

Round hashing for Data Storage

Roberto Grossi and Luca Versari

Presented by Garance Gourdel

Past internships



L3 : Periods in the order-preserving model, Jakub Radoszewski



M1 : Read optimized BWT, Travis Gagie and Gonzalo Navarro



M2 : Similarity measures, Tatiana Starikovskaya

Research to come



ARPE : Prefix search in consistent hashing, Roberto Grossi



PhD : Streaming model with Tatiana Starikovskaya
(conditioned by the finding of a grant)

Article Presented

Round-Hashing for Data Storage: Distributed Servers and External-Memory Tables



Roberto Grossi

Università di Pisa



Luca Versari

Università di Pisa
(Now at Google)

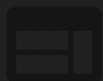
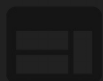
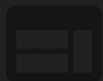
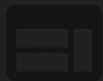
Published at ESA 2018

ESA : European Symposium on Algorithms

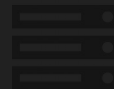
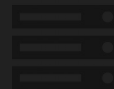
Rank A (Core Ranking)

Consistent hashing

A set of K keys
(web pages)



m Buckets
(servers)

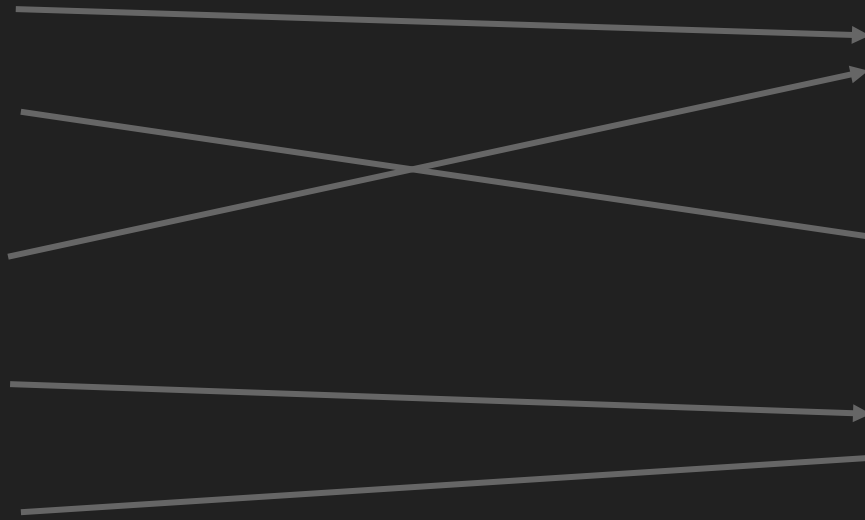
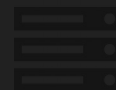
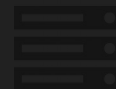
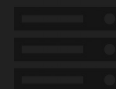


Consistent hashing

A set of K keys
(web pages)



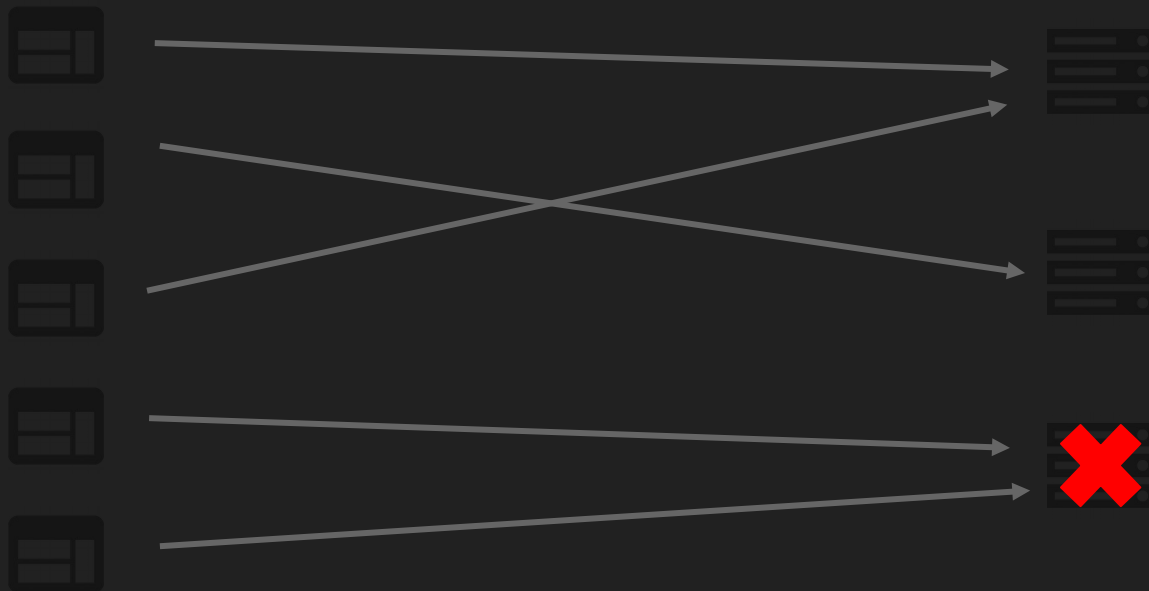
m Buckets
(servers)



Consistent hashing

A set of K keys
(web pages)

