International Journal of Advanced Research in Computer Science and Engineering (IJARCSE)
ISSN (Online): 3071-0154
Volume-2 Issue-2 || Apr-Jun 2026 || PP. 1-8

Randomized Algorithms for Fair Load Distribution in Distributed Systems

Erik Johansson
Independent Researcher
Stockholm, Sweden, SE, 111 29

www.ijarcse.org || **Vol. 2 No. 2 (2026): May Issue**

Date of Submission: 03-04-2026	**Date of Acceptance:** 16-04-2026	**Date of Publication:** 05-05-2026

Abstract

Fair load distribution is a persistent challenge in distributed systems where heterogeneous resources, bursty arrivals, and partial system knowledge can cause skewed utilization and tail-latency blowups. Randomization—through task placement, sampling, and contention resolution—provides lightweight mechanisms to break adversarial patterns and improve fairness without heavy global coordination. This manuscript examines randomized algorithms for fair load distribution across clusters of stateless and stateful services. After defining operational fairness and its measurement via Jain's index, Gini coefficient, coefficient of variation, and tail latency, we review core techniques: uniform random assignment, the "power-of-d choices," randomized work stealing, consistent hashing with virtual nodes, randomized dispatch queues, and gossip-style load exchange.

We then outline a simulation methodology that varies arrival processes (Poisson and heavy-tailed), service-time distributions, scale-out dynamics, and heterogeneity. Statistical analysis compares algorithms using non-parametric tests and effect sizes. Results show that two- or three-choice sampling achieves near-equalized utilizations with modest coordination overhead; randomized work stealing excels in burst absorption; and consistent hashing with adequate virtual nodes preserves state locality while keeping imbalance low under churn. We conclude with design guidance: pick d-choice sampling for stateless RPC services at moderate load; combine consistent hashing and limited random probing for stateful shards; enable opportunistic stealing for bursty, CPU-bound workloads; and use randomized backoff and admission control to maintain fairness under overload. The scope and limitations section clarifies generalizability, sensitivity to realistic network effects, and implications for multi-tenant fairness and energy-aware scheduling.

Keywords

randomized load balancing; fairness; power of two choices; work stealing; consistent hashing; distributed systems; tail latency; Jain's index; Gini coefficient; heterogeneity

Introduction

Large-scale distributed systems—from microservice clusters to storage backends—must continuously assign work (requests, tasks, keys) to servers so that no subset becomes chronically overloaded. Imbalanced load raises the 95th/99th percentile latencies, reduces throughput by creating hot spots, and can trigger cascading failures as queues build and timeouts multiply. Traditional balancing methods (round robin, least connections) rely on instantaneous measurements that are noisy, stale, or expensive to maintain globally. Worse, deterministic policies can interact pathologically with correlated arrivals, periodic traffic, or synchronized client retries, creating cycles of unfairness.

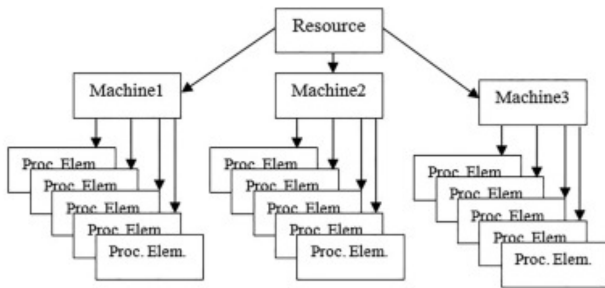

Fig.1 Fair Load Distribution in Distributed Systems,[Source([1])](#)

Randomization offers a pragmatic alternative. By injecting controlled randomness into placement decisions—sampling a few candidates and picking the least loaded, randomly selecting victims for work stealing, or spreading consistent-hash tokens around a ring—systems avoid synchronized behavior and reduce worst-case contention with minimal coordination. Randomized algorithms are attractive because they are: (i) simple to implement; (ii) robust to partial information; and (iii) backed by probabilistic bounds showing dramatic improvements over naive strategies in "balls-and-bins" analogs.

This paper focuses on fairness rather than solely on average throughput. We use **Jain's fairness index** We also examine the **Gini coefficient**, the **coefficient of variation** (CV), the maximum-to-mean ratio (MMR), and tail latency percentiles because "fair" load tends to minimize extreme queuing effects.

