Game Theory

Different Approaches to Solve the k-Server Problem

BEURTHERET Eloi
CHABANE Oualid
KERMADJ Zineddin

(In alphabetical order)

February 2026

Contents

1 Introduction

In the ***k*-server problem**, k servers must serve an online sequence of requests on a metric space. Each request must be served immediately by one server; the cost is the Manhattan distance traveled. The **competitive ratio** $\rho = cost/opt$ measures algorithm quality against the offline optimum. In this project, sites lie on $[0, 99]^2$ and all servers start at $(0, 0)$.

1.1 Historical Background and Conjectures

The k-server problem is a central problem in online and randomized algorithms, with a long and rich development history.

1985 — Sleator and Tarjan [8] proved that the **paging problem** admits a ***k*-competitive deterministic algorithm**.

1988 — Manasse, McGeoch, and Sleator [1] introduced the ***k*-server problem** and formulated the ***k*-server conjecture**. They proved it for $k = 2$ and for metrics with exactly $k + 1$ points.

1990 — Fiat et al. [9] showed existence of an algorithm with a **finite competitive ratio** for any fixed k.

1991 — Chrobak and Larmore [2] proved the conjecture for **tree metrics**.

1995 — Koutsoupias and Papadimitriou [3] introduced the **Work Function Algorithm (WFA)**, proving a competitive ratio of $2k - 1$.

2000 — Bartal and Koutsoupias [4] confirmed k-competitiveness of WFA for special metrics.

2 Algorithms

2.1 Deterministic Algorithms

Greedy. Serve each request with the nearest server. Ignores load entirely.

Balance. Choose server j minimizing $d(s_j, r) + \alpha \cdot L_j$, where L_j is the total distance already traveled by server j ($\alpha = 0.5$).

Balance variants. We explored three penalty modifications: *aggressive* (αL_j^2), *exponential* ($\alpha e^{\beta L_j}$), and *sqrt* ($\alpha\sqrt{L_j}$). All performed worse than linear Balance on our benchmark.

Balance time decay. The weight decays as $\alpha_t = \alpha_0 \gamma^t$ ($\alpha_0 = 0.8$, $\gamma = 0.99995$), favoring load balancing early and proximity later.

Balance reuse zero. Same as time decay, with one extra rule: if a server is already at the request (cost 0), prefer the one that last served this site. **Best overall performance** in our experiments.

Site affinity. Reuses the last server to have visited a site if within $\lambda \cdot d_{\min}$ ($\lambda = 1.4$); otherwise falls back to Balance. Unstable on our instances.

Double coverage. Moves the two nearest servers toward the request simultaneously. Theoretically motivated but very expensive in practice.

Work Function Algorithm (WFA). The WFA [3] is $(2k-1)$-competitive. It maintains a *work function* $w_t(S)$ — the minimum cost to serve all past requests and end in configuration S — and selects the server j minimizing $w_{t+1}(S \leftarrow s_j \rightarrow r) + d(s_j, r)$. We keep at most 500 configurations (pruned by cost) to stay tractable, always retaining the current one. This pruning limits empirical performance despite the strong theoretical guarantee.

Adaptive clustering. Every $T = 10$ steps, recompute k centroids by k-means (Manhattan, 5 iterations, initialized from server positions) on the last $w = 20$ requests, then assign servers to centroids by greedy bipartite matching. Requests are routed to the server owning the nearest centroid zone. Sensitive to the cold-start problem and degenerates to greedy on spread instances.

2.2 Randomized Algorithms

Random among nearest. Pick uniformly among the $m = 2$ nearest servers.

Balance random. Break ties in Balance scores randomly (within $\varepsilon = 10^{-6}$). Closely mirrors deterministic Balance.

Harmonic. Select server j with probability $\propto 1/d(s_j, r)$. Theoretically $O(k^3 \log k)$-competitive. Load-indifferent, which hurts performance on long sequences.

Softmin balance. Same scores as Balance, but server j chosen with probability $\propto \exp(-\text{score}_j/\tau)$, introducing controlled exploration.

