Chapter 13
Randomized Algorithms

13.1 Contention Resolution

Contention Resolution in a Distributed System

Contention resolution. Given n processes P₁, ..., Pₙ, 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.

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 Pr[S(i, t)] ≤ 1/(2n).

**Proof.** By independence, Pr[S[i, t]] = p(1-p)ⁿ⁻¹.

- Setting p = 1/n, we have Pr[S(i, t)] = 1/n (1 - 1/n)ⁿ⁻¹ ≤ 1/(2n).

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

Randomization

Algorithmic design patterns.
- Greedy.
- 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.
Contention Resolution: Randomized Protocol

Claim. The probability that process $i$ fails to access the database in $en$ rounds is at most $1/e$. After $t = \frac{e \cdot n}{c \ln n}$ rounds, the probability is at most $\frac{n}{n+1}$.

**Pf.** Let $F[i, t] = \text{event that process } i \text{ fails to access database in rounds 1 through } t$. By independence and previous claim, we have

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

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

Union bound. Given events $E_1, \ldots, E_n$,

$$\Pr[\bigcup_{i=1}^n E_i] \leq \sum_{i=1}^n \Pr[E_i].$$  

13.2 Global Minimum Cut

**Global Minimum Cut**

Global min cut. Given a connected, 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 $v \neq s$.

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

**Contraction Algorithm** [Karger 1995]

1. Pick an edge $e = (u, v)$ uniformly at random.
2. **Contract** edge $e$.
   - Label the new super-node $w$.
   - Replace $u$ and $v$ by $w$.
   - Keep parallel edges at $w$. Delete self-loops.
3. Repeat until graph has just two nodes.
4. **Return** the cut (all nodes that were contracted to form $w$).
**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 \(k = |F^*| = \text{size of min cut}\).

- 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 \( \leq \frac{2}{n} \) during the first step.

\[ \begin{array}{c}
A^* \\
\vdots \\
B^* \\
f^*
\end{array} \]

**Contraction Algorithm**

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

**Claim.** If we repeat the contraction algorithm \(n^2 \log n\) 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

\[
\left(1 - \frac{2}{n}\right)^{n^2 \log n} \leq \left(1 - \frac{2}{e}\right)^{n^2} = \frac{1}{n^2} \quad \text{for } 1/e < 1/n.
\]

**Global Min Cut: Context**

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

**Improvement.** (Karger-Stein 1996) \(O(n \log^3 n)\).

- Early iterations are less risky than later ones: probability of contracting an edge in min cut hits 50% when \(n / \sqrt{2}\) nodes remain.
- Run contraction algorithm until \(n / \sqrt{2}\) nodes remain.
- Run contraction algorithm twice on resulting graph, and return best of two cuts.

**Extensions.** Naturally generalizes to handle positive weights.

**Best known.** (Karger 2000) \(O(m \log^3 n)\) faster than best known max flow algorithm or deterministic global min cut algorithm.

**13.3 Linearity of Expectation**

**Expectation.** Given a discrete random variables \(X\), its expectation \(E[X]\) is defined by

\[
E[X] = \sum_{x} x \cdot P[X = x].
\]

**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?

\[
E[X] = \sum_{j=0}^{\infty} j \cdot P[X = j] = \sum_{j=0}^{\infty} j \cdot (1-p)^j p = \frac{p}{1-p} \sum_{j=0}^{\infty} j (1-p)^j = \frac{p}{1-p} \cdot \frac{1-p}{p} = \frac{1}{p}
\]
Expectation: Two Properties

Useful property. If $X$ is a 0/1 random variable, $E[X] = \Pr[X = 1]$.

Pf. $E[X] = \sum_{j=0}^\infty j \cdot \Pr[X = j] = \sum_{j=0}^\infty j \cdot \Pr[X = j] = \Pr[X = 1]$.

Not necessarily independent

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

Decouples a complex calculation into simpler pieces.

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. (surprisingly effortless using linearity of expectation)

\[ E[X] = \sum_{i=1}^{n} \Pr[X_i = 1] = \frac{1}{n} + \ldots + \frac{1}{n} = 1. \]

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

Pf.

\[ E[X] = \sum_{i=1}^{n} \Pr[X_i = 1] = \frac{1}{n} + \ldots + \frac{1}{2} + \frac{1}{1} = H(n). \]

\[ \ln(n+1) < H(n) < 1 + \ln n \]

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 $\geq 1$ coupon of each type?

