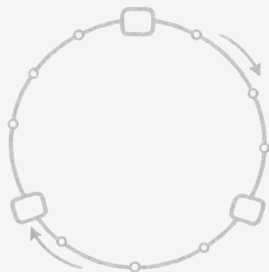


Randomization ► Consistent Hashing

Lecture 9

- A Changing Cache
- Consistent Hashing
- Balancing the Load
- From Theory to Practice





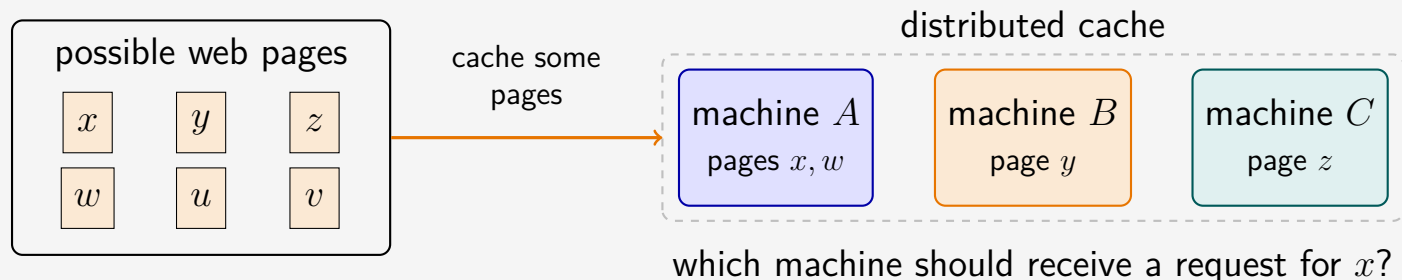
Randomization ► Consistent Hashing

Lecture 9

- ▶ A Changing Cache
- ▶ Consistent Hashing
- ▶ Balancing the Load
- ▶ From Theory to Practice

The Distributed Hashing Setup

A distributed web cache stores copies of pages across several machines.



$$\underbrace{U, |U| = n}_{\text{all possible pages}}, \quad \underbrace{S \subseteq U, |S| = m \ll n}_{\text{pages currently cached}}, \quad \underbrace{q}_{\text{number of machines}}.$$

Compute a page's destination locally, without searching every machine or overloading any one machine.

Hashing, Recalled

Definition: Hash function and collision

A hash function maps a universe U to q locations:

$$h : U \longrightarrow [q] = \{0, 1, \dots, q - 1\}.$$

Distinct items x, y **collide** if $h(x) = h(y)$.

Definition: Load

For a fixed set S of m items, the load of location i is

$$X_i = |\{x \in S : h(x) = i\}|.$$

In our distributed cache, the items are pages and the locations are machines. Many pages may share a machine; we want to avoid concentrating too many there.

What Randomness Gave Us

Fix the pages **before** choosing a fully random hash function.

$$X_i = \sum_{x \in S} \mathbb{1}_{\{h(x)=i\}}, \quad \mathbb{E}[X_i] = \frac{m}{q}.$$

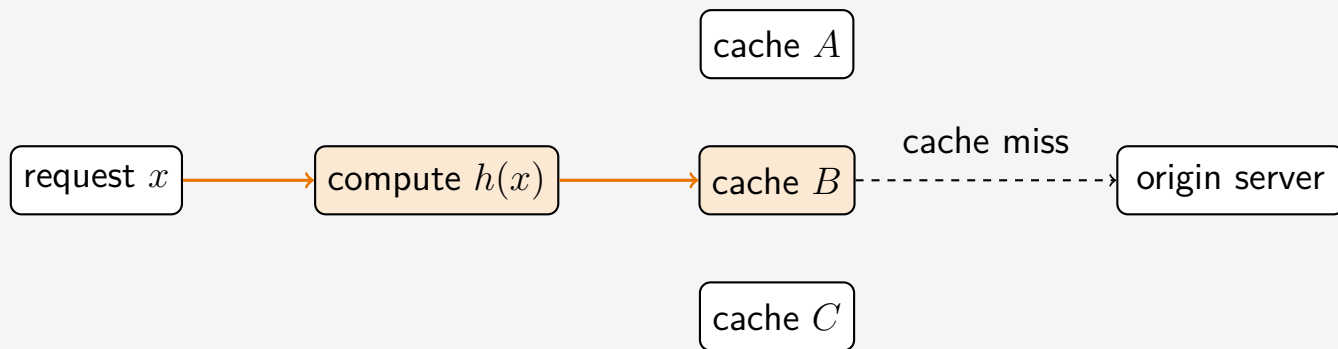
- An expectation describes the average over the random choice of h .
- Concentration controls how far a realized load can be from that average.
- A union bound lets us control **all** machines at once.