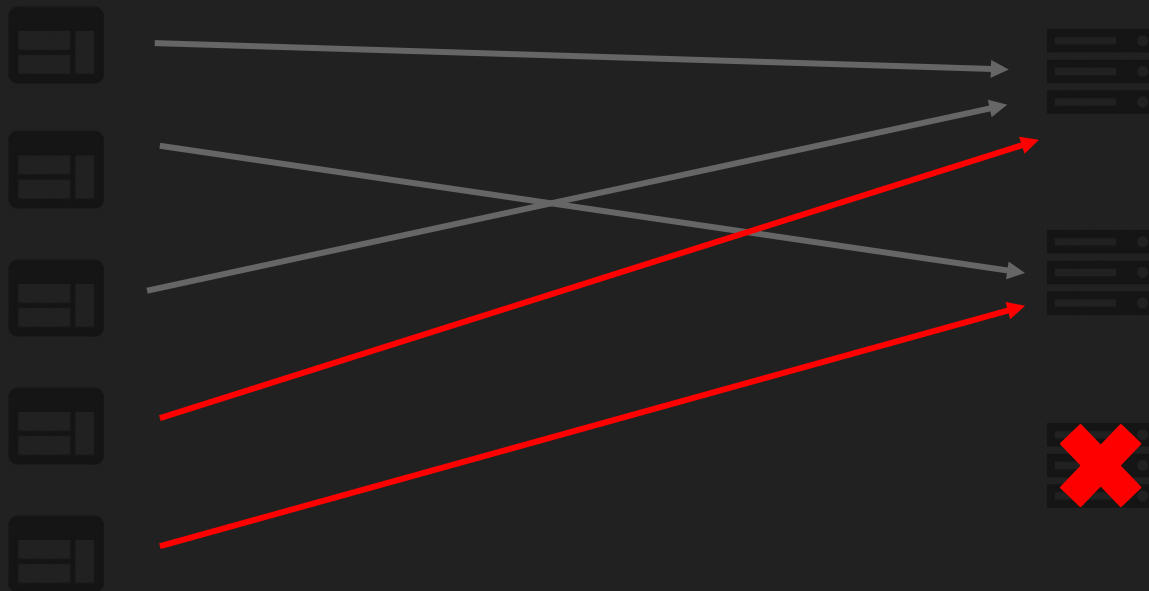
m Buckets
(servers)



Consistent hashing

A set of K keys
(web pages)

m Buckets
(servers)

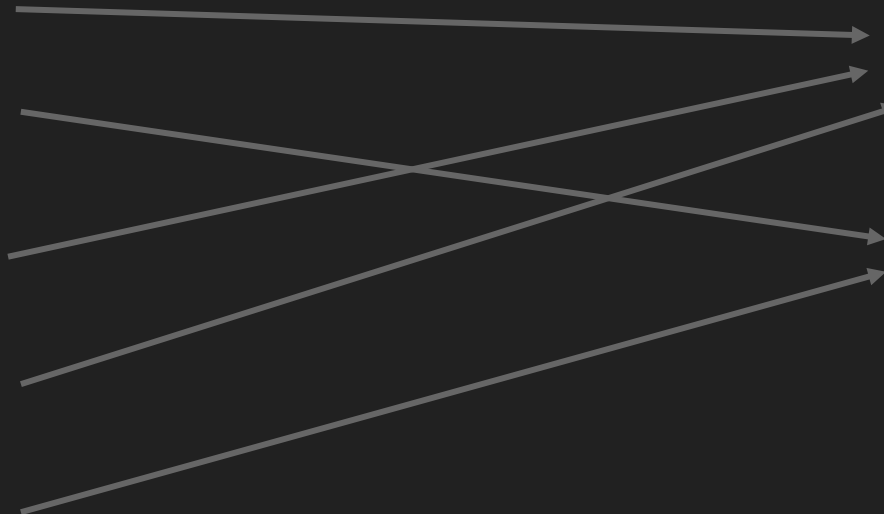
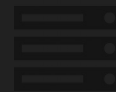
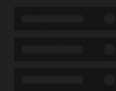


Consistent hashing

A set of K keys
(web pages)

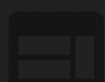


m Buckets
(servers)

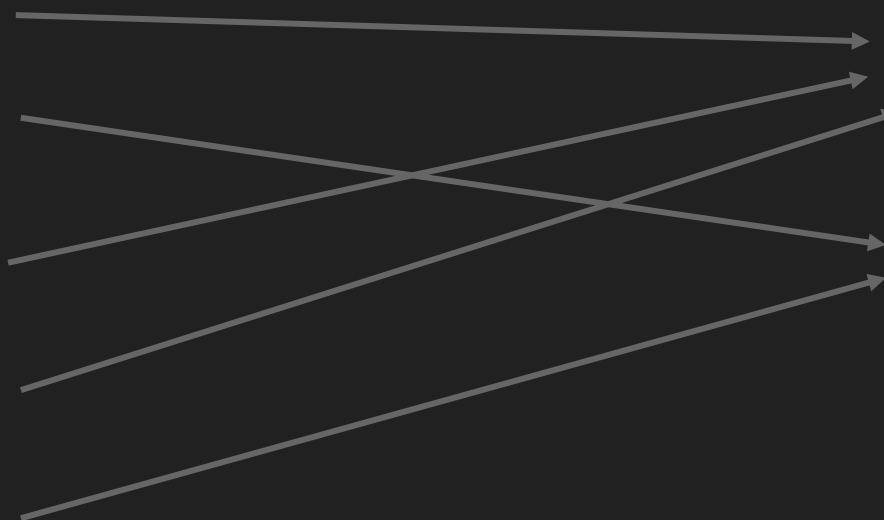
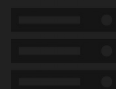
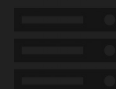
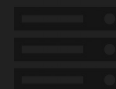