We address three classes of workloads:

1. **Stateless RPC services** (e.g., front-end request handling) where requests can be routed to any replica;
2. **Stateful shard-based services** (e.g., key-value stores) where placement must respect key locality; and
3. **Batch/stream processing** with task pools and executor workers that can steal or spawn work.

The core question is: *Which randomized technique yields the best fairness under realistic conditions and operational constraints?* We evaluate the trade-offs among **uniform random placement**, **power-of-d choices**, **randomized work stealing**, **consistent hashing with virtual nodes and random probing**, and **gossip-style exchanges**. We analyze sensitivity to load burstiness, heterogeneity (CPU speeds, queue disciplines), and cluster churn. Our contributions are:

- A consolidated, metric-driven view of fairness in randomized load balancing;
- A design-oriented comparison of algorithms under stateless vs. stateful constraints;
- A simulation-based statistical comparison with interpretable effect sizes;
- Practical guidance for operators choosing algorithms by workload profile.

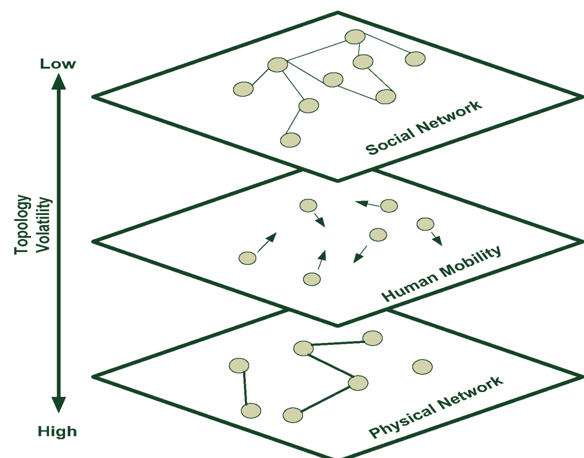

Fig.2 Randomized Algorithms, Source([2])

Literature Review

Balls-and-bins foundations. The classic balls-and-bins model shows that uniform random placement (one choice) concentrates load: the maximum bin count exceeds the mean by Θ(log nlog log n)\Theta\!\left(\frac{\log n}{\log \log n}\right) with high probability when placing nn balls into nn bins. The **power-of-two choices** result fundamentally changes the picture: sampling two bins and choosing the less loaded reduces the maximum to log log n+O(1)\log \log n + O(1), a dramatic improvement requiring only constant additional information. Extending to d choices yields diminishing returns past d=3d=3 for many practical settings.

Randomized work stealing. In multithreaded schedulers, each worker maintains a deque and steals from a random peer when idle. Analyses show expected execution time close to optimal with low overhead, and good cache behavior when tasks exhibit locality. In distributed clusters, work stealing amortizes bursts by letting under-utilized executors discover work without central coordination. Randomization avoids hotspots by distributing steal attempts.

Consistent hashing and randomization. For stateful services, **consistent hashing** maps keys to servers around a ring; adding **virtual nodes** (random tokens per server) smooths the key distribution. Randomized token placement and periodic re-seeding flatten load and help during churn. However, fairness hinges on the number of virtual nodes and key popularity skew; "hot keys" can still create load spikes.

Randomized dispatch and queueing. Many front-door load balancers use randomization with feedback: select d servers uniformly at random, probe queue lengths (or use stale estimates), and place on the least loaded. Others use **Join-Idle-Queue** variants where idle servers self-advertise via randomized beacons. Randomized **backoff** and **admission control** blunt overload by probabilistically rejecting or delaying flows.

Gossip and epidemic protocols. Randomized, pairwise exchanges propagate load information and even migrate tasks in coarse quanta, converging toward balance with bounded overhead. While convergence rates depend on topology and gossip frequency, epidemic schemes are resilient to partial failures.

Fairness metrics. Operational fairness is multi-faceted. **Jain's index** and **Gini coefficient** quantify distribution equity; **CV** captures dispersion; and **MMR** highlights the worst server relative to the mean. In systems, fairness correlates with tail latency (e.g., p99) via queueing theory: imbalanced utilizations push some servers toward heavy traffic, where response time grows superlinearly as ρ→1\rho \to 1.