BBMN [6]. **Implementation of Bansal, Buchbinder, Mądry, and Naor polylogarithmic algorithm (BBMN):** This algorithm achieves ratio $\tilde{O}(\log^2 k \log^3 n)$. Our implementation follows the polylogarithmic randomized algorithm of Bansal, Buchbinder, Mądry, and Naor (BBMN) for the k-server problem. The entire work is built on the *allocation problem* introduced by Coté et al. [5]. The main idea is to reduce the general metric to a structured tree metric where the problem becomes easier to handle (generally all online algorithms that deal with metric spaces are extremely complex to handle, but, converting the metric to an HST tree, the resolution of the problem becomes a simple manipulation of tree algorithms), while preserving distances up to a logarithmic distortion.

First, we probabilistically embed the input metric into a *Hierarchically Well-Separated Tree* (HST) using the FRT embedding. This step transforms the arbitrary metric into a tree metric with expected $O(\log n)$ distortion. Intuitively, instead of moving servers directly in the original space, we simulate their movement on a randomly sampled HST,

where distances correspond to levels in the hierarchy. Since the radius of our data (the maximal distance between a server and a client) can be exponentially big, this might result a linear depth of the HST tree, therefore, we use a weighted HST tree instead of a regular one, this leads to have a depth that is only dependent on the number of client, but not on the distances.

Second, we run a *fractional allocation algorithm* on the tree. Each internal node maintains a fractional distribution of servers among its children. When a request arrives, the update is performed recursively from the root to the requested leaf. The allocation balances transportation cost along tree edges with reallocation cost, controlled by parameters ε and δ, which regulate the smoothness and responsiveness of the fractional updates. This fractional view avoids committing too early to discrete moves and keeps the competitive ratio under control.

Finally, we apply *online randomized rounding* to convert the fractional configuration into an integral server placement. The rounding preserves the expected cost up to a constant factor, ensuring that the overall algorithm remains polylogarithmic competitive.

In our implementation, a fresh random HST is sampled at each run (unless a seed is fixed), and performance is evaluated over multiple runs (e.g., 50) to average out embedding randomness. This combines theoretical guarantees with practical stability in experiments.

3 Experimental Results

Deterministic algorithms: one run per instance. Randomized algorithms: 10,000 runs per instance (mean ratio reported). Table 1 summarizes all algorithms; Table 2 gives per-instance ratios for the most informative ones.

Table 1: Mean competitive ratio by algorithm.

Algorithm	**Type**	**Mean ratio**	**95% CI**
Balance reuse zero	Det.	**1.1932**	[1.11, 1.27]
Balance time decay	Det.	1.1961	[1.12, 1.28]
Balance random	Rand.	1.2352	—
Balance	Det.	1.2435	[1.15, 1.34]
BBMN	Rand.	1.4576	—
WFA	Det.	1.4624	[1.35, 1.57]
Harmonic	Rand.	1.7704	—
Balance exponential	Det.	2.8967	[2.52, 3.27]
Softmin balance	Rand.	3.0299	—
Balance sqrt	Det.	7.1318	[2.99, 11.3]
Site affinity	Det.	10.548	[4.17, 16.93]
Random among nearest ($m = 2$)	Rand.	14.387	—
Balance aggressive	Det.	15.700	[8.08, 23.32]
Adaptive clustering	Det.	18.020	[9.75, 26.3]
Greedy	Det.	20.670	[10.66, 30.68]
Double coverage	Det.	41.482	[21.5, 61.4]

Table 2: Per-instance competitive ratios for key algorithms. (D) = deterministic; (R) = randomized mean over 10,000 runs.

