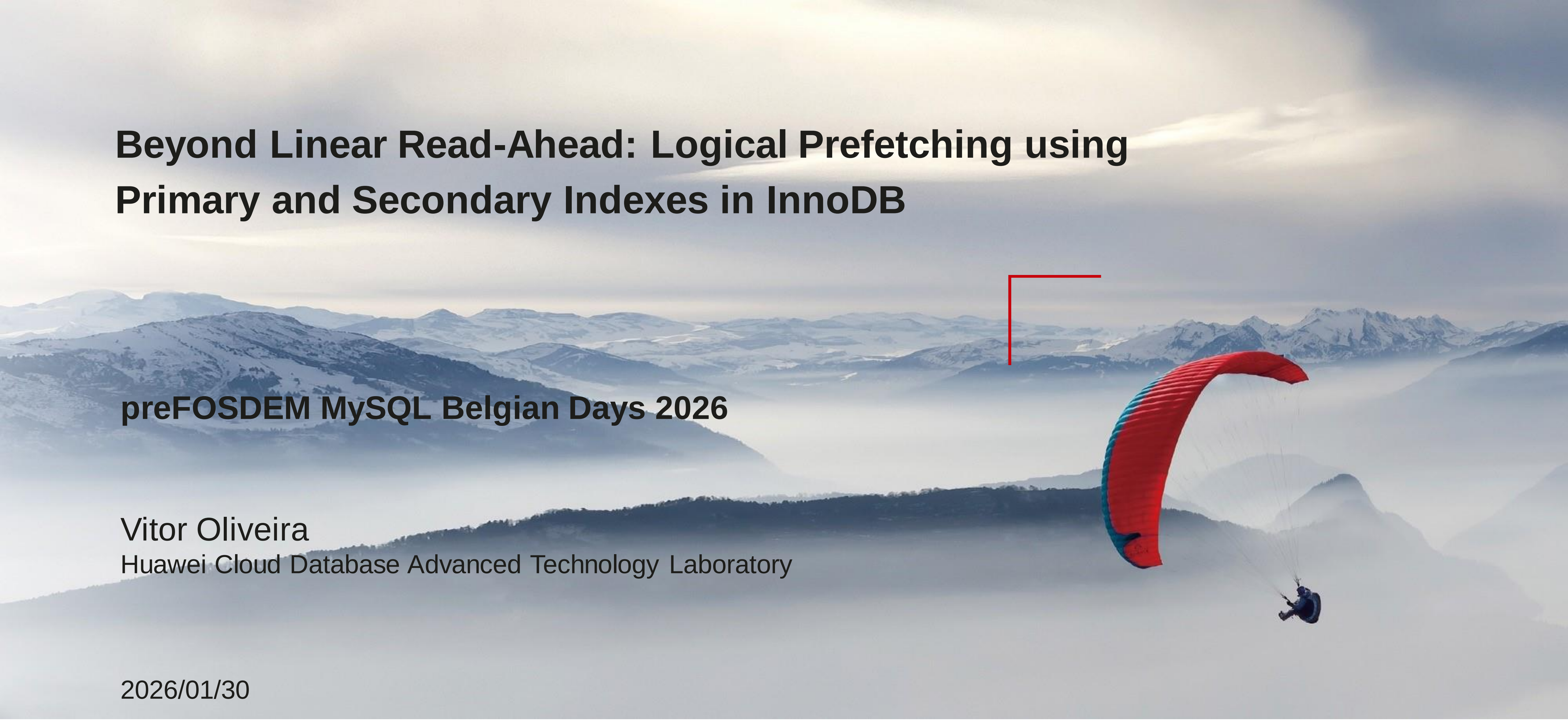


Beyond Linear Read-Ahead: Logical Prefetching using Primary and Secondary Indexes in InnoDB

preFOSDEM MySQL Belgian Days 2026

Vitor Oliveira
Huawei Cloud Database Advanced Technology Laboratory

2026/01/30

HUAWEI

About me

I’m a Performance Architect with background in database and in HPC performance analysis and optimization.

I work at Huawei's Cloud Database Advanced Technology Laboratory, Shannon Research Center since 2021.

My activity has been focused on database optimization for OLTP and analytics workloads, profiling tools and novel architectures involving persistent memory, low-latency networks and memory fabrics.

Prior to that I was a member of Oracle’s MySQL Replication team since 2014.

There I collaborated in projects like Group Replication and MySQL Database Service (HA) on OCI, and worked closely on features like flow control and WRITESET dependency tracking, among others.

I’ve also been teaching sometimes, researching and enjoying music, theatre, physics and also architecture (the real one).

Introduction

MySQL performs at peak performance when the active workset of the database is kept in main memory.

Performance degrades drastically if pages need to be swaped in and out if part of the database is offloaded to storage.

Following our interest in analytics and a strange behavior reported on network storage, we became interested on the impact of prefetching on TPC-H workloads larger than memory.

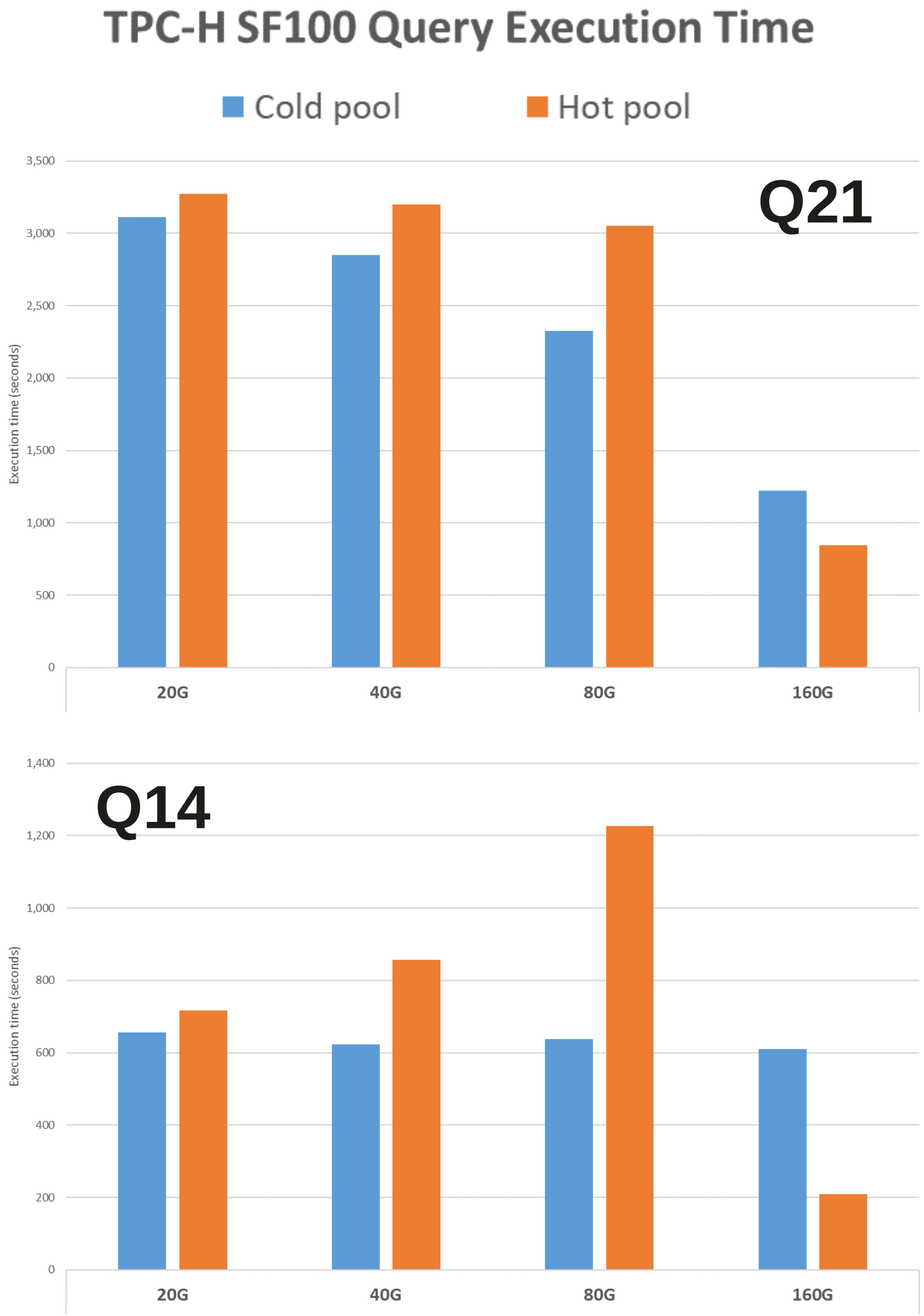
This presentation is about some of the issues we found and about an approach to address them.

The problem

1 TPC-H results showed that some queries would run faster when the system started (cold pool) than on subsequent runs (hot pool).

2 That’s unexpected, as having data already present in memory should improve query time, not make it worse.

3 Even stranger, some results were actually better on smaller buffer pools than on larger buffer pools.

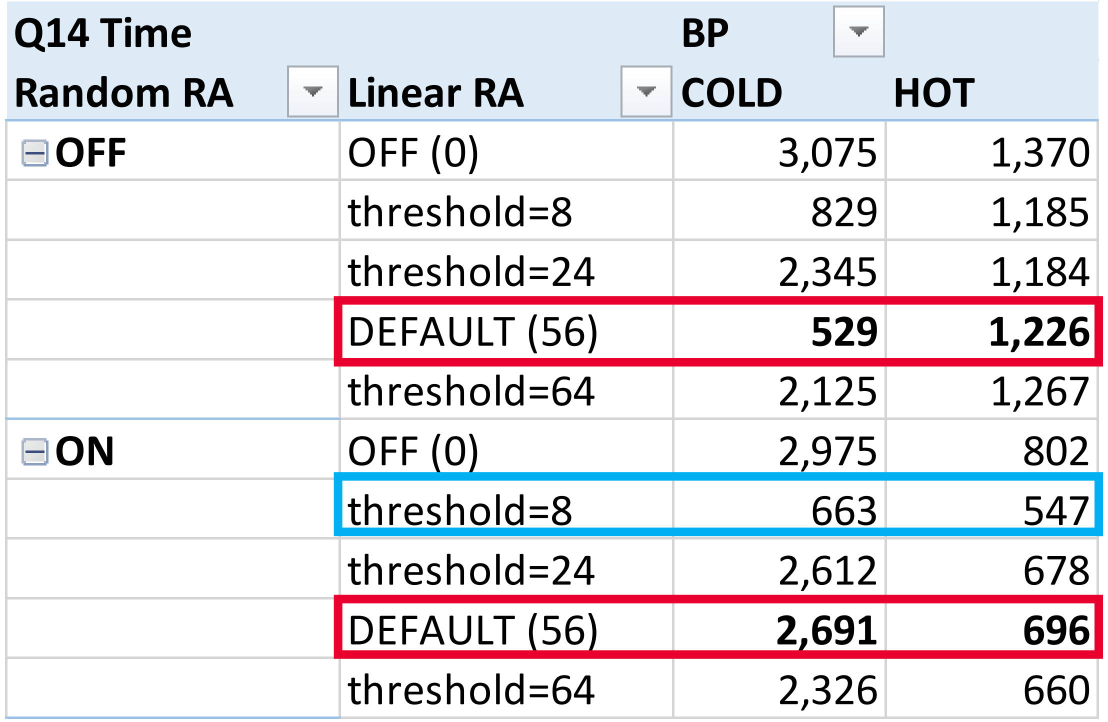

TPC-H SF100 Query Execution Time

Prefetching?

Suspecting prefetcher involvement we experimented with InnoDB's two prefetcher mechanisms:

- **Linear read-ahead**
  Triggers the read of the next extent if it reads sequentially at least X pages of current extent (default X=56).

