Randomized Algorithms

Zhou Jun

Content

13.1 CONTENTION RESOLUTION

1. The Problem
2. Algorithm Design
3. Analysis

The Problem

- We have n processes P_1 to P_n
- A shared database that can be accessed by at most one process in a single round.
- If more than one processes attempt to access – locked out.
- The n processes compete for the access to the database
- Processes cannot communicate with each other.

Algorithm Design

- Define the probability $0<p<1$.
- Each process will attempt to access the database with probability p.
- Each process decide independently from other processes.

Analysis

1. Rounds for a Particular Process to Succeed
2. Rounds for All Process to Succeed

Basic Events

1. $A\ [i,t]$ - P_i attempts to access the database in round t.

   $$\Pr\big[A\ [i,t]\big] = p$$

   $$\Pr\big[\overline{A\ [i,t]}\big] = 1\ - p$$

2. $S\ [i,t]$ - P_i succeeds to access the database in round t.

   $$S\ [i,t] = A\ [i,t] \cap \left(\cap_{j\neq i} \overline{A\ [j,t]}\right)$$

   $$\Pr\big[S\ [i,t]\big] = \ \Pr\big[A\ [i,t]\big] \cdot\ \prod_{j\neq i} \Pr\big[\overline{A\ [j\ t]}\big] = p(1-p)^{n-1}$$

$\Pr\big[S\ [i,t]\big]$ has maximum value when p=1/n, so we set p = 1/n for the following analysis.

(13.1)

1. The function $\left(1-\frac{1}{n}\right)^{n}$ converges monotonically from $\frac{1}{4}$ up to $\frac{1}{e}$ as n increase from 2
2. The function $\left(1-\frac{1}{n}\right)^{n-1}$ converges monotonically from $\frac{1}{2}$ down to $\frac{1}{e}$ as n increase from 2

From (13.1) We have $\frac{1}{en} \leq \Pr\left[S[i,t]\right] \leq \frac{1}{2n}$, and hence $\Pr\left[S[i,t]\right]$ is asymptotically equals to $\Theta\left(\frac{1}{n}\right)$.

Rounds for a Particular Process to Succeed

1. $F[i,t]$ - P_i fails to access the database from round 1 to t.

$$F[i,t] = \cap_{i=1}^{t} \overline{S[i,t]}$$

$$\Pr\left[\overline{F\ [i,t]}\right] = (1 - \Pr\left[S[i,t]\right])^t$$

2. $Pr\left[F[i,t]\right] = (1 - \Pr\left[S[i,t]\right])^t \leq \left(1 - \frac{1}{en}\right)^t$

Set $t = en, Pr\left[F[i,t]\right] \leq \left(1 - \frac{1}{en}\right)^{\lceil en \rceil} \leq \left(1 - \frac{1}{en}\right)^{en} \leq \frac{1}{e}.$

$Increse\ t\ to\ \lceil en \rceil \cdot c \ln n\,,$

$$Pr\left[F[i,t]\right] \leq \left(\left(1 - \frac{1}{en}\right)^{\lceil en \rceil}\right)^{c \ln n} \leq \left(\frac{1}{e}\right)^{c \ln n} = e^{-c \ln n} = n^{-c}.$$

Rounds for a Particular Process to Succeed

Conclusion:

After $\Theta(n)$ rounds, the probability that P_i has not succeeded in any rounds in bounded by a constant; and between $\Theta(n)$ and $\Theta(n \ln n)$, the probability drops to a very small value, bounded by n^{-c}.

Rounds for All Process to Succeed

F_t - Not all processes has succeed after t rounds.

$$F_t = \bigcup_{i=1}^{n} F[i,t]$$

(13.2) (The Union Bound)

$$\Pr\left[\bigcup_{i=1}^{n} \varepsilon_i\right] \leq \sum_{i=1}^{n} \Pr[\varepsilon_i]$$

$$From\ (13.2), \Pr[F_t] \leq \sum_{i=1}^{n} \Pr[F[i,t]].$$

$$If\ we\ take\ t = \lceil en \rceil \cdot c \ln n,\ \Pr[F_t] \leq \sum_{i=1}^{n} \Pr[F[i,t]] \leq \mathrm{n} \cdot \mathrm{n}^{-\mathrm{c}} = \mathrm{n}^{-\mathrm{c}+1}.$$

$$Let's\ say\ t = \lceil en \rceil \cdot 2 \ln n,\ \Pr[F_t] \leq \frac{1}{n}.$$

Rounds for All Process to Succeed

Conclusion:

With Probability at least $1 - \frac{1}{n}$, all processes succeed in accessing the database at least once within $t = 2\lceil en \rceil \ln n$ rounds.

