Mismatched Memory Management of Android Smartphones

Yu Liang, Qiao Li, Chun Jason Xue

City University of Hong Kong

香港城市大學
City University of Hong Kong

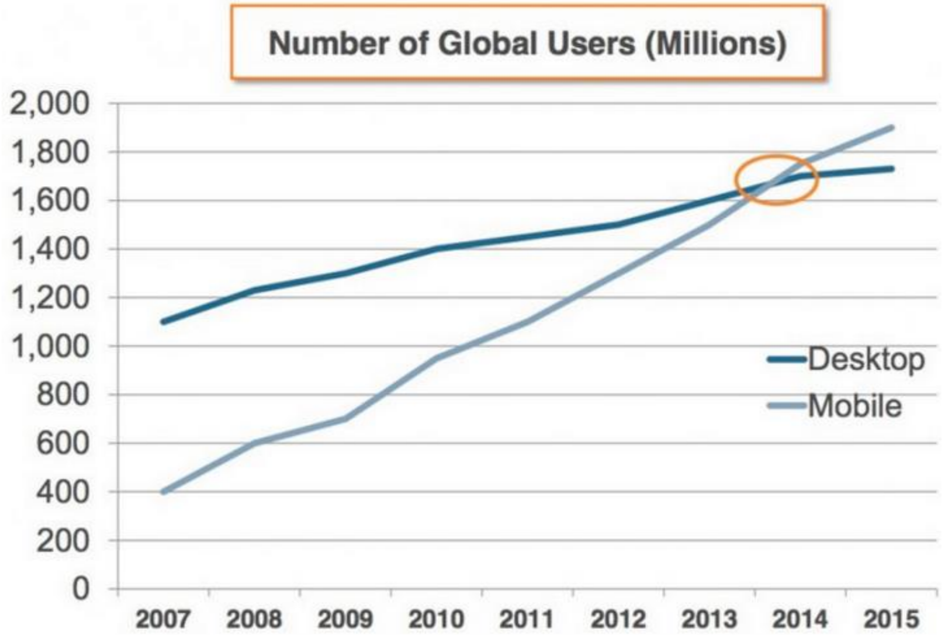
電腦科學系
Department of Computer Science

Outline

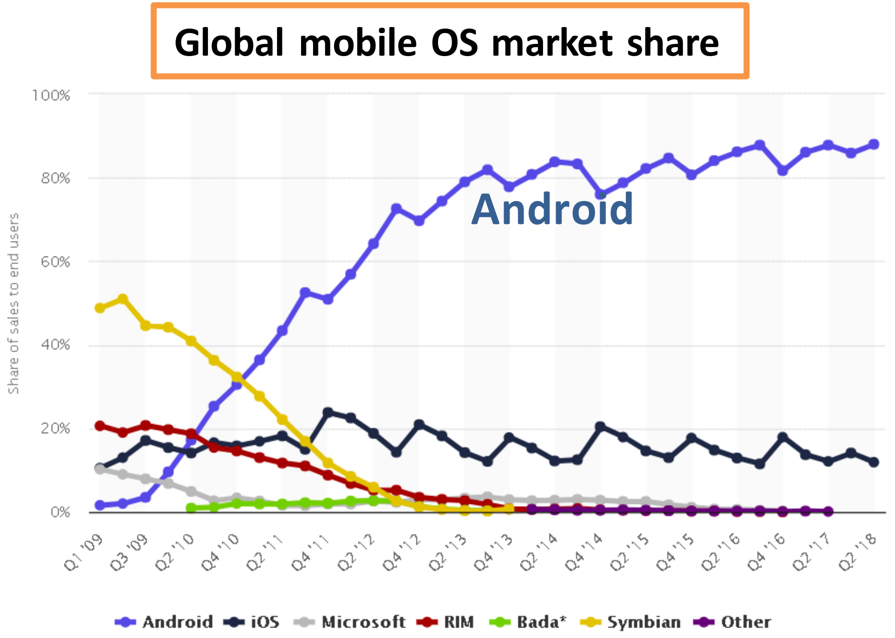
- Choose a metric
- Show the evaluation results
- Analyze the root causes
- Provide serval preliminary ideas
- Conclusion

Android mobile device is popular

Mobile devices are everywhere!