Consistent hashing

A set of K keys
(web pages)



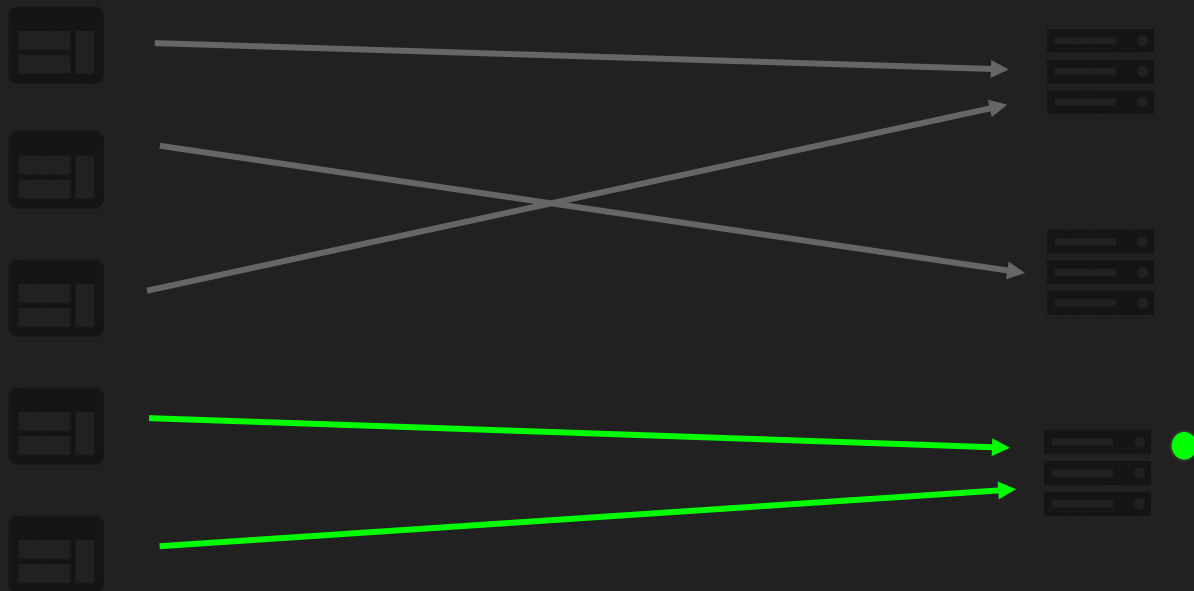
m Buckets
(servers)



Consistent hashing

A set of K keys
(web pages)

m Buckets
(servers)



Operations

- `numBuckets()` - Return the current number of buckets.
- `findBucket(k)` - Given a key `k` find its bucket in $[0, m-1]$.
- `newBucket()` - Add a new bucket `m`.
- `freeBucket()` - Release bucket `m-1`.

Symposium on the Theory of Computing (STOC) 1997

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

Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web.

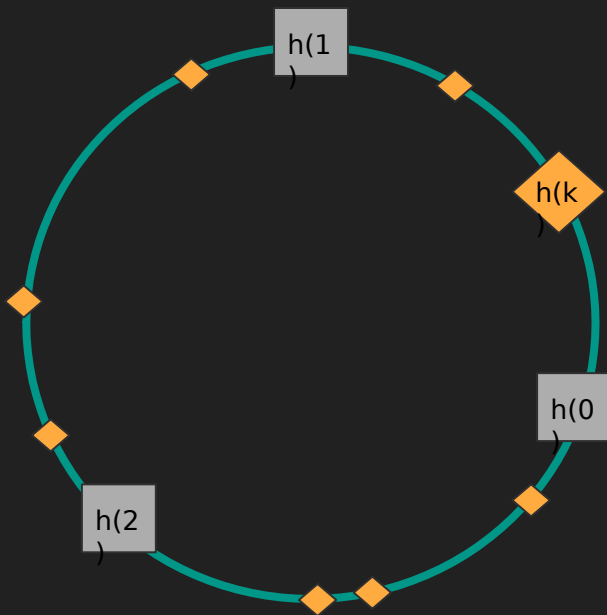
h a uniformly
random hash
function



bucket/server



key/page



Symposium on the Theory of Computing (STOC) 1997

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

Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web.

