which a back end server is able to handle. The metric reflects both, the capacity of the back end server (a faster server will report more Credits) and the current load situation on the back end server (a low loaded server will report more Credits). We have given a precise definition of the Credit metric for TCP applications.

Since the Credit reporting puts an additional burden on the Load Balancer, this has to be implemented in an efficient manner. Therefore, we propose a reporting mechanism, which uses the Remote DMA feature of InfiniBand and writes the updates directly into the (kernel) address space of the load balancer. Hence, no additional message processing is necessary.

We evaluated salbnet with a SLB scenario where the Apache HTTP servers were running in the back end and with workloads derived from a Wikipedia trace. With increasing workloads salbd considerably outperforms the traditional LVS WRR algorithm in a direct comparison. This confirms prior simulation results presented in [5] and shows that self-adapting load balancing is also feasible in praxis.

We also compared two different salbnet scheduling algorithms, the Highest Credits and the simpler Next Credits scheduler. But the expectation that the Highest Credits scheduler performs better than the Next Credits one, could not be validated in general. The Highest Credits scheduler optimizes the benchmark duration, since it results in shorter execution times at the back end servers. But the normalized (First) Response Time is worse compared to Next Credits and hence results in a worse SLB ISP Penalty.

The Credit based approach and the salbnet implementation works application independent in non-specialised environments. Currently, salbnet supports TCP based applications. Further work is done to expand the Credit based approach for UDP applications like DNS as well.

References


