13. Randomized Algorithms

Algorithmic design patterns.
- Greed.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomization.

Randomization. Allow fair coin flip in unit time.

Why randomize? Can lead to simplest, fastest, or only known algorithm for a particular problem.

Ex. Symmetry breaking protocols, graph algorithms, quicksort, hashing, load balancing, Monte Carlo integration, cryptography.

13.1 Contention Resolution

Contention resolution. Given $n$ processes $P_1, ..., P_n$, each competing for access to a shared database. If two or more processes access the database simultaneously, all processes are locked out. Devise protocol to ensure all processes get through on a regular basis.

Restriction. Processes can't communicate.

Challenge. Need symmetry-breaking paradigm.
Contestion Resolution: Randomized Protocol

**Protocol.** Each process requests access to the database at time \( t \) with probability \( p = 1/n \).

**Claim.** Let \( S[i, t] \) = event that process \( i \) succeeds in accessing the database at time \( t \). Then \( 1/(e \cdot n) \leq \Pr[S(i, t)] \leq 1/(2n) \).

**Pf.** By independence, \( \Pr[S(i, t)] = p \cdot (1-p)^{n-1} \).

- Setting \( p = 1/n \), we have \( \Pr[S(i, t)] = 1/n \cdot (1 - 1/n)^{n-1} \).
- Value that maximizes \( \Pr[S(i, t)] \) between 1/e and 1/2.

**Useful fact.** As \( n \) increases from 2, the function:
- \( (1 - 1/n)^n \) converges monotonically from 1/4 up to 1/e.
- \( (1 - 1/n)^{n-1} \) converges monotonically from 1/2 down to 1/e.

Contestion Resolution: Randomized Protocol

**Claim.** The probability that all processes succeed within \( 2e \cdot n \ln n \) rounds is at least \( 1 - 1/n \).

**Pf.** Let \( F[t] \) = event that at least one of the \( n \) processes fails to access database in any of the rounds 1 through \( t \).

\[
\Pr[F[t]] = \Pr[(\bigcup_{i=1}^{n} F[i, t]) = \sum_{i=1}^{n} \Pr[F[i, t]] \leq n \cdot \left(1 - \frac{1}{e}\right)^t
\]

- Choosing \( t = 2 \cdot [\ln n] \cdot [c \ln n] \) yields \( \Pr[F[t]] \leq n \cdot n^2 = 1/n \).

13.2 Global Minimum Cut
Global Minimum Cut

Global min cut. Given an undirected graph $G = (V, E)$ find a cut $(A, B)$ of minimum cardinality.

Applications. Partitioning items in a database, identify clusters of related documents, network reliability, network design, circuit design, TSP solvers.

Network flow solution.
- Replace every edge $(u, v)$ with two antiparallel edges $(u, v)$ and $(v, u)$.
- Pick some vertex $s$ and compute min $s-v$ cut separating $s$ from each other vertex $v \in V$.

False intuition. Global min-cut is harder than min $s-t$ cut.

Contraction Algorithm

Claim. The contraction algorithm returns a min cut with prob $\geq 2/n^2$.

Pf. Consider a global min-cut $(A^*, B^*)$ of $G$. Let $F^*$ be edges with one endpoint in $A^*$ and the other in $B^*$. Let $k = |F^*| = $ size of min cut.
- In first step, algorithm contracts an edge in $F^*$ probability $k / |E|$. Every node has degree $\geq k$ since otherwise $(A^*, B^*)$ would not be min-cut. $\Rightarrow |E| \geq \frac{1}{2}kn$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n$.

Contraction Algorithm

Claim. The contraction algorithm returns a min cut with prob $\geq 2/n^2$.

Pf. Consider a global min-cut $(A^*, B^*)$ of $G$. Let $F^*$ be edges with one endpoint in $A^*$ and the other in $B^*$. Let $k = |F^*| = $ size of min cut.
- Let $G'$ be graph after $j$ iterations. There are $n' = n - j$ supernodes.
- Suppose no edge in $F^*$ has been contracted. The min-cut in $G'$ is still $k$.
- Since value of min-cut is $k$, $|E'| \geq \frac{1}{2}kn'$.
- Thus, algorithm contracts an edge in $F^*$ with probability $\leq 2/n'$.