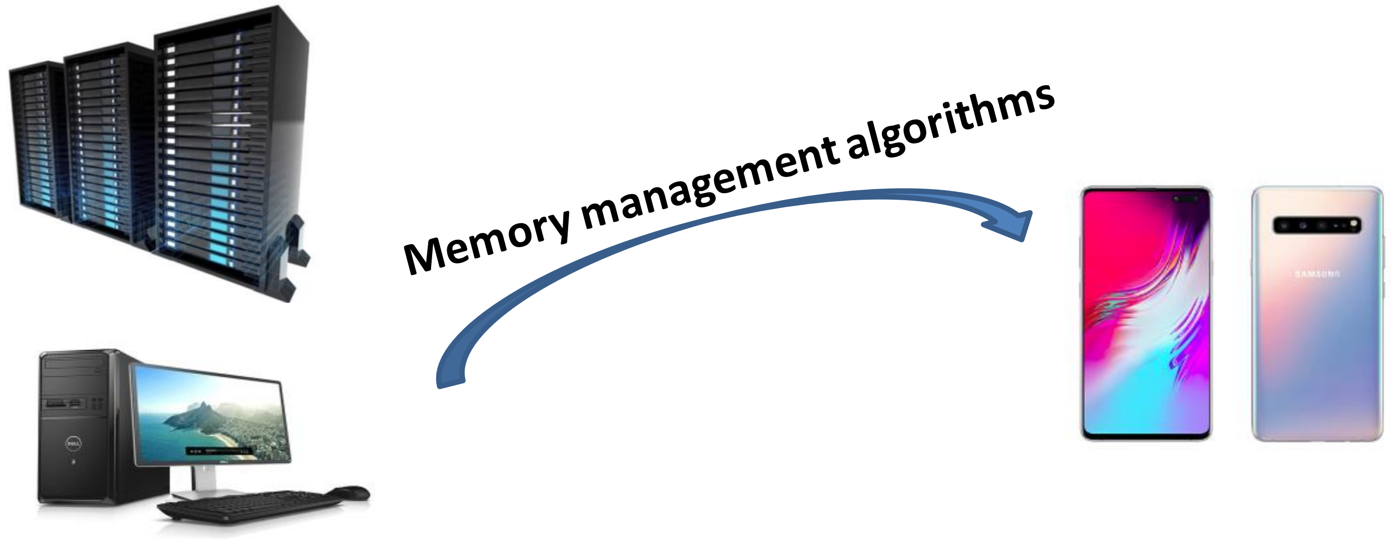
Industry report on mobile market [Morgan Stanley Research 2016]



Industry report on mobile market [© Statista 2018]

