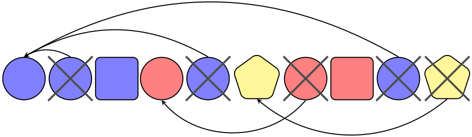
Lazy Exact Deduplication

Jingwei Ma, **Rebecca J. Stones**, Yuxiang Ma,
Jingui Wang, Junjie Ren, Gang Wang, Xiaoguang Liu

College of Computer and Control Engineering,
Nankai University, China.

5 May 2016



Lazy exact deduplication

Lazy exact deduplication

Lead author: Jingwei Ma, PhD student at Nankai University (supervisor: Prof. Gang Wang).

Lazy exact deduplication

Lead author: Jingwei Ma, PhD student at Nankai University (supervisor: Prof. Gang Wang). Couldn't get USA visa in time $\Longrightarrow$ I will present this work.

Lazy exact deduplication

Lead author: Jingwei Ma, PhD student at Nankai University (supervisor: Prof. Gang Wang). Couldn't get USA visa in time $\Longrightarrow$ I will present this work.

Credit where credit is due: Jingwei Ma did the lion's share of this work (development, implementation, experimentation, etc.).

Lazy exact deduplication

Lead author: Jingwei Ma, PhD student at Nankai University (supervisor: Prof. Gang Wang). Couldn't get USA visa in time $\Longrightarrow$ I will present this work.

Credit where credit is due: Jingwei Ma did the lion's share of this work (development, implementation, experimentation, etc.).

Lazy deduplication: 'Lazy' in the sense that we postpone disk lookups, until we can do them as a batch.

Lazy exact deduplication

Lead author: Jingwei Ma, PhD student at Nankai University (supervisor: Prof. Gang Wang). Couldn't get USA visa in time ⟹ I will present this work.

Credit where credit is due: Jingwei Ma did the lion's share of this work (development, implementation, experimentation, etc.).

Lazy deduplication: 'Lazy' in the sense that we postpone disk lookups, until we can do them as a batch. (Lazy is exact.)

Deduplication: What usually happens...

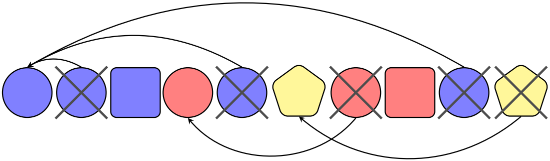
- We have a large amount of data, with lots of duplicate data (e.g. weekly backups).

Deduplication: What usually happens...

- We have a large amount of data, with lots of duplicate data (e.g. weekly backups).
- We read through the data, and if we see something we've seen before, we replace it with an index entry (saving disk space).

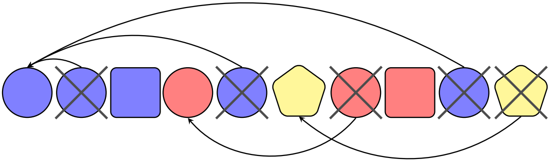
Deduplication: What usually happens...

- We have a large amount of data, with lots of duplicate data (e.g. weekly backups).
- We read through the data, and if we see something we've seen before, we replace it with an index entry (saving disk space).



Deduplication: What usually happens...

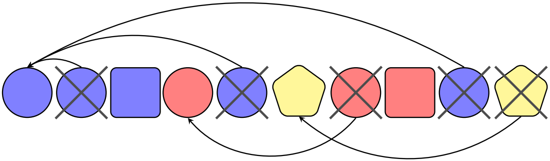
- We have a large amount of data, with lots of duplicate data (e.g. weekly backups).
- We read through the data, and if we see something we've seen before, we replace it with an index entry (saving disk space).



- The data is broken up into *chunks* (Rabin Hash).

Deduplication: What usually happens...

- We have a large amount of data, with lots of duplicate data (e.g. weekly backups).
- We read through the data, and if we see something we've seen before, we replace it with an index entry (saving disk space).



