Caching in Mobile Apps

COS 316: Principles of Computer System Design
Lecture 10
Outline

• Use of caches
• Libraries in mobile apps
• Cache performance in mobile apps
  • Reasons for suboptimal caching
  • Improving cache performance
• Caching more things
  • Computations?
What are caches useful for?
Reusing prior content to improve performance
What are caches useful for?

Reusing prior content to improve performance

- CPU memory
- Databases
- CDNs
- Browsers
- And even mobile apps
Interacting with mobile apps
Apps are similar to web browsers
Interacting with mobile apps

Apps are similar to web browsers

Interaction, e.g., tap(x, y)

GET /img.jpg

User

Origin Servers
Interacting with mobile apps

Apps are similar to web browsers

Interaction, e.g., tap(x, y)
Interacting with mobile apps
Apps are similar to web browsers

Interaction, e.g., tap(x, y)

GET /img.jpg

Similar to web browsing

DNS lookups ✓
HTTP messages ✓
TLS sessions ✓
TCP connection ✓
Client side caches for apps
Client side caches for apps

• Apps incorporate web caches
  • Similar to browsers
Client side caches for apps

• Apps incorporate web caches
  • Similar to browsers
Client side caches for apps

- Apps incorporate web caches
  - Similar to browsers
- Caching libraries
  - OkHttp
  - Ion
  - Picasso
  - .. and more
Caching libraries

- HTTP request/response handling
  - Generate appropriate request headers
  - Parse and handle response headers
- Cache consistency
- Caching algorithm
  - LRU, LFU, etc.
Caching libraries

• HTTP request/response handling
  • Generate appropriate request headers
  • Parse and handle response headers
• Cache consistency
• Caching algorithm
  • LRU, LFU, etc.

How do these caching libraries perform today?
State of cache performance today

Cache Hit Ratio during a series of interactions
State of cache performance today

Cache Hit Ratio during a series of interactions
State of cache performance today

Cache Hit Ratio during a series of interactions
State of cache performance today

Cache Hit Ratio during a series of interactions

CDF

Subsequent series of interactions

Cache hit rate
State of cache performance today

Cache Hit Ratio during a series of interactions

Subsequent series of interactions

Infinite capacity and perfect knowledge of object expirations
State of cache performance today

Cache Hit Ratio during a series of interactions

- Infinite capacity and perfect knowledge of object expirations
- Subsequent series of interactions

Over 2x

Existing Caching
Optimal Caching

Infinite capacity and perfect knowledge of object expirations
State of cache performance today

Cache Hit Ratio during a series of interactions

Current caching is suboptimal

Infinite capacity and perfect knowledge of object expirations

Subsequent series of interactions

Over 2x
App responsiveness
App responsiveness

How does cache hit rate affect responsiveness?

Subsequent series of interactions

Optimal version has infinite capacity, perfect expiration
App responsiveness

How does cache hit rate affect responsiveness?

App response time improvements over a no-cache scenario

Subsequent series of interactions
Optimal version has infinite capacity, perfect expiration
App responsiveness

How does cache hit rate affect responsiveness?

App response time improvements over a no-cache scenario

Subsequent series of interactions

Optimal version has infinite capacity, perfect expiration
Cache performance

Why isn’t cache hit rate optimal?

Interaction, e.g., tap(x, y)
Cache performance

HTTP Response Header for Cache Control

- Whether to cache
  - no store: no cache should store it
- Who should cache
  - private: only a private cache (e.g., browser)
  - public: any cache, including shared ones
- How long to cache
  - max-age=N: for N seconds
  - must-revalidate: check with the server (don’t return stale item)

Cache-Control: public, max-age=86400, must-revalidate
**Cache performance**

**HTTP Response Header for Cache Control**

- Whether to cache
  - no store: no cache should store it
- Who should cache
  - private: only a private cache (e.g., browser)
  - public: any cache, including shared ones
- How long to cache
  - `max-age`: N for N seconds
  - `must-revalidate`: check with the server (don’t return stale item)

**Cache-Control**: `public, max-age=86400, must-revalidate`
Cache performance

Why isn’t cache hit rate optimal?

User

Interaction, e.g., tap(x, y)

Origin Servers

TTL

TTL: Time to Live

Why isn’t cache hit rate optimal?
Cache performance

Why isn’t cache hit rate optimal?

User

Interaction, e.g., tap(x, y)

... 

