Fairness of proportional work-conserving I/O scheduling

Seehwan Yoo, Chuck Yoo and Hyunchan Park

Proposed is a new I/O scheduling for SSD, proportional work-conserving. The proposed scheduling algorithm provides proportional fairness among tasks. To provide proportional fairness, proportional work-conserving differentiates probability to fully utilise the task’s quantum, which is controlled by adjusting opportunistic waiting time. Its proportional fairness is formally proved and its validity through numerical evaluation and experimental results from Linux implementation are presented.

Introduction: The I/O scheduler in modern OS delivers I/O requests from user tasks to device, and the I/O scheduler controls I/O performance of tasks by arbitrating the orders of requests. The I/O scheduler primarily focuses storage devices because disk I/O is usually pointed as a bottleneck in overall performance. Particularly for proportional fairness, no I/O scheduler has been proposed because traditional disk storage has fundamental limitations on controlling performance. Recently the SSD (solid-state drive) has drawn attention as a next-generation storage device [1] because it has several advantages over disk storage such as high bandwidth, low latency and high power-efficiency. Those advantages of the SSD enable the I/O scheduler to finely control I/O performance among tasks.

Although some studies reveal performance characteristics of SSD [2], and present performance enhancements [3, 4], they only focused on enhancing overall performance, instead of achieving fairness. Because fairness is one of the most important objectives of I/O scheduler, this Letter proposes a new I/O scheduling for SSD, proportional work-conserving, that provides proportional fairness among tasks.

A goal of this Letter is to provide proportional storage I/O performance among tasks using proportional work-conserving scheduling, which cannot be achieved with existing I/O schedulers. This Letter provides formal proof for fairness of proportional work-conserving; and also presents validation through numerical evaluation and experimental results from Linux implementation.

Proportional work-conserving: This Section explains our new I/O scheduling for SSD: proportional work-conserving. Proportional work-conserving focuses on differentiation of idle time in anticipatory scheduling [5]. By differentiating opportunistic waiting time, we can differentiate probability to utilise quantum of tasks. Intuitively, if a task has longer waiting time, it is more probable to utilise its quantum; however, if a task has shorter waiting time, it is more easy to switch to another task.

Proportional work-conserving scheduler uses two kinds of queue, S-queue and A-queue. A user task (ti) stacks synchronous requests to its own (per-task) S-queue (Sq,i) and stacks asynchronous requests to global A-queue (AQ) that is shared among all tasks.

The scheduler dispatches I/O requests from S/A-queues to the device queue, according to WRR scheduling. For WRR scheduling, each S-queue has its own quantum (qj), and the quantum is accounted by the size of dispatched request size. If Sq,i has consumed all its quantum, the Sq,i is scheduled out until its quantum is replenished. The quantum is replenished when all tasks have no remaining quantum.

Even though Sq,i has remaining quantum, the scheduler selects another S-queue if Sq,i has no pending requests according to the work-conserving policy. Proportional work-conserving opportunistically waits wby,i amount of time for future requests. In original anticipatory scheduling, all S-queues have the same wby,i, but proportional work-conserving differentiates wby,i values among tasks. To fully utilise SSD, proportional work-conserving puts asynchronous requests from A-queue during wby,i waits.

Proportional work-conserving provides proportional fairness. We can define fairness of proportional work-conserving by the proportionality of throughput as follows. We call tasks ti and tj proportionally fair if Thru(ti)/sij = Thru(tj)/sj, where Thru(ti), Thru(tj) present actual throughput; and sij, sj present the service rates of tasks ti and tj.

Theorem 1 states that we can make tasks ti and tj proportionally fair by controlling wby,i values.

Theorem 1: Tasks ti and tj are proportionally fair if wby,i and wby,j are
cdf(wby,i)/sij = cdf(wby,j)/sj

where cdf(wby,i), cdf(wby,j) are cumulative probabilities of arriving next request at time wby,i and wby,j. In addition, sij, sj are service rate of task ti and tj, respectively.

Proof: We assume that a task makes I/O requests at random intervals, and the intervals are identical and independent from each other. In addition, the average request size is R, and the average latency for request is E[x(i)]. Without why, average throughput is R/E[x]. In a scheduling round, task ti can handle at most Nj number of requests, where Nj = qj/R.

The number of handled requests in a scheduling round is determined by the interval of requests. When the interval is over wby,i, the scheduler dispatches requests from the next queue by the work-conserving policy.