GLOBAL MINIMUM CUT

1. The Problem
2. Algorithm Design
3. Analysis

The Problem

- **Cut:** In graph theory, a cut is a partition of the vertices of a graph into two disjoint subsets A and B.
- **s-t Cut:** a cut that a certain vertices s in subset A and t in subset B.
- **Size of cut (A,B):** number of edges with one end in A and the other in B
- **Global Minimum Cut:** A cut with minimum size among all cuts of a graph.

Problem:

Find the Global Minimum Cut.

The Problem

(13.4) There is a polynomial-time algorithm to find a global min-cut in an undirected graph G

Proof on white board

Algorithm Design

- **Multi-graph G = (V, E):** An undirected graph allowed to have multiple “parallel” edges between the same pair of nodes.
- **Contract (e = (u, v))**

  Combine u and v into a supernode w

  * w is actually a set of nodes, denoted by S(w)
- **Contract Algorithm:**

```
do
    select an edge e uniformly at random
    Contract(e)
until there left only 2 super nodes, say v1, and v2
return cut(S(v1), S(v2))
```

Analysis

(13.5) The Contraction Algorithm returns a global min-cut with probability at least $1/\binom{n}{2}$.

Proof on white board

Further Analysis

The Number of Global Minimum Cuts

(13.6) An undirected graph on n nodes has at most $\binom{n}{2}$. global min-cuts.

Proof

RANDOM VARIABLE AND EXPECTATION

1. Definitions
2. Examples

Definitions

1. **Random Variable**
2. **Expectation**
3. **Linearity of Expectation**

    $E[X+Y] = E[X]+E[Y]$

RANDOMIZED APPROXIMATION ALGORITHM FOR MAX 3-SAT

1. The Problem
2. Algorithm Design
3. Analysis

The Problem

3-SAT Problem:

Given a set of clauses, C_1,… C_k , each of length 3, over a set of variables X={x_1 ,… x_n}, does there exist a satisfying truth assignment.

Max 3-SAT Problem:

When 3-SAT problem has no solution, we want to have an optimized solution.

Design and Analysis

Algorithm: Assign each variable x_1 to x_n independently to 0 or 1 with probability ½ each.

(13.14) Consider a 3-SAT formula, where each clause has three different variables. The expected number of clauses satisfied by a random assignment is within an approximation factor 7/8 if optimal.

Proof on white board

(13.15) For every instance of 3-SAT, there is a truth assignment that satisfies at least a fraction 7/8 fraction of all clauses.

Proof: From (13.14), if there is no such assignment, the expectation cannot be 7/8.

(13.15) Application

- Every instance of 3-SAT with at most 7 clauses is satisfiable.

Waiting to Find a Good Assignment

Algorithm: Repeat until we find the good assignment.

Analysis:

Let p denote the probability of getting a good assignment.

For j = 0, 1, 2 …, k. let p_j denote the probability that a random assignment satisfies exactly j clauses. So the expected number of clauses satisfied is $\sum_{j=0}^{k} jp_j$; and from (13.14) is 7/8k.

We are interested in the quantity $p = \sum_{j \geq \frac{7k}{8}} p_j$.

We start by writing: $\frac{7}{8}k = \sum_{j=0}^{k} jp_j = \sum_{j < \frac{7k}{8}} jp_j + \sum_{j \geq \frac{7k}{8}} jp_j$

Let $k' = \left\lfloor \frac{7}{8}k \right\rfloor$. Then we have $\frac{7}{8}k \leq \sum_{j=0}^{k'} k' p_j + \sum_{j \geq \frac{7k}{8}} kp_j =$

$k'(1-p) + kp \leq k' + kp$

Hence $p \geq \frac{\frac{7}{8}k - k'}{k} \geq \frac{1}{8k}$ $(\frac{7}{8}k - k' \geq \frac{1}{8})$

- From (13.7), the expected number of trials needed to find a satisfying assignment we want is at most 8k.
- **(13.16)** There is a randomized algorithm with polynomial expected running time that is guaranteed to produce a truth assignment satisfying at least a 7/8 fraction of all clauses.