Origin Servers

TTL: Time to Live

TTL

Why isn’t cache hit rate optimal?
Caching correctly is difficult
As TTLs are hard to choose
Caching correctly is difficult
As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes
Caching correctly is difficult
As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes

wsj.com homepage at 9am
Ideal TTL: 30 seconds

wsj.com homepage at 2am
Ideal TTL: 3 hours
Caching correctly is difficult
As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes

wsj.com homepage at 9am
Ideal TTL: 30 seconds

wsj.com homepage at 2am
Ideal TTL: 3 hours

Repeatedly fetch asset every second
Track when content changes
Caching correctly is difficult
As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes

wsj.com homepage at 9am
Ideal TTL: 30 seconds

wsj.com homepage at 2am
Ideal TTL: 3 hours

Repeatedly fetch asset every second
Track when content changes
Caching correctly is difficult
As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes

wsj.com homepage at 9am
Ideal TTL: 30 seconds

wsj.com homepage at 2am
Ideal TTL: 3 hours

Repeatedly fetch asset every second

Track when content changes

Ideal TTLs
{1, 2, ..}
Caching correctly is difficult
As TTLs are hard to choose

Developers need to set TTLs precisely such that they expire exactly when the content changes.

wsj.com homepage at 9am
Ideal TTL: 30 seconds

wsj.com homepage at 2am
Ideal TTL: 3 hours

Repeatedly fetch asset every second

Track when content changes

Ideal TTLs
{1, 2, ..}

Let’s plot the std. dev.
Caching correctly is difficult
Variations in ideal TTL
Caching correctly is difficult

Variations in ideal TTL

Content never changes or changes at periodic intervals
Caching correctly is difficult

Variations in ideal TTL

Content never changes or changes at periodic intervals
Caching correctly is difficult

Variations in ideal TTL

Ideal TTL varies significantly!
Caching correctly is difficult

Variations in ideal TTL

High TTLs $\rightarrow$ good cache performance $\rightarrow$ stale content
Caching correctly is difficult
Variations in ideal TTL

High TTLs $\rightarrow$ good cache performance $\rightarrow$ stale content

Low TTLs $\rightarrow$ fresh content $\rightarrow$ poor cache performance
Caching correctly is difficult
Variations in ideal TTL

High TTLs → good cache performance → stale content
Low TTLs → fresh content → poor cache performance
Making caching better
Anatomy of an app interaction
Making caching better
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app

Origin server
Making caching better
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app
Tap on “Entertainment News” category

Get /entertainment.json

Origin server
Making caching better
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app
Tap on “Entertainment News” category

Origin server
Get /entertainment.json

/entertainment.json

entertainment.json
{
  article1: {
    summary,
    /article1.json
  },
  article2: {
    ...
  }
}
Making caching better
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app
Tap on “Entertainment News” category
Tap on news article

entertainment.json
{
  article1: {
    summary,
  },
  article2: {
    ...
  }
}

Origin server
Making caching better
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app
Tap on “Entertainment News” category
Tap on news article

Get /article1.json

entertainment.json
{
  article1: {
    summary,
    /article1.json
  },
  article2: {
    ...
  }
}
Making caching better
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app
Tap on “Entertainment News” category
Tap on news article

origin server
entertainment.json
{
  article1: {
    summary,
    /article1.json
  },
  article2: {
    ...
  }
}
Making caching better
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app
- Tap on “Entertainment News” category
- Tap on news article

Origin server

get /article1.json

origin server

/article1.json

entertainment.json

```
{  
  article1: {  
    summary,
    /article1.json  
  },  
  article2: {  
    ...
  }
}  
```
Making caching better
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app
- Tap on “Entertainment News” category
- Tap on news article

Origin server

Get /article1.json

/article1.json

Textual files anchor interactions

entertainment.json

{ 
  article1: { 
    summary, 
    /article1.json 
  },
  article2: { 
  ... 
}
Making caching better
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app
- Tap on “Entertainment News” category
- Tap on news article

Characteristics of textual files
- Blocking fetches after interactions
- Dynamic → Small TTLs
- Cheap to fetch

Entertainment.json
{
  article1: {
    summary,
    /article1.json
  },
  article2: {
    ...
  }
}
Making caching better
Anatomy of an app interaction

Task: Loading a news article

Actions by user after opening app
Tap on “Entertainment News” category
Tap on news article

Origin server
Get /entertainment.json

Characteristics of textual files
- Blocking fetches after interactions
- Dynamic
- Small TTLs
- Cheap to fetch

entertainment.json

Textual files anchor interactions

entertainment.json

{ article1: { summary, /article1.json }, article2: { ... } }
Making caching better
Insights about textual files

- Blocking fetches after interactions
- Dynamic → Small TTLs
- Cheap to fetch
Making caching better
Insights about textual files