Remark

Last lecture: how much randomness do we need for a fixed table?
Today: what should happen when the set of machines changes?

The Cache from Last Lecture

Several machines share the job of caching pages. Given a page x , we should compute its machine without searching every cache.

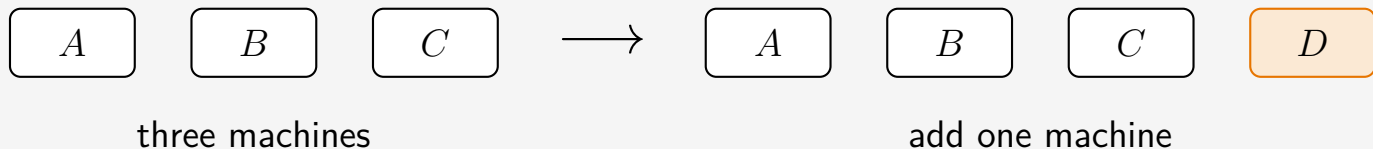


Everyone uses the same hash function and the same machine list.

A cache hit uses an existing copy. A cache miss fetches the page from its origin.

A Successful Service Creates a New Problem

Traffic grows. Three cache machines are no longer enough.



Problem

Assign pages to the available machines, while preserving as many existing assignments as possible when a machine joins or leaves.

Remark

Which pages should move to *D*? Should any pages move between *A*, *B*, and *C*?

The Natural Solution: Change the Modulus

Hash each page to a large integer $H(x)$, then choose machine $H(x) \bmod q$.

For three machines, labels 0, 1, 2 mean A, B, C . Add D with label 3 and change the modulus to 4.

$H(x)$	0	1	2	3	4	5	6	7	8	9	10	11
mod3	A	B	C	A	B	C	A	B	C	A	B	C
mod4	A	B	C	D	A	B	C	D	A	B	C	D

Nine of twelve assignments change. Only three pages go to D ; six others move between machines that were already present.

Why Changing Assignments Is Expensive

- A page may still be cached on A , but its next request goes to B .
- B misses and fetches the page again. Many such changes create a surge of requests to the origin.
- If these were stored records rather than cached copies, changing ownership could require moving the records themselves.

Adding capacity can temporarily make the service **worse**.

Remark

The new assignments are easy to compute. The expensive part is losing what the old assignments had already accomplished.

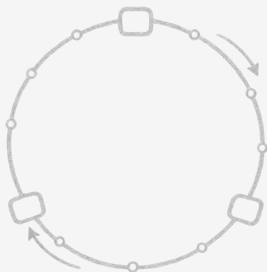
What Should a Solution Preserve?

1. **Balance:** spread the pages across available machines.
2. **Fast lookup:** compute a page's owner efficiently.
3. **Stability:** change few assignments when machines change.

A balanced new machine should receive about $m/(q+1)$ pages. There is no reason for the other pages to change owners.

Remark

Can we use the machines' **identities**, without making every assignment depend so strongly on their **number**?



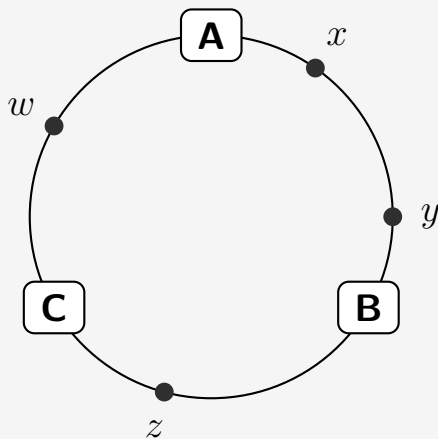
Randomization ► Consistent Hashing

Lecture 9

- ▶ A Changing Cache
- ▶ Consistent Hashing
- ▶ Balancing the Load
- ▶ From Theory to Practice

Imagine a Real Ring

Imagine a circle of circumference 1, with every point available as a position.