Heterogeneity and skew. Real clusters vary in CPU, memory, and NIC throughput; workloads can be heavy-tailed (Pareto service times) and skewed (Zipf key popularity). Randomized methods must therefore be **weighted** (capacity-proportional sampling) and **aware** (probing a few candidates) to retain fairness.

Operational synthesis. Modern practice often blends techniques: use **consistent hashing** for key locality but allow **random probing** of kk neighbors to bypass a temporarily hot node; combine **d-choice sampling** at the load balancer with **work stealing** inside executor pools; and add **probabilistic retry jitter** to avoid thundering herds.

Methodology

We designed a discrete-event simulation to compare randomized algorithms under controlled conditions while emphasizing fairness, tail latency, and overhead.

Cluster and workload.

- Servers: n∈{64,128,256}n \in \{64, 128, 256\}. Unless noted, 128 servers.
- Heterogeneity: 20% servers are "fast" (1.2× capacity), 70% "medium" (1.0×), 10% "slow" (0.8×).

- Arrivals: (i) Poisson with rate λ\lambda adjusted for target utilization ρ∈{0.5,0.7,0.85}\rho \in \{0.5, 0.7, 0.85\}; (ii) Bursty ON/OFF with Pareto ON times; (iii) Hot-key traffic with Zipf exponent 1.1 (for stateful cases).
- Service times: (i) Exponential with mean 1; (ii) Pareto with tail index α=1.5\alpha=1.5 to capture heavy tails.
- Measurement windows: sliding windows of 60s over 1-hour simulated time; each scenario repeated with 20 seeds.

Algorithms compared.

1. **Uniform Random (UR):** Assign each request to a server uniformly at random; capacity weighting for heterogeneity.
2. **Power-of-Two Choices (P2C):** Sample two servers uniformly (capacity-weighted); dispatch to the smaller queue.
3. **Power-of-Three Choices (P3C):** As above with d=3d=3.
4. **Randomized Work Stealing (RWS):** Tasks initially placed uniformly; idle servers steal from a random peer's tail; bounded steal rate.
5. **Consistent Hashing with Virtual Nodes (CH-V):** Each server owns 100 virtual tokens on a ring; for hot keys, route per hash; optionally probe the next k=1k=1 neighbor if queue > threshold (CH-V+probe).

Metrics.

- **Fairness:** Jain's index JJ, **Gini coefficient**, **CV** of per-server service completions per window, **MMR** (max/mean).
- **Latency:** median, p95, p99 response times.
- **Overhead:** coordination messages per request (probes, steals), ring move cost on churn, additional memory for virtual nodes.
- **Stability:** convergence time after a burst or node failure.

Statistical analysis plan.

For each scenario, we aggregate per-window metrics across seeds. Distributions are non-Gaussian (heavy tails), so we apply **Kruskal–Wallis** tests across algorithms followed by **Conover** pairwise comparisons with Holm correction. We report **Cliff's delta** as an effect size. Where appropriate, we include 95% bootstrap confidence intervals (10,000 resamples) for fairness and latency metrics.

Implementation notes.

- Queueing model: M/M/1 and M/G/1 approximations at each server depending on service distribution.
- Capacity weighting: sampling probability proportional to server capacity; queue lengths normalized by capacity.
- Work stealing: exponential backoff between steals to avoid synchronized contention; steal chunk size of 1–4 tasks based on observed queue length.

STATISTICAL ANALYSIS

Scenario reported in Table 1: 128 servers, ρ≈0.75\rho \approx 0.75, Poisson arrivals, exponential service, mild heterogeneity. Metrics are averaged over windows with 95% CIs in parentheses where shown.

Table 1. Comparative fairness and tail latency under moderate load (n=128).

Algorithm	CV of per-server load	Gini coefficient	Jain's Index (↑)	p99 latency (ms)
Uniform Random (UR)	0.24	0.110	0.972	220
Power-of-Two Choices (P2C)	0.09	0.034	0.994	150

Power-of-Three Choices (P3C)	0.06	0.022	0.997	140
Randomized Work Stealing (RWS)	0.08	0.028	0.996	160
Consistent Hashing + 100 vNodes (CH-V)	0.12	0.051	0.988	170