Blocking fetches after interactions → Cache hits are impactful

Dynamic → Small TTLs

Cheap to fetch
Making caching better
Insights about textual files

- Blocking fetches after interactions
- Dynamic → Small TTLs
- Cheap to fetch

→

Cache hits are impactful
Making caching better
Insights about textual files

- Blocking fetches after interactions
- Dynamic → Small TTLs
- Cheap to fetch

Cache hits are impactful

Client

Server

Fetch json
Making caching better
Insights about textual files

- Blocking fetches after interactions
- Dynamic → Small TTLs
- Cheap to fetch

Cache hits are impactful

Client

- Fetch json
- Fetch content linked from json

Server
Making caching better

Insights about textual files

- Blocking fetches after interactions
- Dynamic → Small TTLs
- Cheap to fetch
- Cache hits are impactful
Making caching better
Insights about textual files

- Blocking fetches after interactions
- Dynamic → Small TTLs
- Cheap to fetch

→ Cache hits are impactful
Making caching better

Insights about textual files

- Blocking fetches after interactions
- Dynamic → Small TTLs
- Cheap to fetch

→ Cache hits are impactful
→ Quick to expire
Making caching better
Insights about textual files

- Blocking fetches after interactions → Cache hits are impactful
- Dynamic → Small TTLs → Quick to expire
- Cheap to fetch → Proactive refresh?
Making caching better
Insights about textual files

- Blocking fetches after interactions ➔ Cache hits are impactful
- Dynamic → Small TTLs ➔ Quick to expire
- Cheap to fetch ➔ Proactive refresh?
- Conditional GET
Making caching better
Insights about textual files

Cache Validation: Client Checks Freshness

How do they identify the “version”?  
• Timestamp  
  • When the item was modified by the server  
  • E.g., Last-Modified: Wed, 21 Oct 2015 07:28:00 GMT  
• Version number  
  • Entity tag provided by the server  
  • E.g., ETag: "33a64df551425fcc55e4d42a148795d9f25f89d4"
Making caching better

Insights about textual files

- Blocking fetches after interactions → Cache hits are impactful
- Dynamic → Small TTLs → Quick to expire
- Cheap to fetch → Proactive refresh?

TTLs would reach ideal values
Conditional GET
Making caching better
Insights about textual files

- Blocking fetches after interactions → Cache hits are impactful
- Dynamic → Small TTLs → Quick to expire
- Cheap to fetch → Proactive refresh?

- Outsize impact of such cache hits
- TTLs would reach ideal values
- Conditional GET
Making caching better

Insights about textual files

- Blocking fetches after interactions
- Dynamic → Small TTLs
- Cheap to fetch

→ Cache hits are impactful

→ Improve app interactions by up to 16%

→ Proactive refresh?

↑ Outsize impact of such cache hits

↑ TTLs would reach ideal values

↑ Conditional GET

Cache hits are impactful

Improve app interactions by up to 16%

Proactive refresh?
Making caching better
Insights about textual files

Blocking fetches after interactions → Cache hits are impactful
Dynamic → Small TTLs → Quick to expire
Cheap to fetch → Proactive refresh?

Improve app interactions by up to 16%
TTLs would reach ideal values
Conditional GET

What about other assets in the cache, like images?
Making caching better
Insights about textual files

Blocking fetches after interactions → Cache hits are impactful
Dynamic → Small TTLs → Quick to expire
Cheap to fetch → Proactive refresh?

Improve app interactions by up to 16%
TTLs would reach ideal values
Conditional GET

What about other assets in the cache, like images?
HEAD requests
A different way of caching

Why wait to observe requests?

Caches don’t help first (cold) loads
A different way of caching
Why wait to observe requests?

Caches don’t help first (cold) loads

From last lecture

Client Machine (e.g., Browser)

Advantages
• Very low latency
• Preserves access bandwidth
• Available when disconnected

Disadvantages
• Low hit rate due to “cold” misses
• Many cache consistency checks
• Incomplete logs at the server
A different way of caching
Why wait to observe requests?

Caches don’t help first (cold) loads

Client Machine (e.g., Browser)

Advantages
- Very low latency
- Preserves access bandwidth
- Available when disconnected

Disadvantages
- Low hit rate due to “cold” misses
- Many cache consistency checks
- Incomplete logs at the server

From last lecture
A different way of caching

Why wait to observe requests?

Caches don’t help first (cold) loads

aka “prefetching”

Could we “warm” the cache?
A different way of caching

Why wait to observe requests?

Caches don’t help first (cold) loads

Could we “warm” the cache?

aka “prefetching”

Near-instantaneous response to users
A different way of caching
Why wait to observe requests?