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

Pf.

\[ E[X] = \sum_{i=0}^{n} \Pr[X_i = 1] = \sum_{i=0}^{n} \frac{1}{i+1} = \sum_{i=1}^{n+1} \frac{1}{i} = H(n+1). \]

\[ \ln(n+1) < H(n+1) < 1 + \ln n+1 \]

\[ \text{expected waiting time} = n/(n-1) \]

13.4 MAX 3-SAT

Maximum 3-Satisfiability

exactly 3 distinct literals per clause

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

\[ C_1 = x_1 \lor \overline{x}_2 \lor \overline{x}_3 \]
\[ C_2 = x_2 \lor \overline{x}_3 \lor \overline{x}_4 \]
\[ C_3 = \overline{x}_1 \lor x_2 \lor \overline{x}_3 \]
\[ C_4 = \overline{x}_1 \lor x_2 \lor \overline{x}_4 \]
\[ C_5 = x_1 \lor \overline{x}_2 \lor \overline{x}_3 \]

Remark. NP-hard search problem.

Simple idea. Flip a coin, and set each variable true with probability $\frac{1}{2}$, independently for each variable.
Claim. Given a 3-SAT formula with \( k \) clauses, the expected number of clauses satisfied by a random assignment is \( \frac{7k}{8} \).

\[ \text{Pf.} \quad \text{Consider random variable } Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases} \]

Let \( Z = \text{weight of clauses satisfied by assignment } Z_j \).

\[
\begin{align*}
\mathbb{E}[Z] &= \sum_{j=1}^{k} \mathbb{E}[Z_j] \\
&= \sum_{j=1}^{k} \mathbb{P}[\text{clause } C_j \text{ is satisfied}] \\
&= \frac{7k}{8}.
\end{align*}
\]

Maximum 3-Satisfiability: Analysis

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

**Pf.** Random variable is at least its expectation some of the time.

Probabilistic method. We showed the existence of a non-obvious property of 3-SAT by showing that a random construction produces it with positive probability!

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

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

**Pf.** By previous lemma, each iteration succeeds with probability at least \( \frac{1}{8k} \).

\[
\mathbb{E}[Z] < \left( \frac{7k}{8} - 1 \right) \left( 1 - \frac{1}{8k} \right)^{1 + \frac{7k}{8} - 1} \left( 1 - \frac{1}{8k} \right)^{1 - \frac{7k}{8}} \leq \frac{7}{8}
\]

By the waiting-time bound, the expected number of trials to find the satisfying assignment is at most \( 8k \).

Monte Carlo vs. Las Vegas Algorithms

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

**Ex.** Contraction algorithm for global min cut.

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

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

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

ZPP (Las Vegas) Decision problems solvable in expected poly-time.

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

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

Polynomial Identity Testing

Given a polynomial \(p(x_1, \ldots, x_n)\) we want to know if \(p(x_1, \ldots, x_n) = 0\).

Example 1: \(p(x, y) = (x + y)(x - y) - x^2 + y^2\)
- Answer: \textit{YES!} After expanding and canceling...

Example 2: \(p(x, y) = (x + y)(3y - 3x) - 3xy - 6y^2 + x^2 + 2yz\)
- Answer: \textit{YES!} But checking is getting more complicated

Approach 1: Expand and cancel
- Takes up to \(\sum_{d} \binom{n}{d}\) steps for degree \(d\) polynomial (exponential in \(d\))

Approach 2: Randomize!

Theorem [Schwartz-Zippel]:
- Suppose \(p(x_1, \ldots, x_n) \neq 0\) is not identically zero. Then given any finite set \(S \subseteq \mathbb{R}\) picking \(y_1, \ldots, y_n \sim S\) uniformly at random then
  \[ Pr[p(y_1, \ldots, y_n) = 0] \leq \frac{2}{|S|} \]

Example: if \(S = \{1, \ldots, 2d\}\) then \(Pr[p(y_1, \ldots, y_n) = 0] \leq \frac{1}{2d}\)

One Sided Error: Polynomial Identity testing in RP

Remark: Schwartz-Zippel also holds for other fields \(\mathbb{F}\)

Randomized Primality Test

Input: \(n\)
Output: PRIME or COMPOSITE

Theorem: If \(n\) is a prime then \([x^{n-1} \mod n] = 1\) for any \(x\).

Example: \(n=5, x=2 \rightarrow [2^4 \mod 5] = [16 \mod 5] = 1\)

Attempt 1: Pick random \(x < n\) and check if \([x^{n-1} \mod n] = 1\)

Carmichael Number: Non-prime numbers that satisfy \([x^{n-1} \mod n] = 1\) for any \(x\).