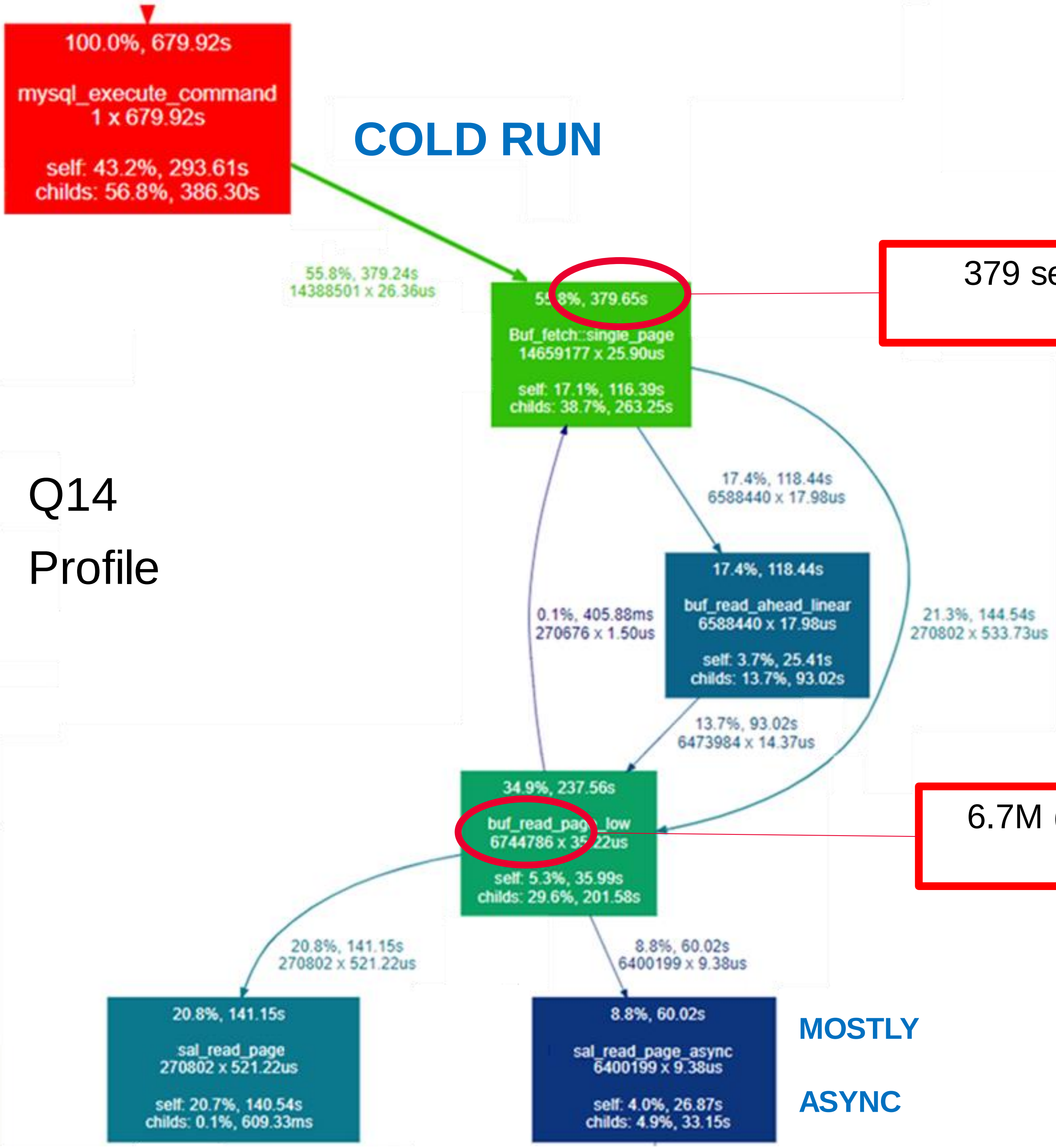
- **Random read-ahead**
  Fetches all pages if more than 13 pages in the extent are already loaded in the buffer pool (disabled by default).



Q14 Time		BP	
Random RA	Linear RA	COLD	HOT
OFF	OFF (0)	3,075	1,370
	threshold=8	829	1,185
	threshold=24	2,345	1,184
	DEFAULT (56)	**529**	**1,226**
	threshold=64	2,125	1,267
ON	OFF (0)	2,975	802
	threshold=8	663	547
	threshold=24	2,612	678
	DEFAULT (56)	**2,691**	**696**
	threshold=64	2,326	660

Q14
Profile

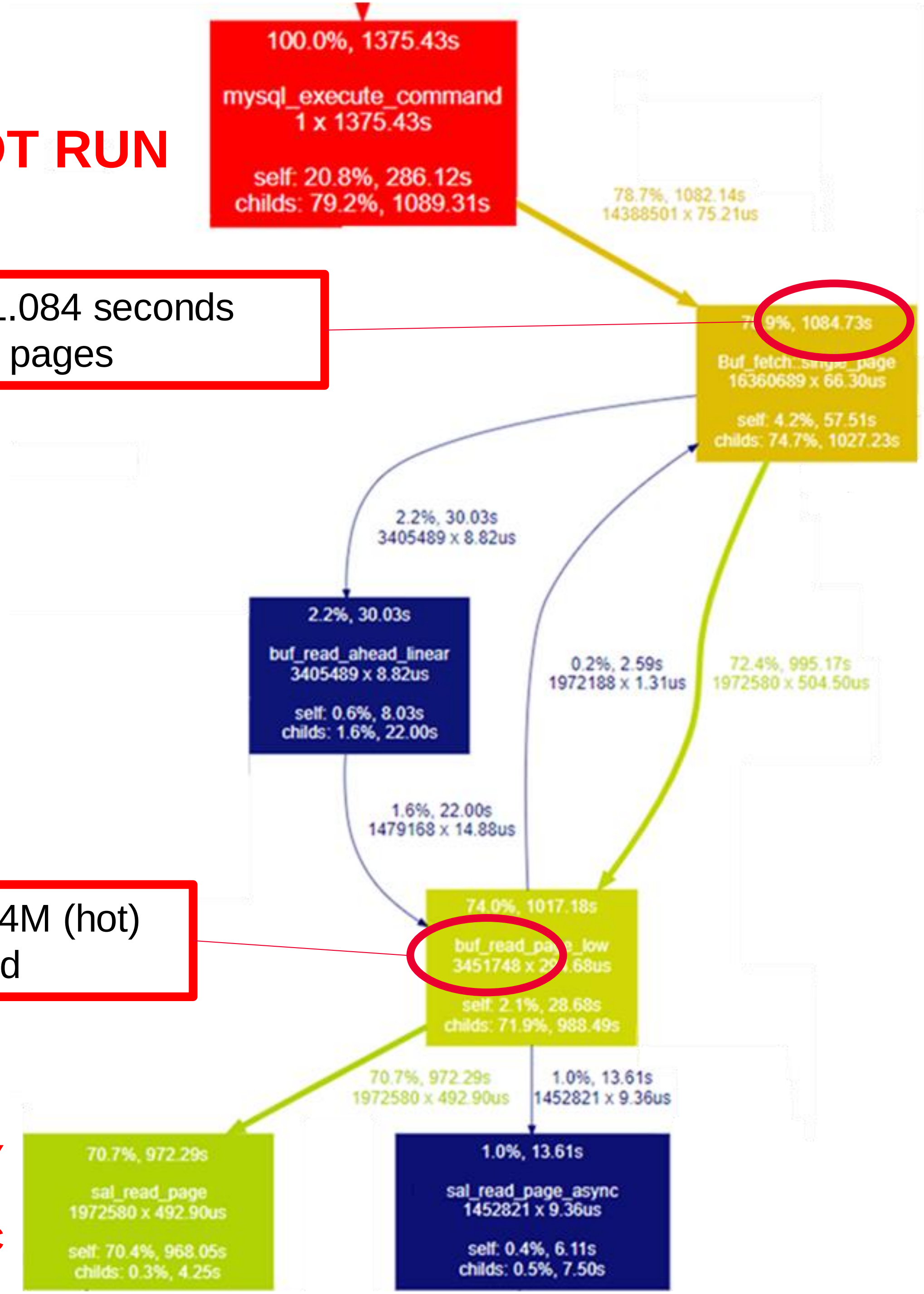
COLD RUN



MOSTLY

ASYNC

HOT RUN



MOSTLY

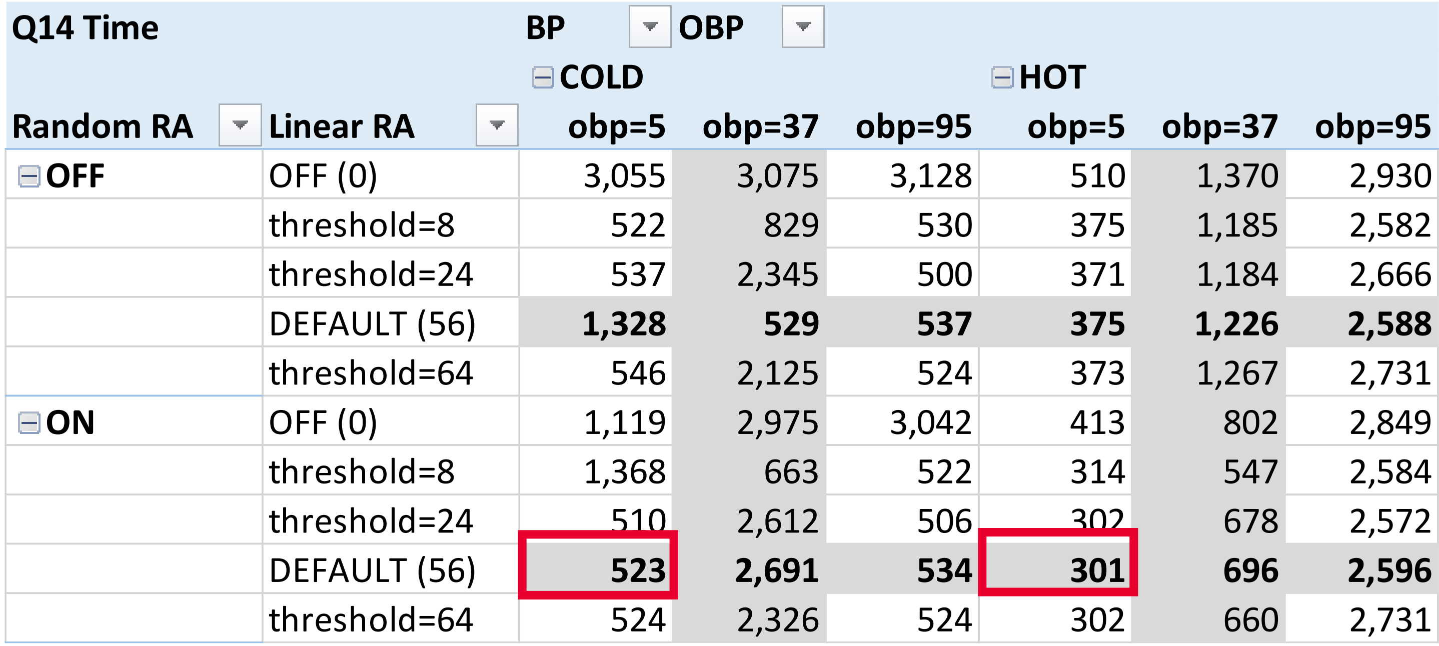
SYNC

379 seconds vs 1.084 seconds accessing pages

6.7M (cold) vs 3.4M (hot) pages read

LRU insertion point

The instability shown suggests the LRU management may be playing a role, so we included also the innodb-old-block-percent (OBP).