- Caches don’t help first (cold) loads
- Could we “warm” the cache? aka “prefetching”
- Near-instantaneous response to users
- But what would the user want?
Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions
Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

4 of top 50 store apps
prefetch
Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

4 of top 50 store apps prefetch

Speedups above 60%
Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

- Speedups above 60%
- 4x data usage
- 4 of top 50 store apps prefetch
Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

- Speedups above 60%
- 4x data usage
- Difficult to predict users
- 4 of top 50 store apps prefetch
Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

- Speedups above 60%
- 4 of top 50 store apps prefetch
- 4x data usage
- Difficult to predict users
- Coupling of speedups and overheads
Prefetching correctly is difficult

Load content in the background at periodic intervals
Avoid network fetches during interactions

Speedups above 60%

4 of top 50 store apps prefetch

Textual files are always fresh in the cache. Could they help?

Difficult to predict users

Coupling of speedups and overheads
Analyzing textual files
Opportunities for prefetching
Analyzing textual files
Opportunities for prefetching

entertainment.json
{
  article1: {
    summary,
    /article1.json
  },
  article2: {
    ...
  }
}

article1.json
{
  title: “Article”
  banner: “/x.jpg”,
  content: “..”
}
Analyzing textual files
Opportunities for prefetching

Text files contain pointers to upcoming requests
Just-in-time prefetching

User

Cache

Origin server
Just-in-time prefetching

User → Interaction (load article1) → Cache → Origin server
Just-in-time prefetching

User → Interaction (load article1) → Cache → Origin server

JSON: article1.json
Referenced content: x.jpg
Just-in-time prefetching

User

Interaction (load article1)

Cache

JSON

referenced content

Origin server

JSON: article1.json
Referenced content: x.jpg
Just-in-time prefetching

Normal Operation

User

Interaction (load article1)

Cache

JSON referenced content

Origin server

JSON: article1.json
Referenced content: x.jpg
Just-in-time prefetching

Normal Operation

Optimization: Parse JSONs and determine referenced content in advance

Interaction (load article1)

JSON: article1.json
Referenced content: x.jpg
Just-in-time prefetching

Normal Operation

Interaction (load article1)

Optimization: Parse JSONs and determine referenced content in advance

JSON: article1.json
Referenced content: x.jpg

User
Cache
Origin server

User
Cache
Origin server
Just-in-time prefetching

Normal Operation

Interaction (load article1)

Cache

User

Origin server

Interaction (load article 1)

Cache

User

Origin server

JSON: article1.json
Referred content: x.jpg

Optimization:
Parse JSONs and determine referenced content in advance
Just-in-time prefetching

Normal Operation

User interacts with the cache and the origin server to load an article.

Optimization:
Parse JSONs and determine referenced content in advance

User interacts with the cache and the origin server to load an article. The cache is pre-fetched in advance to improve load times.

JSON: article1.json
Referenced content: x.jpg
Just-in-time prefetching

Normal Operation

Interaction (load article1)

Optimization: Parse JSONs and determine referenced content in advance

Interaction (load article1)

Cache

User

Origin server

JSON: article1.json
Referenced content: x.jpg

Get JSON + referenced content

Get JSON

Referenced content: x.jpg

Get JSON + referenced content

JSON: article1.json
Just-in-time prefetching

Normal Operation

Optimization: Parse JSONs and determine referenced content in advance

Sidestep the prediction problem

"User requested article1.json so they’ll need the images in it as well"
Just-in-time prefetching

Normal Operation

Optimization: Parse JSONs and determine referenced content in advance

Sidestep the prediction problem

Response times speedups of 27-44%

“User requested article1.json so they’ll need the images in it as well”
Other caching opportunities?
Other caching opportunities?

Interaction, e.g., tap(x, y)
Other caching opportunities?

Interaction, e.g., tap(x, y)

User → Origin Servers
Other caching opportunities?

Interaction, e.g., tap(x, y)

User

Local Computations

Origin Servers
Other caching opportunities?

Interaction, e.g., tap(x, y)

User -> Local Computations -> Network delays -> Origin Servers

User
Other caching opportunities?

Interaction, e.g., tap(x, y)

User ➞ Network delays ➞ Origin Servers

Local Computations

Remote Computations
Other caching opportunities?

Interaction, e.g., tap(x, y)

User

Origin Servers

Local Computations

Network delays

Remote Computations
Other caching opportunities?

Interaction, e.g., tap(x, y)

User → Local Computations → Origin Servers

Network delays

Remote Computations
Other caching opportunities?

Interaction, e.g., tap(x, y)

Content

Network delays

Origin Servers

Local Computations

Remote Computations

User
Other caching opportunities?