- The data is broken up into *chunks* (Rabin Hash).
- The chunks are *fingerprinted* (SHA1): same fingerprint $\Longrightarrow$ duplicate chunk.

Deduplication: What usually happens...

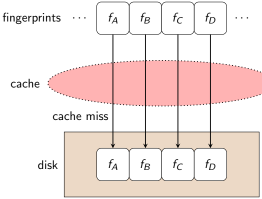
- *Disk bottleneck*: Most fingerprints are stored on disk $\implies$ lots of disk reads ("have I seen this before?") $\implies$ slow.

Deduplication: What usually happens...

- *Disk bottleneck*: Most fingerprints are stored on disk $\Longrightarrow$ lots of disk reads ("have I seen this before?") $\Longrightarrow$ slow.
- *Caching* and *prefetching* reduce the disk bottleneck problem:

Deduplication: What usually happens...

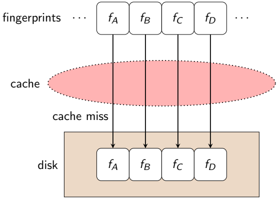
- *Disk bottleneck*: Most fingerprints are stored on disk $\Longrightarrow$ lots of disk reads ("have I seen this before?") $\Longrightarrow$ slow.
- *Caching* and *prefetching* reduce the disk bottleneck problem:



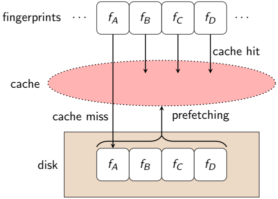
The first time we see fingerprints f_A, f_B, ...

Deduplication: What usually happens...

- *Disk bottleneck*: Most fingerprints are stored on disk $\Longrightarrow$ lots of disk reads ("have I seen this before?") $\Longrightarrow$ slow.
- *Caching* and *prefetching* reduce the disk bottleneck problem:

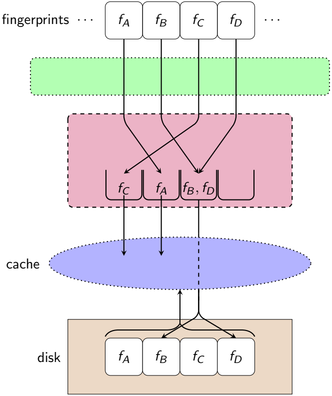


The first time we see fingerprints f_A, f_B, ...



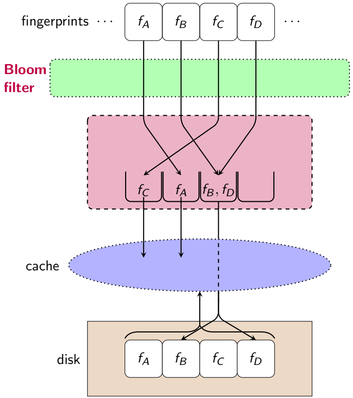
The second time we see fingerprints f_A, f_B, ...

Lazy deduplication...



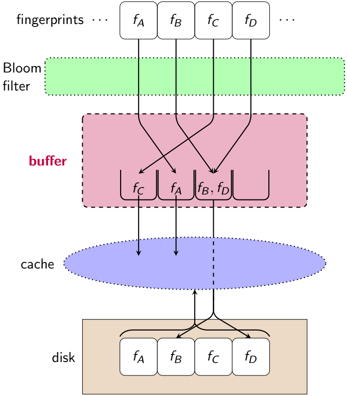
Lazy deduplication...

- Bloom filter: identifies many uniques (not all). [Commonly used.]



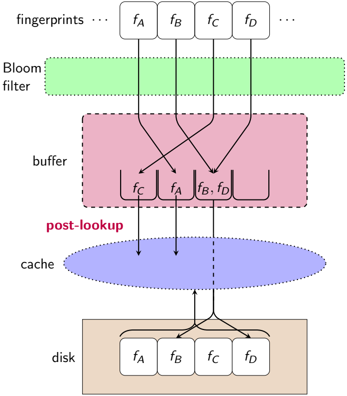
Lazy deduplication...