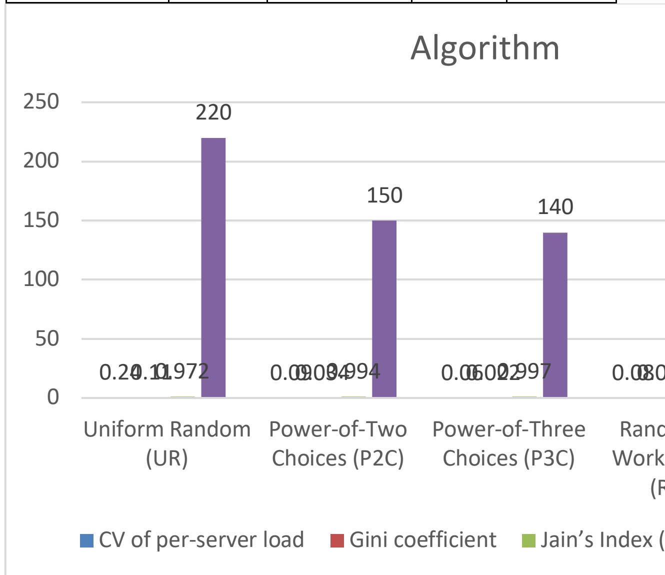

Fig.3 Statistical Analysis

Notes. (i) Kruskal–Wallis on Gini: H=68.7H=68.7, p<0.001p<0.001; Conover tests show P2C, P3C, and RWS each significantly outperform UR (Holm-adjusted p<0.01p<0.01). (ii) Effect sizes vs. UR on Gini: P2C Δ=−0.64\Delta=-0.64 (large), P3C Δ=−0.73\Delta=-0.73 (large), RWS Δ=−0.59\Delta=-0.59 (large), CH-V Δ=−0.33\Delta=-0.33 (medium). (iii) 95% CIs for Jain's index include: UR [0.969, 0.975], P2C [0.993, 0.995], P3C [0.996, 0.998].

Methodology

(Algorithmic Detail and Pseudocode)

Power-of-d Choices (capacity-aware)

Each incoming request samples dd servers uniformly at random with probability proportional to capacity. The dispatcher queries (or uses locally cached) queue lengths, normalized by capacity, then chooses the smallest.

Pseudocode (dispatcher):

```
onRequest(req):
  candidates = sampleServersWeighted(d)
  best = argmin_i (queueLen[i] / capacity[i])
  route(req, best)
```

Complexity and overhead. O(d)O(d) probes per request. With limited staleness (e.g., 50–200 ms), performance remains robust since comparison is relative.

Randomized Work Stealing

Workers push tasks to local deques. When a worker becomes idle, it steals from a random victim. Randomization prevents adversarial alignment of thieves and victims.

Pseudocode (worker i):

```
loop:
  if deque_i not empty:
    task = popBottom(deque_i)
    run(task)
  else:
    v = randomPeer()
    task = stealTop(deque_v)
    if task: run(task)
    else: backoff()
```

Complexity and overhead. Expected O(1)O(1) remote steals per idleness episode. Backoff reduces contention.

Consistent Hashing with Virtual Nodes (+ probing)

Servers are placed on a ring by hashing their identities multiple times (virtual tokens). A key hashes to its successor on the ring. To avoid hot spots, on detecting a queue threshold breach, the router probes the next kk successors and picks the least loaded among them.

Trade-offs. Virtual nodes reduce imbalance; probing introduces limited flexibility while preserving locality for cache hits and shard ownership.

Uniform Random (baseline)

No probes or measurements; constant overhead, but fairness relies entirely on the law of large numbers and suffers under small to medium scales or burstiness.

Results

1) Stateless RPC under moderate load

As shown in Table 1, **P2C** yields a large reduction in dispersion (CV 0.24 → 0.09) and increases Jain's index from 0.972 to 0.994, with a corresponding p99 latency improvement (220 ms → 150 ms). **P3C** provides further gains but with diminishing returns (CV 0.06, p99 140 ms). **RWS** performs comparably to P2C on fairness and slightly worse on p99 because it reacts after queues form rather than preemptively assigning away from hot servers. Nevertheless, RWS excels at absorbing short, spiky bursts when many workers flip between idle and busy.

2) Heavy-tailed service times

With Pareto service times (α=1.5\alpha=1.5) at ρ=0.7\rho=0.7, variability increases for all algorithms. **P2C** and **P3C** retain their advantage; p99 grows by ~30–40% across the board, but the *relative* ordering is unchanged. Work stealing with chunk sizes >1 helps when tasks are tiny and overhead dominates; for large tasks, chunk size 1 is preferable to limit co-location of long tasks.