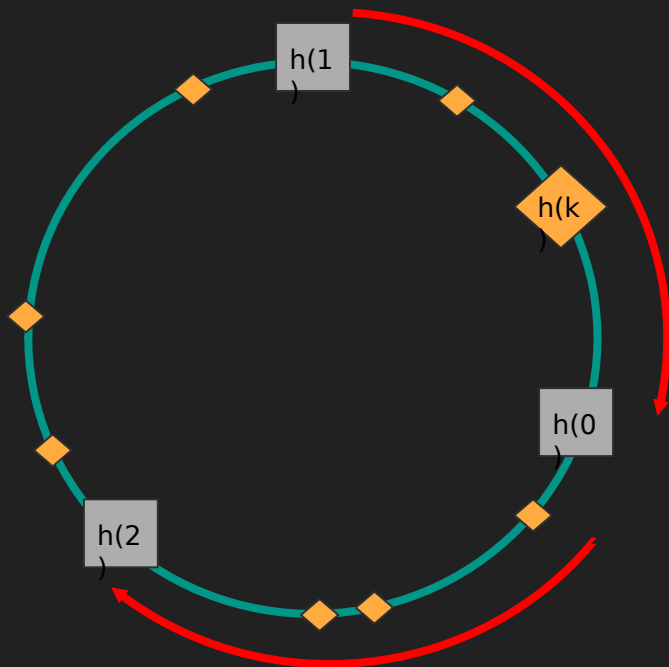
h a uniformly
random hash
function



bucket/server



key/page




Symposium on the Theory of Computing (STOC) 1997

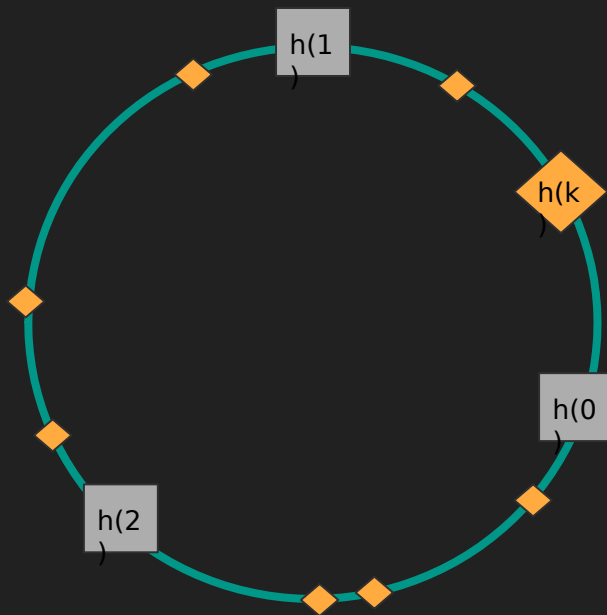
David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin.

Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web.

h a uniformly
random hash
function

 bucket/server

 key/page




Theoretically,
good balancing,
but problematic in
practice with few
servers

Symposium on the Theory of Computing (STOC) 1997

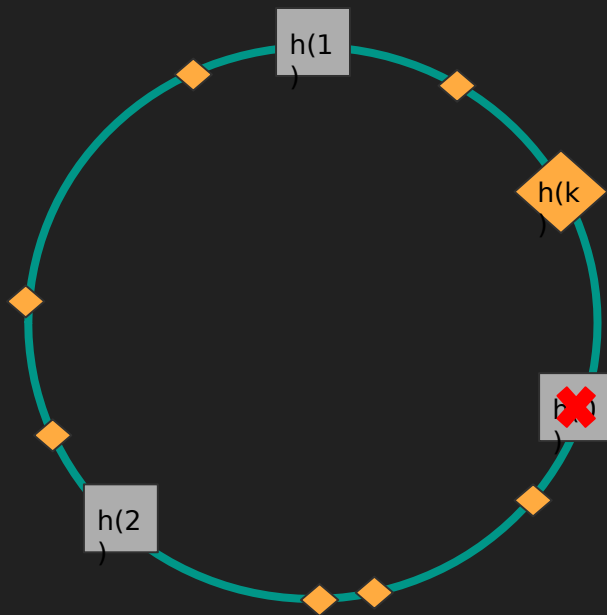
David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin.

Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web.

h a uniformly
random hash
function

 bucket/server

 key/page



Theoretically,
good balancing,
but problematic in
practice with few
servers

Symposium on the Theory of Computing (STOC) 1997

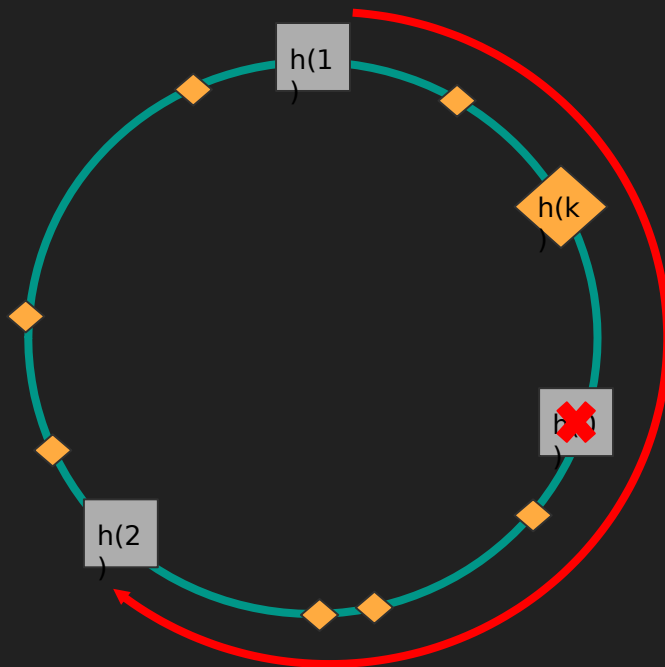
David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin.

Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web.

h a uniformly
random hash
function

■ bucket/server

◆ key/page



Theoretically,
good balancing,
but problematic in
practice with few
servers

IEEE/ACM Transactions on Networking (TON) 1998

*David G. Thaler and Chinya V.
Ravishankar.*

Using Name-Based Mappings to Increase Hit Rates

Highest random weight hashing (Rendez-vous hashing)

For all web page k , and all server s , Compute $h(k,s)$.
 k is assigned to the server that maximize the hash value.

