Scalable and private media consumption with Popcorn

Trinabh Gupta
The University of Texas at Austin
User media consumption has increased …

… leading to large centralized datasets …

… subject to risks such as server hacks, accidental disclosures, etc.
NETFLIX SPILLED YOUR BROKEBACK MOUNTAIN SECRET, LAWSUIT CLAIMS
How can we build a Netflix-like system that

a) provably hides media diet,

b) has low dollar cost, and

c) is compatible with commercial media streaming?
Private Information Retrieval (PIR) provably hides requests but …

- Each request must **touch the entire library**.
- There is a tension between overhead and content protection.
- PIR **assumes fixed-size objects**, but media sizes vary.
Popcorn tailors PIR for media to meet our three requirements.

Its per-request dollar cost is 3.87x times that of a non-private baseline.
Rest of this talk

• Background on PIR.

• Challenges of using PIR (in detail).

• Design (tailoring of PIR) and evaluation of Popcorn.
Background on information-theoretic PIR (ITPIR)

Pick a subset of \{1, 2, 3, 4, 5\} randomly

Ex: \{3, 4\}, 4, 5

\{2, 4\}

Client wants M1

M1 = Reply1 ⊕ Reply2

Ex: \{3, 4\}

Ex: \{1, 2, 4, 5\}

\{2, 4\}

Reply 1 = M2 ⊕ M4

\[\text{Server 1}\]

\[01111001\ldots\]

\[010111000\ldots\]

\[10101011\ldots\]

\[11100000\ldots\]

\[0011000.\ldots\]

Reply 1 = M2 ⊕ M4

\[\text{Server 2}\]

\[01111001\ldots\]

\[010111000\ldots\]

\[10101011\ldots\]

\[11100000\ldots\]

\[0011000.\ldots\]

\[\text{Server 2}\]

No collusion

Reply 2 = M1 ⊕ M2 ⊕ M4

\[\text{Server 2}\]

\[01111001\ldots\]

\[010111000\ldots\]

\[10101011\ldots\]

\[11100000\ldots\]

\[0011000.\ldots\]

\[\text{Server 2}\]

\[01111001\ldots\]

\[010111000\ldots\]

\[10101011\ldots\]

\[11100000\ldots\]

\[0011000.\ldots\]
Computational PIR (CPIR) from 10,000 feet

- one server
- instead of XORs, expensive server-side cryptographic operations
Challenges of using PIR

<table>
<thead>
<tr>
<th>ITPIR</th>
<th>CPIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>content can disseminate in an uncontrolled manner</td>
<td>content disseminates in a controlled manner</td>
</tr>
<tr>
<td>cheap operations (XORs) but process entire library per request</td>
<td>expensive operations and process entire library per request</td>
</tr>
<tr>
<td>assumes fixed-size objects</td>
<td>assumes fixed-size objects</td>
</tr>
</tbody>
</table>

Given these, how can we build a system that controls content and is low cost?
Popcorn composes ITPIR and CPIR to get desirable properties from both

\[
\text{Enc}(K_1, M_1) \quad \text{Enc}(K_2, M_2) \quad \text{Enc}(K_3, M_3) \quad \text{Enc}(K_4, M_4) \quad \text{Enc}(K_5, M_5)
\]
Challenges of using PIR

**ITPIR**
- content can disseminate in an uncontrolled manner
- cheap operations (XORs)
- but process entire library per request
- assumes fixed-size objects

**CPIR**
- content disseminates in a controlled manner
- expensive operations and process entire library per request
- assumes fixed-size objects

**Popcorn**
Popcorn batches requests to amortize the overhead of ITPIR

Pick a subset of \{1, 2, 3, 4, 5\} randomly

Client 1

Client 2

Client 3

\{1, 3, 5\}

\{1, 3, 4, 5\}

\{2, 4\}

\begin{align*}
M1 & : 01110010 \ldots \\
M2 & : 01011100 \ldots \\
M3 & : 10101011 \ldots \\
M4 & : 11100000 \ldots \\
M5 & : 00110000 \ldots 
\end{align*}

Server 1

\begin{align*}
\text{Reply} & = M1 \oplus M3 \oplus M5 \\
\text{Reply} & = M1 \oplus M3 \oplus M4 \oplus M5 \\
\text{Reply} & = M2 \oplus M4
\end{align*}

Observation: Very similar disk I/O for each request!

