draft-ietf-rserpool-policies-00.txt

Definition of Member Selection Policies

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

- Terminology
- Selection Process
- The Policies
  - Round Robin, Weighted Round Robin
  - Random, Weighted Random
  - Least Used, Randomized Least Used
  - Least Used with Degradation, Priority Least Used
  - Anything missing?
- Conclusions and Outlook

Thomas Dreibholz's Reliable Server Pooling Page
http://tdrwww.exp-math.uni-essen.de/dreibholz/rserpool/
Terminology

- **Load:**
  - „How much are a PE's resources currently utilized?“
  - From 0x000000 -> 0% to 0xffffffff -> 100%
  - Utilization to be defined by application (e.g. memory usage, CPU load, ...)

- **Weight:**
  - „A PE's service capacity relatively to other PEs of the same pool“
  - Example: 2*n -> double capacity compared to a PE weighted with n

- **Classification of Policies:**
  - **Static:** Policy information does not change (e.g. CPU power)
  - **Dynamic:** Policy information regularly changes (e.g. server load)

=> Re-registration on change!
Selection Process

- **Step 1:** Name Server
  - On ASAP Name Resolution:
    - NS selects *one or more* PE identities from the pool by its policy
    - Is a selection really needed at the NS? **Yes, it is!**
      - Size of response message limited
      - Inefficient to reply too many elements

- **Step 2:** Pool User's local cache
  - On Name Resolution request (Application wants exactly *one* PE):
    - PU tries to fulfil request by its local cache first (stale cache value ...)
    - If not successful, issue ASAP Name Resolution to NS
    - Propagate result to its cache
    - Apply selection by policy again
Round Robin and Weighted Round Robin

- **Round Robin (the default policy)**
  - PE references can be hold in a circular list, pointer to current element
  - Selection at NS:
    - Pointer to be forwarded by *one*, regardless of the amount of elements actually selected -> necessary to avoid degeneration!
    - No duplicate entries in the list of returned elements
  - Selection at PU:
    - Pointer to be forwarded by the amount of elements selected

- **Weighted Round Robin:**
  - Policy Information per PE: *weight*
  - Each PE gets as many entries in the list as its weight constant specifies
  - Then: Handling like Round Robin
  - Again: No duplicate entries in the list of returned elements
Random and Weighted Random

- **Weighted Random:**
  - Policy Information per PE: *weight*
  - Selection at NS:
    - *Weight* constant defines PE's selection probability relative to other elements in the pool
    - Randomly select based on these probabilities
    - No duplicate entries in the list of returned elements
  - Selection at PU:
    - Same behaviour as for NS

- **Random:**
  - Special case of Weighted Random:
    *All weights* are set to same value (e.g. 1)
Least Used and Randomized Least Used

- Least Used:
  - Policy Information per PE: load
  - Selection at NS:
    - Get fraction of the pool's PE entries, sorted ascending by their load values
    - Should make round robin selection between equal-loaded PEs
    - No duplicate entries in the list of returned elements
  - Selection at PU:
    - Same behaviour as for NS

- Randomized Least Used
  - Same as Weighted Random selection with weight := 0xffffffff - load
Least Used with Degradation and Priority Least Used

- **Least Used with Degradation:**
  - Policy Information per PE: load, load degradation
  - Each selection component maintains per-PE local degradation counter
    - Initialized with 0, reset to 0 on re-registration, incremented by PE's load degradation on selection
  - Selection at NS and PU:
    - Like Least Used with (load + degradation counter) instead of load only
  - **Difficulty:** Dependencies between load degradation, request rate and stale cache value -> Finding optimal parameters is not easy!

- **Priority Least Used:**
  - Load degradation is only constant. **No** local counters!
  - Handle like Least Used with (load + load degradation) instead of load only
Anything missing?
Your ideas are welcome!

- Is any policy missing?
- Does your application require a special policy?
- Do you have ideas for additional policies?

Do not hesitate to contact us!

We are always interested to include

- additional,
- new,
- better,
- ...

policies!
Any Questions?

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

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