Policy Management in the Reliable Server Pooling Architecture

Thomas Dreibholz
Institute for Experimental Mathematics
University of Duisburg-Essen, Germany
dreibh@exp-math.uni-essen.de
http://www.exp-math.uni-essen.de/~dreibh
# Table of Contents

- Introduction - What is Reliable Server Pooling
- An Important RSerPool Task - Server Selection by Pool Policies
- Namespace and Policy Management – How to implement it efficiently?
  - Requirements
  - Our Proposed Concept
  - Performance Evaluation Results
- Conclusions and Outlook

---

Thomas Dreibholz's Reliable Server Pooling Page
[http://tdrwww.exp-math.uni-essen.de/dreibholz/rserpool/](http://tdrwww.exp-math.uni-essen.de/dreibholz/rserpool/)
What is Reliable Server Pooling (RSerPool)?

- Some applications require high availability, e.g.
  - e-Commerce
  - Medicine
  - ...

- No single point of failure => multiple redundant servers for same service (server pool) => **RSerPool** – A unified solution for server pool management

- Based on SCTP (Stream Control Transmission Protocol)
- Under Standardization by IETF RSerPool WG
- Important RSerPool task: Selection of servers ...
  - Load Balancing, application-specific policies

- RSerPool architecture also usable for new applications:
  - Mobility Management
  - Distributed Computing
What is Reliable Server Pooling (RSerPool)?

**Terminology:**
- Pool Element (PE): Server
- Pool
- PE ID: Unique ID of PE
- Pool Handle: Unique ID of pool
- Namespace
- Name Server (NS)
- Pool User (PU): Client

**Protocols:**
- ASAP (Aggregate Server Access Protocol)
- ENRP (Endpoint Name Resolution Protocol)
Server Selection and Pool Policies

- How does a PU access a pool's service
  - PU asks an arbitrary NS to select *appropriate* PEs of a certain pool
  - PU may add them to its cache (optional) and selects one *appropriate* PE
  - PU connects to selected PE

- How is a PE selected *appropriately*?
  - Pool Policies:
    - Weighted Round Robin (defined in RSerPool Internet Draft)
    - Least Used (defined in RSerPool Internet Draft)
    - Weighted Random (will be defined in RSerPool Internet Draft)
    - and many more; possibly service-specific extensions ...

- Many PEs in pools of many different policies ...
  
  How can a namespace be managed efficiently?
  (Internet Drafts only define policy behaviour, but not implementation ...)
Namespace Management - What are the requirements?

- For Pool Elements:
  - (Re-)Registration, i.e. lookup (by PE ID) + insertion of PE entry
  - Deregistration, i.e. removal of PE entry

- For Pool Users:
  - Resolution of Pool Handle to set of PE entries, appropriately selected by the pool's policy

- For Name Servers:
  - Step-wise traversal of Namespace, e.g. get first 100 PE entries, continue with next 100, and so on ... 

- Main Observations:
  1. for PEs: pool access by pool element ID
  2. for PUs: pool access by selection order (depending on pool policy)
Our Namespace Management Concept

- **Namespace**:  
  - **Pool Set**, sorted by pool handle

- **Pool**:  
  - **PE Index Set**  
    - sorted by: **PE ID**  
  - **PE Selection Set**  
    - sorted by: **Sorting Order**
  - **Selection Procedure**

- **Quite straightforward, but ...**
  How can certain policies (Least Used, Weighted Round Robin) be expressed as „Sorting Order“ and „Selection Procedure“?
Defining „Sorting Order“

- **Part 1: Policy-Specific Sorting Key**
  - Policy-dependent sorting key, e.g. *load* for Least Used

- **Part 2: Sequence Number**
  - For every pool: pool-wide global sequence number
  - For every PE entry: PE sequence number
  - **New PE entry or PE entry selected:**
    - PE's sequence number := pool's sequence number
    - **Increment** pool's sequence number
  - Note: A PE entry's sequence number is *unique* within its pool!

- **Sorting Order** := Sorting by *composite key* (Pol.-Spec. Key, PE Seq.No.)

- **Usual Selection Procedure** :=
  - Simply take first PE entry from the Selection Set
  - **Update** its sequence number + possibly its pol.-spec. key; **re-insert** it
Our Policy Realizations

- IETF drafts define what policies mean, but not how to implement them!

- **Least Used:**
  - **Sorting Order:** Sorting by (PE load, Seq.No.)
  - **Selection Procedure:** Take first PE of the Selection Set
  - Note: Seq.No. ensures round robin selection between equal-loaded PEs

- **Weighted Round Robin**
  - For each PE: Round Counter $r$, Virtual Counter $v$ (Selections to go for current round)
  - **Sorting Order:** Sorting by $(r, v\text{ (descending)}, \text{ Seq.No.})$
  - **Selection Procedure:** Take first PE of the Selection Set

- **Weighted Random:**
  - For each PE: weight specifies proportional selection probability
  - For each pool: WeightSum := Sum of all PEs' weights
  - **Sorting Order:** PE ID only (ensures unique order)
  - **Selection Procedure:** Random number $r \in \{0, \ldots, \text{WeightSum}\} \subseteq \mathbb{R}$ exactly maps to one PE
Example 1: Least Used Policy

- **Sorting Order:** Sorting by (PE load, Seq.No.)
- **Selection Procedure:** Simply take the **first** PE of the Selection Set
- **Before Selection:**

<table>
<thead>
<tr>
<th>Pool „Example“</th>
<th>Policy LU seq=8</th>
<th>PE #7</th>
<th>load=10%</th>
<th>seq=6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PE #2</td>
<td>load=10%</td>
<td>seq=7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PE #11</td>
<td>load=44%</td>
<td>seq=3</td>
</tr>
</tbody>
</table>

