ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2010 Prof. R. Wattenhofer / J. Smula, T. Langner

Discrete Event Systems

Solution to Exercise Sheet 9

1 Waiting Problem

To see why $\Pr[Y > y] = e^{-\lambda y}$ if Y is exponentially distributed with parameter λ, observe that

$$\Pr[Y \leq y] = F_Y(y) = 1 - e^{-\lambda y}$$

for exponentially distributed random variables and further

$$\Pr[Y > y] = 1 - \Pr[Y \leq y] = e^{-\lambda y} \ .$$

We use this formulation in the following because it makes the solution more succinct.

a) We have $X := \min\{X_1, X_2\}$. The X_i are exponentially distributed with parameter λ_i.

$$\Pr[X_i > t] = e^{-\lambda_i t}$$

The probability $\Pr[X > t] = \Pr[\min\{X_1, X_2\} > t]$ corresponds to the event, that both X_1 and X_2 attain values larger than t. Hence we have

$$\begin{aligned}
\Pr[X > t] &= \Pr[\min\{X_1, X_2\} > t] \\
&= \Pr[X_1 > t \wedge X_2 > t] \\
&= \Pr[X_1 > t] \cdot \Pr[X_2 > t] && \text{(since } X_1 \text{ and } X_2 \text{ are independent)} \\
&= e^{-\lambda_1 t} \cdot e^{-\lambda_2 t} \\
&= e^{-(\lambda_1 + \lambda_2)t}
\end{aligned}$$

which means that X is exponentially distributed with $\lambda = \lambda_1 + \lambda_2$.

b) Let $X(n)$ be the minimum of n exponentially distributed random variables with parameter $\lambda_1, \ldots, \lambda_n$. We can use induction over n to show the theorem for arbitrary n. Part **a)** gives us the base case. As induction hypothesis, we assume

$$\Pr[X(n-1) > t] = e^{-(\lambda_1 + \cdots + \lambda_{n-1})t} \ .$$

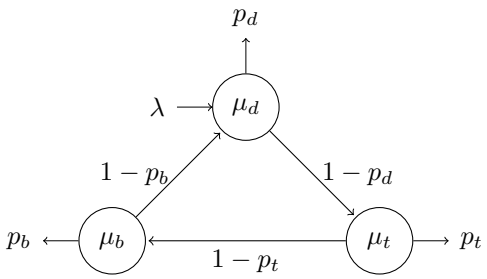
Then we can calculate the distribution for $X(n)$ as follows.

$$\begin{aligned}
\Pr[X(n) > t] &= \Pr[\min\{X_1, \ldots, X_n\} > t] \\
&= \Pr[\min\{X_1, \ldots, X_{n-1}\} > t \wedge X_n > t] \\
&= \Pr[\min\{X_1, \ldots, X_{n-1}\} > t] \cdot \Pr[X_n > t] && \text{(independence of } X_i\text{)} \\
&= \Pr[X(n-1) > t] \cdot \Pr[X_n > t] \\
&= e^{-(\lambda_1 + \cdots + \lambda_{n-1})t} \cdot e^{-\lambda_n t} && \text{(induction hypothesis)} \\
&= e^{-(\lambda_1 + \cdots + \lambda_n)t}
\end{aligned}$$

Hence, X is exponentially distributed with parameter $\lambda = \lambda_1 + \cdots + \lambda_n$.

2 Queuing Networks

a)



b) We have an open queuing network and hence we can apply Jackson's theorem (slides 97ff):

$$\begin{aligned} \lambda_d &= \lambda + \lambda_b(1 - p_b) \\ \lambda_t &= \lambda_d(1 - p_d) \\ \lambda_b &= \lambda_t(1 - p_t) \end{aligned}$$

Solving this equation system gives:

$$\lambda_d = \frac{\lambda}{1 - (1 - p_d)(1 - p_t)(1 - p_b)}$$

$$\lambda_t = \frac{(1 - p_d)\lambda}{1 - (1 - p_d)(1 - p_t)(1 - p_b)}$$

$$\lambda_b = \frac{(1 - p_d)(1 - p_t)\lambda}{1 - (1 - p_d)(1 - p_t)(1 - p_b)}$$

c) The waiting time is given by $W_t = \rho_t/(\mu_t - \lambda_t)$, where $\rho_t = \lambda_t/\mu_t$.