Q14 Time		BP ▾	OBP ▾				
		⊟ COLD			⊟ HOT		
Random RA ▾	Linear RA ▾	obp=5	obp=37	obp=95	obp=5	obp=37	obp=95
⊟ **OFF**	OFF (0)	3,055	3,075	3,128	510	1,370	2,930
	threshold=8	522	829	530	375	1,185	2,582
	threshold=24	537	2,345	500	371	1,184	2,666
	DEFAULT (56)	**1,328**	**529**	**537**	**375**	**1,226**	**2,588**
	threshold=64	546	2,125	524	373	1,267	2,731
⊟ **ON**	OFF (0)	1,119	2,975	3,042	413	802	2,849
	threshold=8	1,368	663	522	314	547	2,584
	threshold=24	510	2,612	506	302	678	2,572
	DEFAULT (56)	**523**	**2,691**	**534**	**301**	**696**	**2,596**
	threshold=64	524	2,326	524	302	660	2,731

1. The LRU insertion point has indeed a big impact, with 5% showing much lower numbers than the default 37%.

2. That does not apply to all cases, and in fact it does apply to MySQL's default prefetcher configuration.

3. But it does apply to when random read ahead, which is also unexpected since the workload is mostly sequential.

The pool is on fire

METRIC	COLD POOL		HOT POOL	
	obp5	obp37	obp5	obp37
page uses / storage read	4.0	4.1	12.4	6.9
pages loaded / buffer pool size	127.2%	126.4%	34.0%	65.3%
pages read synchronously	4.3%	3.5%	6.3%	30.5%
pages read asynchronously	122.9%	122.9%	27.7%	34.8%
unused read-ahead pages	0.4%	0.4%	0.4%	0.7%

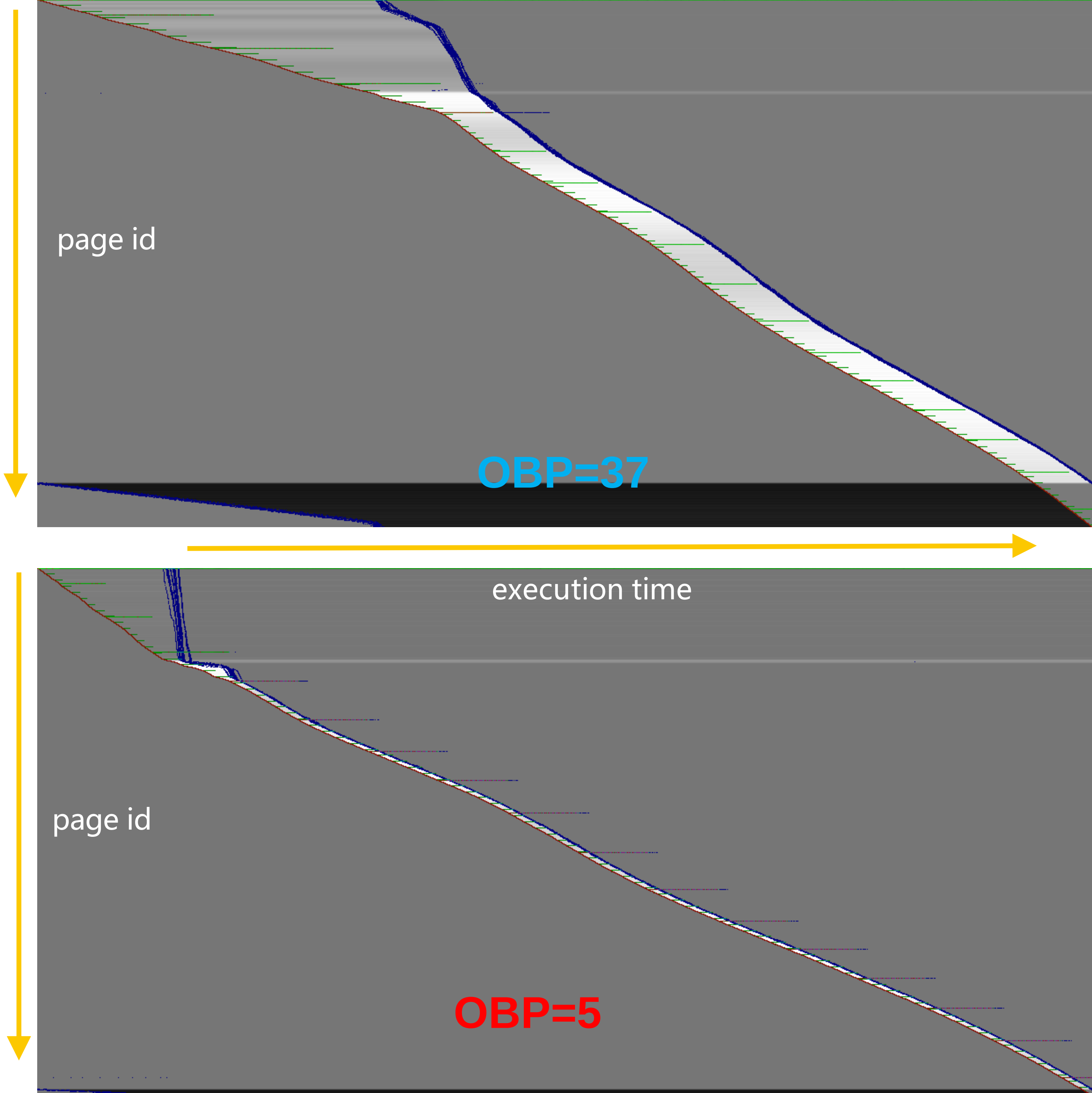
background = black -> grey -> white

number of pages in memory in that range in the **orders** table

red = read from storage

green = read from buffer pool

blue = page freed



Linear read-ahead

1. The linear read-ahead prefetcher is invoked when trying to access a page that is missing from the buffer pool, but it is only triggered if called for the boundary pages of the extent;

2. Then it scans the pages in the current extent and counts the number of pages present in the buffer pool that have a sequential access time, either descending order if it is the first page of the extent, or ascending order if it is the last page of the extent;

3. If that number exceeds the **innodb_read_ahead_threshold**, the successor/predecessor of the current page is fetched and if that is also a boundary page of the extent it belongs to, it will trigger the asynchronous read of all 64 pages in that extent.

Limitations

1 **It depends on the physical layout of the data**

- Fragmented databases, as long running database tend to be, will not benefit much from the prefetcher as the sequence information will be wrong.

2 **Linear read-ahead can be disabled by the pages previously loaded in the BP**

- if the boundary page is present in the buffer pool, nothing is prefetched;
- non sequential access time between new and old pages counts as a sequence interruption, which counts for the maximum number in innodb_read_ahead_threshold.

3 **The query may have to wait for storage even when linear read-ahead is working**

- Prefetch is only triggered on the last page of the previous extent;
- The next page will take time to load, even more when overloaded with 64 simultaneous requests.

Can we do something about this?...

Enhanced read-ahead (ERA)

To try to improve on the previous issues, we developed an enhanced read-ahead mechanism to address, mostly as exploratory tool to evaluate the impact of different alternative approaches to prefetching.

Initial requirements:

1. prefetch pages even if theirs neighbours are already in the buffer pool;
2. Use logical prefetching to benefit the workload even on fragmented databases, not just the physical layout;
3. be friendly to storage, with patterns fit for cloud-based deployments.

Things to avoid:

- don’t read too late to avoid waiting for storage;
- don’t read too early to avoid prematurely discarding pages from the buffer pool;
- don’t overload the system when prefetching is not needed or is ineffective.

Logical prefetching

A main problem is knowing which page follows another without actually reading that pages from disk, which is why linear read-ahead uses physical proximity as an heuristic.

> That sequence can be read ahead from the clustered index, similar to what facebook did in https://yoshinorimatsunobu.blogspot.com/2013/10/making-full-table-scan-10x-faster-in.html.

> We also wanted to try using secondary indexes, even if those have to scan both the secondary index and the corresponding clustered index to find the pages that are next in a sequence.

> But to avoid some overhead, we also considered a page sequence cache to keep a cache of the pages sequences which have been loaded over time

Ideally the prefetcher should also work with the semantic information provided by the MySQL layer, that would be used to guide the prefetching more closely.

Design

1. Track all page accesses per thread and store the page and additional positioning information in a sequence history list;

2. If at least N predecessors of the current page/position are in the history list, make sure that at least M successors of the current page/position is in the global prefetch list, except for pages that:
   - > are already in memory or
   - > pre-fetch was already been requested.

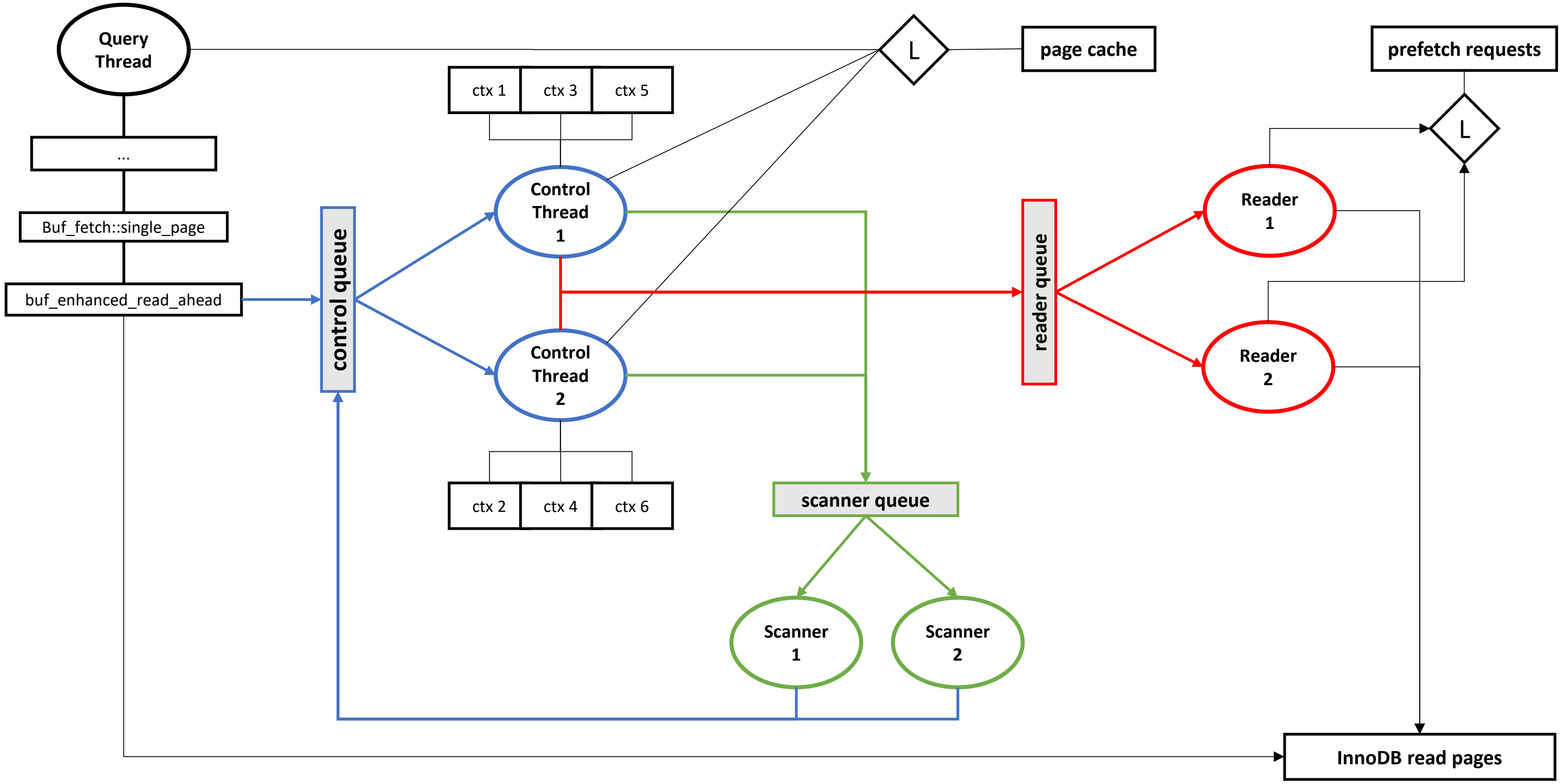
3. Discover the predecessor/successor page sequences ahead of the requests, using page sequence information obtained from multiple sources using independent threads.

Design

1. Control thread does history checking and selects the list of successors to pre-fetch;

2. Reader thread takes the pre-fetch requests and issues the storage reads and tracks their success;

3. Scanner thread fetches sequence information from indexes when that is found missing.



The threads communicate through messages:

- Query threads sends the page_id and the frontend context information to the control thread when accessing a page;
- When the control thread needs to read from disk it sends the list of pages to read to the read queue, and when it needs index pages to be processed it sends the index page_id to the scanner queue;
- When the scanner thread has read the sequences, it sends the list to the control thread to import.

