Alohamora: Reviving HTTP/2 Push and Preload by Adapting Policies On-the-Fly

Nikhil Kansal
Murali Ramanujam
Ravi Netravali
Serial dependency chains in page loads

Parse page and discover dependencies
Serial dependency chains in page loads

Load dependencies from remote server
Serial dependency chains in page loads

REPEAT!

1 HTML

2 JS

3 IMG 3 IMG

Need img1.jpg and img2.jpg!

GET http://b.com/img1.jpg

GET http://c.com/img2.jpg
Serial dependency chains in page loads

**REPEAT!**

1. HTML
2. JS
3. IMG
3. IMG

Serial dependency chains

**Slow page load**

Need img1.jpg and img2.jpg!

GET http://b.com/img1.jpg
GET http://c.com/img2.jpg
Page loads with HTTP/2 push + preload

Push

GET http://a.com/index.html

Page will need to load more resources
Page loads with HTTP/2 push + preload

**Push**

GET http://a.com/index.html

**Page will need to load more resources**

**Push + Preload**

GET http://a.com/index.html

**preload:** http://c.com/img2.jpg

GET http://c.com/img2.jpg

**Page will need to load more resources**
Page loads with HTTP/2 push + preload

**Push**

GET http://a.com/index.html

Page will need to load more resources

**Push + Preload**

GET http://a.com/index.html

Preload: http://c.com/img2.jpg

GET http://c.com/img2.jpg

Page will need to load more resources

**Policy:** specification of which objects to push/preload from which parents
Static policies don't generalize
Static policies don't generalize
Static policies don't generalize

GET index.html
Static policies don't generalize

GET index.html

GET dynamic.js
Static policies don't generalize

GET index.html

Compute delay

GET dynamic.js
Static policies don't generalize

GET index.html

GET dynamic.js

Compute delay
Static policies don't generalize
Static policies don't generalize

Found best policy for each environment and applied it to a different environment

**Baseline environment**: [2.2 GHz CPU, 100 ms, 12 Mbps, cold cache]
**Best Policy**: brute force for that condition
**No Push/Preload**: default browser
Static policies don't generalize

Found best policy for each environment and applied it to a different environment

Policies from the wrong environment **forego 18-30%** of potential benefits

Performance can **degrade over 20%** compared to default load

**Baseline environment**: [2.2 GHz CPU, 100 ms, 12 Mbps, cold cache]

**Best Policy**: brute force for that condition

**No Push/Preload**: default browser
Alohamora

Generates dynamic push + preload policies on-the-fly

Page dependency graph
Client device
Network characteristics
Cache contents

Input

Alohamora

Push/preload policy

Output
Alohamora

Generates dynamic push + preload policies on-the-fly

Page dependency graph
Client device
Network characteristics
Cache contents

Input

Alohamora

Output

Push/preload policy

Online Inference

Offline Training

Reinforcement Learning
Reinforcement Learning (RL)

Agent

\[ a_t = \max P_A(s_t) \]

Observe state \( s_t \)

Environment

\[ s_t \in E \]

Take action \( a_t \)

Receive reward \( r_t = R(a_t, s_t) \)

**Goal:** Find \( P_A \) that produces \( \max \sum_t r_t \)
RL in Alohamora

State $s_t$ → Agent → Reward $r_t$ → Environment

Action $a_t$
RL in Alohamora

**State** $S_t$

**Agent**

**Reward** $r_t$

**Action** $a_t$
RL in Alohamora

State $S_t$

Agent

Reward $r_t$

Push/preload action
push a
preload b
preload c
push d

Push/preload policy

Action $a_t$
RL in Alohamora

**State** $S_t$

**Agent**

**Reward**: web performance metric: page load time or speed index

**Push/preload action**
- push a
- preload b
- preload c
- push d

**Push/preload policy**
RL in Alohamaura

**State** $S_t$
- Network bandwidth
- Network latency
- Device CPU speed
- Browser cache contents
- Page dependency graph

**Agent**

**Reward**: web performance metric: page load time or speed index

**Push/preload action**
- push a
- preload b
- preload c
- push d

**Push/preload policy**
- ...
- ...
- ...

**Action** $a_t$
Training challenges
Training challenges

Challenge 1: Too many pages to consider
Training challenges

Challenge 1: Too many pages to consider

Structurally similar pages ➔
similar push/preload policies
Training challenges

Challenge 1: Too many pages to consider

- Articles
- Subsections
- Blog posts

Structurally similar pages ➔ similar push/preload policies
Training challenges

Challenge 1: Too many pages to consider

Structurally similar pages ➔
similar push/preload policies

Solution: train a single model per page cluster
Training challenges

Challenge 1: Too many pages to consider

Challenge 2: Too many policies per page

Solution: train a single model per page cluster
Training challenges

Challenge 1: Too many pages to consider

Structurally similar pages ➔ similar push/preload policies

Solution: train a single model per page cluster

Challenge 2: Too many policies per page

Number of policies: exponential in # of page objects

Testing time per policy: 10 seconds on average

New York Times homepage: 10 days to train for a single environment
Training challenges

Challenge 1: Too many pages to consider

- Articles
- Subsections
- Blog posts

Structurally similar pages ➔ similar push/preload policies

Solution: train a single model per page cluster

Challenge 2: Too many policies per page

Number of policies:
exponential in # of page objects

Testing time per policy:
10 seconds on average

New York Times homepage:
10 days to train for a single environment

Solution: simulate page loads without a real device/browser
Generalization

Goal
Avoid training one model per page

Challenge
Find clusters that optimize policy performance and number of clusters

Insight
Policy efficacy is correlated with resource utilization ➔ page structure
Generalization