PE #7 will be selected next (lowest load and lowest seq.no. for this load)

- **After Selection:**

<table>
<thead>
<tr>
<th>Pool „Example“</th>
<th>Policy LU seq=9</th>
<th>PE #2</th>
<th>load=10%</th>
<th>seq=7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PE #7</td>
<td>load=10%</td>
<td>seq=8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PE #11</td>
<td>load=44%</td>
<td>seq=3</td>
</tr>
</tbody>
</table>

- PE #2 will be next one, then again PE #7 and so on ...
- Seq-No. ensures round-robin selection between PEs of equal load!
Example 2: Weighted Round Robin

- For each PE entry:
  Round Counter \( r \), Virtual Counter \( v \) (Selections to go for current round)
- Sorting Order: Sorting by (Rd.Cntr, Vrt.Cntr. descending, Seq.No.)
- Selection Procedure: Take first PE
- Example:

<table>
<thead>
<tr>
<th>Pool „Example“</th>
<th>Policy WRR</th>
<th>PE #5</th>
<th>weight=2</th>
<th>r=20</th>
<th>v=2</th>
<th>seq=6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PE #1</td>
<td>weight=1</td>
<td>r=20</td>
<td>v=1</td>
<td>seq=7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PE #9</td>
<td>weight=1</td>
<td>r=20</td>
<td>v=1</td>
<td>seq=8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pool „Example“</th>
<th>Policy WRR</th>
<th>PE #1</th>
<th>weight=1</th>
<th>r=20</th>
<th>v=1</th>
<th>seq=7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PE #9</td>
<td>weight=1</td>
<td>r=20</td>
<td>v=1</td>
<td>seq=8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PE #5</td>
<td>weight=2</td>
<td>r=20</td>
<td>v=1</td>
<td>seq=9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pool „Example“</th>
<th>Policy WRR</th>
<th>PE #9</th>
<th>weight=1</th>
<th>r=20</th>
<th>v=1</th>
<th>seq=8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PE #5</td>
<td>weight=2</td>
<td>r=20</td>
<td>v=1</td>
<td>seq=9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PE #1</td>
<td>weight=1</td>
<td></td>
<td></td>
<td>seq=10</td>
</tr>
</tbody>
</table>

Next: PE #9, finally PE #5. End of WRR round 20.
Example 3: Weighted Random

- Sorting Order: Sorting by PE ID only (for unique identification)
- For each PE entry:
  - value, i.e. its selection probability
  - For each pool: Value Sum := Sum of all PEs' value settings
- Selection Procedure:
  - Get random number $r \in [0, \ldots, \text{ValueSum}] \subseteq \mathbb{R}$
  - $r$ maps to exactly one PE

Example:

<table>
<thead>
<tr>
<th>Pool „Example“</th>
<th>Policy WRAND</th>
<th>PE #17</th>
<th>PE #8</th>
<th>PE #11</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq=10</td>
<td>ValueSum=6</td>
<td>weight=1</td>
<td>weight=3</td>
<td>weight=2</td>
</tr>
<tr>
<td>ValueSum=6</td>
<td></td>
<td>value=1</td>
<td>value=3</td>
<td>value=2</td>
</tr>
</tbody>
</table>

$r=5.25$ => $[0, 1]$ for PE #17; $[1, 4]$ for PE #8; $[4, 6]$ for PE #11

=> Selection of PE #11
Implementation

- We use *sets* for Pools, Index and Selection, but ...
  
  ... How should we implement a *set*?

- Possible Data Structures:
  - Linear List
  - Unbalanced Binary Tree
  - Balanced Binary Tree (Red-Black)
  - Randomized Binary Tree (Treap)

- Question now:
  - Which is most efficient?
  - What is average namespace operation runtime on „standard PC“ hardware (AMD Athlon 1.3 GHz)?

=> Performance Evaluation!
Performance Evaluation

- **Transactions Scenario**
- **Operations Ratio:**
  - Registrations: 1
  - Reregistrations: 30
  - PE Selections: 5
  - Traversal: 10
- **Avg. Operation Runtime:**
  10 pools
  2 to 202 PEs per pool

- **Results:**
  - Avg. runtime **less than 20μs** for 10 pools of 202 PEs (balanced trees)!
  - Unbalanced trees unsuitable (insertion/removal too systematic)
Performance Evaluation (Scalability)

- Distributed Computing Scen.

- Operations Ratio:
  - Registrations: 1
  - Reregistrations: 300
  - PE Selections: 5000
  - Traversal: 1

- Avg. Operation Runtime:
  - 1 pool
  - 10 to 100010 PEs

- Results:
  - Acceptable runtime even for very large pools (< 70µs for 100010 PEs)!
Conclusions & Outlook

- Namespace and Policy Management is basic task of RSerPool
  - Must be efficient -> Large pools (e.g. for distributed computing)
  - Must be extendable -> New policies for new applications

- Proposed Solution: Reduction of problem to ...
  - Definition of policy-specific **sorting orders** and **selection procedures**
  - Storage of **sorted sets**
  - Efficiency shown by performance evaluation => best for **balanced trees**

- Current Status
  - Implementation of Namespace and Policy Management as C Library
  - Usage for our OMNeT++ RSerPool simulation model **rspsim**

- Future Plans
  - Usage of our library also in our Open Source RSerPool Prototype **rsplib**
  - Full implementation of the RSerPool standard by 09/2004.
Any Questions?

Project Homepage:
http://tdrwww.exp-math.uni-essen.de/dreibholz/rserpool/

Thomas Dreibholz, dreibh@exp-math.uni-essen.de