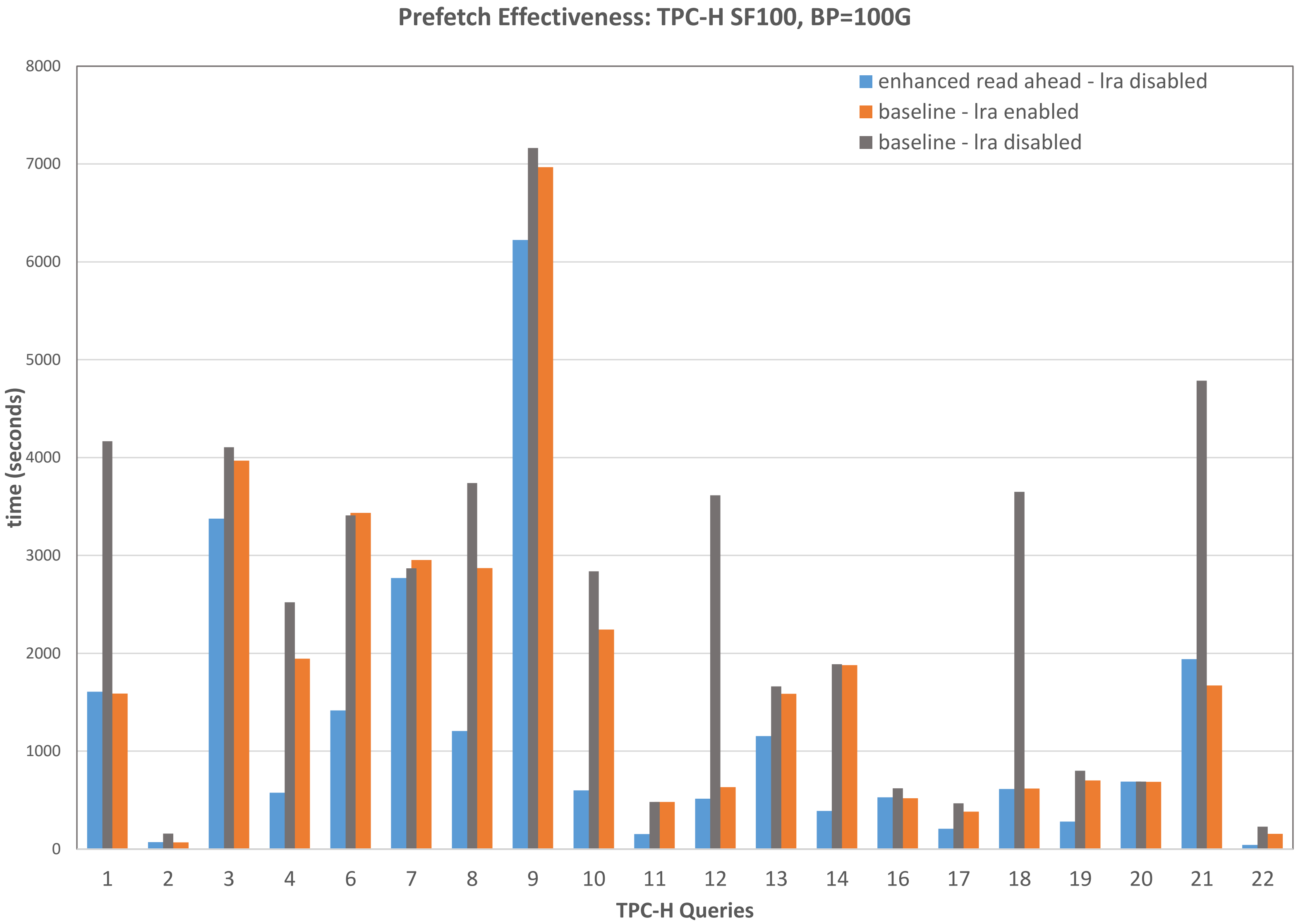
Evaluation

TPC-H SF100 on a 100GB buffer pool, a little less than half of the database on the buffer pool.

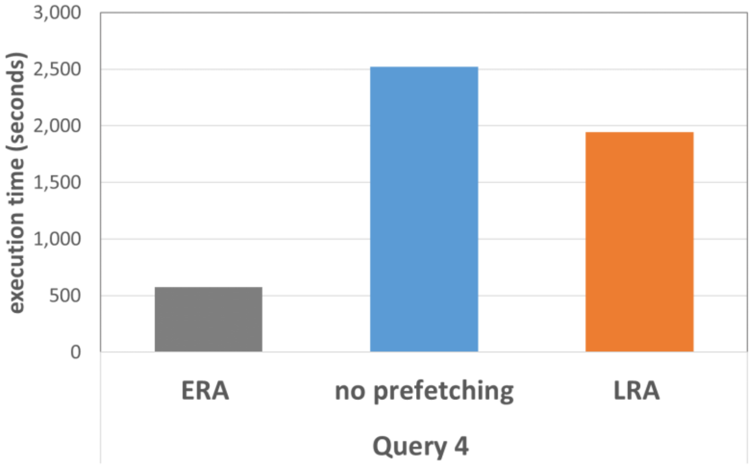
1. LRA saves 30% of the time, while ERA 52% of the time, compared to disabling prefetching;

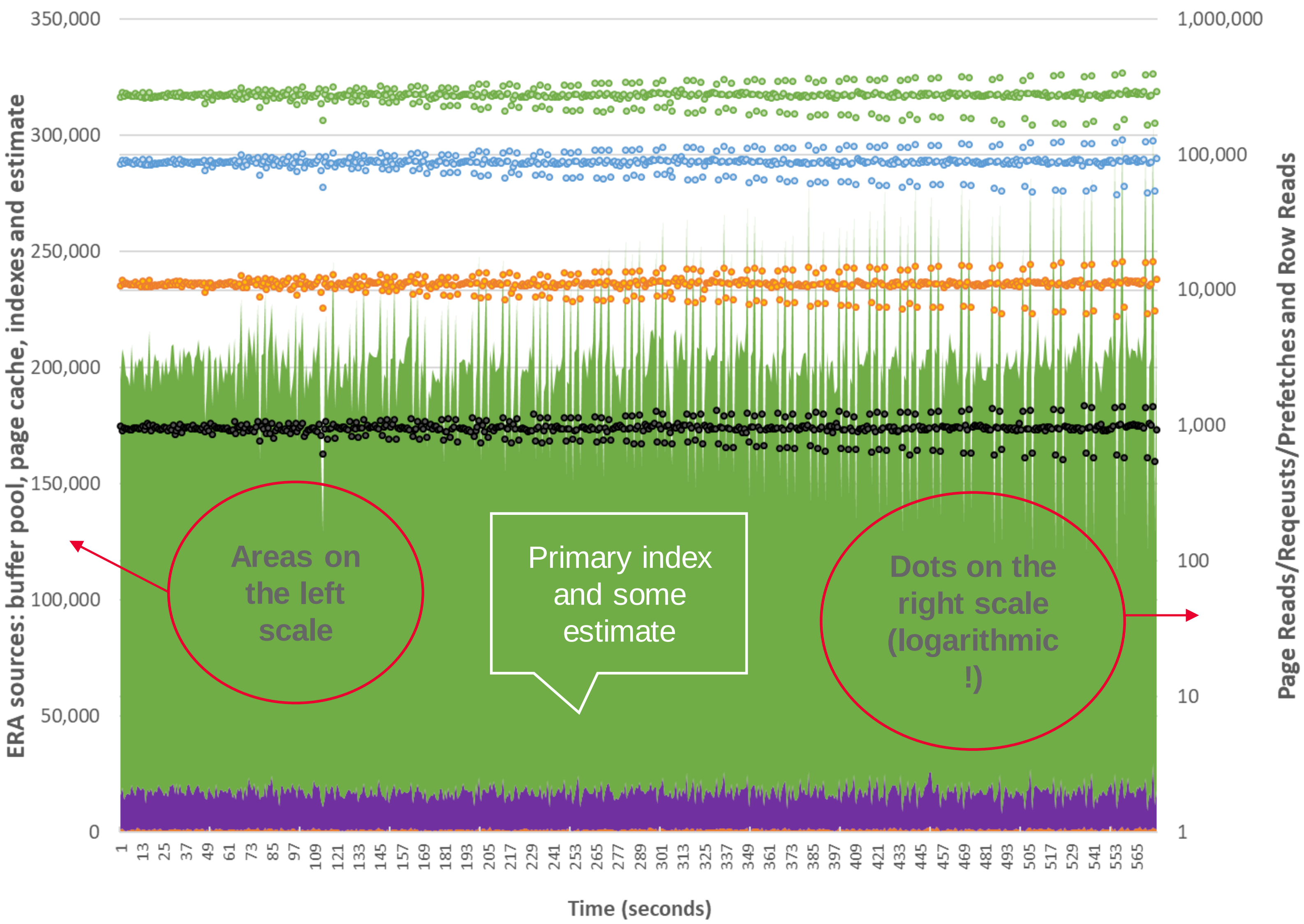
2. ERA benefit more queries than LRA, or in greater amount;

3. In queries 4, 6, 8, 10, 11, 14, 17, 19 and 22 the difference is particularly significant.



Evaluation





Areas on the left scale

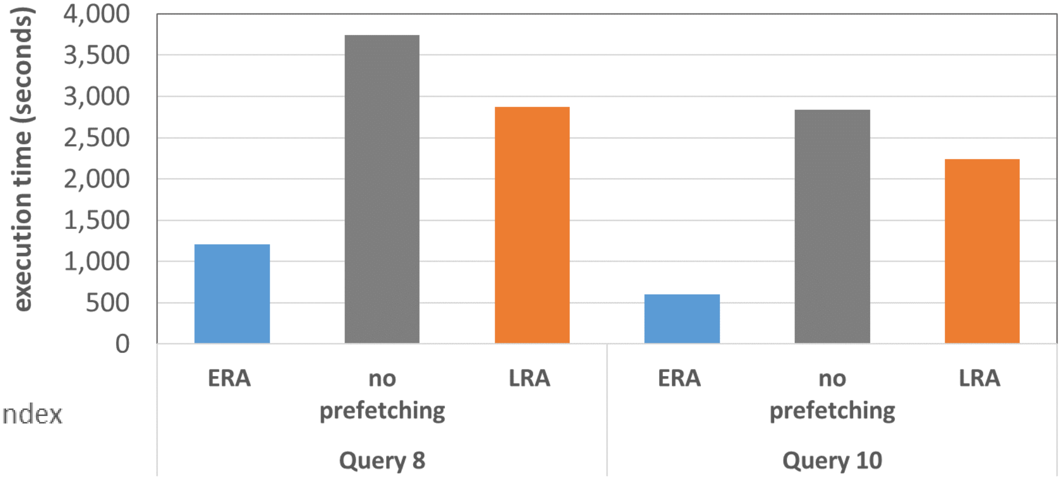
Primary index and some estimate

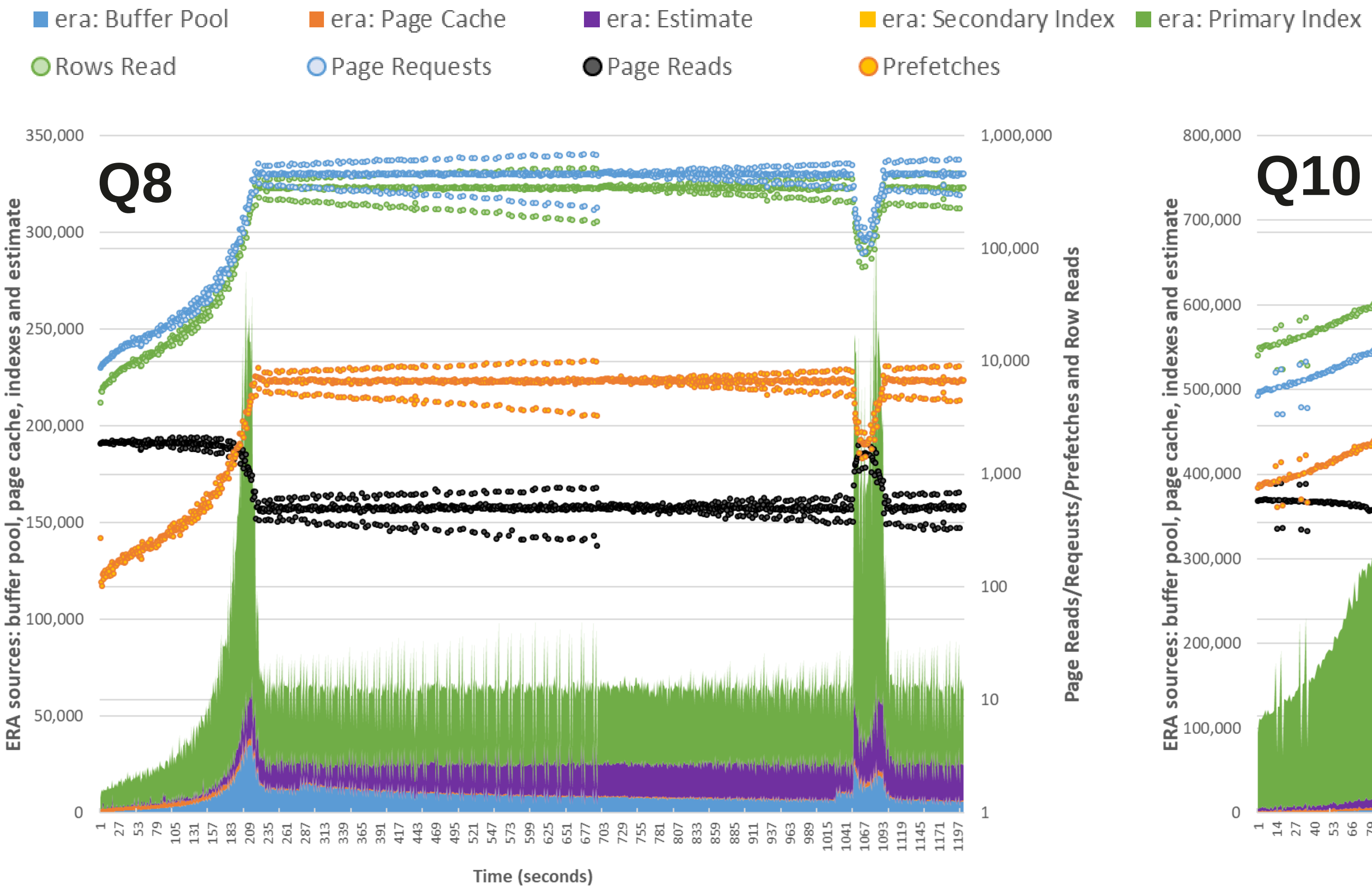
Dots on the right scale (logarithmic !)