- Let $E_j$ = event that an edge in $F^*$ is not contracted in iteration $j$.

$$Pr[E_1 \cap E_2 \cdots \cap E_{n-2}] = Pr[E_1] \times Pr[E_2 \mid E_1] \times \cdots \times Pr[E_{n-2} \mid E_1 \cap E_2 \cdots \cap E_{n-3}]$$

$$= \left(1 - \frac{1}{2}\right) \left(1 - \frac{3}{2^{n-1}}\right) \cdots \left(1 - \frac{2}{2^{n-3}}\right)$$

$$= \frac{\left(\frac{2}{2^{n-1}}\right) \left(\frac{4}{2^{n-2}}\right) \cdots \left(\frac{2^{n-2}}{2^{n-1}}\right)}{\left(\frac{2}{2^{n-1}}\right) \left(\frac{4}{2^{n-1}}\right) \cdots \left(\frac{2^{n-1}}{2^{n-1}}\right)}$$

$$= \frac{2}{2^n}$$
Contraction Algorithm

Amplification. To amplify the probability of success, run the contraction algorithm many times.

Claim. If we repeat the contraction algorithm n^2 \ln n times with independent random choices, the probability of failing to find the global min-cut is at most 1/n^2.

Pf. By independence, the probability of failure is at most
\[
\left(1 - \frac{2}{n^2}\right)^{n^2 \ln n} \leq \left(e^{-1}\right)^{2\ln n} = \frac{1}{n^2} \\
(1 - 1/x)^x \leq 1/e
\]

13.3 Linearity of Expectation

Global Min Cut: Context

Remark. Overall running time is slow since we perform Θ(n^2 \log n) iterations and each takes Ω(m) time.

Improvement. [Karger-Stein 1996] O(n^2 \log^3 n).
- Early iterations are less risky than later ones: probability of contracting an edge in min cut hits 50% when n / √ 2 nodes remain.
- Run contraction algorithm until n / √ 2 nodes remain.
- Run contraction algorithm twice on resulting graph, and return best of two cuts.

Extensions. Naturally generalizes to handles positive weights.

Best known. [Karger 2000] O(m \log^5 n).
- Faster than best known max flow algorithm or deterministic global min cut algorithm.

Expectation

Expectation. Given a discrete random variables X, its expectation E[X] is defined by:
\[ E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] \]

Linearity of expectation. Given two random variables X and Y defined over the same probability space, E[X + Y] = E[X] + E[Y].

Waiting for a first success. Coin is heads with probability p and tails with probability 1-p. How many independent flips X until first heads?
\[
E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{\infty} j \cdot \left(1 - p\right)^{j-1} p = \frac{p}{1 - p} \sum_{j=0}^{\infty} \left(1 - p\right)^j = \frac{p}{1 - p} \cdot \frac{1 - p}{p^2} = \frac{1}{p}
\]

**Guessing Cards**

**Game.** Shuffle a deck of $n$ cards; turn them over one at a time; try to guess each card.

**Memoryless guessing.** No psychic abilities; can’t even remember what’s been turned over already. Guess a card from full deck uniformly at random.

**Claim.** The expected number of correct guesses is $1$.

**Pf.**
- Let $X_i = 1$ if $i^{th}$ prediction is correct and $0$ otherwise.
- Let $X = \text{number of correct guesses} = X_1 + \ldots + X_n$.
- $E[X] = 0 \cdot \Pr[X_i = 0] + 1 \cdot \Pr[X_i = 1] = \Pr[X_i = 1] = 1/n$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1$.

**Guessing with memory.** Guess a card uniformly at random from cards not yet seen.

**Claim.** The expected number of correct guesses is $\Theta(\log n)$.

**Pf.**
- Let $X_i = 1$ if $i^{th}$ prediction is correct and $0$ otherwise.
- Let $X = \text{number of correct guesses} = X_1 + \ldots + X_n$.
- $E[X_i] = 1 / (n - i - 1)$.
- $E[X] = 1/n + \ldots + 1/2 + 1/1 = H(n)$.
- \[ \ln(n+1) < H(n) < 1 + \ln n \]

**Coupon Collector**

**Coupon collector.** Each box of cereal contains a coupon. There are $n$ different types of coupons. Assuming all boxes are equally likely to contain each coupon, how many boxes before you have $1$ coupon of each type?