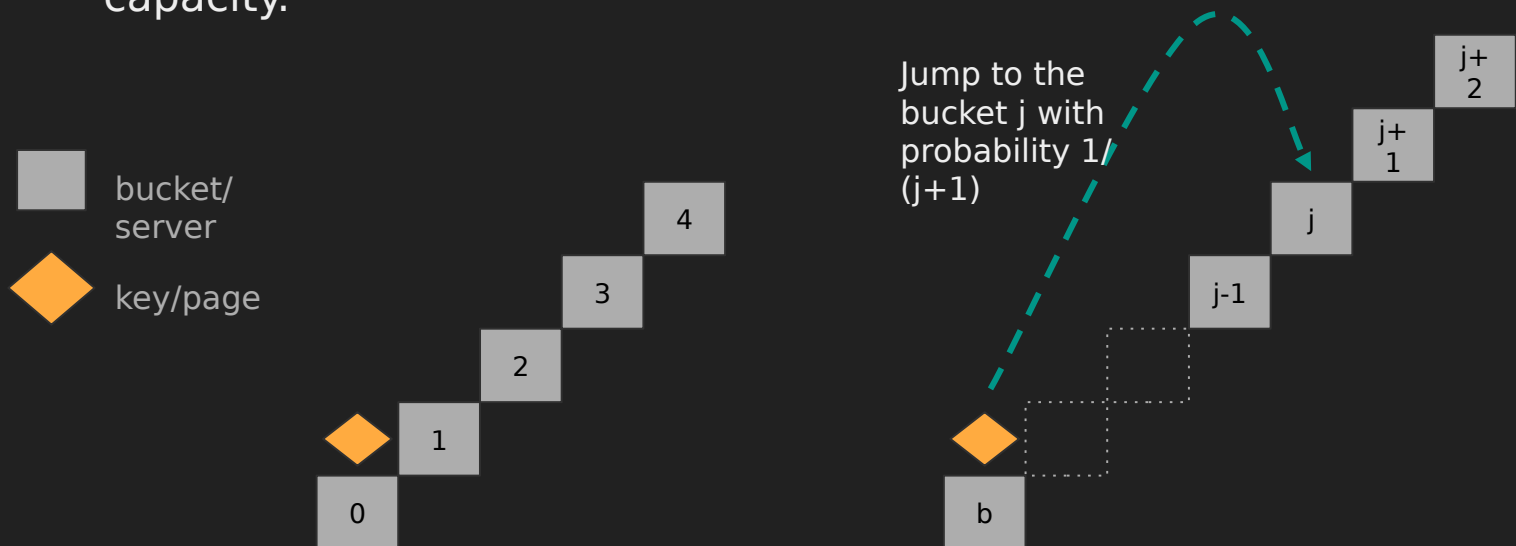
When a server is added each bucket scans its key, if they have a new maximum they move to the new bucket.

Easy redistribution for deleted server.

A fast, minimal memory, consistent hash algorithm

Jump consistent hashing

For data center, servers cannot disappear, it would mean loss of valuable data, they can only be added to increase storage capacity.



New setting: consistent hashing for data storage.

1980 *Witold Litwin*

Linear hashing: A new tool for file and table addressing.

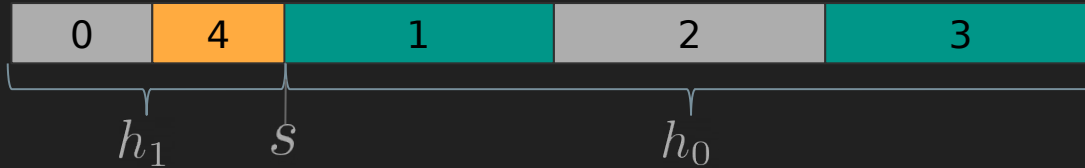
Initial number
of bucket $N = 4$

$$h_0(k) = k \bmod N$$



$N = 5 \quad L = 1$

$$h_1(k) = k \bmod 2N$$



At any time S delimits the frontier between the current level and the previous one,
and :

$$h_L(k) = k \bmod 2^L N$$

Complexity

	find bucket	space	new bucket
consistent hashing	$O(\log m)$	$O(m)$	$\tilde{O}(\alpha + \log m)$
rendezvous hashing	$O(\log m)$	$O(m)$	$O(m\alpha)$
jump consistent hash	$\tilde{O}(\log m)$	$O(1)$	$O(m\alpha)$
linear hashing	$O(\log(m/s_0))$	$O(1)$	$\tilde{O}(s_0\alpha)$
round-hashing	$O(1)$	$O(1)$	$\tilde{O}(s_0\alpha)$

m Number of buckets

α Load factor (K/m)

s_0 Constant user selectable (typically 64 or 128)

Round hashing - Contributions

Round hashing - Contributions

- `findBucket(k)` takes constant time and space in the worst-case (important for security)

Round hashing - Contributions

- `findBucket(k)` takes constant time and space in the worst-case (important for security)
- The factor between the most and the least populated bucket is at most $1 + 1/s_0$

Round hashing - Contributions

- findBucket(k) takes constant time and space in the worst-case (important for security)
- The factor between the most and the least populated bucket is at most $1 + 1/s_0$
- It avoids general divisions and modulo operations.

On Intel processors, euclidean divisions and modulo take from 85 to 100 cycles compared to 1 cycle for addition.