We just handled this

Interaction, e.g., tap(x, y)

Content

Network delays

Origin Servers

Local Computations

Remote Computations

What about these?
Intro to memoization
Intro to memoization

```c
int fn(int input) {
    return input*2;
}
```
Intro to memoization

```java
int fn(int input) {
    return input * 2;
}
```

```java
int fn(int input) {
    if (memo.contains(input)) {
        return memo.get(input);
    } else {
        int tmp = input * 2;
        memo.set(input, tmp);
        return tmp;
    }
}
```
Intro to memoization

```cpp
int fn(int input) {
    return input*2;
}

int fn(int input) {
    if (memo.contains(input)) {
        return memo.get(input);
    } else {
        int tmp = input * 2;
        memo.set(input, tmp);
        return tmp;
    }
}
```

If read state matches a prior run

Apply prior writes
Potential of memoization
Potential of memoization

Perform a series of interactions on the app

Wait for delay $\delta$

Perform another series of interactions

During the 2nd run, how much compute (per interaction) was memoizable?
Potential of memoization

Perform a series of interactions on the app
Wait for delay $\delta$
Perform another series of interactions
During the 2nd run, how much compute (per interaction) was memoizable?

Upper bound of memoization benefits with zero overhead
Potential of memoization

Perform a series of interactions on the app
Wait for delay $\delta$
Perform another series of interactions
During the 2nd run, how much compute (per interaction) was memoizable?

Upper bound of memoization benefits with zero overhead

![Graph showing memoizable percentage of compute delays for different delays: Back-to-back, 4 hours, 12 hours, and 24 hours.](image)
Potential of memoization

Perform a series of interactions on the app
Wait for delay $\delta$
Perform another series of interactions

During the 2nd run, how much compute (per interaction) was memoizable?

Upper bound of memoization benefits with zero overhead

Benefits persist over time due to stable computations
Implementing memoization
Implementing memoization

Original app

Program Analysis
Implementing memoization

Original app

Program Analysis

Analysis on each function

Function reads and writes

F1 reads: \{x, y\} writes: \{z\}

F2 reads: \{z\} writes: {}  

F3 reads: {} writes: \{x\}
Implementing memoization

Program Analysis

Rewriter to include memoization

All functions with memoization

All functions

Function reads and writes

F1
reads: \{x, y\}
writes: \{z\}

F2
reads: \{z\}
writes: \{\}\n
F3
reads: \{\}\nwrites: \{x\}
Implementing memoization

Original app

Program Analysis

All functions

Rewriter to include memoization

All functions with memoization

Analysis on each function

Modified app

Function reads and writes

F1 reads: \{x, y\} writes: \{z\}

F2 reads: \{z\} writes: \{

F3 reads: \{\} writes: \{x\}
Straightforward memoization

The result
Straightforward memoization

The result
Straightforward memoization

The result

Blocking, expensive, potentially unnecessary cache queries
Challenges with memoization

An experimental result
Challenges with memoization
An experimental result

Function invocation runtimes for 5 exemplar apps
Challenges with memoization
An experimental result

Function invocation runtimes for 5 exemplar apps

- Low function runtimes → Overheads can exceed runtimes
- Large number of invocations → Queries can overwhelm resources
Lookaheads
Making cache lookups fast!
Lookaheads
Making cache lookups fast!

- Offline callgraph generation. Used Online.
- Annotate nodes.
  - Variable dependencies
Lookaheads
Making cache lookups fast!

- Offline callgraph generation. Used Online.
- Annotate nodes.
  - Variable dependencies
- Identify *potentially* upcoming functions.
- Query these functions $\rightarrow$ Warm the cache
Lookaheads
Making cache lookups fast!

- Offline callgraph generation. Used Online.
- Annotate nodes.
  - Variable dependencies
- Identify potentially upcoming functions.
- Query these functions → Warm the cache
  - *Read state of upcoming functions may vary!*

Reads: \{x, y\}
Writes: \{z\}

onTouch()
Lookaheads
Lookaheads

Static callgraph
Lookaheads

Current Computation

onTouch( )

F1

F9

F6

F2

writes: {c}

reads: {a, b, c}

F5

• F2 may run shortly. F2’s reads: {a, b, c}
• F6 may write to {c}
  • No writes to {a, b} before F2
• Lookup {a, b, c=*}
  • Use current values {a=5, b=1}
• Results → Microcache

Static callgraph
Lookaheads

- F2 may run shortly. F2’s reads: \{a, b, c\}
- F6 may write to \{c\}
  - No writes to \{a, b\} before F2
- Lookup \{a, b, c=*\}
  - Use current values \{a=5, b=1\}
  - Results → Microcache

Static callgraph

Current Computation

onTouch( )

F1

F5

F9

F6

writes: \{c\}

F2

reads: \{a, b, c\}

Cache

Microcache

F2’s entries

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
Lookaheads

Current Computation

onTouch()