**Claim.** The expected number of steps is $\Theta(n \log n)$.

**Pf.**
- Phase $j$ = time between $j$ and $j+1$ distinct coupons.
- Let $X_j = \text{number of steps you spend in phase } j$.
- Let $X = \text{number of steps in total} = X_0 + X_1 + \ldots + X_{n-1}$.

\[
E[X] = \sum_{j=0}^{n-1} E[X_j] = \sum_{j=0}^{n-1} \frac{n}{n - j} = n \sum_{i=1}^{n} \frac{1}{i} = n H(n)
\]

\[ \text{prob of success} = \frac{n-n}{n} \]
\[ \Rightarrow \text{expected waiting time} = n H(n) \]

---

13.4 MAX 3-SAT
Maximum 3-Satisfiability

**MAX-3SAT.** Given 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.

|= x_2 \lor \overline{x_3} \lor \overline{x_4} \\
| = x_2 \lor x_3 \lor \overline{x_4} \\
| = \overline{x_1} \lor x_2 \lor x_4 \\
|= x_1 \lor x_2 \lor x_4

**Remark.** NP-hard search problem.

**Simple idea.** Flip a coin, and set each variable true with probability \(\frac{1}{2}\), independently for each variable.

### Maximum 3-Satisfiability: Analysis

**Claim.** Given a 3-SAT formula with \(k\) clauses, the expected number of clauses satisfied by a random assignment is \(7k/8\).

**Pf.** Consider random variable \(Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases}\)

- Let \(E[Z] = \text{weight of clauses satisfied by assignment } Z\).

\[
E[Z] = \sum_{j=1}^{k} E[Z_j]
\]

linearity of expectation

\[
= \sum_{j=1}^{k} \Pr[\text{clause } C_j \text{ is satisfied}]
= \frac{7k}{8}
\]

**Corollary.** For any instance of 3-SAT, there exists a truth assignment that satisfies at least a \(7/8\) fraction of all clauses.

### Maximum 3-Satisfiability: Analysis

**Johnson’s algorithm.** Repeatedly generate random truth assignments until one of them satisfies \(\geq 7k/8\) clauses.

**Theorem.** Johnson’s algorithm is a \(7/8\)-approximation algorithm.

**Pf.** By previous lemma, each iteration succeeds with probability at least \(1/(8k)\). By the waiting-time bound, the expected number of trials to find the satisfying assignment is at most \(8k\).
Maximum Satisfiability

Extensions.
- Allow one, two, or more literals per clause.
- Find max weighted set of satisfied clauses.

Theorem. [Asano-Williamson 2000] There exists a 0.784-approximation algorithm for MAX-SAT.

Theorem. [Karloff-Zwick 1997, Zwick+computer 2002] There exists a 7/8-approximation algorithm for version of MAX-3SAT where each clause has at most 3 literals.

Theorem. [Håstad 1997] Unless \( P = NP \), no \( \rho \)-approximation algorithm for MAX-3SAT (and hence MAX-SAT) for any \( \rho > 7/8 \).

\( \uparrow \)

very unlikely to improve over simple randomized algorithm for MAX-3SAT

Monte Carlo vs. Las Vegas Algorithms

Monte Carlo algorithms. Guaranteed to run in poly-time, likely to find correct answer.

Ex: Contraction algorithm for global min cut.

Las Vegas algorithms. Guaranteed to find correct answer, likely to run in poly-time.

Ex: Randomized quicksort, Johnson’s MAX-3SAT algorithm.

stop algorithm after a certain point

Remark. Can always convert a Las Vegas algorithm into Monte Carlo, but no known method to convert the other way.

RP and ZPP

RP. [Monte Carlo] Decision problems solvable with one-sided error in poly-time.