HASHING: A RANDOMIZED IMPLEMENTATION OF DICTIONARIES

1. The Problem
2. Algorithm Design
3. Analysis

The Problem

Universe: The set of all possible elements.

Dictionary: A data structure supporting the following operation:

- MakeDictionary
- Insert(u)
- Delete(u)
- Lookup(u)

Hashing

Hashing: The basic idea of hashing is to work with an array of size |S|, rather than one comparable to |U|.

We want to be able to store a set S of size up to n. We set up an array **H** of size n to store the information, and a function **h** from U to {0, 1,…,n-1}.

H: hash table. h: hash function.

Goal: Find a good hash function

(13.22) With a uniform random hashing scheme, the probability that two selected value collide – that is, h(u) = h(v) – is exactly 1/n.

Proof

Good Hash Function

The key idea is to choose a hash function from a carefully selected class of functions H. Each function h in H should have two properties:

1. For any pair of elements u, v in U, the probability that a randomly chosen h satisfies h(u) = h(v) is at most 1/n
2. Each h can be compactly represented and, for a given h and, we can compute the value h(u) efficiently.

All the random functions cannot satisfy the second properties. The only way to represent an arbitrary function is to write down all its values.

Design Hash

- We use a prime number $p \approx n$ as the size of hash table. We identify the universe with vectors of the form $x = (x_{1,}\ x_2, \ldots, x_r)$ for some integer r where $0 \leq x_i < p$ for all i.
- Let A be the set of all vectors of the form $a=(a_1,\ldots, a_r)$, where a_i is an integer in the range [0, p - 1] for each i = 1, …, r. For each a in A, we define the linear function

$$h_a(x) = \left(\sum_{i=1}^{r} a_i x_i\right) mod\ p$$

Design Hash

- Now we can define the family of hash functions $\mathrm{H} = \{h_a : a \in A\}$
- To define A, we need to have prime number p>=n. There are methods for generating such p, so we do not go into here.
- Now we can build a dictionary by randomly selecting an h_a from H.

Analysis

- Apparently, this class of hash functions satisfy the second property. We can represent it compactly and compute h(u) efficiently. Now we only need to show it satisfy the first property:
  - For any pair of elements u, v in U, the probability that a randomly chosen h satisfies h(u) = h(v) is at most 1/n

Analysis

$(\mathbf{13.24})$ $For\ any\ prime\ p\ and\ any\ integer\ z \neq 0\ mod\ p, and\ any\ two\ integers\ \alpha\ and\ \beta, if\ \alpha z = \beta z\ mod\ p, then\ \alpha = \beta\ mod\ p$.

Proof: From αz=βz mod p, we have z(α−β)=0 mod p, hence z(α−β) is divisible by p. Since z is not divisiable by p, (α−β) is divisibale by p. Thus α=β mod p

Analysis

(13.25) The class of linear functions H defined above is universal.

Proof

(13.23) Let H be a universal class of hash functions mapping a universe U to the set $\{0, 1, \ldots, n-1\}$, let S be an arbitrary subset of U of size at most n, and let u be any element in U. We define X to be a random variable equal to the number of elements $s \in S$ for which $h(s) = h(u)$, for a random choice of hash function $h \in H$. (Here S and u are fixed, and the randomness is in the choice of $h \in H$.) Then $E[X] \leq 1$.

FINDING THE CLOSEST PAIR OF POINTS: A RANDOMIZED APPROACH

1. The Problem
2. Algorithm Design
3. Analysis

The Problem

Given n points in a plane, we wish to find the pair that closest to each other.

Notations:

$P = \{p_1, p_2, \ldots, p_n\}$

p_i is denoted by (x_i, y_i)

$d(p_i, p_j)$ is the distance

To simplify the discussion, we assume all points are in a unit square.