Android inherits Linux kernel

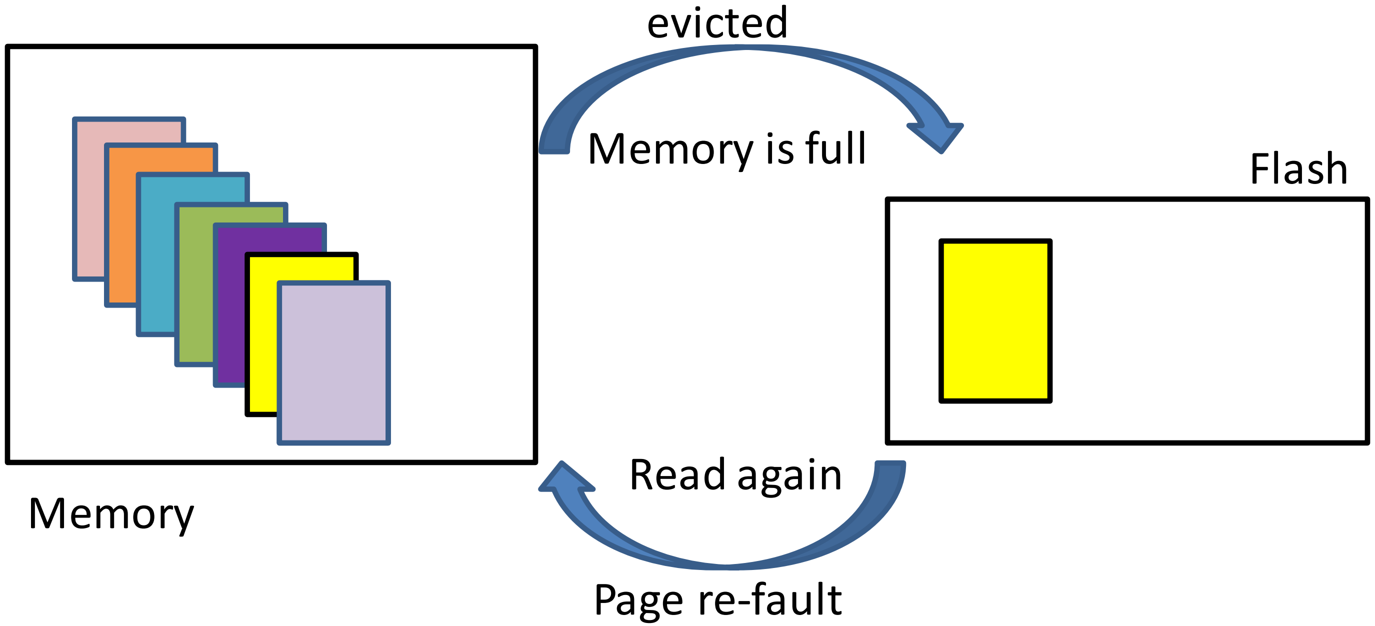
- Linux memory management algorithms are transplanted to Android smartphones



Are these algorithms working well on Android Smartphones?

Page re-fault is used to evaluate the efficiency of algorithms

- Page re-fault is defined as the case that the page fault happens on the previously evicted page.



Page re-fault is the chosen metric

- Page fault happens in three cases
  1. Reading a page for the first time
     - Physical memory is not allocated for this page.
  2. Reading a wrong address
     - This process will be killed.
  3. Reading an evicted page (page re-fault) ✓
     - This case could be avoided.

How often page re-fault happens on real Android smartphone?