Every machine picks an independent, uniformly random point on the ring.

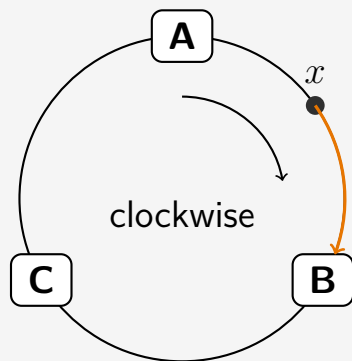
Every page does the same, independently of the machines and other pages.

Choose the points **once**. Everyone uses the same positions, and an existing position never changes when a machine joins.

Algorithm: Consistent Hashing on a Ring

Algorithm: The clockwise successor rule

1. Choose an independent uniform point for every page and machine, and keep these positions fixed.
2. Assign a page to the **first machine clockwise** from its position.

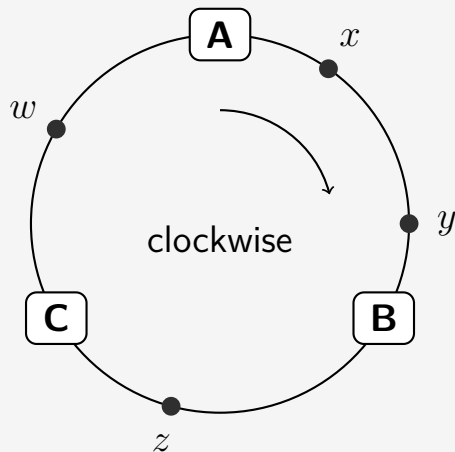


x belongs to B .

Each machine owns the interval from its predecessor up to itself.

Everyone with the same machine list computes the same answer.

Example: Who Owns Each Page?



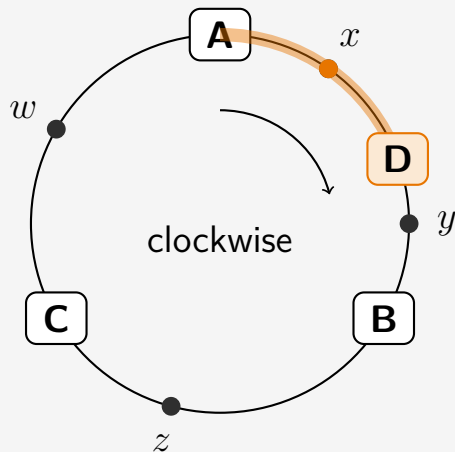
Remark

Which machine owns each page?

$$\begin{aligned}x &\mapsto B, & y &\mapsto B, \\z &\mapsto C, & w &\mapsto A.\end{aligned}$$

The circle makes wraparound part of the same rule.

Adding a Machine Changes One Interval



Insert D between A and B .

Remark

Which pages change owners?

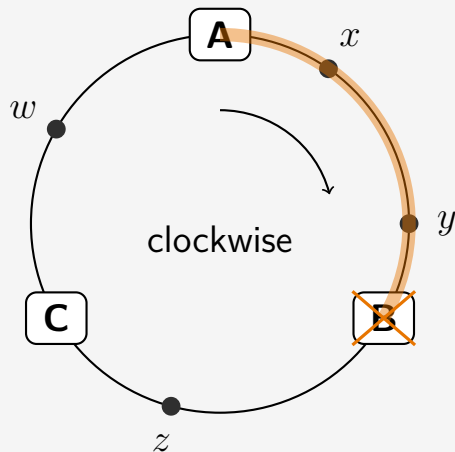
D takes part of B 's interval:

$$x : B \longrightarrow D.$$

y, z, w keep their owners.

No page moves between two old machines.

Removing a Machine Changes One Interval



Return to the original ring and remove B .

Remark

Which pages change owners?

B 's pages go to its successor C :

$$x, y : B \longrightarrow C.$$

z stays on C ; w stays on A .

The Stability Guarantee

Theorem: Local changes

Keep all ring positions fixed.

- When a machine joins, any page that changes owner goes to that machine.
- When a machine leaves, only its pages change owners.

An insertion splits an existing ownership interval. A deletion merges the deleted machine's interval into its successor's.

Every other interval keeps the same owner. This holds for **every** placement of the machines, even an unbalanced one.

How Much Changes?