Algorithm Design

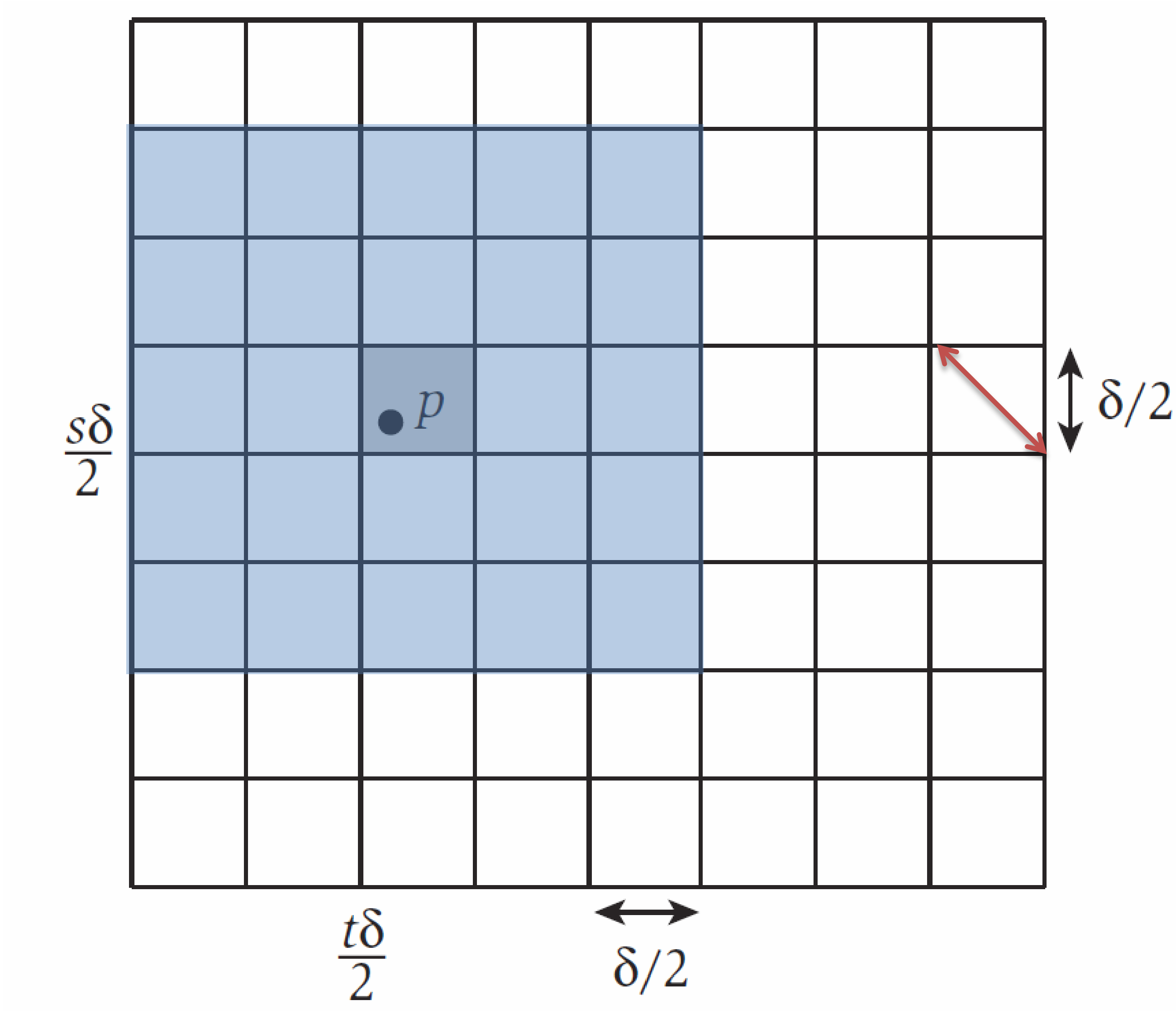
- subdivide the unit square into sub-squares whose sides have length $\delta/2$
- There are totally $\left\lceil \frac{2}{\delta} \right\rceil^2$ subsquares.
- We index the squares by

$$S_{st} = \{(x, y) : s\delta/2 \leq x < (s+1)\delta/2;\ t\delta/2 \leq y < (t+1)\delta/2\}$$

(13.26) If two points p and q belong to the same sub-square S_{st}, then $d(p, q) < \delta$.