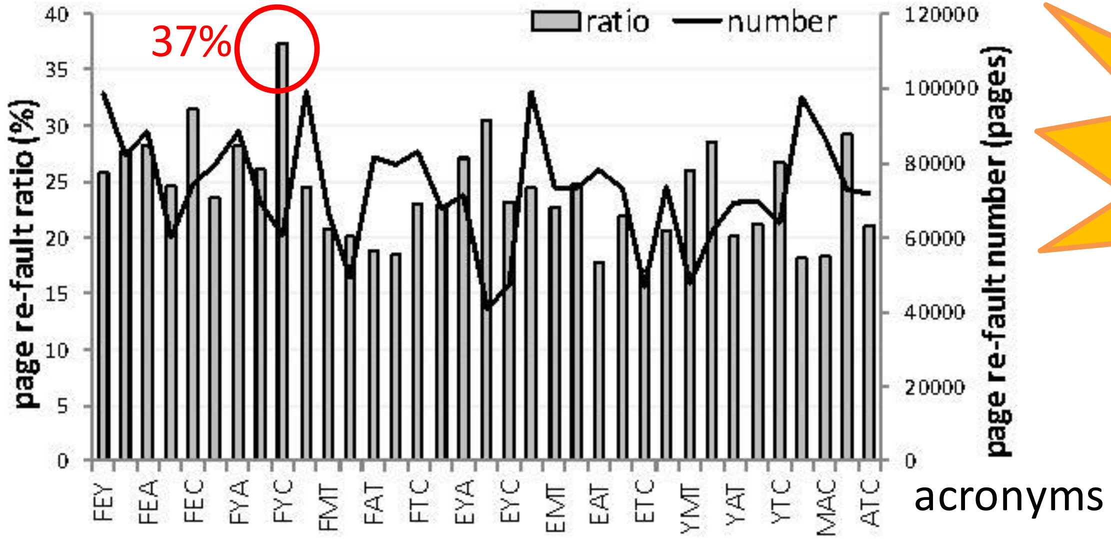
Page Re-fault on Android Smartphone

- Experimental setup:
  - Huawei P9 smartphone with ARM's Cortex-A72 CPU, 32GB internal memory and 3GB RAM, running Android 7.0 on Linux 4.1.18 kernel.
- Applications:
  - Facebook, Twitter, Chrome, Google Earth, Google Map, Angrybirds, YouTube

Applications	Memory	Workloads
Launching one app	empty	light
Using two apps	empty	light
Launching five apps and using two apps	full	moderate
Launching ten apps	full	moderate
Launching twenty apps and using three apps	full	heavy

Page re-fault is <u>severe</u> on Android smartphones

- For severity: after launching twenty and only using three apps (full and heavy)



Wow! That is pretty high!

  - Facebook, Earth, YouTube, Map, Twitter, Chrome, Angrybirds

Memory management algorithms often reclaim the pages, which will be used in the near future.

Previous works

- Prior researches have focused on reducing the number of page faults by optimizing eviction algorithms [1] [2].
- Eviction algorithms decide how to select the victim pages which will be evicted out of memory.
- The optimized LRU is known as a good eviction algorithm and is applied to Android.
- However, the experimental results show that page re-fault ratio is still high on Android smartphones from daily usage.

What is the root cause of high page re-fault ratio?

Uncovered by this paper

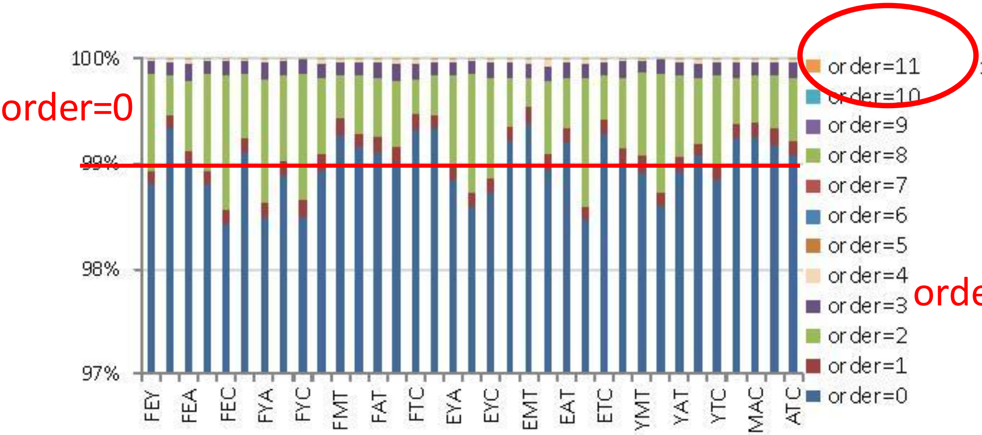
- The causes of high page re-fault ratio on smartphones

1. The reclaim size is often too large for the requests of smartphones;
   - Reclaim size represents the number of pages freed by each reclaim operation.

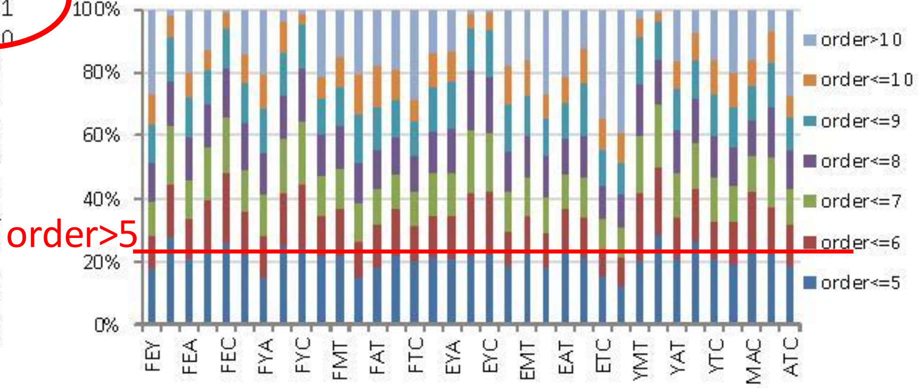
2. The limited reclaim scope aggravates the punishment.
   - Reclaim scope represents the region of pages freed by each reclaim operation, such as the pages in the *inactive_file_lru* list.