F1

F2
reads: \{a, b, c\}

F9

writes: \{c\}

F6

F5

• F2 may run shortly. F2’s reads: \{a, b, c\}
• F6 may write to \{c\}
  • No writes to \{a, b\} before F2
• Lookup \{a, b, c=\}*
  • Use current values \{a=5, b=1\}
• Results → Microcache

• F2’s entries
  a=5
  b=1
  c=11

• Cache
  a=1
  b=1
  c=7

Microcache

Static callgraph
Lookaheads

- F2 may run shortly. F2’s reads: \{a, b, c\}
- F6 may write to \{c\}
  - No writes to \{a, b\} before F2
- Lookup \{a, b, c=\}
  - Use current values \{a=5, b=1\}
  - Results $\rightarrow$ Microcache

```
Current Computation
```

```
F2's entries
a=5
b=1
c=11
```

```
F2 may run shortly. F2’s reads: \{a, b, c\}
F6 may write to \{c\}
  - No writes to \{a, b\} before F2
Lookup \{a, b, c=\}
  - Use current values \{a=5, b=1\}
  - Results $\rightarrow$ Microcache
```

```
Microcache
a=5
b=1
c=11
```

```
Cache
a=1
b=1
c=7
```

```
Current Computation
```

```
F2's entries
a=5
b=1
c=11
```

```
F2 may run shortly. F2’s reads: \{a, b, c\}
F6 may write to \{c\}
  - No writes to \{a, b\} before F2
Lookup \{a, b, c=\}
  - Use current values \{a=5, b=1\}
  - Results $\rightarrow$ Microcache
```

```
Microcache
a=5
b=1
c=11
```

```
Cache
a=1
b=1
c=7
```

```
Current Computation
```

```
F2's entries
a=5
b=1
c=11
```

```
F2 may run shortly. F2’s reads: \{a, b, c\}
F6 may write to \{c\}
  - No writes to \{a, b\} before F2
Lookup \{a, b, c=\}
  - Use current values \{a=5, b=1\}
  - Results $\rightarrow$ Microcache
```

```
Microcache
a=5
b=1
c=11
```

```
Cache
a=1
b=1
c=7
```

```
Current Computation
```

```
F2's entries
a=5
b=1
c=11
```

```
F2 may run shortly. F2’s reads: \{a, b, c\}
F6 may write to \{c\}
  - No writes to \{a, b\} before F2
Lookup \{a, b, c=\}
  - Use current values \{a=5, b=1\}
  - Results $\rightarrow$ Microcache
```

```
Microcache
a=5
b=1
c=11
```

```
Cache
a=1
b=1
c=7
```

```
Current Computation
```

```
F2's entries
a=5
b=1
c=11
```

```
F2 may run shortly. F2’s reads: \{a, b, c\}
F6 may write to \{c\}
  - No writes to \{a, b\} before F2
Lookup \{a, b, c=\}
  - Use current values \{a=5, b=1\}
  - Results $\rightarrow$ Microcache
```

```
Microcache
a=5
b=1
c=11
```

```
Cache
a=1
b=1
c=7
```

```
Current Computation
```

```
F2's entries
a=5
b=1
c=11
```

```
F2 may run shortly. F2’s reads: \{a, b, c\}
F6 may write to \{c\}
  - No writes to \{a, b\} before F2
Lookup \{a, b, c=\}
  - Use current values \{a=5, b=1\}
  - Results $\rightarrow$ Microcache
```

```
Microcache
a=5
b=1
c=11
```

```
Cache
a=1
b=1
c=7
```

```
Current Computation
```
• F2 may run shortly. F2’s reads: \{a, b, c\}
• F6 may write to \{c\}
  • No writes to \{a, b\} before F2
• Lookup \{a, b, c=\}
  • Use current values \{a=5, b=1\}
• Results \rightarrow Microcache
Lookaheads

onTouch()

Low function runtimes → Overheads can exceed runtimes

Microcache

F2’s entries

Current Computation

{a=5, b=1, c=11}
Lookaheads

onTouch()

\{a=5, b=1, c=11\}

Low function runtimes → Overheads can exceed runtimes

Similar to prefetching!

Current Computation

F2’s entries

a=5
b=1
c=11

a=1
b=1
c=7
Caching Policies
Caching Policies

Adding to the cache

Removing from the cache
What functions do we memoize?

Removing from the cache
Caching Policies

What functions do we memoize?

How to evict cache entries?
Caching Policies

What functions do we memoize?

Given a function, what fraction of its invocations were memoized?
Caching Policies

What functions do we memoize?

Given a function, what fraction of its invocations were memoized?
Caching Policies

What functions do we memoize?

