Making JavaScript WARC

David Calano

CS 895 - Web Archiving Forensics

Week 5

September 26, 2022

Invasion of the Byte Snatchers!

- Median of 44% bytes devoted to JS (2020)
- 4x increase in JavaScript utilization
- Trend towards complexity
- Significant portion of JavaScript is irrelevant to web archiving

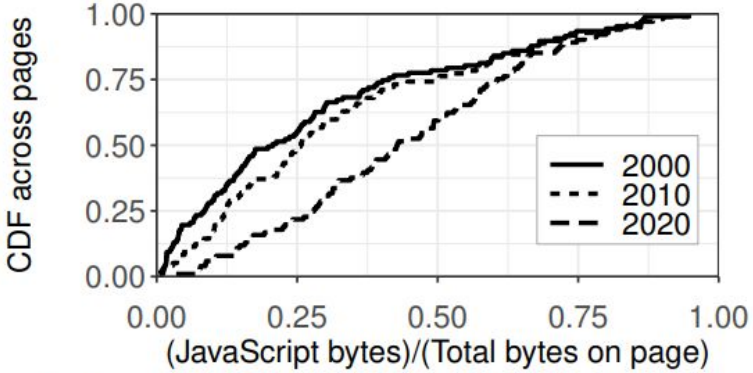


Figure 1: **Across the landing pages of 300 sites, distribution of fraction of bytes on the page accounted for by JavaScript.**

https://www.usenix.org/system/files/osdi22-goel.pdf

What’s the Problem?

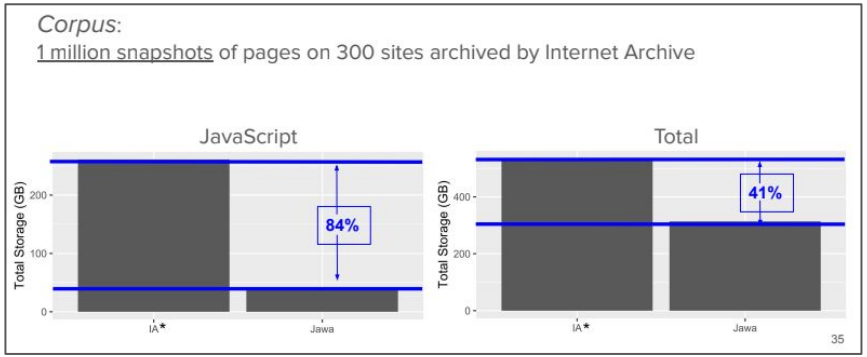
Problems with Web Archives

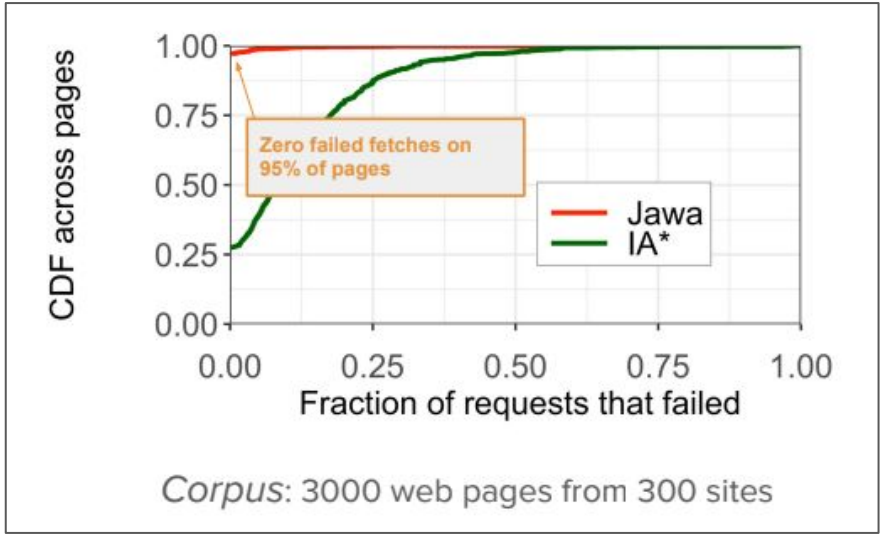
Limited page snapshots	Poor Page Fidelity
 2010-2020: **4x** increase in **JavaScript**  **Per page snapshot expensive**	**JavaScript non-determinism**: Date, math.random, user-agent  

https://goelayu.github.io/files/jawa-poster.pdf

What Can a “JavaScript-Aware Web Archive” Do For You?