Cause1: Reclaiming size is often too large

The distribution of allocation size:



The distribution of reclaim sizes :

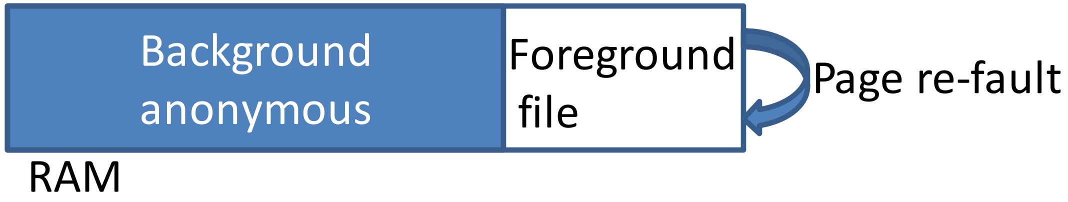


- 99% of allocation sizes are 1 page (order=0)
- 80% of reclaiming sizes are larger than 32 pages (order=5).
- **The requests on Android smartphones are usually small (more than 60% requests <16 KB).**

Compared to allocation size, reclaim size is often too large.

Cause2: Limited reclaim scope

- Current reclaim scope are not fully considering foreground background differentiation
- The page frames are maintained with four lists: *inactive_anonymous_list, active_anonymous_list,* *inactive_file_list, active_file_list.*



The limited reclaim scope aggravates the punishment.

Idea 1: reduce reclaiming size

- A tradeoff between reclaim size and overall performance:
  - Reclaim size is too small, free pages will be consumed quickly, the heavy-weight direct reclaiming will be triggered, which degrades the performance.
  - Reclaim size is too large, the ratio of page re-fault will be high and thus degrades the performance.
- To exploit the tradeoff to find the optimal reclaim size for Android smartphones.

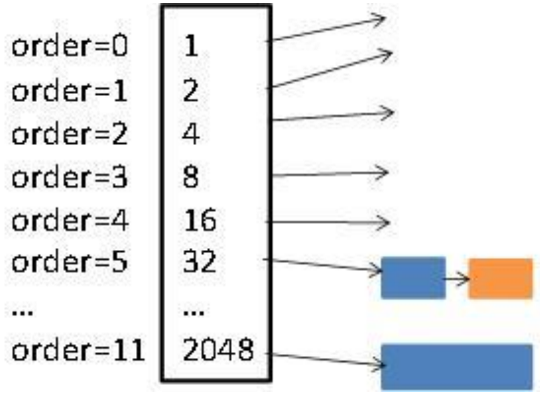
Idea 2: reclaiming considers status of app

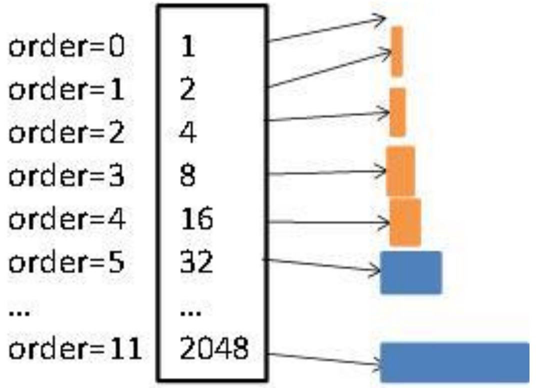
- For mobile devices, background and foreground status should be considered in the priority decision of reclaiming.

- For example, the reclaiming procedure should evict some *active_anonymous* pages of background processes before the *active_file* pages of foreground processes.