Given a function, what fraction of its invocations were memoized?

Certain functions never hit in the cache
Caching Policies

What functions do we memoize?

Given a function, what fraction of its invocations were memoized?

Certain functions never hit in the cache

Track hit rate with cache entries: Deactivate low hit-rate functions
Caching Policies

What functions do we memoize?

Given a function, what fraction of its invocations were memoized?

Large number of invocations →
Queries can overwhelm resources

Track hit rate with cache entries: Deactivate low hit-rate functions
Caching Policies
Caching Policies

How to evict cache entries?
Caching Policies

How to evict cache entries?

Annotate cache entries with utility/potential benefits ➔ Rank cache entries by utility
Caching Policies

- Annotate cache entries with utility/potential benefits
- Rank cache entries by utility
Caching Policies

Annotate cache entries with utility/potential benefits → Rank cache entries by utility

How does the runtime of a particular function vary across its invocations?
Caching Policies

Annotate cache entries with utility/potential benefits

Rank cache entries by utility

How does the runtime of a particular function vary across its invocations?
Caching Policies

Annotate cache entries with utility/potential benefits

→

Rank cache entries by utility

How does the runtime of a particular function vary across its invocations?

![Coefficient of variation of runtime (fraction) CDF]

- With memoization
- Without memoization

Low median → Stable runtime
Caching Policies

Annotate cache entries with utility/potential benefits → Rank cache entries by utility

How does the runtime of a particular function vary across its invocations?

Low median → Stable runtime

Store first runtimes → Diff is the speedup

First memoized and unmemoized invocation
Median computation improvements per interaction

Computation Improvements

Devices: Note 9, Pixel 5. Networks: WiFi, LTE.

<table>
<thead>
<tr>
<th></th>
<th>LTE</th>
<th>WiFi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel 5</td>
<td>36.8%</td>
<td>42.9%</td>
</tr>
<tr>
<td>Note 9</td>
<td>41.2%</td>
<td>43.8%</td>
</tr>
</tbody>
</table>
Computation Improvements

Devices: Note 9, Pixel 5. Networks: WiFi, LTE.

Median computation improvements per interaction

- **Pixel 5**
  - LTE: 36.8%
  - WiFi: 42.9%

- **Note 9**
  - LTE: 41.2%
  - WiFi: 43.8%
Performance Improvements
With Network and Compute optimizations

% improvement in IRT

Network optimizations  Compute opt.  Both

δ=0hr,LTE  δ=12hr,LTE  δ=0hr,WiFi  δ=12hr,WiFi

20  40  60  80
Caching in mobile apps

Summary

• Apps could cache content, just like web browsers and CDN. But this potential is not utilized optimally
  • Techniques exist to optimize caching and prefetching
  • For web content:
    • improve TTLs
    • prefetch just-in-time
  • For computations:
    • improve cache lookup time with lookaheads
    • optimized cache entries
    • domain specific caching policies
More content on compute caching (Floo)
Floo: Kinds of local computations
Floo: Kinds of local computations

Compute bottlenecks are critical
Floo: Kinds of local computations

Compute bottlenecks are critical

But what are these computations exactly?
Floo: Kinds of local computations

But what are these computations exactly?

- App-defined code
- AOSP platform code

% of compute time

0 20 40

1st party code 3rd party libraries OpenJDK Android Platform
Floo: Kinds of local computations

But what are these computations exactly?

![Bar chart showing the percentage of compute time for different types of code: 1st party code, 3rd party libraries, OpenJDK, and Android Platform. The chart indicates that App-defined and AOSP functions are substantial.]

App-defined and AOSP functions are substantial
Implementing memoization
Implementing memoization

Exclude functions using nondeterministic or native APIs

```java
Fn f() {
    return Time.now() + 1
}
```
Implementing memoization

Exclude functions using nondeterministic or native APIs

Correctness

```java
Fn f() {
    return Time.now() + 1
}
```
Implementing memoization

Exclude functions using nondeterministic or native APIs

Memo writes include side effects

Correctness

```javascript
Fn f() {
    return Time.now() + 1
}
```

```javascript
Fn f() {
    this.var = 5;
    print(this.var);
}
```
Implementing memoization

Exclude functions using nondeterministic or native APIs

Memo writes include side effects

Comprehensive read state in addition to parameters

Correctness

```
Fn f() {
    return Time.now() + 1
}
```

```
Fn f() {
    this.var = 5;
    print(this.var);
}
```

```
Fn f(int param) {
    if (this.x == 5)
        return 10;
    else return param;
}
```
Implementing memoization

