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

By Jason Nieh, etc

Xiaojun Wang
10/07/2005

Virginia Tech

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_A The proportional share of client A
  - $W_A(t_1,t_2)$ The amount of service received by A during time interval (t1, t2).
  - $E_A(t_1,t_2)$ The service time error for A over (t1, t2)

$$E_A(t_1,t_2) = W_A(t_1,t_2) - (t_2 - t_1)\frac{S_A}{\sum_i S_i}$$

  - 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.

$$VT_A(t) = \frac{W_A(t)}{S_A}$$

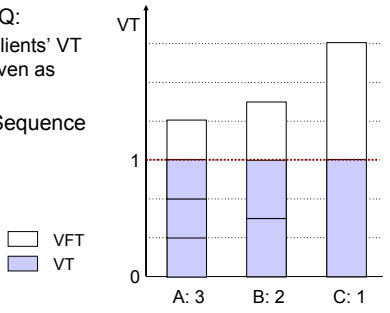
- Virtual Finishing Time: the virtual time the client would have after executing for one time quantum.
  - Q: time quantum, the duration of a standard time slice assigned to a client to execute

$$VFT_A(t) = VT_A(t+Q) = VT_A(t) + \frac{Q}{S_A}$$

suppose *A* is executed during (t, t+*Q*)

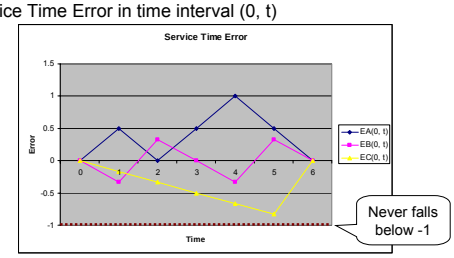
An Example of WFQ

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



An Example of WFQ

- 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 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.

$$QVT(t+Q) = QVT(t) + \frac{Q}{\sum_i S_i}$$

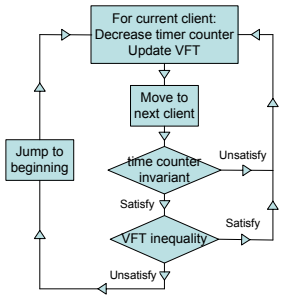
- Goal of the algorithm: to make the VT of each client to be as close to QVT as possible.

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:*

$$VFT_C(t) - QVT(t+Q) < \frac{Q}{S_C}$$

$$= VT_C(t+Q)$$



An Example of VTRR

T	QVT	A: 3		B: 2		C: 1	
		TC	VFT	TC	VFT	TC	VFT
0	0	3	1/3	2	1/2	1	1
1	1/6	2	2/3	2	1/2	1	1
2	2/6	2	2/3	1	1	1	1
3	3/6	2	2/3	1	1	0	2
4	4/6	1	1	1	1	0	2
5	5/6	1	1	0	3/2	0	2
6	1	0	4/3	0	3/2	0	2

Execute Check conditions

- Time counter invariant

$$TC \le TC_{previous}$$

1 <= 1

- *VFT inequality:*

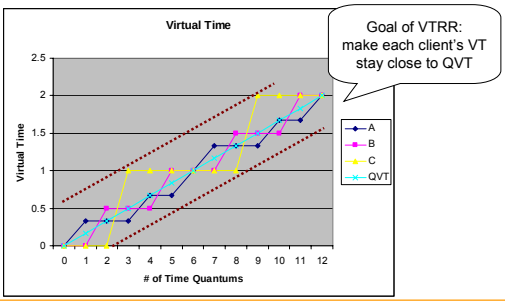
$$VFT_{Current}(t) - QVT(t+Q) < \frac{Q}{S_{Current}}$$

1 - 5/6 < 1/2

- Execution Sequence
  - ABCABA

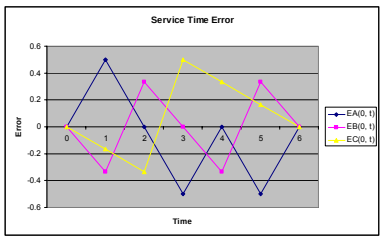
An Example of VTRR

- Virtual Time Grow Pattern



An Example of VTRR

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



Dynamic Considerations

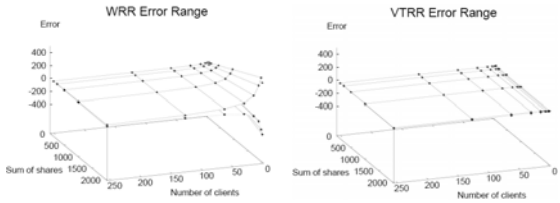
- An on-line scheduling algorithm allows clients to be dynamically created, terminated, change run state, and change their share values.
- Insert client to running queue
  - How to determine new client's initial VFT and time counter?
- Remove non-runnable client from the queue
  - *Last-previous* and *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 logN) 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

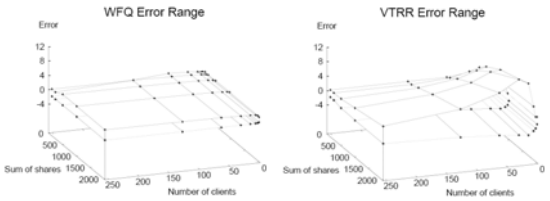
Accuracy: Simulation Result

- VTRR vs. WRR service time error



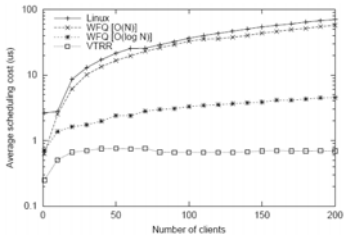
Accuracy: Simulation Result

- 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?