Idea 3: Reduce the maximum order

- Too large allocation size will degrade the efficiency of allocation operations.





Allocation procedure of buddy system for 2 free pages

Conclusion

- Page re-fault is unexpectedly high (up to 37%) on Android smartphones when running popular apps;
- Priority flip between background apps and foreground apps;
- The maximum allocation size of buddy system is often too large for the requests of apps running on Android smartphones.

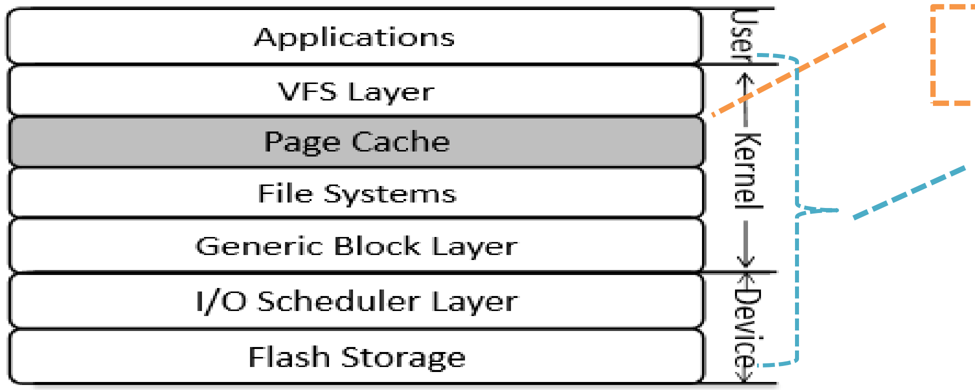
open discussion

Thank you!

Welcome discussion !

Page fault is the root cause of the long read latency

- Overview of Android I/O Stack:



Cache hit: read from the page cache

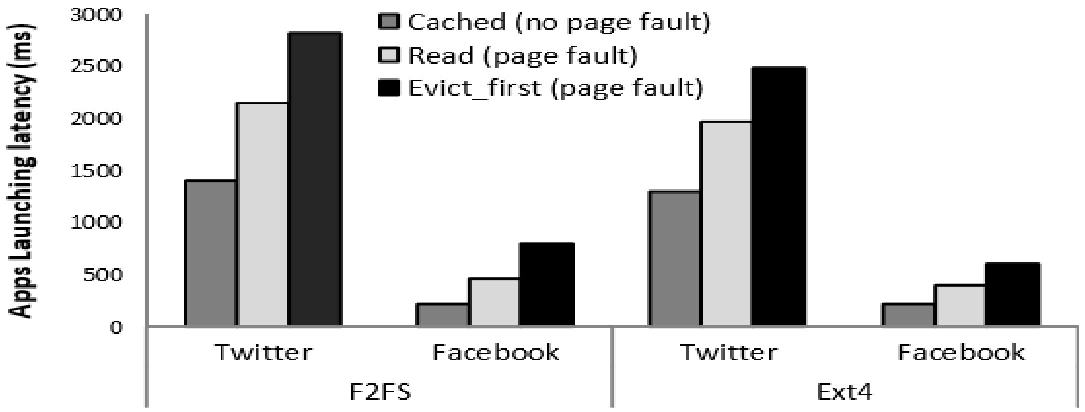
Page fault : go through all of these layers

- Handling page fault is on microsecond scale;
- Handling cache hit is on nanoseconds level.

Page fault is the read latency bottleneck of Android smartphones.

