Virtual-Time Round-Robin: An O(1) Proportional Share Scheduler

By Jason Nieh, etc

Xiaojun Wang
10/07/2005

Outline

- Proportional Share Scheduling
- Weighted Round Robin
- Weighted Fair Queueing
- Virtual-Time Round-Robin
- Summary

Proportional Share Scheduling

- Given a set of clients with associated weights, a proportional share scheduler should allocate resources to each client in proportion to its respective weight.
- Why useful?
  - Administrative Purposes
    - Allocate resource to users according to their weights (for example, money they pay)
  - QoS Goals
    - Video, audio applications (minimize jitter)

Evaluation Criteria

- Two Evaluation Criteria
  - Accuracy of proportional sharing
  - Scheduling overhead
- VTRR
  - High proportional sharing accuracy (WFQ)
  - Low scheduling overhead (RR)

Proportional Fairness

- Some notations
  - $S_i$ The proportional share of client $A$
  - $W_i(t_1, t_2)$ The amount of service received by $A$ during time interval $(t_1, t_2)$.
  - $E_i(t_1, t_2)$ The service time error for $A$ over $(t_1, t_2)$
  $E_i(t_1, t_2) = W_i(t_1, t_2) - (t_2 - t_1) \frac{S_i}{\sum S_i}$
  - The goal of a proportional share scheduler is to minimize $E$ over all time intervals.

Weighted Round Robin

- Algorithm
  - Clients are placed in a queue and executed in turn (same frequency).
  - When being executed, each client is assigned a time quantum equal to its share (adjustable time quantum size).
- Evaluation
  - Low scheduling overhead: $O(1)$
  - Weak proportional fairness guarantee. The service time error can be quite large, especially when the share values are large.
Weighted Fair Queueing

- Originally invented for scheduling network packets.
- Maintain a queue of clients sorted by their *virtual finishing time*, each time select the client with the smallest VFT.
- Different frequency, same time quantum size.
- Evaluation
  - Good proportional sharing accuracy.
  - High scheduling overhead: $O(n)$, $O(\log n)$

Virtual Time

- Virtual Time: a measure of the degree to which a client has received its proportional allocation, relative to other clients.
  \[ VT_c(t) = \frac{W_c(t)}{S_c} \]
- Virtual Finishing Time: the virtual time the client would have after executing for one time quantum.
  \[ VFT_c(t) = VT_c(t) + \frac{Q}{S_c} \]
  (suppose $A$ is executed during $(t, t+Q)$)

An Example of WFQ

- Goal of WFQ:
  - Make all clients’ VT grow as even as possible.
- Executing Sequence
  - ABAABC

Service Time Error in time interval $(0, t)$

- Note the sum of the Errors of all clients is 0, and each client’s Error becomes 0 at the end of each cycle.

Virtual-Time Round-Robin

- An accurate, low-overhead proportional share scheduler which combines the benefits of WRR and WFQ.
- Overview of Algorithm
  - Sort clients in the run queue in descending order of their shares.
  - Starting from the beginning, run each client for one fixed time quantum. (reason of $O(1)$)
  - Jump back to the beginning if a client has received more allocation than ideal case.

Client’s State

- Five values for each client:
  - Share: used to sort the clients in the queue.
  - VFT: used to decide when to jump back to the beginning (VFT inequality).
  - Time counter:
    - Reset to share value at the beginning of each scheduling cycle.
    - Decrease by one when received a time quantum.
    - Become 0 at the end of each scheduling cycle.
  - Used to ensure perfect fairness is achieved at the end of each cycle.
  - ID number
  - Run state: runnable or not
Scheduler’s State

- Run queue, time quantum, total shares
- Queue virtual time: a measure of what a client’s VFT should be if it has received exactly its proportional share allocation.
  \[ QVT(t + Q) = QVT(t) + \frac{Q}{\sum S_i} \]
- Goal of the algorithm: to make the VT of each client to be as close to QVT as possible.

VTRR Algorithm

- Time counter invariant:
  for any two consecutive clients in the queue A and B, the counter value for B must always be no greater than the counter value for A.
- VFT inequality:
  \[ \frac{VFT_st(t)}{S_c} - QVT(t + Q) < \frac{Q}{S_c} \]
  \[ = VTC(t + Q) \]

An Example of VTRR

<table>
<thead>
<tr>
<th>T</th>
<th>QVT</th>
<th>A: 3</th>
<th>B: 2</th>
<th>C: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3/1</td>
<td>2/1</td>
<td>1/1</td>
</tr>
<tr>
<td>1</td>
<td>1/6</td>
<td>2/3</td>
<td>1/1</td>
<td>1/1</td>
</tr>
<tr>
<td>2</td>
<td>3/6</td>
<td>2/2</td>
<td>1/1</td>
<td>1/1</td>
</tr>
<tr>
<td>3</td>
<td>1/6</td>
<td>1/1</td>
<td>1/1</td>
<td>1/1</td>
</tr>
<tr>
<td>4</td>
<td>5/6</td>
<td>0/0</td>
<td>3/2</td>
<td>0/2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4/3</td>
<td>0/3</td>
<td>0/2</td>
</tr>
</tbody>
</table>

- Time counter invariant
  \[ TC \leq TC_{\text{previous}} \]
  \[ 1 \leq 1 \]
- VFT inequality:
  \[ VFT_{\text{current}}(t) - QVT(t + Q) < \frac{Q}{S_c_{\text{current}}} \]
  \[ 1 - 5/6 < 1/2 \]
- Execution Sequence
  ABCABA

Dynamic Considerations

- An on-line scheduling algorithm allows clients to be dynamically created, terminated, change run state, and change their share values.
- Insert client to running queue
  - How to determine new client’s initial VFT and time counter?
- Remove non-runnable client from the queue
  - Last-previous and last-next pointers.
- Change client’s position in the queue
  - Remove, re-insert.
Complexity

- $O(1)$
  - Select a client for execution
  - Update current client’s variables
  - Check next client
  - Remove client from the queue
- Higher order operations:
  - Sort the running queue: $O(N \log N)$ infrequent
  - Reset time counter: $O(N)$ / length(scheduling cycle)
  - Insert client to the queue:
    - $O(N)$ or $O(\log N)$, infrequent
    - $O(1)$ last-previous & last-next
    - can be done in $O(1)$ if the range of share values is fixed

Accuracy: Simulation Result

- VTRR vs. WRR service time error

Accuracy: Simulation Result

- VTRR vs. WFQ service time error

Overhead: Experiment Result

- Scheduling overhead

Summary

- VTRR
  - High proportional sharing accuracy (comparable to WFQ)
  - Low scheduling overhead ($O(1)$)
  - Easy to implement (add/change < 100 lines of code in Linux)
- A promising solution for scheduling in large scale server systems.
- Progress
  - Group Ratio Round-Robin: $O(1)$ Proportional Share Scheduling for Uniprocessor and Multiprocessor Systems

Thank you!
Questions?