Theorem: If \(n \geq 3\) is a prime then \([x^{n-1} \mod n] = 1\) for any \(x\).

Example: \(n=3, x=2 \rightarrow [2^4 \mod 3] = [16 \mod 3] = 1\)

Attempt 1: Pick random \(x < n\) and check if \([x^{n-1} \mod n] = 1\)

Carmichael Number: Non-prime numbers that satisfy \([x^{n-1} \mod n] = 1\) for any \(x\).

Theorem: If \(n \geq 3\) is a prime then \(n - 1\) is even and can be written as \(n - 1 = 2d\) for any \(x\) if \(d\) holds that either
- \([x^d \mod n] = 1\), or
- \([x^d \mod n] = n - 1\) for some \(0 \leq d < n\)
Randomized Primality Test

Input: n
Output: PRIME or COMPOSITE

Theorem: If n is a prime then \( x^{n-1} \mod n = 1 \) for any x.

Theorem: If \( n \geq 3 \) is a prime then \( n-1 \) is even and can be written as \( n-1 = 2d \) for any \( d \) it holds that either
- \( x^d \mod n = 1 \), or
- \( x^d \mod n = n-1 \) for some \( 0 \leq r < s \)

Witness of Non-Primality: \( x < n \) such that \( x^d \mod n = 1 \) and \( x^d \mod n = n-1 \) for all \( 0 \leq r < s \) (Strong Liar for n: if \( x < n \) is not a witness, but \( n \geq 3 \) is a prime)

Theorem: If \( n \geq 3 \) is not a prime and \( x < n \) is randomly picked then
\[ \Pr[x \text{ is strong liar for } n] \leq \frac{1}{4} \]

Miller-Rabin Primality Test

Witness of Non-Primality: \( x < n \) such that \( x^d \mod n = 1 \) and \( x^d \mod n = n-1 \) for all \( 0 \leq r < s \) (Strong Liar for n: if \( x < n \) is not a witness, but \( n \geq 3 \) is a prime)

Theorem: If \( n \geq 3 \) is not a prime and \( x < n \) is randomly picked then
\[ \Pr[x \text{ is strong liar for } n] \leq \frac{1}{4} \]

Miller-Rabin test runs in time \( O(kn\log n) \) and mistakenly identifies a composite as prime with probability at most \( 4^{-k} \)

FFT-Multiplication: Reduces running time to \( O(kn\log n) \)

There is a polynomial time algorithm to test if a n-bit number is prime.

Miller-Rabin is used in practice in crypto libraries

13.5 Randomized Divide-and-Conquer

QuickSort

Running time.
- [Best case.] Select the median element as the splitter:
  quicksort makes \( \Theta(n \log n) \) comparisons.
- [Worst case.] Select the smallest element as the splitter:
  quicksort makes \( \Theta(n^2) \) comparisons.

Randomize. Protect against worst case by choosing splitter at random.

Intuition. If we always select an element that is bigger than 25% of the elements and smaller than 25% of the elements, then quicksort makes \( \Theta(n \log n) \) comparisons.

Notation. Label elements so that \( x_1 < x_2 < ... < x_n \).

QuickSort: BST Representation of Splitters

BST representation. Draw recursive BST of splitters.
Quick Sort: BST Representation of Splitters

Observation. Element only compared with its ancestors and descendants.
- \( x_2 \) and \( x_7 \) are compared if their LCA = \( x_2 \) or \( x_7 \).
- \( x_2 \) and \( x_7 \) are not compared if their LCA = \( x_3 \) or \( x_4 \) or \( x_5 \) or \( x_6 \).

Claim. \( P(\text{\( x_i \) and \( x_j \) are compared}) = \frac{2}{|j - i + 1|} \).

Intuition: Consider first time splitter selected from interval \( x_i \ldots x_j \).

Theorem. Expected number of comparisons is \( O(n \log n) \).

Proof. Theorem. [Knuth 1973] Stddev of number of comparisons is \( \sim 0.65N \).

Ex. If \( n = 1 \) million, the probability that randomized quicksort takes less than \( 4n \ln n \) comparisons is at least 99.94%.

Chebyshev's inequality. \( \Pr[|X - \mu| \geq k\delta] \leq \frac{1}{k^2} \).

13.6 Universal Hashing

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.

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 \( u \neq v \).
- A collision is expected after \( \Theta(n^2) \) random insertions. This phenomenon is known as the "birthday paradox."
- Separate chaining: \( H[i] \) stores linked list of elements \( u \) with \( h(u) = i \).

