CSE515 Advanced Algorithms
Lecture 28: Randomized Distributed Algorithms

Antoine Vigneron

antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

July 27, 2021

Introduction

- In this lecture, we present randomized algorithms for two distributed computing problems.
- References:
  - Section 13.1 and 13.10 of Algorithm Design by Kleinberg and Tardos.

Contention Resolution

- Processes $P_1, \ldots, P_n$ need to access a shared database.
- There are several rounds, at each round, at most one process can access the database.
- If two or more processes try to access the database at round t, they are all blocked.

Contention Resolution

Example

- Round 1: Only P_2 tries to access database $\Rightarrow$ P_2 succeeds
- Round 2: P_1 and P_3 try to access database $\Rightarrow$ both fail
- Round 3: No process tries $\Rightarrow$ nobody succeeds
- Round 4: Only P_1 tries $\Rightarrow$ P_1 succeeds
- Round 5: Only P_2 tries to access database $\Rightarrow$ P_2 succeeds again
- ...

- We would like each process to be able to access the database reasonably often.
- If all requests are coordinated centrally, it is easy: Process P_i accesses the database at rounds i, $i + n$, $i + 2n$ ...
- It is also easy if the processes can communicate.

Contention Resolution

- What if the processes cannot communicate?
- If all processes follow the same deterministic algorithm, they all attempt to access the database at the same time, and fail.
- Solution: use randomization to break symmetry.

Algorithm

At each round, each process tries to access the database with probability p, for some fixed p.

- What is the probability that P_i succeeds at round t?

$$\Pr(S_{i,t}) = p(1-p)^{n-1}$$

because it is the probability that P_i tries to access the database, and none of the other processes does.

Contention Resolution

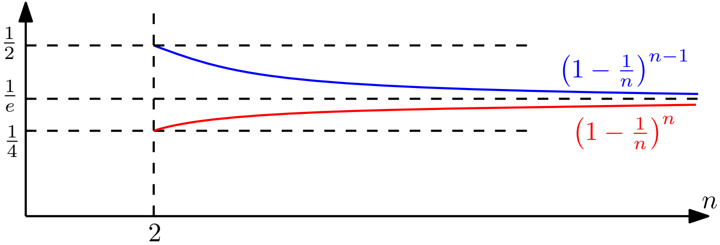
- How to choose p?
- Maximize $p(1-p)^{n-1}$:

$$\begin{aligned}(p(1-p)^{n-1})' &= (1-p)^{n-1} - p(n-1)(1-p)^{n-2} \\ &= (1-p)^{n-2}(1-pn).\end{aligned}$$

- So the maximum is achieved at $p = 1/n$, and then the probability of success is

$$\Pr(S_{i,t}) = \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1}.$$

Contention Resolution



Lemma

When n goes from 2 to ∞:

- $\left(1-\frac{1}{n}\right)^{n}$ *increases from 1/4.*
- $\left(1-\frac{1}{n}\right)^{n-1}$ *decreases from 1/2.*
- $\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^{n-1} = \lim_{n\to\infty}\left(1-\frac{1}{n}\right)^{n} = \frac{1}{e}.$

Waiting for a Particular Process to Succeed

- Let $F_{i,t}$ be the event that P_i does not succeed at any round from 1 to t.

$$
\begin{aligned}
\Pr\left(F_{i,t}\right) &= \prod_{r=1}^{t} \Pr\left(\overline{S_{i,t}}\right) = \prod_{r=1}^{t} 1 - \Pr\left(S_{i,t}\right) \\
&= \left(1 - \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1}\right)^{t} \\
&\leqslant \left(1 - \frac{1}{en}\right)^{t} && \text{by our lemma.}
\end{aligned}
$$

- We would like to choose t so that this probability is low.

Waiting for a Particular Process to Succeed

- We choose $t = \lceil en \rceil$, so that we can apply the lemma again.

$$\Pr\left(F_{i,t}\right) \leqslant \left(1 - \frac{1}{en}\right)^{\lceil en \rceil} \leqslant \left(1 - \frac{1}{en}\right)^{en} \leqslant \frac{1}{e}$$

- So process i succeeds at least once during the first $t = \lceil en \rceil$ rounds with probability at least $1 - 1/e \approx 0.63$
- To increase this probability, we can increase t.
- Let $t = \lceil en \rceil \cdot \lceil c \ln n \rceil$. Then

$$\Pr\left(F_{i,t}\right) \leqslant \left(\left(1 - \frac{1}{en}\right)^{en}\right)^{c \ln n} \leqslant \left(\frac{1}{e}\right)^{c \ln n} = \frac{1}{n^c}$$

Waiting for a Particular Process to Succeed

- So P_i succeeds at least once during the first $t = \lceil en \rceil \cdot \lceil c \ln n \rceil$ with probability $\geqslant 1 - 1/n^c$.
- In other words, it succeeds with high probability.