With $q + 1$ independent, identically distributed machine positions, each machine is equally likely to own any fixed page.

$$\Pr[\text{page } x \text{ changes owner on insertion}] = \Pr[\text{new machine owns } x] = \frac{1}{q + 1}.$$

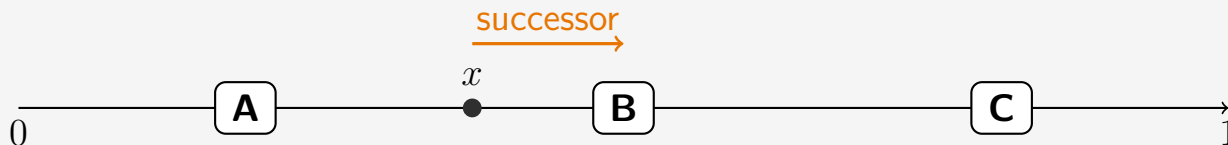
$$\mathbb{E}[\text{number of changed assignments}] = \sum_{x \in S} \Pr[x \text{ changes owner}] = \frac{m}{q + 1}.$$

Method	Expected fraction changing on insertion
Change modulus $q \rightarrow q + 1$	$q/(q + 1)$
Consistent hashing	$1/(q + 1)$

Finding a Page's Owner

How do we efficiently implement the successor rule? A naive search takes $O(q)$ time.

Pick an origin on the ring and give positions real coordinates in $[0, 1)$. Store the machines in a balanced binary search tree (BST), keyed by position.



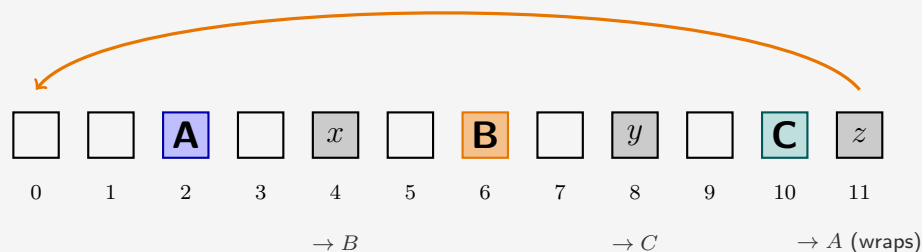
- Search for the first machine at or after the page's coordinate. If there is none, return the machine with the smallest coordinate.
- A balanced BST supports this lookup in $O(\log q)$ time.
- Insert or delete a machine position in $O(\log q)$ time.
- Store $O(q)$ machine positions, not one entry per possible page.

Updating this directory is cheap. Moving or refetching affected pages is a separate cost.

From a Real Ring to Finite Hash Values

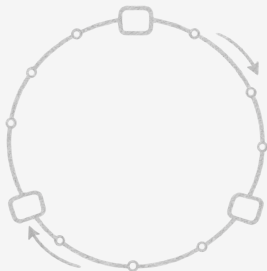
In an implementation, replace real positions by a **large fixed range** of integers, such as $\{0, \dots, R - 1\}$ with $R = 2^{64}$.

the positions continue up to $R - 1$, then wrap to 0



Algorithm: A discrete ring

1. Use a shared hash function H to choose positions for pages and machine identifiers.
 2. Arrange the R possible positions in a circle. Keep R fixed when the number of machines changes.
 3. Use the same successor rule and BST, now with integer keys.
- Store only the q occupied machine positions, not all R slots. Lookup takes $O(\log q)$ time.
 - Use a fixed order of machine identifiers to break position ties.

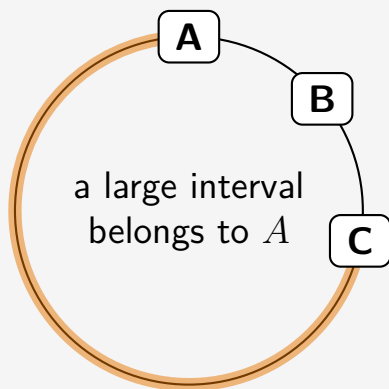


Randomization ► Consistent Hashing

Lecture 9

- ▶ A Changing Cache
- ▶ Consistent Hashing
- ▶ Balancing the Load
- ▶ From Theory to Practice

Stable Does Not Necessarily Mean Balanced



Analyze independent uniform positions on the real ring. By symmetry, every machine has expected load m/q .