Instance	**Bal. reuse** (D)	**Bal. decay** (D)	**Balance** (D)	**WFA** (D)	**Harmonic** (R)	**BBMN** (R)	**Clustering** (D)	**Greedy** (D)
N200_OPT221	1.000	1.000	1.000	1.262	1.134	1.000	17.91	17.91
N200_OPT286	1.147	1.294	1.392	1.902	1.224	1.000	30.73	30.73
N200_OPT347	1.029	1.029	1.029	1.945	1.243	1.000	17.89	33.97
N200_OPT5166	1.264	1.283	1.257	1.272	2.054	1.707	1.333	1.190
N200_OPT5266	1.435	1.368	1.332	1.060	2.154	1.740	1.145	1.112
N200_OPT5298	1.244	1.231	1.227	1.134	1.876	1.633	1.273	1.122
N250_OPT134	1.000	1.000	1.000	1.343	1.301	1.000	29.27	29.27
N250_OPT4262	1.307	1.291	1.334	1.373	2.194	2.011	1.960	1.858
N300_OPT246	1.098	1.098	1.098	1.707	1.248	1.000	46.53	46.53
N300_OPT337	1.000	1.000	1.000	1.404	1.283	1.000	40.82	40.82
N300_OPT394	1.000	1.000	1.000	1.599	1.261	1.000	30.43	30.43
N300_OPT5645	1.336	1.336	1.468	1.425	2.150	1.967	1.317	1.380
N300_OPT6260	1.227	1.206	1.311	1.044	2.247	1.792	2.326	2.246
N300_OPT7236	1.419	1.395	1.391	1.515	2.110	1.793	1.173	1.236
N350_OPT277	1.000	1.000	1.000	1.657	1.242	1.000	38.82	76.63
N350_OPT5552	1.408	1.431	1.514	1.437	2.241	2.118	1.542	1.385
N400_OPT3683	1.467	1.539	1.649	1.421	3.181	2.087	2.435	2.123
N400_OPT3717	1.484	1.422	1.568	1.580	3.017	2.304	2.496	2.454
N400_OPT377	1.000	1.000	1.000	1.424	1.130	1.000	31.77	31.77
N400_OPT398	1.000	1.000	1.302	1.744	1.119	1.000	59.24	59.24
Mean	**1.193**	**1.196**	**1.244**	**1.462**	**1.770**	**1.458**	**18.02**	**20.67**

3.1 Observations

Balance family. Balance reuse zero (1.19) and time decay (1.20) are the top performers. Fixed-α Balance (1.24) is already strong; aggressive, exponential, and sqrt variants backfire on irregular patterns.

WFA and BBMN. Both achieve a mean around 1.46. WFA is limited by configuration pruning; BBMN is near-optimal on small instances but reaches 2–2.3 on larger ones.

Harmonic. Competitive on small instances ($\approx$ 1.1–1.3) but degrades on larger ones (up to 3.2) due to load-indifferent weighting.

Adaptive clustering. Bimodal: near-optimal on large-OPT instances, catastrophic on small-OPT ones (cold-start at $(0, 0)$). Mean ratio of 18.

Greedy and double coverage. Both very poor when requests are spatially spread.

4 Conclusions

1. Load-balancing heuristics (Balance family) clearly dominate pure greedy and distance-only strategies.
2. Time-decaying the load weight and reusing zero-cost servers (Balance reuse zero, mean $\approx$ 1.19) are the most effective refinements.
3. WFA and BBMN, despite strong theoretical backing, are limited here by configuration pruning and instance structure respectively.
4. Harmonic works well on small instances but is load-indifferent, limiting performance on longer sequences.
5. Adaptive clustering suffers from cold-start sensitivity; a warm-start phase would significantly help.
6. For practical use, **Balance reuse zero** offers the best trade-off between performance and simplicity.

References

[1] M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 1988.

[2] M. Chrobak and L. Larmore. The server problem and on-line games. *Combinatorica*, 1991.

[3] E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 1995.

[4] Y. Bartal and E. Koutsoupias. On the competitive ratio of the work function algorithm. *STOC*, 2000.

[5] P. Coté, A. Meyerson, and L. Poplawski. Randomized k-server on hierarchical binary trees. *STOC*, 2008.

[6] N. Bansal, N. Buchbinder, A. Mądry, and J. Naor. A polylogarithmic-competitive algorithm for the k-server problem. *FOCS*, 2011.

[7] Lower bound result showing the randomized k-server conjecture is false, 2022.

[8] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. Communications of the ACM, 1985.

[9] A. Fiat et al. Competitive algorithms for the k-server problem. FOCS, 1990.