Page fault is the root cause of the long read latency

- The influence of page fault on Android smartphones

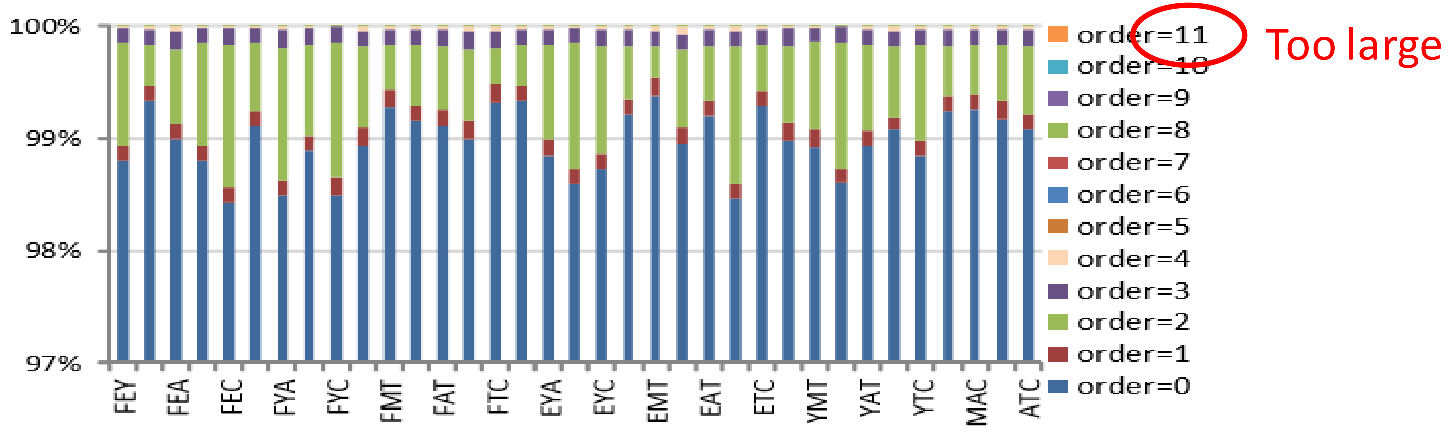


- “Read” and “Evict_first” cases cause longer launching latency
- This additional latency is mainly caused by page fault.

Page fault prolongs the latency of app launching.

Cause1: Reclaiming size is often too large

- The distribution of allocation size:



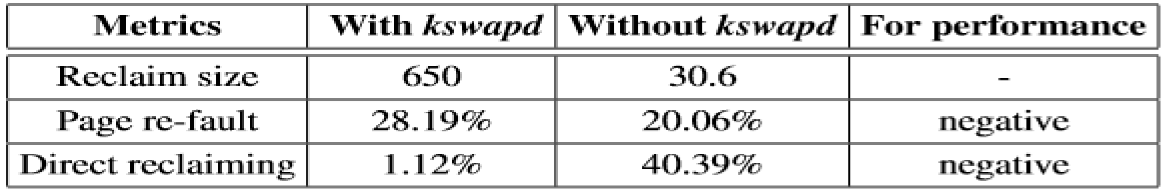
- The corresponding allocation size equals to 2^{order}
- 99% of allocation orders are 0 (1 page), and more than 99.9% of orders are smaller than 4 (16 pages).

The size of requests on Android smartphones is usually small.

A preliminary idea is proposed to improve performance

- There is a tradeoff between reclaim size and overall performance:
  - Reclaim size is too small, free pages will be consumed quickly, the heavy-weight direct reclaiming will be triggered, which degrades the performance.
  - Reclaim size is too large, the ratio of page re-fault will be high and thus degrades the performance.

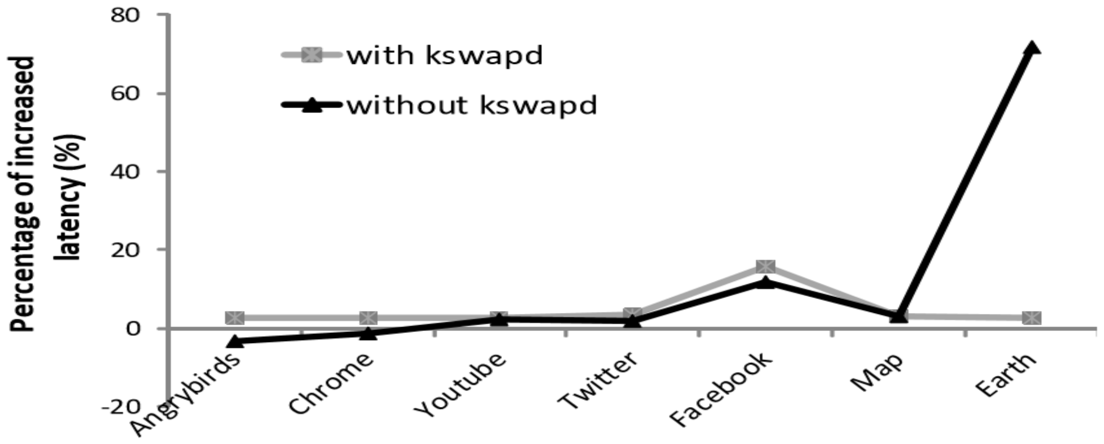
- Influence of *kswapd* on performance.