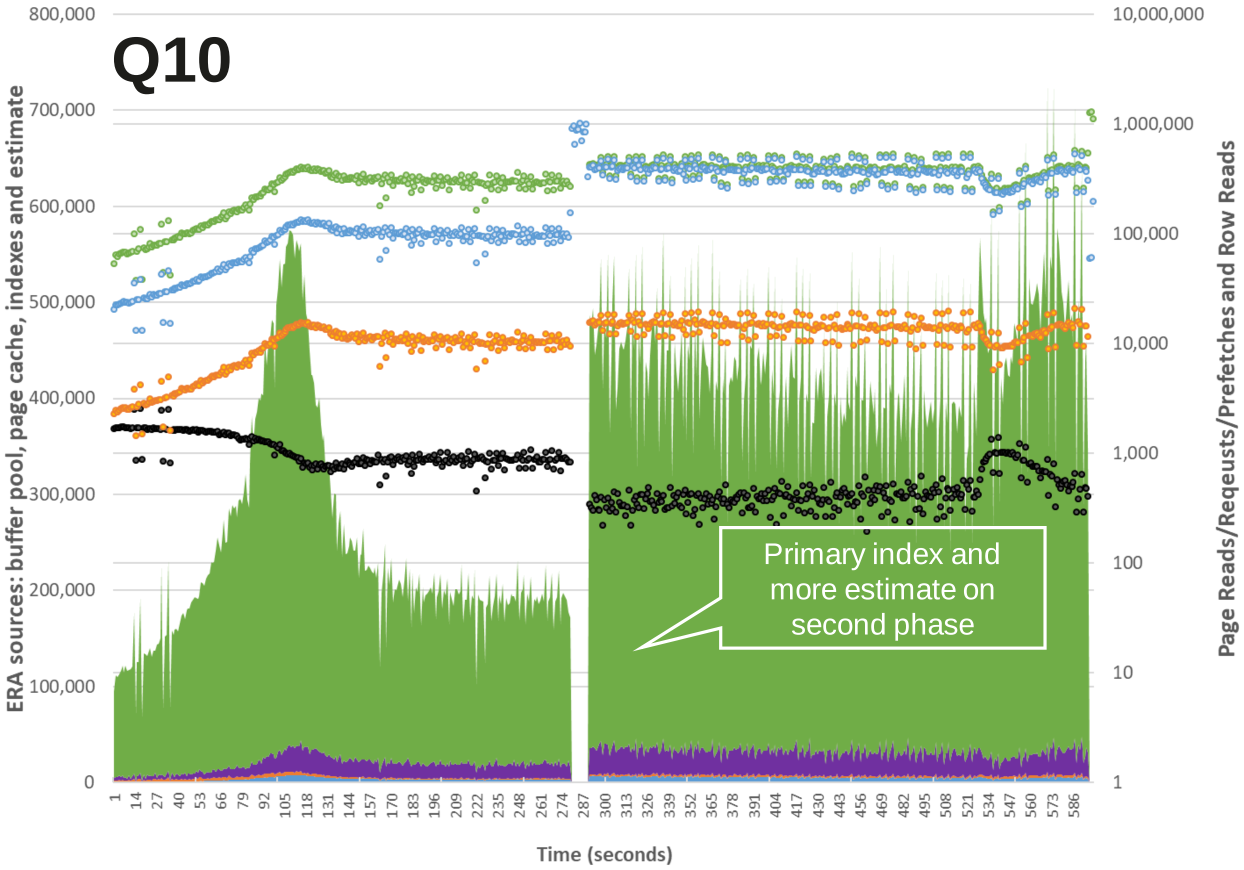
🟦 era: Buffer Pool 🟧 era: Page Cache 🟪 era: Estimate 🟨 era: Secondary Index 🟩 era: Primary Index

○ Rows Read ○ Page Requests ● Page Reads ● Prefetches

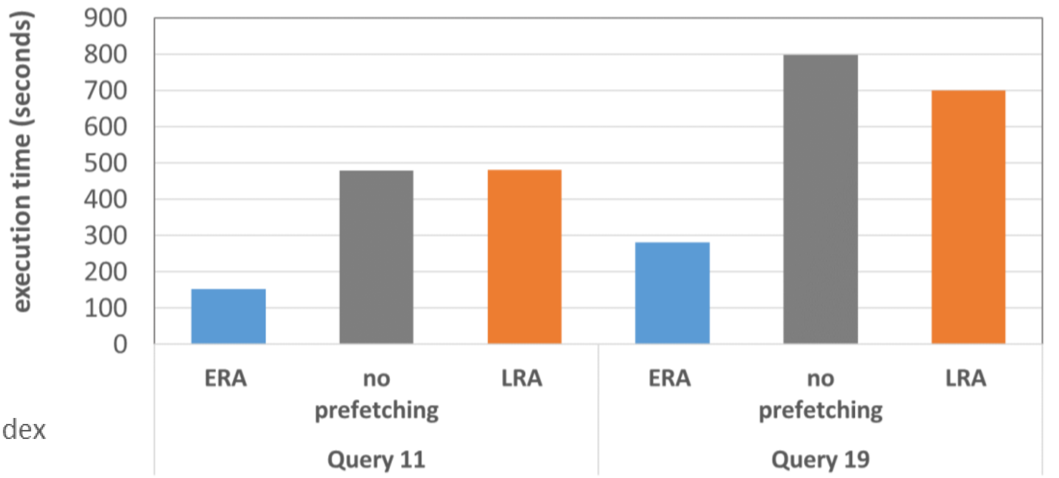
Evaluation

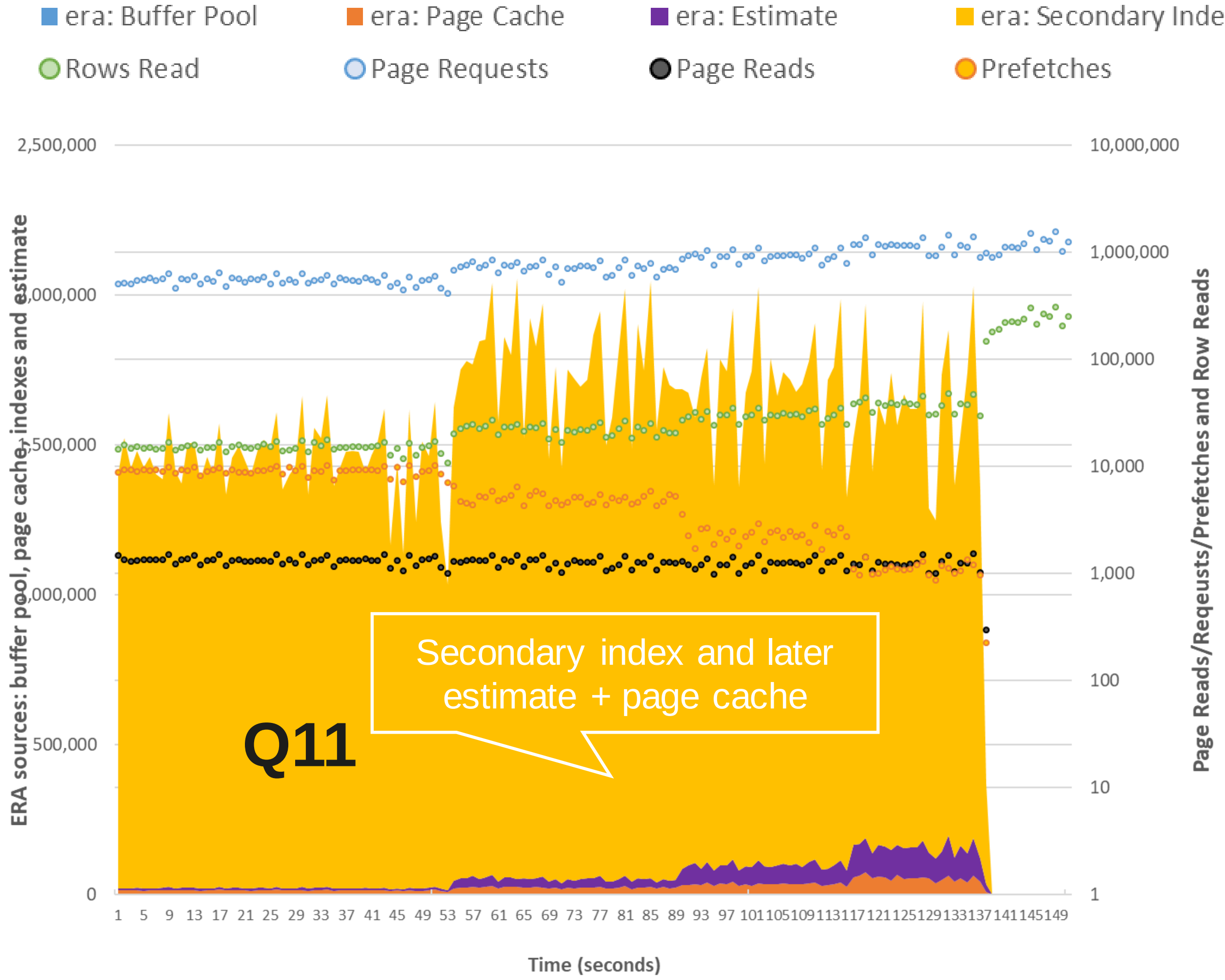

execution time (seconds)
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
ERA
no prefetching
LRA
ERA
no prefetching
LRA
Query 8
Query 10


🟦 era: Buffer Pool
🟧 era: Page Cache
🟪 era: Estimate
🟨 era: Secondary Index
🟩 era: Primary Index
○ Rows Read
○ Page Requests
● Page Reads
○ Prefetches
Q8
ERA sources: buffer pool, page cache, indexes and estimate
350,000
300,000
250,000
200,000
150,000
100,000
50,000
0
1,000,000
100,000
10,000
1,000
100
10
1
Page Reads/Reqeusts/Prefetches and Row Reads
1
27
53
79
105
131
157
183
209
235
261
287
313
339
365
391
417
443
469
495
521
547
573
599
625
651
677
703
729
755
781
807
833
859
885
911
937
963
989
1015
1041
1067
1093
1119
1145
1171
1197
Time (seconds)


Q10
ERA sources: buffer pool, page cache, indexes and estimate
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000
0
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
Page Reads/Reqeusts/Prefetches and Row Reads
1
14
27
40
53
66
79
92
105
118
131
144
157
170
183
196
209
222
235
248
261
274
287
300
313
326
339
352
365
378
391
404
417
430
443
456
469
482
495
508
521
534
547
560
573
586
Time (seconds)
Primary index and more estimate on second phase