Exclude functions using nondeterministic or native APIs

Memo writes include side effects

Comprehensive read state in addition to parameters

Correctness

Support for impure functions

Fn f() {
    return Time.now() + 1
}

Fn f() {
    this.var = 5;
    print(this.var);
}

Fn f(int param) {
    if (this.x == 5)
        return 10;
    else return param;
}
Evaluation

Floo - mobile computation caching

• How does Floo improve the compute time?
• How does this improvement translate into responsiveness?
• How does Floo perform with fewer device resources?
• Do Floo’s transformations preserve correct app behavior?
• What is the source of Floo’s wins?
• How does Floo compare against prior work?
• Does the improvement change based on user access delays?
• What are the typical app characteristics that enable memoization?
Responsiveness (IRT) Improvements

![Graph showing CDF of IRT improvement with different devices and network conditions]
Responsiveness (IRT) Improvements

![Diagram showing CDF of IRT improvement]
Responsiveness (IRT) Improvements

![Graph showing CDF and IRT improvement percentage for Pixel 5 (LTE), Pixel 5 (WiFi), Note 9 (LTE), and Note 9 (WiFi).]
Responsiveness (IRT) Improvements

Floo optimizes response times by 33-72% for an interaction.
Varying device resources

% improvement in IRT vs # threads

% improvement in IRT vs Compute cache size (MB)

Floo, LRU
Correctness

Does Floo preserve developer intent?

Interaction, e.g., tap(x, y)

Origin Servers

Network delays

Local Computations

Remote Computations

Content
Correctness

Does Floo preserve developer intent?

Replay same interactions

Interaction, e.g., tap(x, y)

Content

Network Cache
Replay same responses
Correctness

Does Floo preserve developer intent?

- Same server side content
- Same user interactions
- Single thread to prevent races
- Enforce determinism
Correctness

Does Floo preserve developer intent?

- Same server side content
- Same user interactions
- Single thread to prevent races
- Enforce determinism

Compare behavior with and without Floo
Correctness

Does Floo preserve developer intent?
Correctness

Does Floo preserve developer intent?
Correctness

Does Floo preserve developer intent?

Enforce determinism
Force single thread

User perspective
- Pixel-wise screen comparison
- Identical Android View classes

App state
- Full heap equivalence
Correctness

Does Floo preserve developer intent?

Enforce determinism
Force single thread

User perspective

- Pixel-wise screen comparison
- Identical Android View classes

Match after each interaction

App state

- Full heap equivalence
- Match after a series of interactions
Correctness
Does Floo preserve developer intent?

Enforce determinism
Force single thread

User perspective
Pixel-wise screen comparison
Identical Android View classes
Match after each interaction

App state
Full heap equivalence
Match after a series of interactions

Enforce determinism
Allow multiple threads

App state
Does Floo preserve developer intent?

Correctness

User perspective
- Pixel-wise screen comparison
- Identical Android View classes
  - Match after each interaction

App state
- Full heap equivalence
  - Match after a series of interactions

Enforce determinism
- Force single thread

Enforce determinism
- Allow multiple threads

App state
- Heap mismatch
  - After a series of interactions
Correctness

Investigating app state mismatches
Correctness

Investigating app state mismatches

Thread schedule variations
Correctness

Investigating app state mismatches

Thread schedule variations

Run 1: Without Floo
Run 2: With Floo

Heap and potential visual mismatch across two runs
Correctness

Investigating app state mismatches

Run 1: Without Floo
Run 2: With Floo

Heap and potential visual mismatch across two runs
Correctness

Investigating app state mismatches

Thread schedule variations

Run 1: Without Floo

Run 2: With Floo

Undetected synchronizations

while(!x.contentReady); // busy wait
y = x.contentReady; // y must be true

Heap and potential visual mismatch across two runs
Correctness

Investigating app state mismatches

Thread schedule variations

Run 1: Without Floo

Run 2: With Floo

Undetected synchronizations

while(!x.contentReady); // busy wait
y = x.contentReady; // y must be true

We entered the function with x.contentReady as false and set y to be true

Heap and potential visual mismatch across two runs
Correctness
Investigating app state mismatches

Thread schedule variations

Run 1: Without Floo
Run 2: With Floo

Undetected synchronizations

while(!x.contentReady); // busy wait
y = x.contentReady; // y must be true

We entered the function with x.contentReady as false and set y to be true

if x.contentReady is false
set y=true

Incorrect behavior

Heap and potential visual mismatch across two runs
More content on mobile web caching (Marauder)
Bandwidth overheads

![CDF plot showing bandwidth overheads for different methods.](image)

- **Cache Refreshing**
- **JIT Prefetching**
- **Marauder**
Constituent benefits

% improvement in IRT

- JIT Prefetching
- Cache Refreshing
- Marauder

\( \delta \)