Round hashing

Initialisation:

$s = s_0$

To add a bucket:

If no long arcs :

If $s < 2s_0 - 1$:

$s := s + 1$

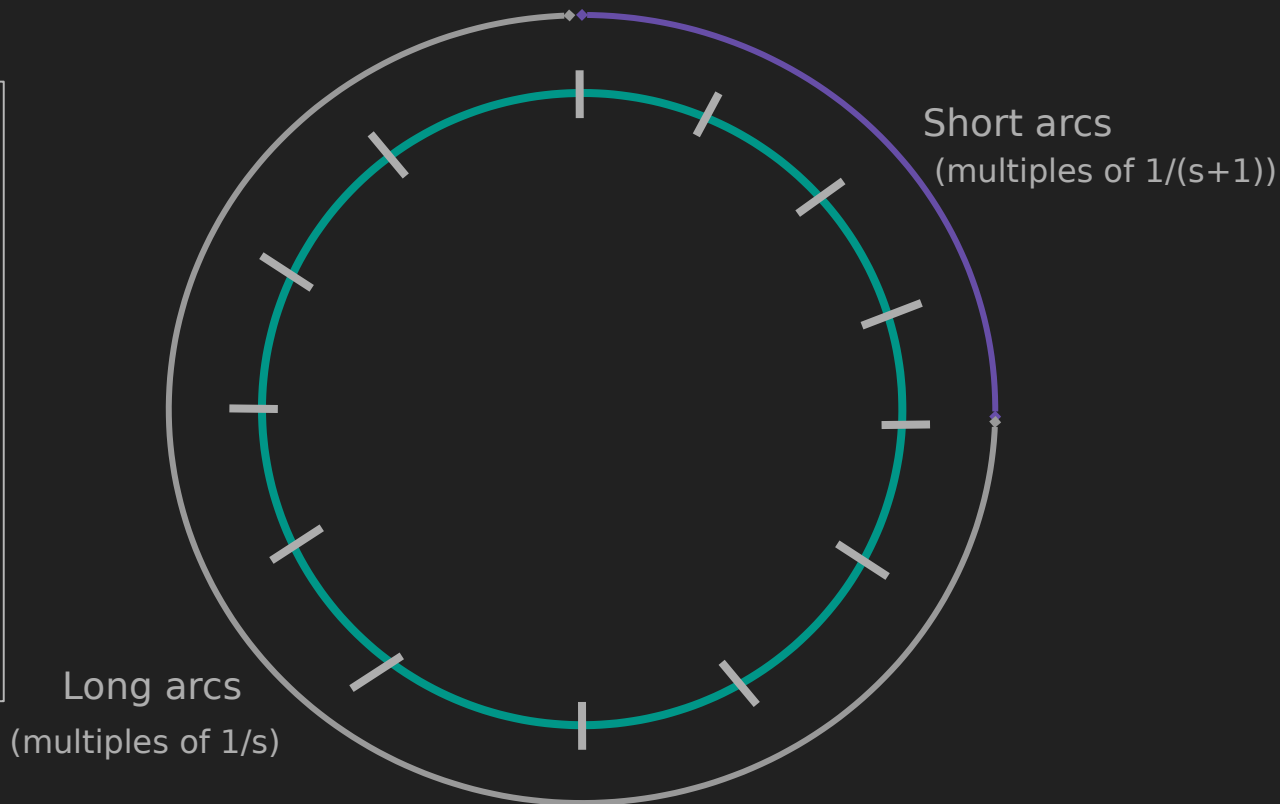
Else :

$s = s_0$

Take the first s long arcs

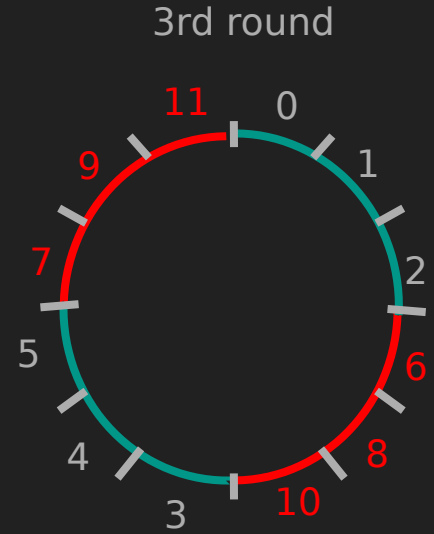
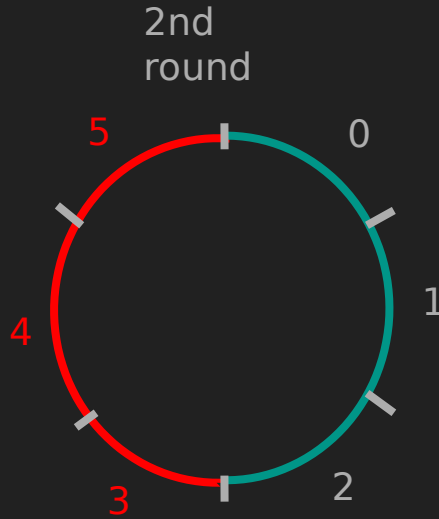
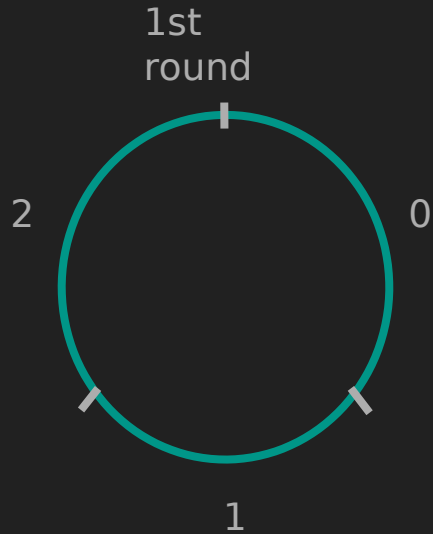
and form $s + 1$ short arcs

with them.