One-sided error.
- If the correct answer is no, always return no.
- If the correct answer is yes, return yes with probability \( \geq \frac{1}{2} \).

ZPP. [Las Vegas] Decision problems solvable in expected poly-time.

\( \uparrow \)

running time can be unbounded, but on average it is fast

Theorem. \( P \subseteq ZPP \subseteq RP \subseteq NP \).

Fundamental open questions. To what extent does randomization help?
Does \( P = ZPP \)? Does \( ZPP = RP \)? Does \( RP = NP \)?
**Dictionary Data Type**

**Dictionary.** Given a universe $U$ of possible elements, maintain a subset $S \subseteq U$ so that **inserting**, deleting, and **searching** in $S$ is efficient.

**Dictionary interface.**
- **Create()**: Initialize a dictionary with $S = \emptyset$.
- **Insert (u)**: Add element $u \in U$ to $S$.
- **Delete (u)**: Delete $u$ from $S$, if $u$ is currently in $S$.
- **Lookup (u)**: Determine whether $u$ is in $S$.

**Challenge.** Universe $U$ can be extremely large so defining an array of size $|U|$ is infeasible.

**Applications.** File systems, databases, Google, compilers, checksums, P2P networks, associative arrays, cryptography, web caching, etc.

**Algorithmic Complexity Attacks**

**Deterministic hashing.** If $|U| \geq n^2$, then for any fixed hash function $h$, there is a subset $S \subseteq U$ of $n$ elements that all hash to same slot. Thus, $\Theta(n)$ time per search in worst-case.

**When do we need stronger performance guarantees?**
- Obvious situations: aircraft control, nuclear reactors.
- Surprising situations: denial-of-service attacks.

**Real-world exploits.** [Crosby-Wallach 2003]
- Bro server: send carefully chosen packets to DOS the server, using less bandwidth than a dial-up modem.
- Perl 5.8.0: insert carefully chosen strings into associative array.
- Linux 2.4.20 kernel: save files with carefully chosen names.

**Hashing**

**Hash function.** $h : U \rightarrow \{0, 1, \ldots, n-1\}$.

**Hashing.** Create an array $H$ of size $n$. When processing element $u$, access array element $H[h(u)]$.

**Collision.** When $h(u) = h(v)$ but $u \neq v$.
- Birthday paradox $\Rightarrow$ expect a collision after $\sqrt{n}$ random insertions.
- Separate chaining: $H[i]$ stores linked list of elements $u$ with $h(u) = i$.

**Idealistic hash function.** Maps $m$ elements uniformly at random to $n$ hash slots.
- Running time depends on length of chains.
- Average length of chain $= \alpha = m / n$.
- Choose $n = m$ $\Rightarrow$ on average $O(1)$ per insert, lookup, or delete.
- Max length of chain $= O(\log m / \log \log m)$, assuming $\alpha$ is a constant.

**Challenge.** Find an efficiently computable hash function $h$ that has idealized properties.

**Approach.** Use randomization in the choice of $h$.
Universal Hashing

Universal class of hash functions. [Carter-Wegman, 1980s]
- For any pair of elements \( u, v \in U \), \( \Pr_{h \in H} [ h(u) = h(v) ] \leq 1/n \)
- Can select random \( h \) efficiently.
- Can compute \( h(u) \) efficiently.

Ex. \( U = \{ a, b, c, d, e, f \} \), \( n = 2 \).

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_a(x) )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( h_b(x) )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ H = \{ h_a, h_b \} \]
\[ \Pr_{h \in H} [ h(a) = h(b) ] = 1/2 \]
\[ \Pr_{h \in H} [ h(a) = h(c) ] = 1 \]
\[ \Pr_{h \in H} [ h(a) = h(d) ] = 0 \]

Not universal

\[ H = \{ h_a, h_2, h_3, h_4 \} \]
\[ \Pr_{h \in H} [ h(a) = h(b) ] = 1/2 \]
\[ \Pr_{h \in H} [ h(a) = h(c) ] = 1/2 \]
\[ \Pr_{h \in H} [ h(a) = h(d) ] = 1/2 \]
\[ \Pr_{h \in H} [ h(a) = h(e) ] = 1/2 \]
\[ \Pr_{h \in H} [ h(a) = h(f) ] = 0 \]