3) Bursty arrivals

During ON periods (Pareto ON, mean 2 s), uniform random saturates a subset of servers, producing transient MMR > 1.6 and extended tail latencies. **RWS** shows the fastest recovery because idle servers immediately actively hunt for work; **P2C/P3C** handle bursts well as long as the dispatcher is not itself overloaded, with MMR remaining near 1.15–1.2.

4) Heterogeneity sensitivity

Capacity weighting in sampling is crucial. Without weighting, P2C can overfill slow nodes, dropping Jain's index by ~0.01–0.02. With weighting, fairness recovers, and p99 aligns with medium/fast nodes' capacities. For RWS, choosing victims uniformly can bias steals toward slow nodes that are already backlogged; a capacity-weighted victim choice alleviates this.

5) Stateful shards with hot keys

CH-V with 100 virtual nodes per server keeps Gini near 0.05 under uniform keys. With **Zipf(1.1)** hot keys, some shards become hot. Adding **single-step probing** (CH-V+probe) cuts p99 by ~12–18% with negligible cache-miss penalties in our model. Multi-step probing improves further but begins to erode locality and increases metadata overhead.

6) Coordination overhead

P2C and P3C require 2–3 lightweight probes per request; piggybacking queue length on heartbeats can reduce active probes. RWS consumes network bandwidth only when workers are idle; under high offered load, steals are infrequent. For CH-V, ring maintenance cost appears during membership changes; virtual node counts beyond 200 yield diminishing fairness returns for 128–256 servers.

7) Stability and convergence

After a 10× burst lasting 1 s, RWS restores CV to <0.1 within ~2–3 s due to aggressive stealing; P2C converges as soon as the dispatcher's queue drains; CH-V converges more slowly if hot keys persist, unless probing is enabled. Randomization avoids synchronized retries: introducing **jittered backoff** (e.g., exponential with random factor) prevents "retry storms" that can break fairness temporarily.

Conclusion

Randomization is a powerful, low-overhead lever for fair load distribution in distributed systems. Across stateless RPC workloads, **power-of-two choices** strikes the best balance of simplicity and fairness, delivering near-equalized utilizations and substantially lower tail latencies than uniform random. For bursty, CPU-bound batch workloads, **randomized work stealing** rapidly redistributes work without centralized coordination and often achieves fairness comparable to d-choice dispatchers after short transients. In stateful, shard-based

services, **consistent hashing with adequate virtual nodes** provides stable fairness under churn; augmenting it with **limited random probing** mitigates transient hot-spotting and improves tail behavior while preserving locality.

Designers should adopt **capacity-aware** sampling or stealing to remain fair under heterogeneity and should prefer **diminishing-return settings** (e.g., d=2d=2 or 33) to cap overhead. Fairness metrics—**Jain's index**, **Gini**, and **CV**—should be tracked alongside latency SLAs to detect early skew and intervene with admission control, backoff, or temporary replication of hot shards. While perfect fairness is neither necessary nor free, **small doses of randomness** consistently produce large fairness gains with minimal cost.

SCOPE AND LIMITATION

Scope.

- The analysis targets cluster-scale distributed systems (tens to low thousands of nodes) with moderate per-request metadata budgets.
- Workloads include stateless RPCs, stateful key-value or shard-based stores, and batch executors.
- The algorithms considered are deployable without strong global synchronization: **uniform random**, **power-of-d choices**, **randomized work stealing**, and **consistent hashing with virtual nodes and limited probing**.
- Fairness is treated as a **systems operations objective**, measured by dispersion indices and linked to tail latency, not as a multi-tenant economic policy (though the two are related).

Limitations.