But a particular random ring can have very unequal intervals.

Adding many pages does not fix a large ownership interval.

Bounding the Largest Ownership Interval

Lemma: One position per machine: an upper bound

For $q \geq 2$, with probability at least $1 - q^{-1}$, every machine owns **at most** an $O(\log q/q)$ fraction of the circle.

Proof

Let L_i be machine i 's ownership fraction. Fix i and condition on its position; the other $q - 1$ points remain independent and uniform.

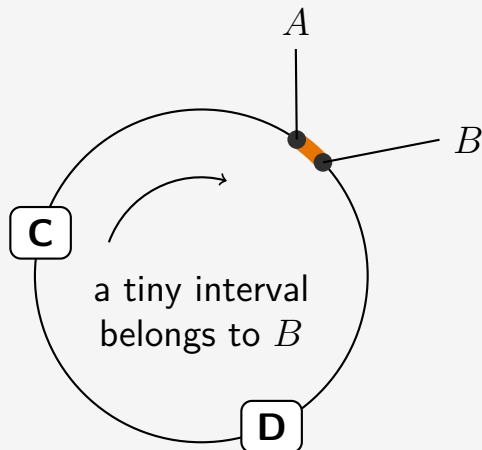
For $0 < t < 1$, $L_i > t$ exactly when the arc of length t ending at i contains **no other machine**. Each other machine independently avoids that arc with probability $1 - t$:

$$\Pr[L_i > t \mid \text{position of } i] = (1 - t)^{q-1} \leq e^{-(q-1)t}.$$

Choose $t = 2 \ln q / (q - 1)$, so this probability is at most q^{-2} . The bound holds for every position of i ; union-bound over the machines:

$$\Pr[\max_i L_i > t] \leq q \cdot q^{-2} = q^{-1}.$$

Some Machines Receive Almost No Share



Two nearby machines leave the second one with a very small ownership interval.

With constant probability, some machine owns at most $1/q^2$ of the ring:

$$\Pr\left[\min_i L_i \leq \frac{1}{q^2}\right] \rightarrow 1 - e^{-1} \approx 63\%.$$

This is much smaller than the average share $1/q$.

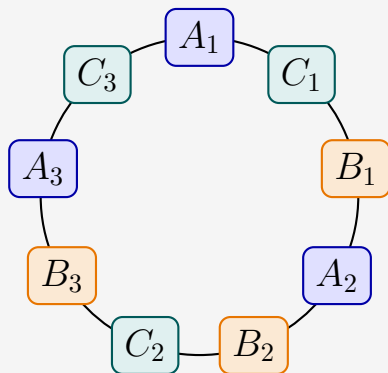
Exercise: Prove the small-gap probability statement above.

How can we slightly modify the algorithm to ensure that every machine owns closer to $1/q$ of the ring?

Give Each Machine Several Positions

Definition: Virtual nodes

Give each physical machine v independently chosen random positions: $(A, 1), \dots, (A, v)$ denote the positions belonging to A , and similarly for every other machine. The page's owner is the physical machine of its clockwise successor.



Each machine owns several smaller intervals around the ring.

An unusually large interval matters less when it is one of many.

These are **positions**, not replicas of the pages.

Concentration Improves the Balance

As before, L_i is the fraction of the circle owned by physical machine i . For any fixed ring, independent uniform page positions give

$$\mathbb{E}[X_i \mid \text{ring}] = mL_i.$$

Theorem: Balance with virtual nodes

For any fixed constant $a > 0$, sufficiently many positions per machine, $v = \Theta(\log q)$, ensure that

$$L_i = O(1/q) \quad \text{for every machine } i$$

with probability at least $1 - q^{-a}$.

Once the intervals are balanced, concentration of the page counts gives

$$\max_i X_i = O(m/q + \log q) \quad \text{with high probability.}$$

Machines with Different Capacities

A machine with more storage should own a larger share of the pages.

machine A

capacity 1



v positions

machine B

capacity 2



$2v$ positions

machine C

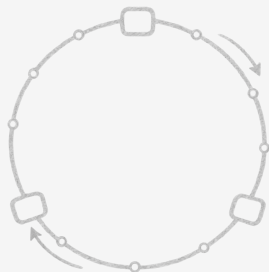
capacity 1



v positions

Give machines numbers of virtual nodes proportional to their capacities. Here B receives half the ring **in expectation**.