d) We apply the given values to the equations for λ_d, λ_t and λ_b and obtain:

$$\lambda_d = 10, \qquad \lambda_t = 25/3, \qquad \lambda_b = 20/3.$$

Therefore, by the formula of slide 73, the expected number of customers in the system is given by

$$N = \frac{\lambda_d}{\mu_d - \lambda_d} + \frac{\lambda_t}{\mu_t - \lambda_t} + \frac{\lambda_b}{\mu_b - \lambda_b} = 8.$$

Applying Little's formula to the entire system gives $T = N/\lambda = 8/5$ hours.

e) We require $\lambda_t = 1$ and therefore

$$\frac{\lambda(1 - p_d)}{1 - (1 - p_d)(1 - p_t)(1 - p_b)} = 1.$$

Solving the equation for p_d yields:

$$p_d = 1 - \frac{1}{\lambda + (1 - p_t)(1 - p_b)} = 1 - \frac{1}{5 + \frac{4}{5} \cdot \frac{3}{4}} = 1 - \frac{1}{\frac{28}{5}} = \frac{23}{28}.$$

3 Bin Packing

The algorithm mentioned in the exercise is 2-competitive. The proof works as follows: Consider the bins in the order in which they were closed. Consider two consecutive bins i and $i+1$. Assume that the algorithm fills bin i up to level $x \leq 1$. The next item (the first to be put into bin $i+1$) must be of size *larger* than $1-x$. Otherwise, the algorithm would not have opened a new bin. Therefore, any *two consecutive bins* must have total size strictly more than 1. Because the optimum must also use at least one bin to store this size, the algorithm requires at most twice as many bins.

To show that there is indeed a sequence which the optimum can serve using half the number of bins as the algorithm, assume the following input sequence:

$$I = \underbrace{\left(1, \frac{2}{n}, 1, \frac{2}{n}, 1, \frac{2}{n}, \ldots, 1, \frac{2}{n}\right)}_{n \text{ items}}.$$

Clearly, our algorithm ALG needs to open a new bin after every item. That is, the number of bins opened by ALG is $\text{cost}_{\text{ALG}}(I) = n$. On the other hand, the optimal algorithm OPT can put all $n/2$ objects of size 1 into one bin each and all the $n/2$ objects of size $2/n$ together into one bin. Hence, $\text{cost}_{\text{OPT}}(I) = n/2 + 1$. The competitive ratio c for input sequence I is therefore

$$c \geq \frac{\text{cost}_{\text{ALG}}(I)}{\text{cost}_{\text{OPT}}(I)} = \frac{n}{\frac{n}{2}+1} = \frac{2n}{n+2}.$$

For large n, this tends to 2.

4 Paging

a) (i) FIFO *(First-in/First-out): Replace the page that has been in the cache longest.*
The FIFO strategy is 3-*competitive*. In order to prove this, observe that on any consecutive input subsequence containing three or fewer distinct page references, FIFO incurs three or fewer page faults. Now, consider a 3-*phase partition* of the input sequence I. A 3-phase partition is defined as follows: Phase 0 is the empty sequence. For every $i \geq 1$, phase i is the maximal sequence following phase $i-1$ that contains at most three distinct page requests; that is, if it exists phase $i+1$ begins on the request that constitutes the fourth distinct page request since the start of the i-th phase.