**Goal**
Avoid training one model per page

**Challenge**
Find clusters that optimize policy performance and number of clusters

**Insight**
Policy efficacy is correlated with resource utilization ➔ page structure

Page dependency tree captures resource utilization
Generalization

**Goal**
Avoid training one model per page

**Challenge**
Find clusters that optimize policy performance and number of clusters

**Insight**
Policy efficacy is correlated with resource utilization ➔ page structure

Page dependency tree captures resource utilization

\[
\begin{bmatrix}
D(T_1, T_1) & \cdots & D(T_1, T_j) & \cdots & D(T_1, T_n) \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
D(T_i, T_1) & \cdots & D(T_i, T_j) & \cdots & D(T_i, T_n) \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
D(T_n, T_1) & \cdots & D(T_n, T_j) & \cdots & D(T_n, T_n)
\end{bmatrix}
\]

Distance matrix where \( D \) is the edit distance between two trees
Generalization

**Goal**
Avoid training one model per page

**Challenge**
Find clusters that optimize policy performance and number of clusters

**Insight**
Policy efficacy is correlated with resource utilization → page structure

Object Size
Execution Time
Request Delay
Server Proc Delay

1 HTML
2 CSS
3 JS
4 IMG
5 IMG
6 IMG

Page dependency tree captures resource utilization

Distance matrix where $D$ is the edit distance between two trees
Generalization

Policies within the same cluster achieve **82-100%** of benefits

Policies applied across clusters forego **31-64%** of benefits

We can train **one model** using the "median page" **per cluster**
Page Load Simulator

**Goal**
Predict policy performance on a page load in a specific environment

**Challenge**
Be accurate across arbitrary environments, modeling cross-stack constraints

**Insight**
Model the browser as a multi-queue system and modulate unshaped loads
Page Load Simulator

**Goal**
Predict policy performance on a page load in a specific environment

**Challenge**
Be accurate across arbitrary environments, modeling cross-stack constraints

**Insight**
Model the browser as a multi-queue system and modulate unshaped loads
Page Load Simulator

**Goal**
Predict policy performance on a page load in a specific environment

**Challenge**
Be accurate across arbitrary environments, modeling cross-stack constraints

**Insight**
Model the browser as a multi-queue system and modulate unshaped loads
Page Load Simulator

**Goal**
Predict policy performance on a page load in a specific environment

**Challenge**
Be accurate across arbitrary environments, modeling cross-stack constraints

**Insight**
Model the browser as a multi-queue system and modulate unshaped loads
Page Load Simulator

**Goal**
Predict policy performance on a page load in a specific environment

**Challenge**
Be accurate across arbitrary environments, modeling cross-stack constraints

**Insight**
Model the browser as a multi-queue system and modulate unshaped loads
Page Load Simulator

**Goal**
Predict policy performance on a page load in a specific environment

**Challenge**
Be accurate across arbitrary environments, modeling cross-stack constraints

**Insight**
Model the browser as a multi-queue system and modulate unshaped loads

Object Size
Execution Time
Request Delay
Server Proc Delay

1. HTML
2. CSS
3. JS
4. IMG
5. IMG
6. IMG

Delayed

- 2
- 3
- 4

Downloading

- 2
- 3

Downloaded

- 1

Delayed

- 4

Downloaded

- 1

Delayed

- 5
- 6

Downloaded

- 1
- 2
- 3

Delayed

- 5
- 6

Downloaded

- 1
- 2
- 3
- 4
Page Load Simulator

**Goal**
Predict policy performance on a page load in a specific environment

**Challenge**
Be accurate across arbitrary environments, modeling cross-stack constraints

**Insight**
Model the browser as a multi-queue system and modulate unshaped loads

**Speed**
Time to load page: **10s → 5ms.**

**Faithfulness**
Median error: **0.4%.**

**Time to train:** **10 days → 20 mins**
System design (online inference)
System design (online inference)
System design (online inference)

<table>
<thead>
<tr>
<th>Alohamora policy generator</th>
<th>Web server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alohamora (online)</td>
<td>Web property</td>
</tr>
</tbody>
</table>
System design (online inference)
System design (online inference)

Alohamora (offline)
- Clustering + model training
- Dependency graph generation

Alohamora policy generator

Alohamora (online)

Web server

Web property
System design (online inference)

GET /index.html
+ Network info
+ CPU speed
+ Cache contents

Alohamora (offline)
- Clustering + model training
- Dependency graph generation

Alohamora policy generator

Alohamora (online)

Web server

Web property
System design (online inference)

GET /index.html
+ Network info
+ CPU speed
+ Cache contents
System design (online inference)

Alohamora (offline)
- Clustering + model training
- Dependency graph generation

Alohamora policy generator

Web server

GET /index.html
- Network info
- CPU speed
- Cache contents

Web property
System design (online inference)

GET /index.html
+ Network info
+ CPU speed
+ Cache contents

Response w/ push + preload policy

Alohamora (offline)
- Clustering + model training
- Dependency graph generation

Alohamora policy generator

Alohamora (online)

Web server

Web property
Results

24-61% improvement in PLT, 19-48% improvement in SI, 1.4 to 3.6x faster than Vroom
**Results**

**24-61% improvement in PLT, 19-48% improvement in SI, 1.4 to 3.6x faster than Vroom**

Validated performance in real-world settings
Results

24-61% improvement in PLT, 19-48% improvement in SI, 1.4 to 3.6x faster than Vroom

Validated performance in real-world settings

Alohamora never degrades page load performance
Conclusion

- Alohamora adapts push/preload policies based on the target page and execution environment
- Ensures practicality via:
  - Cross-page clustering
  - Cross-stack page load simulation
- Alohamora can speed up page loads up to 61%, and never slows them down.

https://github.com/nkansal96/alohamora