Universal

Designing a Universal Class of Hash Functions

**Theorem.** \( H = \{ h_a : a \in A \} \) is a universal class of hash functions.

**Proof.** Let \( x = (x_1, x_2, ..., x_r) \) and \( y = (y_1, y_2, ..., y_r) \) be two distinct elements of \( U \). We need to show that \( \Pr[h_a(x) = h_a(y)] \leq 1/n \).
- Since \( x \neq y \), there exists an integer \( j \) such that \( x_j \neq y_j \).
- We have \( h_a(x) = h_a(y) \) iff
\[
\sum_{i=1}^{r} a_i (x_i - y_i) \mod p = 0
\]
- Can assume \( a_i \) was chosen uniformly at random by first selecting all coordinates \( a_i \) where \( i \neq j \), then selecting \( a_j \) at random. Thus, we can assume \( a_j \) is fixed for all coordinates \( i \neq j \).
- Since \( p \) is prime, \( a_j = m z^{-1} \) is the unique solution among \( p \) possibilities.
- Thus \( \Pr[h_a(x) = h_a(y)] = 1/p = 1/n \).
Bloom Filters

Bloom Filter

**Bloom filter.** [Bloom, 1978] Ingenious data structure to maximize space efficiency by allowing some false positives.
- Maintain one bit-array of size $n \approx 8 |S|$.
- Choose $k$ independent hash functions $h_1, ..., h_k : U \rightarrow \{1, ..., n\}$.
- **Insert**($u$): set $H[h_i(u)] = 1$ for each $i = 1, ..., k$
- **Exists**($u$): check $H[h_i(u)] = 1$ for each $i = 1, ..., k$
  - if any are 0, $u$ is definitely not in set
  - if all are 1, $u$ is probably in the set

![Bloom Filter Diagram]

Set Membership

**Set membership.** Represent a set $S$ of $m$ elements from universe $U$ where $|U| >> |S|$ to support membership queries.

**Ex.** ISP caches web pages, especially large data files like images and video. Client requests URL. Server needs to quickly determine whether the page is in its cache. False positive undesirable, but acceptable; don’t want false negatives.

More applications.
- Packet routing.
- Unsuitable password list.
- URLs in web proxy cache.
- Early Unix spell checkers.
- Network intrusion detection.
- Set intersection for keyword search.
- peed up semijoin operation in databases.
- Collaborating in overlay and P2P networks.

**Bloom Filter: Analysis**

**Parameters.** $m$ elements in $S$, $n$ bits in table, $k$ hash functions.

**Claim.** The probability of a false positive is at most $(0.62)^{m/k}$.

**Pf.**
- $Pr[H[i] = 0] \approx (1 - \frac{1}{e})^n = e^{-k \cdot m/n}$
- $Pr[\text{false positive for element } u] = Pr[h_1(u) = ... = h_k(u) = 1] = (1 - e^{-k \cdot m/n})^k$
- Optimal choice is $k = \ln 2 (n/m)$.
- $Pr[\text{false positive for element } u] = (1/2)^k \approx (0.62)^{n/m}$.

**Ex.** $n = 8m$, $k = 5$, false positive rate $= 2.2\%$. 

13.9 Chernoff Bounds

**Bloom Filter Summary**

**Advantage.** No collisions to deal with, efficient use of space, provides privacy since no way to reconstruct S from bit array.

**Disadvantage.** No easy way to store info with keys.

**Hashing.** Use \( \log m \) bits per element \( \Rightarrow \) vanishingly small error.

**Bloom filter.** Use \( O(1) \) bits per element \( \Rightarrow \) constant error.

**Bloom filter principle.** Whenever a list or set is used and space is a consideration, a Bloom filter should be considered. When using a Bloom filter, consider the potential effects of false positives. — Broder and Mitzenmacher

---

**Theorem.** Suppose \( X_1, \ldots, X_n \) are independent 0-1 random variables. Let \( X = X_1 + \ldots + X_n \). Then for any \( \mu \geq E[X] \) and \( \delta > 0 \), we have