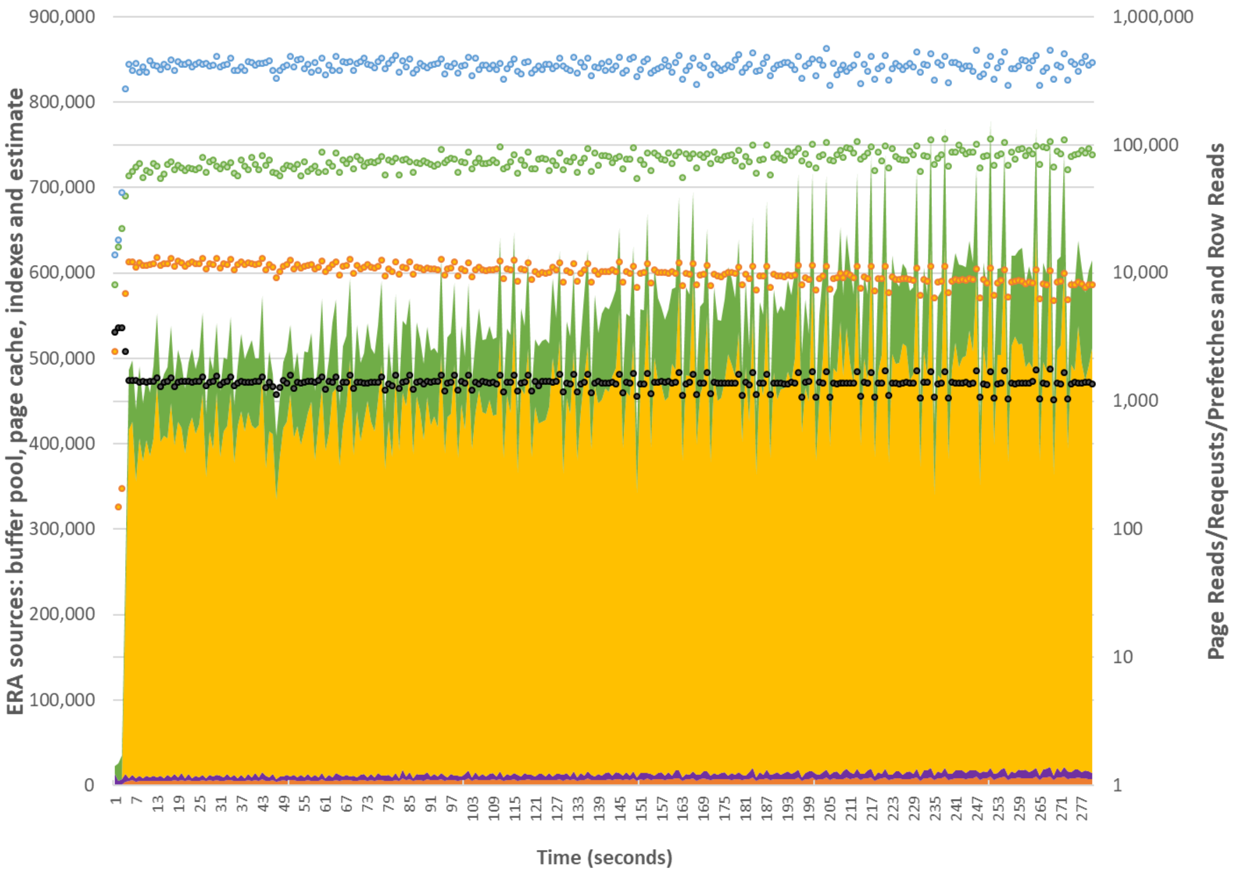
Evaluation



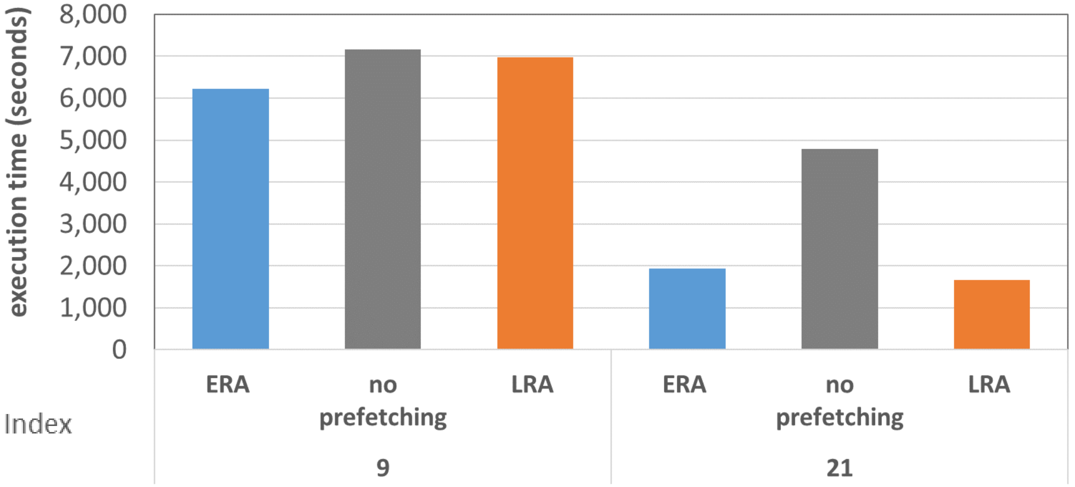


Secondary index and later estimate + page cache

Q11

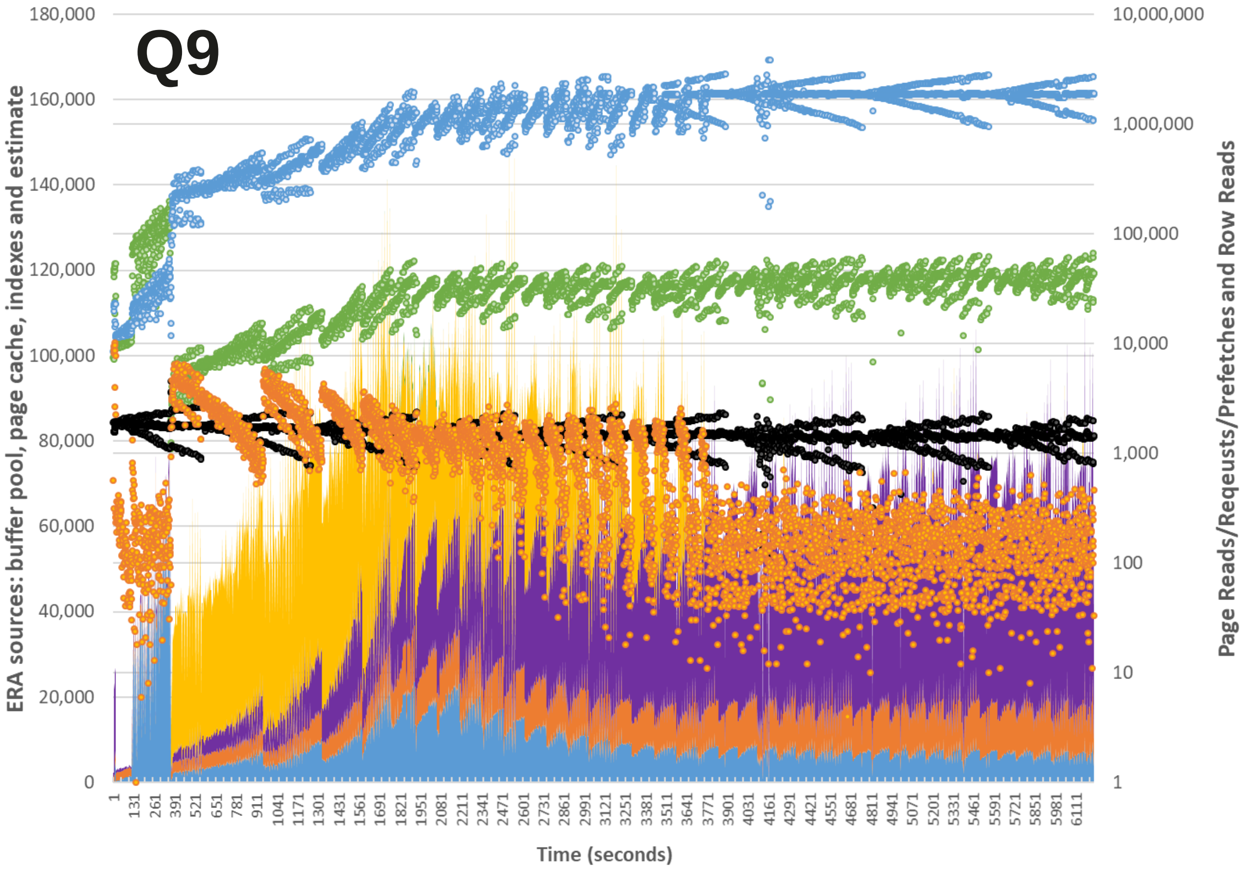


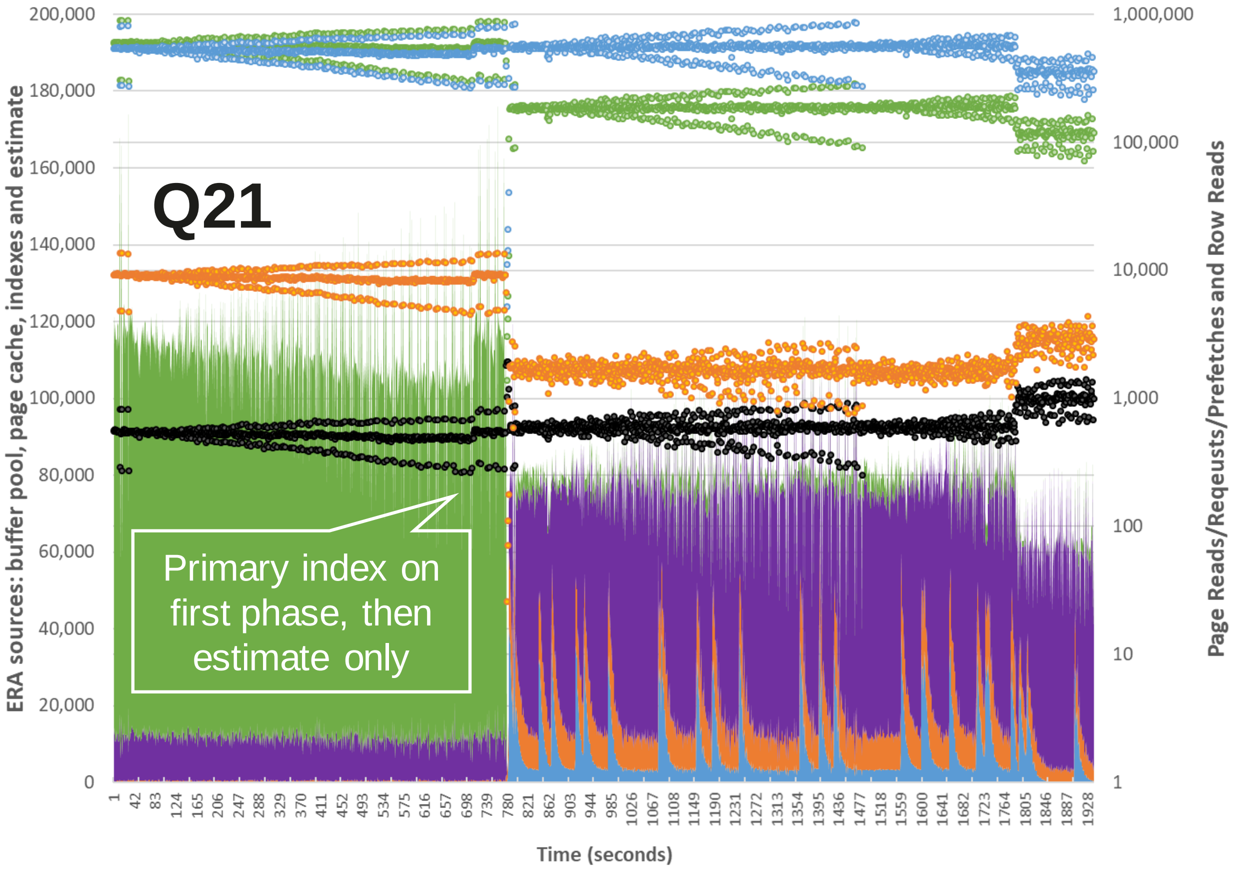
Evaluation



🟦 era: Buffer Pool 🟧 era: Page Cache 🟪 era: Estimate 🟨 era: Secondary Index 🟩 era: Primary Index

● Rows Read ○ Page Requests ● Page Reads ● Prefetches





Primary index on first phase, then estimate only

Conclusion

1 Prefetching is an important tool to hide storage latency and it is well fit to the sequential nature of many large analytics queries.

2 The enhanced read-ahead mechanism proposes a mechanism that is more robust than InnoDB's linear read head, applies to more queries and in a larger scale;

3 But that comes at the cost of being complex and computationally demanding, the overhead only makes sense if large databases are used.