- Bloom filter: identifies many uniques (not all). [Commonly used.]
- buffer: stores fingerprints in hash buckets; searched later on disk ("lazy")—when full, whole buckets are searched in one go (stored on-disk in hash buckets)



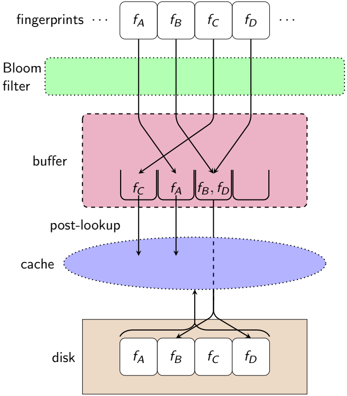
Lazy deduplication...

- Bloom filter: identifies many uniques (not all). [Commonly used.]
- buffer: stores fingerprints in hash buckets; searched later on disk ("lazy")—when full, whole buckets are searched in one go (stored on-disk in hash buckets)
- post-lookup: searching the cache after buffering (maybe multiple times)



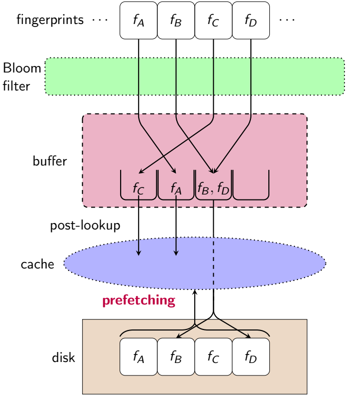
Lazy deduplication...

- Bloom filter: identifies many uniques (not all). [Commonly used.]
- buffer: stores fingerprints in hash buckets; searched later on disk ("lazy")—when full, whole buckets are searched in one go (stored on-disk in hash buckets)
- post-lookup: searching the cache after buffering (maybe multiple times)
- pre-lookup: searching the cache before buffering [not shown]



Lazy deduplication...

- Bloom filter: identifies many uniques (not all). [Commonly used.]
- buffer: stores fingerprints in hash buckets; searched later on disk ("lazy")—when full, whole buckets are searched in one go (stored on-disk in hash buckets)
- post-lookup: searching the cache after buffering (maybe multiple times)
- pre-lookup: searching the cache before buffering [not shown]
- prefetching: bidirectional; triggers post-lookup



Prefetching...

- Ordinarily, we prefetch the subsequent on-disk fingerprints after a duplicate is found on disk

Prefetching...

- Ordinarily, we prefetch the subsequent on-disk fingerprints after a duplicate is found on disk—these will probably be the next incoming fingerprints.

Prefetching...

- Ordinarily, we prefetch the subsequent on-disk fingerprints after a duplicate is found on disk—these will probably be the next incoming fingerprints. But this doesn't work with the lazy method (where fingerprints are buffered).

Prefetching...

- Ordinarily, we prefetch the subsequent on-disk fingerprints after a duplicate is found on disk—these will probably be the next incoming fingerprints. But this doesn't work with the lazy method (where fingerprints are buffered).
- To overcome this obstacle, each buffered fingerprint is given a...

Prefetching...

- Ordinarily, we prefetch the subsequent on-disk fingerprints after a duplicate is found on disk—these will probably be the next incoming fingerprints. But this doesn't work with the lazy method (where fingerprints are buffered).
- To overcome this obstacle, each buffered fingerprint is given a...
  - *rank*, used to determine the on-disk search range;

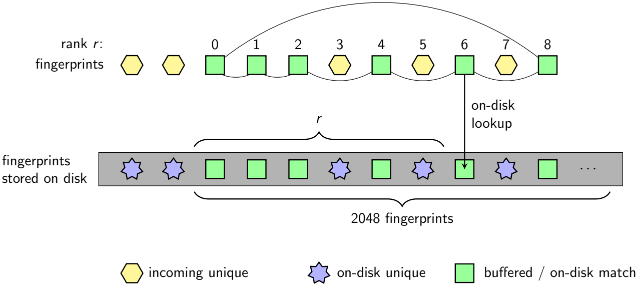
Prefetching...