(13.27) If for two points $p, q \in P$ we have $d(p, q) < \delta$, then the subs-quares containing them are close.

Algorithm Design



Algorithm Design

```
Order the points in a random sequence p_1, p_2, ..., p_n
Let δ denote the minimum distance found so far
Initialize δ = d(p_1, p_2)
Invoke MakeDictionary for storing subsquares of side length δ/2
For i = 1, 2, ..., n:
  Determine the subsquare S_st containing p_i
  Look up the 25 subsquares close to p_i
  Compute the distance from p_i to any points found in these subsquares
  If there is a point p_j (j < i) such that δ' = d(p_j, p_i) < δ then
    Delete the current dictionary
    Invoke MakeDictionary for storing subsquares of side length δ'/2
    For each of the points p_1, p_2, ..., p_i:
      Determine the subsquare of side length δ'/2 that contains it
      Insert this subsquare into the new dictionary
    Endfor
  Else
    Insert p_i into the current dictionary
  Endif
Endfor
```

Analysis

For Each point p we pick, the have the following operations:

1. Look up dictionary for points in 5*5 grid: O(1)
2. Compute the distance of the points: O(1)
3. Insert p to the set: 1

*. If δ change, we make a new dictionary: 1

We will pick n points, therefore:

Analysis

(13.28) The algorithm correctly maintains the closest pair at all times, and it performs at most O(n) distance computations, O(n) Lookup operations, and O(n) MakeDictionary operations.***Plus n insert operations.**

- Let random variable X be the number of total insert operations.
- let X_i be equal to 1 if the ith point causes δ to change, and equal to 0 otherwise.
- **(13.29)** $X = n + \sum_i iX_i$
- **(13.30)** $Pr[X_i = 1] <= 2/i$
- $E[X] = n + \sum_{\boldsymbol{i=1}}^{n} iE[X_i] \leq n + 2n = 3n$

Analysis

(13.31) In expectation, the randomized closest-pair algorithm requires O(n) time plus O(n) dictionary operations.

Now we will prove the O(n) dictionary operations will take O(n) time.

Analysis

(13.32) Assume we implement the randomized closest-pair algorithm using a universal hashing scheme. In expectation, the total number of points considered during the Lookup operations is bounded by O(n).

Proof on white board

(13.33) In expectation, the algorithm uses O(n) hash-function computations and O(n) additional time for finding the closest pair of points.

RANDOMIZED CACHING

The Problem

- Suppose a processor has n memories and k cache slots.
- The optimal algorithm is Farthest-in-Future policy, which is not practical
- Suppose a sequence σ of memory request
- f (σ) denotes the minimum number of missing which is achieved by the optimal Farthest-in-Future policy

Marking Algorithm

Design:

```
Each memory item can be either marked or unmarked
At the beginning of the phase, all items are unmarked
On a request to item s:
   Mark s
   If s is in the cache, then evict nothing
   Else s is not in the cache:
      If all items currently in the cache are marked then
         Declare the phase over
         Processing of s is deferred to start of next phase
      Else evict an unmarked item from the cache
      Endif
   Endif
```

Marking Algorithm

Analysis:

(13.35) In each phase, σ contains accesses to exactly k distinct items. The subsequent phase begins with an access to a different (k+1)th item.

(13.36) The marking algorithm incurs at most k misses per phase, for a total of at most kr misses over all r phases.

(13.37) The optimum incurs at least $r - 1$ misses. In other words, $f(\sigma) \geq r - 1$.

(13.38) For any marking algorithm, the number of misses it incurs on any sequence σ is at most $k \cdot f(\sigma) + k$

Randomized Marking Algorithm

Design:

```
Each memory item can be either marked or unmarked
At the beginning of the phase, all items are unmarked
On a request to item s:
  Mark s
  If s is in the cache, then evict nothing
  Else s is not in the cache:
    If all items currently in the cache are marked then
      Declare the phase over
      Processing of s is deferred to start of next phase
    Else evict an unmarked item chosen uniformly at random
        from the cache
    Endif
  Endif
```

Randomized Marking Algorithm

Analysis:

- We call an unmarked item **fresh** if it was not marked in the previous phase either, and **stale** if it was marked.
- Among k accesses to unmarked items in phase j, c_j denote number of fresh items.