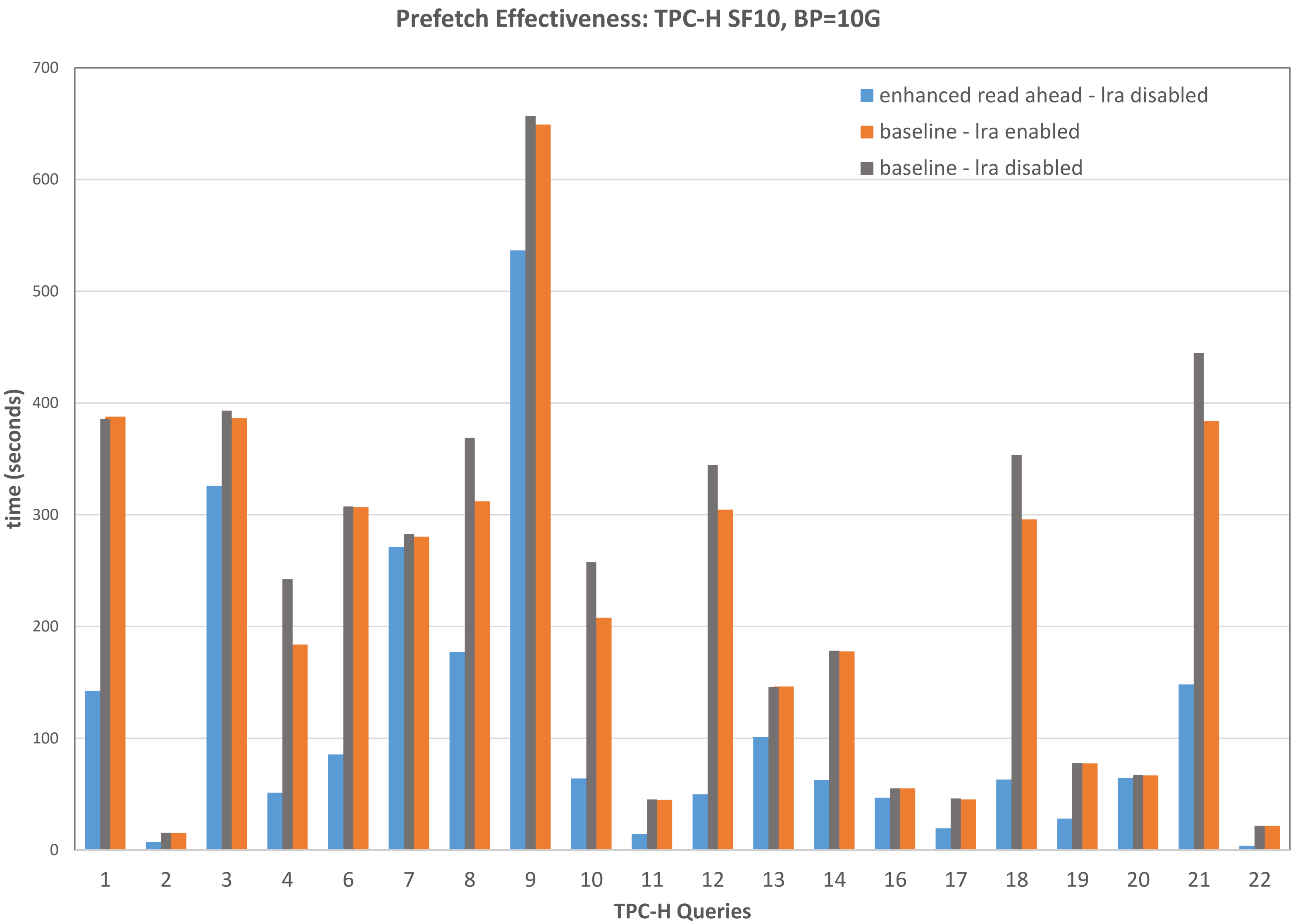
Thank you.

Bring digital to every person, home and organization for a fully connected, intelligent world.

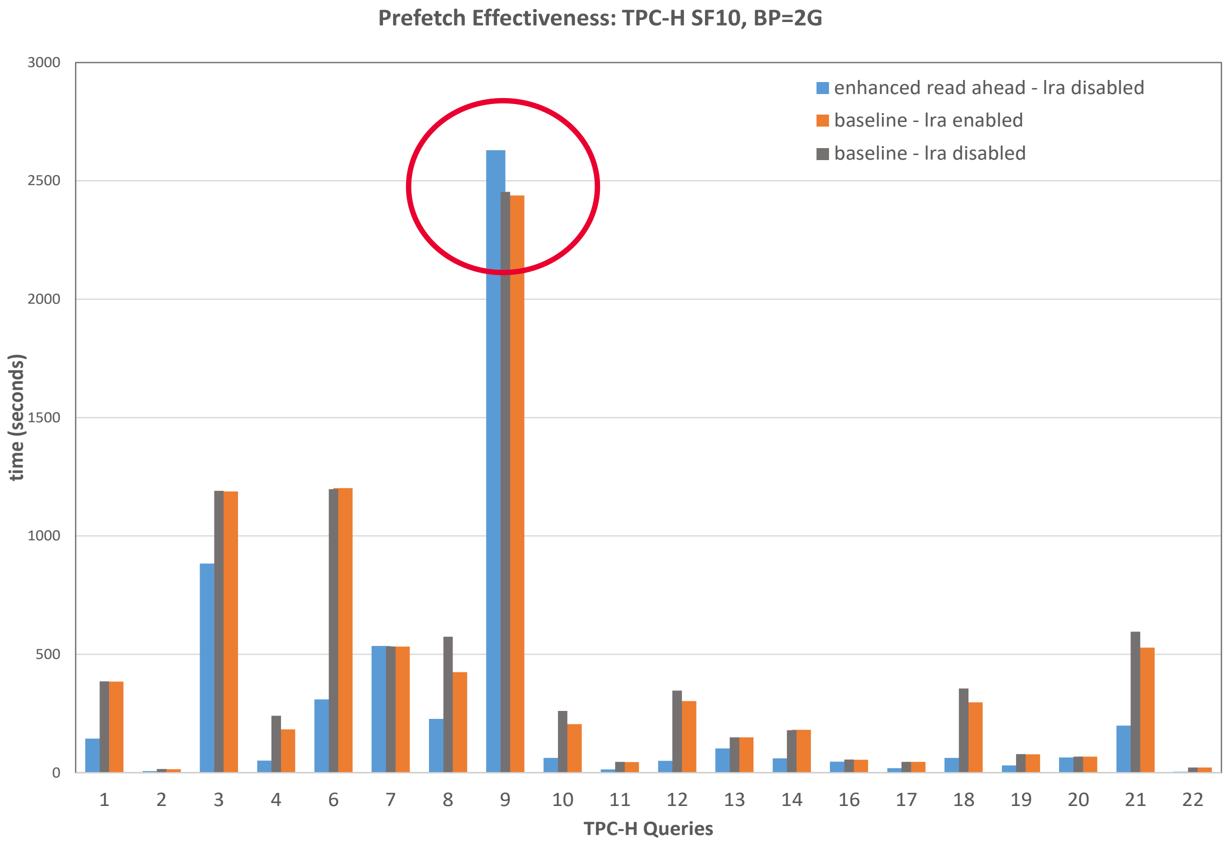
The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.

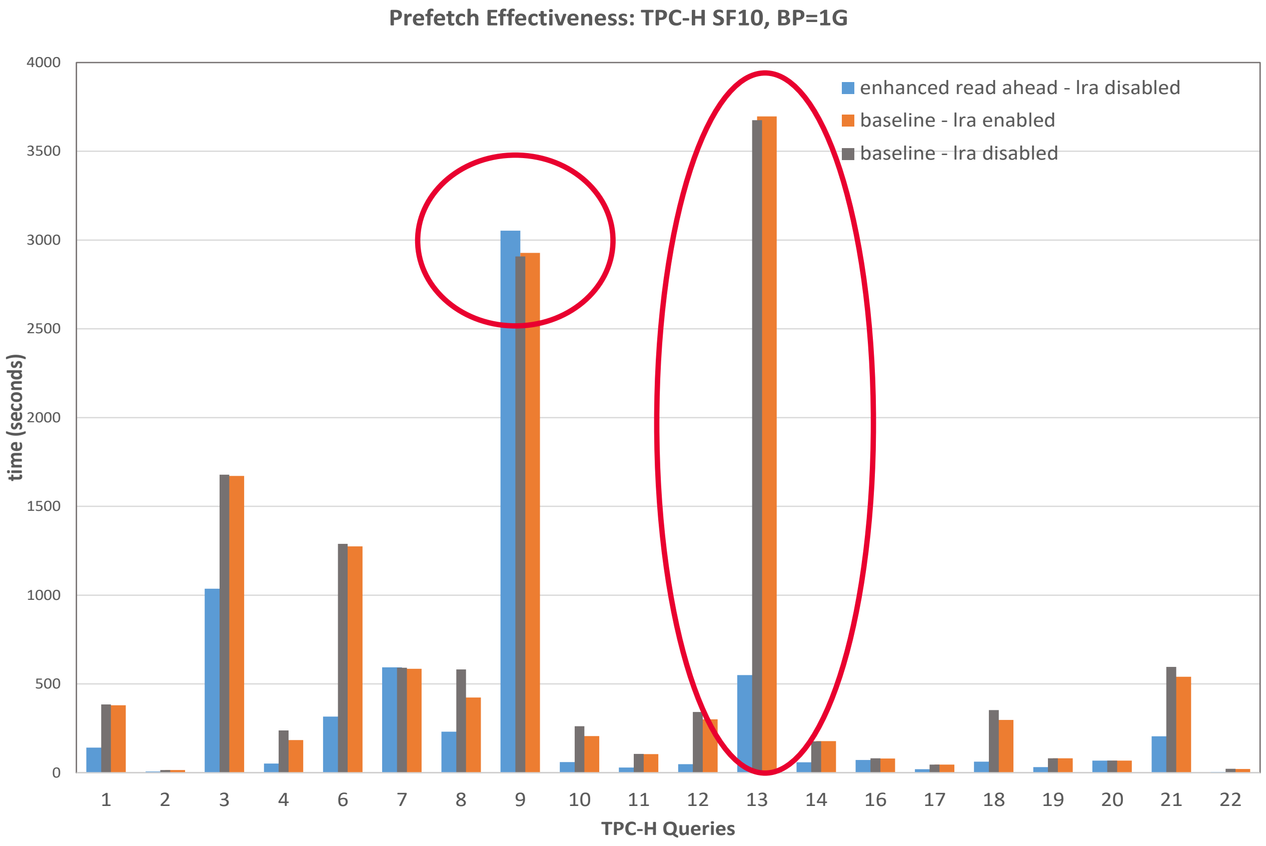
SF10, 10G

- Overall the existing linear read-ahead (LRA) saves 8% of the time compared to running without prefetcher;
- The enhanced read-ahead (ERA) saves 52% compared to no prefetching, and 48% compared to linear read-ahead.