- Ordinarily, we prefetch the subsequent on-disk fingerprints after a duplicate is found on disk—these will probably be the next incoming fingerprints. But this doesn't work with the lazy method (where fingerprints are buffered).
- To overcome this obstacle, each buffered fingerprint is given a...
    - *rank*, used to determine the on-disk search range; and a
    - *buffer cycle*, indicating where duplicates might be on-disk.

Prefetching...

- Ordinarily, we prefetch the subsequent on-disk fingerprints after a duplicate is found on disk—these will probably be the next incoming fingerprints. But this doesn't work with the lazy method (where fingerprints are buffered).
- To overcome this obstacle, each buffered fingerprint is given a...
  - *rank*, used to determine the on-disk search range; and a
  - *buffer cycle*, indicating where duplicates might be on-disk.

It looks like this:



Experimental results...

(See our paper for the details and further experiments.)

Experimental results...

(See our paper for the details and further experiments.)

The time it takes to deduplicate a dataset (on SSD):

	Vm (220GB)	Src (343GB)	FSLHomes (3.58TB)
eager way	282 sec.	476 sec.	5824 sec.
lazy way	151 sec.	226 sec.	3939 sec.

(eager = non-lazy [exact] way—i.e., no buffering before accessing the disk)

Conclusion: Lazy is faster.

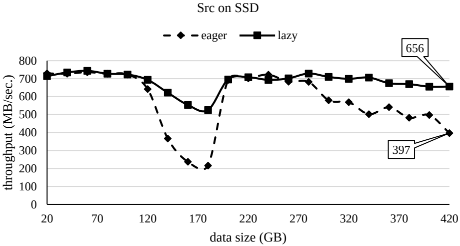
On-disk lookups...

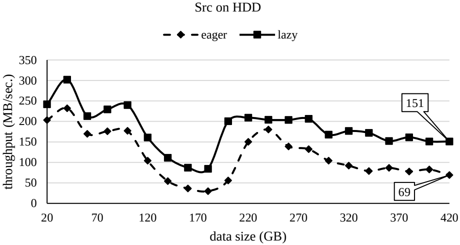
Disk access time (sec.) on SSD:

	Vm		Src		FSLHomes	
	eager	lazy	eager	lazy	eager	lazy
on-disk lookup	176	20	325	45	4598	1639
prefetching	46	60	52	68	298	655
other	59	71	99	113	928	1645
total disk access	222	80	377	113	4896	2294
total dedup.	282	151	476	226	5824	3939

Conclusion: Lazy reduces the disk bottleneck.

Throughput...





Conclusion: Lazy has better throughput on both SSD and HDD, but moreso on slower HDD.

рахмат
danke
謝謝
ngiyabonga
شكراً جزيلاً
Баярлалаа
спасибо
faafetai lava
kiitos
dankie
nandri
mersi
kia ora
barka
welalin
tack
teşekkür ederim
mahalo
tapadh leat
vinaka
спасибі
blagodaram
dank je
misaotra
matondo
paldies
thank you
gracias
хвала
asante
manana
bedankt
bayarlalaa
grazie
hvala
maururu
köszönöm
dankon
akun
enkosi
dziękuję
djiere dieuf
tau
mochchakkeram
noakoze
lenki
дякую
mamnun
chnorakaloutioun
gràcies
sulpay
go raibh maith agat
sobodi
dekuji
obrigado
sagolun
sukriya
kop khun krap
taiku
arigatō
takk
dakujem
trugarez
mèsi
didi madloba
ありがとう
najis tuke
terima kasih
lanemirt
rahmet
grazie
diolch
dhanyavadagalu
shukriya
merce
мерси
kam sah hamnida
rahmat
তোমাকে ধন্যবাদ
감사합니다
xiexie
ευχαριστώ
merci