Metrics	With *kswapd*	Without *kswapd*	For performance
Reclaim size	650	30.6	-
Page re-fault	28.19%	20.06%	negative
Direct reclaiming	1.12%	40.39%	negative

Tradeoff between reclaiming size and overall performance

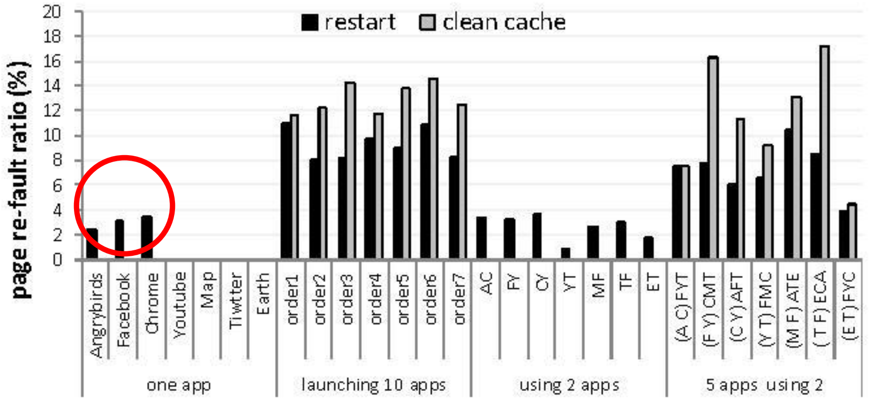
- The latency of launching seven apps are tracked when *kswapd* is turned on or turned off.



To exploit the tradeoff to find the optimal reclaiming size.

Page re-fault is prevalent on Android smartphones

- For universality: empty and light/moderate workloads

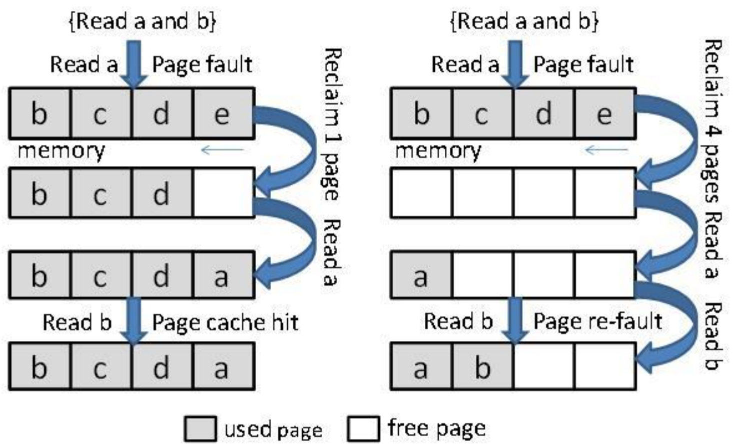


- In “launching 10 apps” case, 10 apps are launched in different orders.
- In “5 apps using 2” case, (AC) FYT represents launching Angrybirds, Chrome, Facebook, Youtube, and Twitter, but only using Angrybirds and Chrome.

- This is because some data will be pre-loaded into memory after restart

Large-size reclaiming scheme induces more page re-faults

- An example of this case:



- Reclaiming size=1, page b will be cache hit.
- Reclaiming size=4, page b will be re-fault.
- The large-size reclaiming scheme could induce more page re-faults.