Benefits of batching:

- Disk I/O transfers are amortized.
- CPU cycles are reduced as matrix multiplication algorithms exploit cache locality.
Straw man: Group requests that arrive during an epoch

Client's view:

Client A, Client B, Client C

Epoch

Start handling A, B, C

First chunk of movie

Client A's playback buffer

Wait for server to form batch

Client perceived delay = epoch + epsilon
Straw man: Group requests that arrive during an epoch

Server’s choices:

Small batch, small delay

Large batch, large delay

**Issue:** Hard to get both small delay and large batch
Popcorn exploits streaming to form large batches with small startup delay

\[ t = \text{times at which a client needs movie chunks} \]

\[ t = 0 \]

= time it takes to consume a single chunk

\[ t = 2 \]

\[ t = 3 \]

Observation: Client needs only the first chunk immediately.
Narrow first column => small startup delay

Wider columns => longer processing times …
… but bigger batches
<table>
<thead>
<tr>
<th>ITPIR</th>
<th>CPIR</th>
<th>Popcorn</th>
</tr>
</thead>
<tbody>
<tr>
<td>content can disseminate in an uncontrolled manner</td>
<td>content disseminates in a controlled manner</td>
<td>content disseminates in a controlled manner</td>
</tr>
<tr>
<td>cheap operations but process entire library <em>per request</em></td>
<td>expensive operations, process entire library <em>per request</em></td>
<td>cheap operations, process entire library <em>per batch</em></td>
</tr>
<tr>
<td>assumes fixed-size objects</td>
<td>assumes fixed-size objects</td>
<td>?</td>
</tr>
</tbody>
</table>
Popcorn exploits compression to address fixed-size requirement

- Small variations in bitrate have limited impact on user satisfaction [SIGCOMM 11, LANC 11, CCNC 12].

- 85% of movies close to the average size.
Outline

✓ Background on PIR.

✓ Design (tailoring of PIR) of Popcorn.

• Evaluation of Popcorn.
Experiment method

Baselines:

• Non-private system (Apache server)
• State-of-the-art CPIR [XPIR PETS16]
• State-of-the-art ITPIR [Percy++]
• ITPIR++: ITPIR extended with the straw man batching scheme

Netflix-like library: 8000 movies, 90 minutes, 4Mbps

Workload: 10K clients arrive within 90 minutes according to a Poisson process

Estimate per-request dollar cost using Amazon’s pricing model

• CPU: $0.0076/hour
• Disk I/O bandwidth: $0.042/Gbps-hour
• Network: $0.006/GB
<table>
<thead>
<tr>
<th>System</th>
<th># of CPUs</th>
<th>Disk I/O (Gbps)</th>
<th>Network (relative to non-private)</th>
<th>$ relative to non-private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-private</td>
<td>0</td>
<td>0</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>CPIR</td>
<td>11.6</td>
<td>64</td>
<td>5x</td>
<td>265x</td>
</tr>
<tr>
<td>ITPIR</td>
<td>3.1</td>
<td>64</td>
<td>2x</td>
<td>256x</td>
</tr>
<tr>
<td>ITPIR++ (delay 15s)</td>
<td>0.65</td>
<td>3</td>
<td>2x</td>
<td>14x</td>
</tr>
<tr>
<td>Popcorn (delay 15s)</td>
<td>0.74</td>
<td>0.23</td>
<td>2x</td>
<td>3.87x</td>
</tr>
</tbody>
</table>
Popcorn is private and affordable but …

- Assumes that the ITPIR servers do not collude.

- Incurs costs that are linear in the size of the library.

- Does not support recommendations, aggregate view statistics.

Solution: Use prior work [Canny S&P ’02, Toubiana et al. NDSS ‘10]
Related work

• Improving performance of PIR.
  • Distributing work [FC13, TDSC12], cheaper crypto [PETS16, ESORICS14, ISC10, TKDE13, WEWoRC07], bucketing [DBSec10, PETS10], batching [FC15, JoC04], secure co-processors [PET03, FAST13, NDSS08, IBM Systems Journal01]

• Protecting library content in ITPIR [RANDOM98, S&P07, WPES13]

• Handling variable-sized objects [CCSW14, NDSS13]

• Prior PIR implementations [Percy++, PETS16, CCSW14]

• Video-on-demand [MMCN95]
Take-away points from Popcorn

• It is possible to build a private, functional, and low-cost media delivery system …

• … by tailoring PIR to media delivery.

• The per-request cost in Popcorn is 3.87x that of a non-private baseline.