Round hashing

$s_0 = 3$



Round hashing

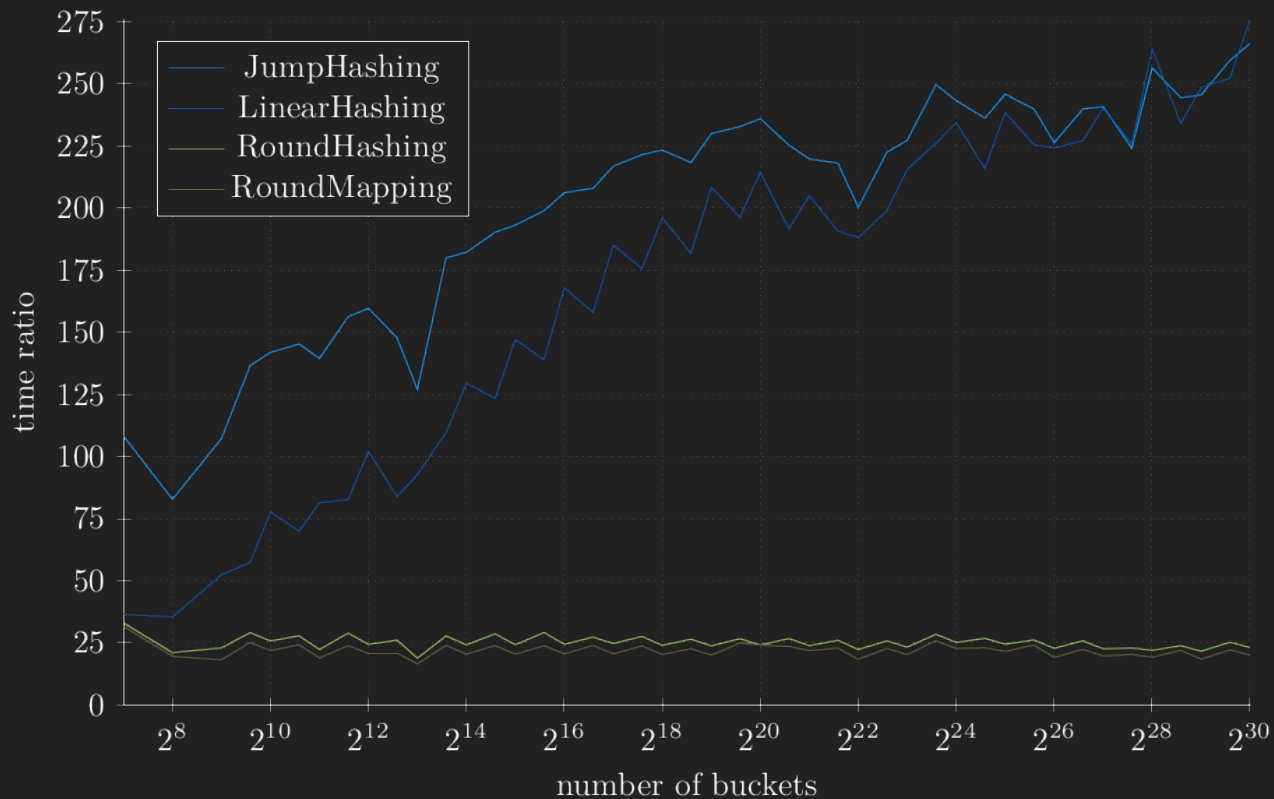
Algorithm 1: Mapping from arcs to buckets.

```
1 Function findBucket( $u$ )
2    $j \leftarrow$  arc hit by  $u$ 
3   if  $j < s_0$  then return  $j$ 
4   if  $j > p$  then  $j' \leftarrow j - \frac{p+1}{s+1}$ ,  $s' = s$ 
5   else  $j' \leftarrow j$ ,  $s' = s + 1$ 
6    $x \leftarrow (j' \% s') \% s_0$ 
7    $q' \leftarrow q + \left\lfloor \frac{s'-1}{s_0} \right\rfloor$ 
8    $i = \left(1 + \left\lfloor \frac{s'-1}{s_0} \right\rfloor\right) \cdot \left\lfloor \frac{j'}{s'} \right\rfloor + \left\lfloor \frac{j' \% s'}{s_0} \right\rfloor$ 
9   return  $pos(i, x, q')$ 

10 Function  $pos(i, x, q)$ 
11    $e \leftarrow$  position of the least significant bit 1 in  $i$ 
12   return  $\left\lfloor \frac{(s_0+x)2^q+i}{2^{e+1}} \right\rfloor$ 
```