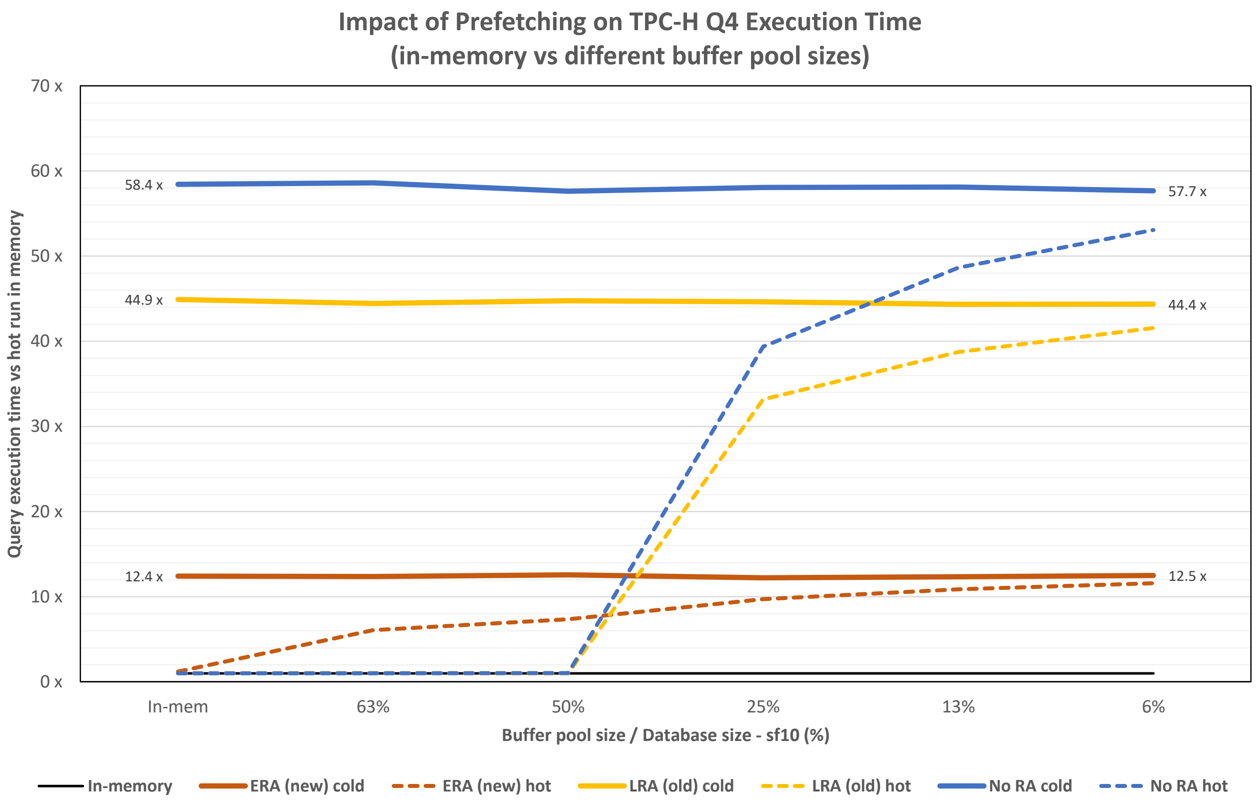


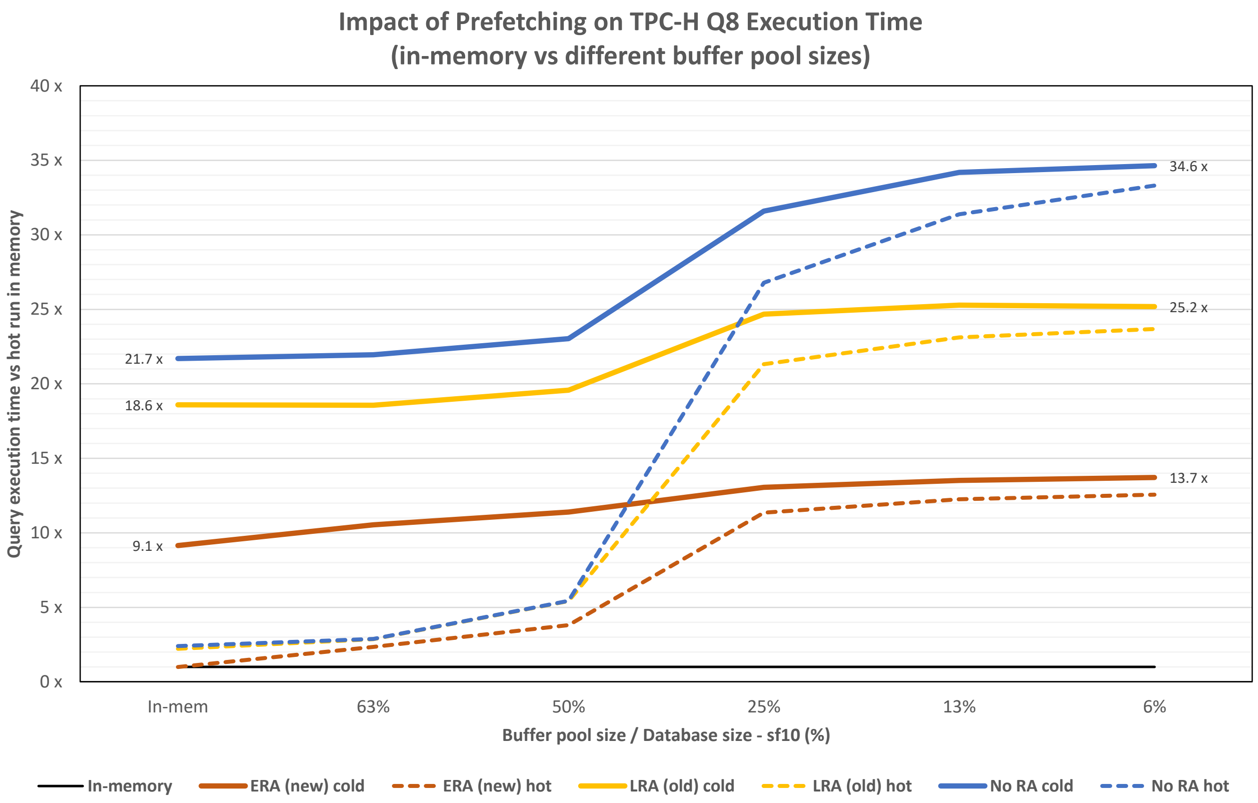
SF10, 2GB and 1GB



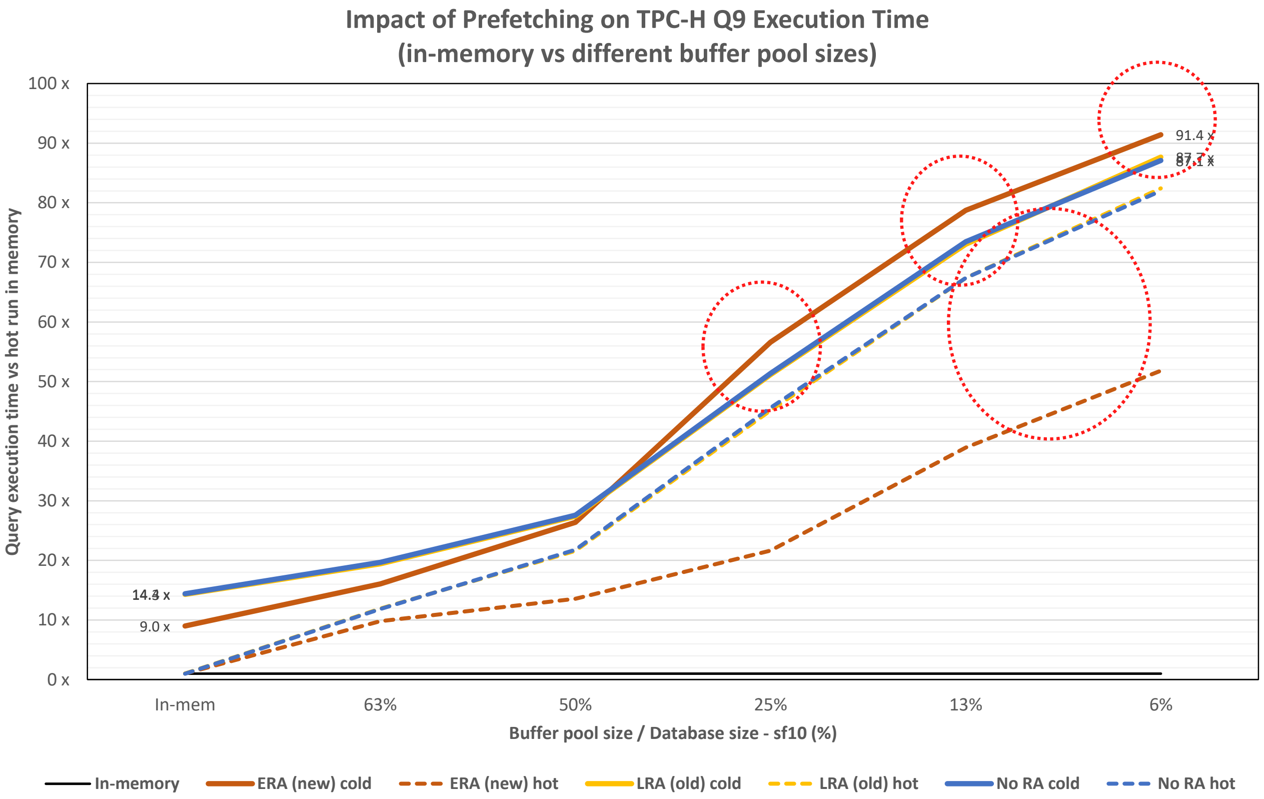


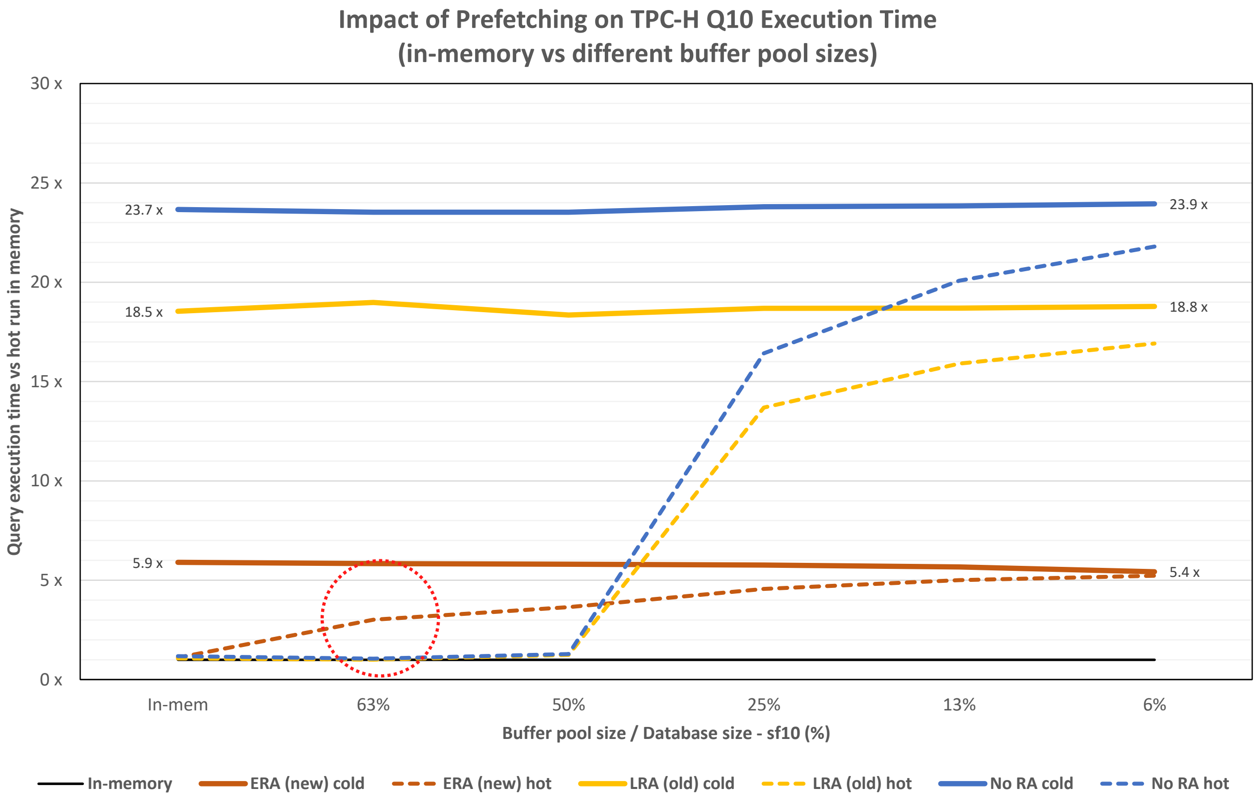
SF10


Impact of Prefetching on TPC-H Q4 Execution Time
(in-memory vs different buffer pool sizes)
70 x
60 x
50 x
40 x
30 x
20 x
10 x
0 x
Query execution time vs hot run in memory
58.4 x
57.7 x
44.9 x
44.4 x
12.4 x
12.5 x
In-mem
63%
50%
25%
13%
6%
Buffer pool size / Database size - sf10 (%)
In-memory
ERA (new) cold
ERA (new) hot
LRA (old) cold
LRA (old) hot
No RA cold
No RA hot


Impact of Prefetching on TPC-H Q8 Execution Time
(in-memory vs different buffer pool sizes)
40 x
35 x
30 x
25 x
20 x
15 x
10 x
5 x
0 x
Query execution time vs hot run in memory
21.7 x
34.6 x
18.6 x
25.2 x
9.1 x
13.7 x
In-mem
63%
50%
25%
13%
6%
Buffer pool size / Database size - sf10 (%)
In-memory
ERA (new) cold
ERA (new) hot
LRA (old) cold
LRA (old) hot
No RA cold
No RA hot

Not so good


Impact of Prefetching on TPC-H Q9 Execution Time
(in-memory vs different buffer pool sizes)
Query execution time vs hot run in memory
100 x
90 x
80 x
70 x
60 x
50 x
40 x
30 x
20 x
10 x
0 x
91.4 x
87.7 x
87.1 x
14.3 x
14.4 x
9.0 x
In-mem
63%
50%
25%
13%
6%
Buffer pool size / Database size - sf10 (%)
In-memory
ERA (new) cold
ERA (new) hot
LRA (old) cold
LRA (old) hot
No RA cold
No RA hot


Impact of Prefetching on TPC-H Q10 Execution Time
(in-memory vs different buffer pool sizes)
Query execution time vs hot run in memory
30 x
25 x
20 x
15 x
10 x
5 x
0 x
23.7 x
23.9 x
18.5 x
18.8 x
5.9 x
5.4 x
In-mem
63%
50%
25%
13%
6%
Buffer pool size / Database size - sf10 (%)
In-memory
ERA (new) cold
ERA (new) hot
LRA (old) cold
LRA (old) hot
No RA cold
No RA hot