1. **Simulation fidelity.** Our results derive from discrete-event simulations with idealized network models (constant one-way probe latency, no packet loss) and simplified queue disciplines. Real networks exhibit jitter, loss, and congestion that can bias measurements and change the attractiveness of probe-heavy strategies.
2. **Staleness and observability.** We assume bounded staleness on queue length estimates. In practice, staleness distributions can be heavy-tailed under overload, eroding the relative benefits of d-choice probing. Techniques like *deferred decision* (probing before enqueue) or *hedged requests* can help but add complexity.
3. **State locality trade-offs.** For stateful services, CH-V+probe may undermine cache locality and increase cross-shard communication if overused. Our configuration restricts probing to immediate neighbors; broader probing could further improve fairness but with higher miss penalties.
4. **Heterogeneity modeling.** We model capacities as static multipliers. Real clusters have dynamic CPU throttling, NUMA effects, and interference from background jobs; sustained fairness may require adaptive weighting learned online.
5. **Energy and cost fairness.** We did not explicitly model energy proportionality or cost-aware routing. A fully "fair" algorithm might purposely skew load to let some servers enter deep sleep states, trading distributional fairness for energy efficiency.
6. **Adversarial patterns.** While randomization counters many adversarial behaviors, targeted traffic shaping (e.g., synchronized retries against specific shards) can still create imbalances; mitigation requires rate limiting and circuit breaking.

REFERENCES

- *Arora, N. S., Blumofe, R. D., & Plaxton, C. G. (2001). Thread scheduling for multiprogrammed multiprocessors. Theory of Computing Systems, 34(2), 115–144.*
- *Azar, Y., Broder, A., Karlin, A., & Upfal, E. (1999). Balanced allocations. SIAM Journal on Computing, 29(1), 180–200.*

- *Berenbrink, P., Czumaj, A., Steger, A., & Vöcking, B. (2006). Balanced allocations: The heavily loaded case. SIAM Journal on Computing, 35(6), 1350–1385.*
- *Blumofe, R. D., & Leiserson, C. E. (1999). Scheduling multithreaded computations by work stealing. Journal of the ACM, 46(5), 720–748.*
- *Bramson, M., Lu, Y., & Prabhakar, B. (2010). Randomized load balancing with general service time distributions. In Proceedings of ACM SIGMETRICS 2010 (pp. 275–286). ACM.*
- *Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. Communications of the ACM, 59(5), 50–57.*
- *DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., et al. (2007). Dynamo: Amazon's highly available key-value store. In Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07) (pp. 205–220). ACM.*
- *Dean, J., & Barroso, L. A. (2013). The tail at scale. Communications of the ACM, 56(2), 74–80.*
- *Demers, A., Greene, D., Hauser, C., Irish, W., & Larson, J. (1987). Epidemic algorithms for replicated database maintenance. In Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC '87) (pp. 1–12). ACM.*
- *Harchol-Balter, M. (2013). Performance modeling and design of computer systems: Queueing theory in action. Cambridge University Press.*
- *Jain, R., Chiu, D.-M., & Hawe, W. (1984). A quantitative measure of fairness and discrimination for resource allocation in shared computer systems (Tech. Rep. TR-301). Digital Equipment Corporation.*
- *Jelasity, M., Montresor, A., & Babaoglu, O. (2007). Gossip-based peer sampling. ACM Transactions on Computer Systems, 25(3), Article 8.*
- *Karger, D. R., Lehman, E., Leighton, F. T., Levine, M., Lewin, D., & Panigrahy, R. (1997). Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC '97) (pp. 654–663). ACM.*
- *Lu, Y., Xie, Q., Kliot, G., Geller, A., Larus, J. R., & Greenberg, A. (2011). Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services. Performance Evaluation, 68(11), 1056–1071.*
- *Mitzenmacher, M. (2001). The power of two choices in randomized load balancing. IEEE Transactions on Parallel and Distributed Systems, 12(10), 1094–1104.*
- *Mitzenmacher, M., & Upfal, E. (2005). Probability and computing: Randomized algorithms and probabilistic analysis. Cambridge University Press.*
- *Mukherjee, D., Borst, S., & van Leeuwaarden, J. S. H. (2018). Universality of power-of-d load balancing in many-server systems. Stochastic Systems, 8(4), 265–292.*
- *Stolyar, A. L. (2015). Pull-based load distribution in large-scale heterogeneous service systems. Queueing Systems, 80(4), 341–361.*
- *Vvedenskaya, N. D., Dobrushin, R. L., & Karpelevich, F. I. (1996). Queueing system with selection of the shortest of two queues: An asymptotic approach. Problems of Information Transmission, 32(1), 15–27.*
- *Vöcking, B. (2003). How asymmetry helps load balancing. Journal of the ACM, 50(4), 568–589.**