Ad Hoc Hash Function

Ad hoc hash function.

```java
int h(String s, int n) {
    int hash = 0;
    for (int i = 0; i < s.length(); i++)
        hash = (31 * hash) + s[i];
    return hash % n;
}
```

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

Q. But isn’t ad hoc hash function good enough in practice?
Algorithmic Complexity Attacks

When can’t we live with ad hoc hash function?

- 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.

- Obvious situations: aircraft control, nuclear reactors.
- Surprising situations: denial-of-service attacks.

Universal Hashing

Universal class of hash functions. [Carter-Wegman 1980s]

- For any pair of elements x, y \in U, Pr_{h \in H} [h(x) = h(y)] \leq 1/n
- Can select random h efficiently.
- Can compute h(u) efficiently.

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

Designing a Universal Family of Hash Functions

Theorem. [Chebyshev 1850] There exists a prime between n and 2n.

Modulus. Choose a prime number p = \alpha n. – no need for randomness here.

Integer encoding. Identify each element u \in U with a base-p integer of r digits: x = (x_1, x_2, \ldots, x_r).

Hash function. Let A be a set of all r-digit base-p integers. For each \alpha \in A, define

\[ h_\alpha(x) = \sum_{i=1}^r a_i x_i \mod p \]

Hash function family. H = \{h_\alpha : \alpha \in A\}.

Hashing Performance

Idealistic hash function. Maps m elements uniformly at random to n hash slots.

- Running time depends on length of chains.
- Average length of chain \leq m / n.
- Choose n = m \implies average O(1) per insert, lookup, or delete.

Challenge. Achieve idealized randomized guarantees, but with a hash function where you can easily find items where you put them.

Approach. Use randomization in the choice of h.

- adversary learns the randomized algorithm you're using, but doesn’t know random choices that the algorithm makes.
Number Theory Facts

**Fact.** Let $p$ be prime, and let $z \equiv 0 \pmod{p}$. Then $uz \equiv m \pmod{p}$ has at most one solution $0 \leq u < p$.

**Proof.**
- Suppose $u$ and $\beta$ are two different solutions.
- Then $(u - \beta)z \equiv 0 \pmod{p}$; hence $(u - \beta)z$ is divisible by $p$.
- Since $z \equiv 0 \pmod{p}$, we know that $z$ is not divisible by $p$.
- It follows that $(u - \beta)$ is divisible by $p$.
- This implies $u = \beta$.

**Bonus fact.** Can replace "at most one" with "exactly one" in above fact.

**Proof idea.** Euclid's algorithm.

Chernoff Bounds (above mean)

**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 for any $\delta > 0$, we have

$$
P(X > (1 + \delta)\mu) < \left(1 - \frac{\delta}{\mu + \delta}\right)^\mu.
$$

**Proof.** We apply a number of simple transformations.
- For any $\delta > 0$,
  $$
P(X > (1 + \delta)\mu) = \Pr\left[\sum_{i=1}^n \epsilon_i > (1+\delta)\mu\right] < \epsilon^{(1+\delta)\mu} E[\epsilon^\mu]
  \quad \text{Markov's inequality: } E[\epsilon^\mu] = \sum_{i=1}^n \Pr[\epsilon_i = 1] E[\epsilon^\mu].$$
- We use the independence of $X_i$.
- Now
  $$
  E[\epsilon^\mu] = E\left[\sum_{i=1}^n \epsilon_i^\mu\right] = \sum_{i=1}^n E[\epsilon_i^\mu].
  \quad \text{definition of } X_i
  \quad \text{independence}.
  $$

Chernoff Bounds (below mean)

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

$$
P(X < (1 - \delta)\mu) < e^{-\delta^2\mu/2}.
$$

**Proof idea.** Similar.

**Remark.** Not quite symmetric since only makes sense to consider $\delta > 1$.

13.9 Chernoff Bounds

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 \(\left\lfloor \frac{m}{n} \right\rfloor\) jobs.

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

Load Balancing: Many Jobs

Theorem. Suppose the number of jobs \(m = 16n \ln n\). Then on average, each of the \(n\) processors handles \(\mu = 16 \ln n\) jobs. With high probability every processor will have between half and twice the average load.

Proof.

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

\[
\Pr[X_i > 2\mu] < \left(\frac{1}{2}\right)^{2\mu} = \frac{1}{n}
\]

\[
\Pr[X_i < \frac{\mu}{2}] < \frac{1}{n}
\]

Union bound \(\Rightarrow\) every processor has load between half and twice the average with probability \(\geq 1 - 2/n\).