(13.39) $f(\sigma) \geq \frac{1}{2} \sum_{i=1}^{r} c_j$

Analysis

1. Let random variable M_σ denote the number of cache misses incurred.
2. Let X_j denote the number of misses in phase j
3. There are at least c_j misses.
4. For an ith request to a stale item, suppose there have been ***c*** $\leq c_j$ requests to fresh items. Then the cache contains the ***c*** formerly fresh items that are now marked, **i−1** stale items now marked, and ***k* − *c* − *i* + 1** items that are stale and not marked
5. There are k − i + 1 items are still stale not yet marked.
6. The probability of not in cache is

$$\frac{(k-i+1)-(k-i+1-c)}{k-i+1} = \frac{c}{k-i+1} \leq \frac{c_j}{k-i+1}$$

Analysis

7. $\mathrm{E}\left[\mathrm{X_j}\right] \leq \mathrm{c_j} + \sum_{\mathrm{i=1}}^{\mathrm{k-c_j}} \frac{\mathrm{cj}}{\mathrm{k-i+1}} \leq \mathrm{c_j}\left[1 + \sum_{\mathrm{l=c_j+1}}^{\mathrm{k}} \frac{1}{\mathrm{l}}\right] = \mathrm{c_j}\left(1 + \log \mathrm{k} - \log \mathrm{c_j}\right) \leq \mathrm{c_j} \log \mathrm{k}$

(l=cj+i)

8. $E[M_\sigma] = \sum_{j=1}^{r} E\left[X_j\right] \leq \log \mathrm{k} \sum_{j=1}^{r} \mathrm{c_j}$

9. We have **(13.39)** $f(\sigma) \geq \frac{1}{2}\sum_{i=1}^{r} c_j$

10. $E[M_\sigma] \leq 2 \log \mathrm{k}\, f(\sigma)$

(13.41) The expected number of misses incurred by the Randomized Marking Algorithm is at most 2logk·f(σ)=O(log k)·f(σ).

CHERNOFF BOUNDS

Problem

A random variable X that is a sum of several independent 0-1valued random variables: $X = X_1 + X_2 + X_3 + \ldots + X_n$, where X_i takes the value 1 with probability p_i, and the value 0 otherwise.

Analysis

(13.42) Let $X_1, X_2, X_3, . . . , X_n$ be defined as above, and assume that μ ≥ E [X]. Then, for any δ > 0, we have

$$\Pr[X > (1 + \delta)\mu] < \left[\frac{e^{\delta}}{(1 + \delta)^{1+\delta}}\right]^{u}$$

(13.43) Let $X_1, X_2, X_3, . . . , X_n$ be defined as above, 0< δ<1, we have

$$\Pr[X < (1-\delta)\mu] < e^{-\frac{1}{2}\mu\delta^2}$$

LOAD BALANCING

1. The Problem
2. Analysis

The Problem

- We distribute m jobs to totally n processors randomly.
- Analyze how well this algorithm will work

Analysis: m=n

- Let X_i be the random variable equal to the number of jobs assigned to processor i.
- Let Y_{ij} be the random variable equal to 1 if job j is assigned to processor i, and 0 otherwise.
- Clearly $E[X_i]=1$. But what is the probability that $X_i >$ c?
- With (13.42) $\Pr[X > (1+\delta)\mu] < \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{u}$, we let u=1 and c=1+δ, therefore
- (13.44) $\Pr[X_i > c] < \frac{e^{c-1}}{c^{c}}$

Analysis m=n

$$\Pr\left[X_i > c\right] < \left(\frac{e^{c-1}}{c^c}\right) < \left(\frac{e}{c}\right)^c = \left(\frac{1}{\gamma(n)}\right)^{e\gamma(n)} < \left(\frac{1}{\gamma(n)}\right)^{2\gamma(n)} = \frac{1}{n^2}.$$

- **(13.45)** With Probability at least $1-n^{-1}$, no processor receives more than $e\gamma(n) = \Theta\left(\frac{\log n}{\log\log n}\right)$ jobs.