By the above observation, it is clear that FIFO incurs at most three page faults for any phase $i \geq 1$, because it cannot fault twice on the same page. For any $i \geq 1$, let q be the first request of phase i and consider the input sequence starting with the second request of phase i up to and including the first request of phase $i+1$. The optimal algorithm OPT can only have two additional pages cached (on top of q), and there are three different requests (except if phase i is the last phase) in this sequence not counting request q. Hence, OPT must incur at least one page fault in this phase. Combining the two bounds, we have

$$\text{cost}_{\text{FIFO}}(I) \leq 3 \cdot \text{cost}_{\text{OPT}}(I) + \alpha,$$

where $\alpha \leq 3$ is the maximal number of page faults incurred by FIFO in the last phase. A sequence for which FIFO actually yields three times as many page faults as OPT is given by

$$I = (p_1,\ p_2,\ p_3,\ p_4,\ p_1,\ p_2,\ \ldots)\ .$$

(ii) LFU *(Least Frequently Used): Replace the page that has been requested the smallest number of times since entering the fast memory.*
The LFU strategy is *not competitive*. Consider the following request sequence:

$$I = (p_1,\ p_1,\ p_2,\ p_2,\ p_3,\ p_4,\ p_3,\ p_4,\ p_3,\ \ldots)$$

In this sequence, LFU keeps on exchanging p_3 and p_4 ad infinitum, while the optimum can keep these two pages in the cache.

(iii) LIFO *(Last-in/First-out): Replace the page most recently moved to the cache.*

For the same reason as LFU, the LIFO strategy is also *not competitive*. Consider the following request sequence:

$$I = (p_1,\ p_2,\ p_3,\ p_4,\ p_3,\ p_4,\ p_3,\ \ldots)$$

In this sequence, LIFO keeps on exchanging p_3 and p_4 ad forever. It is therefore not competitive.

(iv) LRU *(Least Recently Used): When eviction is necessary, replace the page whose most recent request was the earliest.*

Like the FIFO strategy, LRU is 3-*competitive*. The reason is that (like FIFO), LRU has the property that on any consecutive input subsequence containing three or fewer distinct page references, it incurs at most three page faults. The remainder of the proof is then equivalent to the FIFO case.

(v) FWF *(Flush When Full): Whenever there is a page fault and there is no space left in the cache, evict* all *pages currently in the cache.*

The FWF algorithm is also 3-*competitive*. Consider the first phase, i.e., the first consecutive input subsequence containing three distinct page references. Clearly, FWF does not operate a flush. Now, consider the subsequent phase. In this phase, there can be at most one flush and hence, three page faults. Similarly, in every subsequent phase, there can be at most one flush and three page faults. From this observation, the proof follows like in the FIFO case.

b) We prove the following theorem:

Theorem. *There exists no deterministic online paging algorithm* ALG *with a competitive ratio better than* 3.

Proof. Assume that there are four pages, $p_1, \ldots, p_4$. We prove that there is an arbitrarily long request sequence I for which $|I| = \text{cost}_{\text{ALG}}(I) \geq 3 \cdot \text{cost}_{\text{OPT}}(I)$. Without loss of generality, assume that ALG initially holds p_1, p_2, and p_3 in its cache. We define a "cruel" request sequence $I = (r_1, r_2, r_3, \ldots)$ inductively: $r_1 = p_4$, and r_{i+1} is defined to be the unique page that is not in ALG's cache just after serving the request sequence $r_1, \ldots, r_i$. In other words, the adversary always requests the one page that ALG does not have in its cache. Clearly, I can be made arbitrarily long and ALG has a page fault on each request in I, hence $|I| = \text{cost}_{\text{ALG}}(I)$.

We now show, however, that it is possible to serve every request sequence I with at most $|I|/3$ page faults. Specifically, consider the offline algorithm NRL that knows the complete request sequence in advance and always evicts the one page from the cache whose *next request is latest*. Suppose that for serving the i-th request r_i, NRL evicts the page p. By the definition of NRL, and since there are four pages in total, it must be that all the pages in the cache (except perhaps r_i) must be requested prior to the next request of p. Hence, NRL has a page fault at most once every three requests.

Clearly, it holds that

$$\text{cost}_{\text{OPT}}(I) \leq \text{cost}_{\text{NRL}}(I) \leq \frac{|I|}{3} = \frac{\text{cost}_{\text{ALG}}(I)}{3},$$

which finishes the proof of the theorem. □