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

**Pf.** We apply a number of simple transformations.

- For any \( t > 0 \),
  \[
  Pr[X > (1 + \delta)\mu] = Pr[e^{tX} > e^{t(1+\delta)\mu}] \leq e^{t(1+\delta)\mu} \cdot E[e^{tX}] \\
  f(x) = e^{tx} \text{ is monotone in } x \quad \text{Markov's inequality: } Pr[X > a] = E[X] / a
  
  \]

- Now
  \[
  E[e^{tX}] = E[e^{t\sum X_i}] = \prod_i E[e^{tX_i}] \\
  \text{definition of } X \quad \text{independence}
  \]
Chernoff Bounds

Pf. (cont)

Let \( p_i = \Pr[X_i = 1] \). Then,

\[
E[e^{X_i}] = p_i e^0 + (1-p_i) e^0 = 1 + p_i (e^x - 1) \leq e^{p_i (e^x - 1)}
\]

for any \( x > 0 \).

\[
\text{Combining everything:}
\]

\[
\Pr[X > (1+\delta)\mu] \leq e^{-(1+\delta)\mu} \prod_i E[e^{X_i}] \leq e^{-(1+\delta)\mu} \prod_i e^{p_i (e^x - 1)} \leq e^{-(1+\delta)\mu + \mu (e^x - 1)}
\]

previous slide  

inequality above

\[
\sum_i p_i = E[X] = \mu
\]

Finally, choose \( t = \ln(1+\delta) \).

Chernoff Bounds

**Theorem.** Suppose \( X_1, \ldots, X_n \) are independent 0-1 random variables. Let \( X = X_1 + \ldots + X_n \). Then for any \( \mu \geq E[X] \) and \( 0 < \delta < 1 \), we have

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

Pf idea. Similar.

**Remark.** Not quite symmetric since only makes sense to consider \( \delta < 1 \).

13.10 Load Balancing

Load balancing. System in which \( m \) jobs arrive in a stream and need to be processed immediately on \( n \) identical processors. Find an assignment that balances the workload across processors.

**Centralized controller.** Assign jobs in round-robin manner. Each processor receives at most \( \lfloor m/n \rfloor \) jobs.

**Decentralized controller.** Assign jobs to processors uniformly at random. How likely is it that some processor is assigned "too many" jobs?
Analysis.

- Let \( X_i \) = number of jobs assigned to processor \( i \).
- Let \( Y_{ij} = 1 \) if job \( j \) assigned to processor \( i \), and 0 otherwise.
- We have \( E[Y_{ij}] = 1/n \)
- Thus, \( X_i = \sum_j Y_{ij} \) and \( \mu = E[X_i] = 1 \).
- Applying Chernoff bounds with \( \delta = c - 1 \) yields
  \[
  Pr[X_i > c] < \frac{e^{c-1}}{c^c}
  \]
- Let \( \gamma(n) \) be number \( x \) such that \( x^n = n \), and choose \( c = e \gamma(n) \).

\[
Pr[X_i > c] < \left(\frac{e}{c}\right)^c < \left(\frac{1}{\gamma(n)}\right)^{\gamma(n)} < \left(\frac{1}{n} \ln n\right)^{\gamma(n)} = \frac{1}{n^c}
\]
- Union bound \( \Rightarrow \) with probability \( \geq 1 - 1/n \) no processor receives more than \( e \gamma(n) = \Theta(\log n / \log \log n) \) jobs.

Load Balancing

Many jobs. Suppose the number of jobs \( m = 16n \ln n \). Then on average, each processor handles \( \mu = 16 \ln n \) jobs. The probability that any processor exceeds \( 2 \mu \) jobs is at most \( 1/n \).

Analysis.

- Let \( X_i, Y_{ij} \) be as before.
- Applying Chernoff bounds with \( \delta = 1 \) yields

\[
Pr[X_i > 2\mu] < \left(\frac{e}{4}\right)^{16\ln n} < \left(\frac{1}{e}\right)^{16\ln n} = \frac{1}{n^2}
\]
- Applying the union bound, we conclude that all \( n \) machines will have load between half and twice the average.