Adding positions still moves pages only to the machine receiving them.



Randomization ► Consistent Hashing

Lecture 9

- A Changing Cache
- Consistent Hashing
- Balancing the Load
- From Theory to Practice

From a Theory Paper to a Company

Story adapted from [Tim Roughgarden](#) and [Gregory Valiant's](#) lecture notes.

STOC 1997: *Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web.*

David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, and Rina Panigrahy.

An earlier conference submission was rejected: a reviewer claimed that the work “had no hope of being practical.”

1998: Leighton and Lewin help found [Akamai](#), taking algorithms for distributed caching into a commercial service.



1997
paper

1998
company

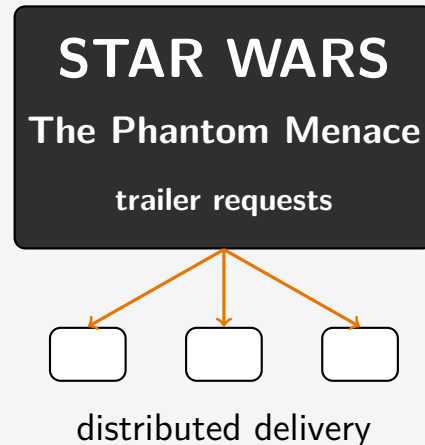
1999
traffic

A Star Wars Trailer and an April Fools' Call

March 1999: demand for the *Phantom Menace* trailer creates an early test of large-scale content delivery.

Apple hosts the trailer, but the rush of fans overwhelms its site. A copy served through Akamai remains available: distributing the cached content lets more machines handle the demand.

April 1, 1999: Apple's Steve Jobs calls Akamai's Paul Sagan, who initially thinks the call is an April Fools' prank. The successful trailer delivery has caught Apple's attention.

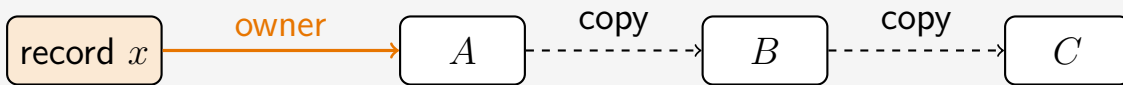


The application combines stable placement with caching, replication, and routing. Consistent hashing is one useful ingredient.

The Idea Outgrows Caching: Dynamo

Dynamo (2007): Amazon's distributed key-value storage system, described by DeCandia and coauthors.

- Assign records to machines using consistent hashing and virtual nodes.
- Replicate records across several **distinct physical machines**.
- Support changes to the machine set, including failures.



Remark

A cache can refetch a missing page, but a storage system needs to preserve the record. This is a different problem, but the underlying solution using consistent hashing is similar.

What If Nobody Knows All the Machines?

So far, a requester stores all q machine positions in a local directory. At a very large scale, even this directory can be too expensive.

Problem

Find a page's owner when each machine knows only a small part of the network. A request may need to contact other machines to reach its destination.

Theorem: Chord (Stoica and coauthors, 2001)

With random identifiers and correct routing information in a stable network:

- Each machine stores $O(\log q)$ contacts.
- A lookup contacts $O(\log q)$ machines, with high probability.

Idea: Use consistent hashing on the directory of machines itself.

Can We Impose a Hard Load Limit?

Virtual nodes balance ownership shares. They do not forbid a large page count. Suppose every machine has capacity

$$C = \left\lceil (1 + \varepsilon) \frac{m}{q} \right\rceil, \quad 0 < \varepsilon \leq 1.$$

A natural idea: if the next machine is full, continue clockwise. But one update can trigger a **cascade** of relocations.

Theorem: Mirrokni–Thorup–Zadimoghaddam (2018)

For $m \geq q$, a dynamic hashing scheme maintains maximum load at most C , with expected relocation costs

$$O(\varepsilon^{-2}) \text{ per item update,} \quad O\left(\frac{m}{q\varepsilon^2}\right) \text{ per machine update.}$$

For $m = q$ and $\varepsilon = 1$: maximum load 2, with constant expected moves.

2021: Aamand, Knudsen, and Thorup improve the dependence on spare capacity from $1/\varepsilon^2$ to $1/\varepsilon$.