For n number of handled requests in a scheduling round, throughput of ti is

n = 1, R(E[x(i)] + wby,j)cdf(wby,i)(1 - cdf(wby,j))

(2)

n = 2, 2R/(2E[x(i)] + wby,j)cdf(wby,i)2(1 - cdf(wby,j))

(3)

n = Nj, NR/(NjE[x(i)] + wby,j)cdf(wby,i)Nj(1 - cdf(wby,j))

(4)

n > Nj, 0

(5)

Therefore, the expectation for the actual throughput of task ti is

Thru(i) = \sum_{i=1}^{Nj} nR/E[x(i)]cdf(wby,i)(1 - cdf(wby,j))

(6)

Because nE[x(i)] + wby,i is always larger than nE[x],

Thru(i) < \sum_{i=1}^{Nj} nR/E[x(i)]cdf(wby,i)(1 - cdf(wby,j))

(7)

= R \sum_{i=1}^{Nj} cdf(wby,i)1 - cdf(wby,j)

(8)

= R \sum_{i=1}^{Nj} cdf(wby,i)1 - cdf(wby,j)(1 - cdf(wby,j)Nj-1)

(9)

In addition, nE[x(i)] + wby,j is always larger than nE[x] + wby:

Thru(i) > \sum_{i=1}^{Nj} nR/E[x(i)]cdf(wby,i)(1 - cdf(wby,j))

(11)

= R \sum_{i=1}^{Nj} cdf(wby,i)1 - cdf(wby,j)

(12)

= R \sum_{i=1}^{Nj} cdf(wby,i)1 - cdf(wby,j)Nj-1

(13)

Thus, we get the range of Thru(i) as follows:

R E[x(i)] + wby,j cdf(wby,i)(1 - cdf(wby,j)Nj-1)

< Thru(i) < R E[x(i)]1 - cdf(wby,j)Nj-1

(14)

Similarly, the range of Thru(j) is given as follows:

R E[x(j)] + wby,j cdf(wby,j)(1 - cdf(wby,j)Nj-1)

< Thru(j) < R E[x(j)]1 - cdf(wby,j)Nj-1

(15)

From (14) and (15), we obtain

\left\{ \begin{array}{l}
\frac{E[x(i)]}{E[x(j)]} \frac{cdf(wby,i)(1 - cdf(wby,j)Nj-1)}{cdf(wby,j)(1 - cdf(wby,j)Nj-1)} Thru(i) < Thru(j)
\\
\frac{E[x(j)]}{E[x(i)]} \frac{cdf(wby,i)(1 - cdf(wby,j)Nj-1)}{cdf(wby,j)(1 - cdf(wby,j)Nj-1)} Thru(i) > Thru(j)
\end{array} \right.

(16)
For large $N_i, N_j$, 

\[
\left( \frac{\mathbb{E}[X]}{\mathbb{E}[X] + \text{cdf}(\text{wb}_j)} \right) < \frac{\text{Thru}(i)}{\text{Thru}(j)}
\] \tag{17}

Because we assume \( \text{cdf}(\text{wb}_j)/\text{cdf}(\text{wb}_i) = s_i/s_j \) in (1), we can rewrite (17) as follows:

\[
\left( \frac{\mathbb{E}[X]}{\mathbb{E}[X] + \text{wb}_j} \right) < \frac{\text{Thru}(i)}{\text{Thru}(j)}
\] \tag{18}

For reasonably small \( \text{wb}_i, \text{wb}_j \),

\[
\frac{\text{Thru}(i)}{\text{Thru}(j)} < \frac{s_i}{s_j}
\] \tag{19}

Thus, (20) proves Theorem 1:

\[
\frac{\text{Thru}(i)}{s_i} = \frac{\text{Thru}(j)}{s_j}
\] \tag{20}

**Evaluation:** To validate our theorem, we perform an experiment with postmark that is a popular filesystem benchmark program. For five tasks $t_1$–$t_5$, we give five different service rates as 1.2, 1.1, 1.0, 0.9, 0.8. Why values taken from cdf are 1,329, 1,097, 715, 373 ms, at which probability of tasks are 83%, 76%, 69%, 62%, 55%, respectively.

The achieved throughput are presented in Fig. 1. In the Figure, throughputs of $t_1$–$t_5$ are 50.56, 46.18, 42.18, 37.96, 33.28 Mbit/s. If we normalise each throughput by $t_1$, we get 1.2, 1.09, 1.0, 0.9, 0.79, for $t_1$–$t_5$, which is very similar to provided service rates. The achieved fairness values $\text{Thru}/s_i$ are 42.13, 41.98, 42.18, 42.18, 41.6. Because fairness values are almost similar, we can say that proportional work-conserving achieves proportional fairness.

**Conclusion:** This Letter presents a new I/O scheduling for SSD, proportional work-conserving. We show that proportional work-conserving achieves better fairness and aggregated throughput by controlling opportunistic waiting time. We also presents a formal proof for its fairness and validation through numerical evaluation and experimental results.

© The Institution of Engineering and Technology 2012

January 2012
doi: 10.1049/el.2012.0261

One or more of the Figures in this Letter are available in colour online.

Seehwan Yoo, Chuck Yoo and Hyunchan Park (College of Information and Communication, Korea University, Seoul 136-713, Republic of Korea)

E-mail: shyoo@os.korea.ac.kr

**References**