Waiting for all the Processes to Succeed

- Suppose now we want to wait long enough, so that all processes succeed with high probability.
- Let F_t be the event that at least one process did not succeed at any round from 1 to t:

$$F_t = \bigcup_{i=1}^{n} F_{i,t}.$$

- How to bound $\Pr(F_t)$? We use:

Proposition (Union bound)

For any events $E_1, \dots, E_n$, we have $\Pr\left(\bigcup_{i=1}^{n} E_i\right) \leqslant \sum_{i=1}^{n} \Pr(E_i)$.

Waiting for all the Processes to Succeed

- So if we set $c = 2$ and thus $t = \lceil en \rceil \cdot \lceil 2 \ln n \rceil$, we find

$$\Pr(F_t) \leqslant \sum_{i=1}^{n} \Pr(F_{i,t}) \leqslant n \frac{1}{n^2} = \frac{1}{n}.$$

- It follows that:

Theorem

With probability at least $1 - 1/n$, all processes succeed at least once during the first $\lceil en \rceil \cdot \lceil 2 \ln n \rceil$ rounds.

- Remark: This is a factor $O(\log n)$ from optimal, as it takes at least n rounds for all the processes to succeed.

Load Balancing

Problem

- *Suppose a stream of m jobs has to be processed by n processors.*
- *The system is not controlled centrally.*
- *How to assign the jobs?*

- If the system was controlled centrally, we could easily ensure that each processor handles $\lceil m/n \rceil$ jobs.
- What can we do if the system is distributed?
- Answer: Assign each job to a random processor.

Load Balancing: Case $m = n$

- Suppose $m = n$, where $m = \#$ jobs and $n = \#$ processors.
- $X_i =$ number of jobs handled by processor i.
- $Y_{ij} = \begin{cases} 1 & \text{if job } j \text{ is assigned to processor } i \\ 0 & \text{otherwise.} \end{cases}$

(Remark: Y_{ij} is an indicator variable.)

- So $X_i = \sum_{j=1}^{m} Y_j$, $E(Y_{ij}) = 1/n$ and thus

$$E(X_i) = \sum_{i=1}^{m} E(Y_{ij}) = \frac{m}{n} = 1.$$

Load Balancing: Case $m = n$

- How to bound $\Pr(X_i \geqslant c)$?
- We use Chernoff's bound for upper tail (see previous lecture), using $\mu = 1$ and $\delta = c - 1$:

$$\Pr(X_i \geqslant c) \leqslant \frac{e^{c-1}}{c^c}.$$

- Now we would like to find a bound on the probability that no processor takes more than c jobs.
- So we would like $\Pr(X_i \geqslant c) \leqslant \frac{e^{c-1}}{c^c}$ to be small enough, i.e. less than $1/n^2$.

Load Balancing: Case $m = n$

- Let $\gamma(n)$ be the number x such that $x^x = n$.

Lemma

$$\gamma(n) = \Theta\left(\frac{\log n}{\log\log n}\right)$$

Theorem

With probability at least $1 - 1/n$, no processor receives more than $e\gamma(n)$ jobs. In other words, with high probability, every processor receives $O(\log n/\log\log n)$ jobs.

- (Proofs done in class. See textbook.)

Increasing the Number of Processors

- We take $m = 16n \ln n$ jobs.
- Then the average number of jobs per processor is $\mu = 16 \ln n$.
- Applying Chernoff's bound for upper tail with $\delta = 1$, we get

$$\Pr(X_i > 2\mu) < \left(\frac{e}{4}\right)^{16 \ln n} < \left(\frac{1}{e^2}\right)^{\ln n} = \frac{1}{n^2}$$

  because $(e/4)^{16} < 1/e^2$.
- Applying Chernoff's bound for lower tail with $\delta = 1/2$, we get

$$\Pr(X_i < \mu/2) < e^{-\frac{1}{2}\left(\frac{1}{2}\right)^2 16 \ln n} = e^{-2 \ln n} = \frac{1}{n^2}.$$

Increasing the Number of Processors

- After applying the union bound, we obtain:

Theorem

If there are n processors and at least $\Omega(n \log n)$ jobs, then with high probability, every processor has a load between half and twice the average.