Analysis: m>n

if we have m = 16nln n jobs, then the expected load per processor is μ = 16 ln n

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

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

(13.46) When there are n processors and Ω(nlog n) jobs, then with high probability, every processor will have a load between half and twice the average.

PACKET ROUTING

1. The Problem
2. Algorithm Design
3. Analysis

The Problem

- A single edge e can only transmit a single packet per time step
- Given packets labeled 1, 2, . . . , N and associated paths P_1, P_2, . . . , P_N, a packet schedule specifies, for each edge e and each time step t, which packet will cross edge e in step t.
- the duration of the schedule is the number of steps that elapse until every packet reaches its destination
- **Goal:** Find a schedule of minimum duration

The Problem

Obstacles:

1. Dilation d: the maximum length of any P_i
2. Congestion c: the maximum number that have any single edge in common

The duration is at least Ω (c + d)

Algorithm Design

Each packet i behaves as follows:

i chooses a random delay s between 1 and r

i waits at its source for s time steps

i then moves full speed ahead, one edge per time step until it reaches its destination

---

For a parameter b, group intervals of b consecutive time steps into single blocks of time

Each packet i behaves as follows:

i chooses a random delay s between 1 and r

i waits at its source for s blocks

i then moves forward one edge per block, until it reaches its destination

The Problem

- **(13.47)** Let ε denote the event that more than b packets are required to be at the same edge e at the start of the same block. If ε does not occur, then the duration of the schedule is at most b(r+d)
- Our goal is now to choose values of r and b so that both the probability Pr [ε] and the duration b(r + d) are small quantities

Analysis

1. let F_{et} denote the event that more than b packets are required to be at e at the start of block t. Clearly, $\varepsilon = \bigcup e,t \; F_{e,t}$
2. N_{et} is equal to the number of packets scheduled at e at the start of block t, then F_{et} is equivalent to the event $[N_{et} > b]$.
3. X_{eti} equal to 1if packet i is required to be at edge e at the start of block t, and equal to 0 otherwise. $E[X_{eti}] = 1/r$
4. We say at most c packets have paths that include e, $E[N_{et}] <= c/r$

Analysis

1. let F_{et} denote the event that more than b packets are required to be at e at the start of block t. Clearly, $\varepsilon = \bigcup e, t \ F_{e,t}$
2. N_{et} is equal to the number of packets scheduled at e at the start of block t, then F_{et} is equivalent to the event $[N_{et} > b]$.
3. X_{eti} equal to 1if packet i is required to be at edge e at the start of block t, and equal to 0 otherwise. $E[X_{eti}] = 1/r$
4. We say at most c packets have paths that include e, $E[N_{et}] <= c/r$

Analysis

5. $r = \frac{c}{qlog(mN)}$

6. We define μ = c/r, and observe that $E[N_{et}] \le \mu$. Choose δ = 2, so that $(1+\delta)\mu = \frac{3c}{r} = 3qlog(mN)$

7. $\Pr\left[N_{et} > \frac{3c}{r}\right] = \Pr[N_{et} > (1+\delta)\mu]$

$$\Pr\left[N_{et} > \frac{3c}{r}\right] < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu} < \left[\frac{e^{1+\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu} = \left(\frac{e}{1+\delta}\right)^{(1+\delta)\mu}$$

$$= \left(\frac{e}{3}\right)^{(1+\delta)\mu} = \left(\frac{e}{3}\right)^{3c/r} = \left(\frac{e}{3}\right)^{3q\log(mN)} = \frac{1}{(mN)^{z}},$$

Analysis

8. Here we can choose b=3c/r

9. There are m different choices for e, and d + r different choice for t, where we observe that d + r ≤ d + c − 1≤ N. Thus we have

$$\Pr[\mathcal{E}] = \Pr\left[\bigcup_{e,t} \mathcal{F}_{et}\right] \le \sum_{e,t} \Pr[\mathcal{F}_{et}] \le mN \cdot \frac{1}{(mN)^z} = \frac{1}{(mN)^{z-1}}$$

Analysis

(13.48) With high probability, the duration of the schedule for the packets is O(c + d log (mN)).

$$b(r+d) = \frac{3c}{r}\left(r+d\right) = 3c + d \cdot \frac{3c}{r} = 3c + d(3q\log(mN)) = O(c + d\log(mN))$$