Experimental results - time

Computing 10 millions hash values



Experimental results - load balancing

	s_0	$\frac{\sigma}{\mu}$	min	max	1%	99%	percentile ratio
jump consistent h.		0.316	0.988	1.012	0.993	1.007	1.014
round-hashing	1	29.325	0.610	1.221	0.610	1.221	2.001
	2	20.272	0.814	1.221	0.814	1.221	1.500
	4	7.192	0.977	1.221	0.977	1.221	1.250
	8	4.465	0.976	1.085	0.976	1.085	1.112
	16	2.560	0.976	1.028	0.976	1.028	1.053
	32	0.613	0.976	1.002	0.976	1.002	1.027
	64	0.421	0.989	1.002	0.989	1.002	1.013
	128	0.277	0.995	1.002	0.995	1.002	1.007
linear hashing	1	29.329	0.602	1.232	0.605	1.228	2.030
	2	20.274	0.803	1.234	0.808	1.228	1.520
	4	7.203	0.964	1.232	0.969	1.225	1.264
	8	4.476	0.965	1.095	0.970	1.090	1.124
	16	2.583	0.965	1.041	0.970	1.034	1.066
	32	0.685	0.968	1.014	0.973	1.009	1.037
	64	0.527	0.980	1.014	0.984	1.009	1.025
	128	0.417	0.985	1.014	0.990	1.009	1.019

External memory tables: round table

- High-throughput servers with many lookup request, few updates
- Some keys can be kept in a stash in main memory

n the total number of keys in the table.

B the maximum number of keys that fit inside one block transfer.

k the number of keys in the stash.

- Look up: Reads just 1 block of EM, thus constant CPU time.
- Update: $O(s_0(B + \log n / \log \log n))$ in the worst case, $O(s_0 B)$ expected.
- The number of keys in the stash : $k \approx n / \exp(B)$

My ARPE Research subject

Prefix search in consistent hashing

My ARPE Research subject

Prefix search in consistent hashing

Several possible definitions, Given P a string:

My ARPE Research subject

Prefix search in consistent hashing

Several possible definitions, Given P a string:

- Return a bucket containing a key that has P as a prefix.

My ARPE Research subject

Prefix search in consistent hashing

Several possible definitions, Given P a string:

- Return a bucket containing a key that has P as a prefix.
- Return the list of all buckets which contain a key that has P as a prefix.

My ARPE Research subject

Prefix search in consistent hashing

Several possible definitions, Given P a string:

- Return a bucket containing a key that has P as a prefix.
- Return the list of all buckets which contain a key that has P as a prefix.
- Return the bucket containing the key which has the longest common prefix with P .

My ARPE Research subject

Prefix search in consistent hashing

Several possible definitions, Given P a string:

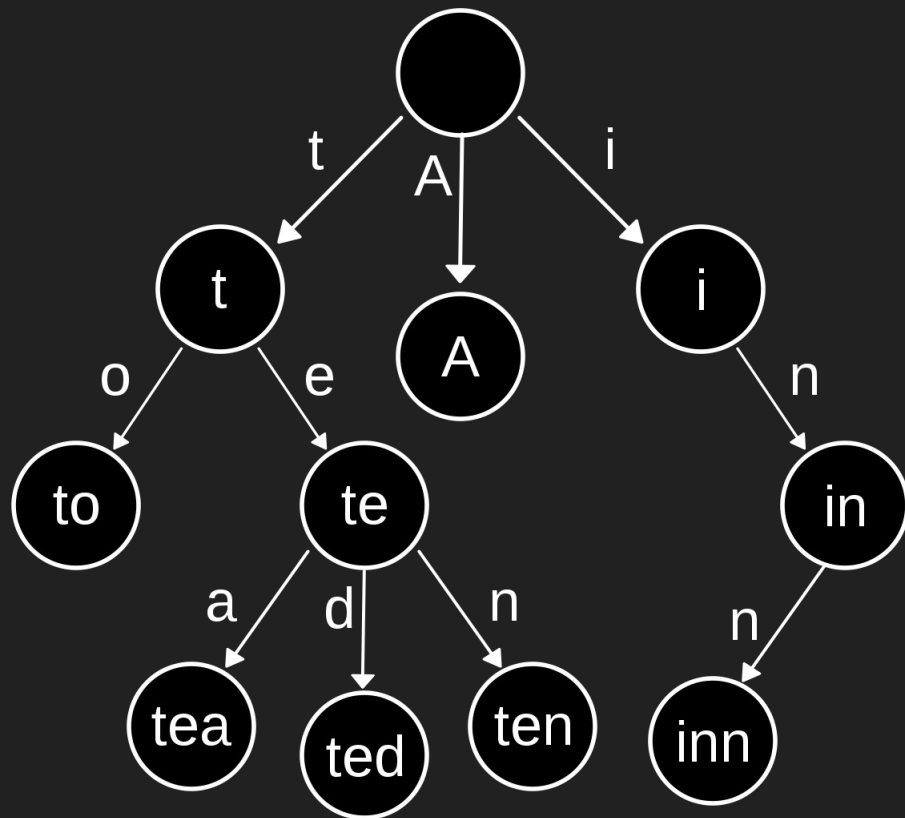
- Return a bucket containing a key that has P as a prefix.
- Return the list of all buckets which contain a key that has P as a prefix.
- Return the bucket containing the key which has the longest common prefix with P .

Trie

The keys get sorted in lexicographical order.

You can easily do a prefix search.

But how do you map to different buckets ?



We need to combine the efficient tools for prefix search and efficient consistent hashing.

Inspirations:

- Hashed Patricia Trie
- Zuffix arrays
- Idea ?
 - Projection of the key preserved in lexicographic order
 - Find the Longest common prefix
 - Return it's bucket

The answer will partially depends on the possible applications.

Thank you for your attention !

Questions ?