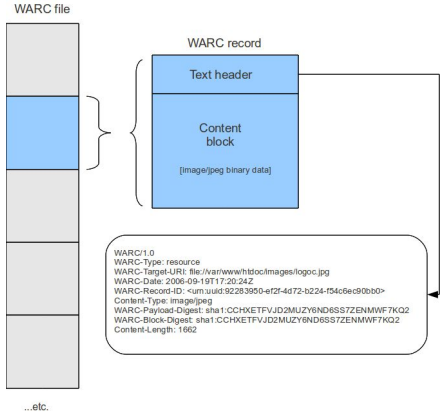
- Lower storage and data costs
- Reduced overhead
- Increased archival capacity
- Future proofing
- Increased visual fidelity





https://www.usenix.org/sites/default/files/conference/protected-files/osdi22_slides_goel.pdf

WARC vs Jawa



https://wiki.archivematica.org/Significant_characteristics_of_websites

Crawl index		
	Key	*Value*
IA	URL	List of (content hash, WARC file ID) tuples
Jawa	(URL, content hash)	List of (start byte offset, end byte offset, WARC file ID) tuples

Serving index		
	Key	*Value*
IA	(URL, timestamp)	(WARC file ID, byte offset)
Jawa	(URL, timestamp)	List of (WARC file ID, byte offset) tuples

Table 2: **Comparison of indices maintained by IA and Jawa.**

https://www.usenix.org/system/files/osdi22-goel.pdf

WARC

- Stores deduplicated files for each page snapshot
- File URLs rewritten
- Deterministic content extraction

Jawa

- JavaScript stored in partitioned byte offsets
- JavaScript served from compiled partitions
- Index I/O efficiency improvements
- Non-deterministic content extraction

Datasets

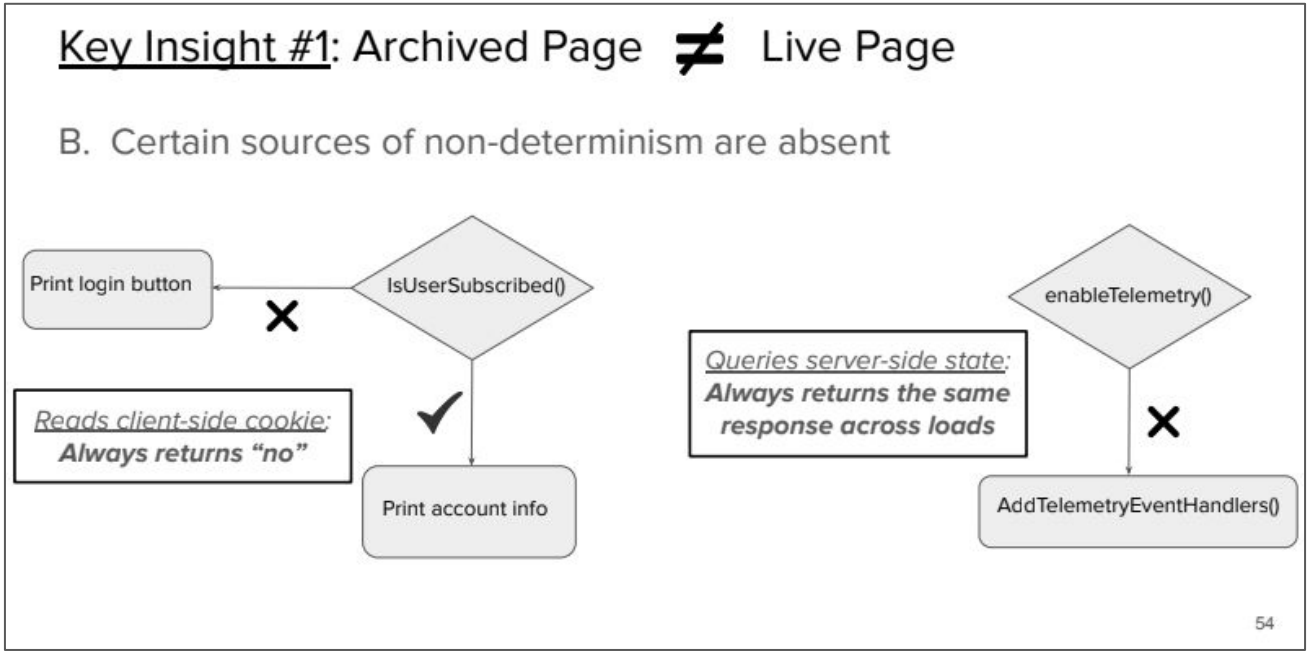
$Corpus_{3K}$

- 300 random websites sampled across 3 Alexa top site rank stratifications
- Single snapshot selected from Sept. 2021
- 1 landing page, 9 internal pages

$Corpus_{1M}$

- 3,500 snapshots for each Internet Archive page selected from Sep. 2020

The Past and Present Web



https://www.usenix.org/sites/default/files/conference/protected-files/osdi22_slides_goel.pdf

Client-side Variance

Improve fidelity: Fix sources of JavaScript variation

Randomness (date, math.random, performance)		Client characteristics	
Insight	**Solution**	**Insight**	**Solution**
1-1 mapping between diverged URLs	*Use server-side matching techniques* a.com/b.js?ts=5467 ⇕ a.com/b.js?ts=8967	Key contributor to URL divergence.	*Enforce same values across loads* Crawling device ⇕ Replay device APIs: - User-agent - Screen size - Operating system - Geolocation - ....

https://goelayu.github.io/files/jawa-poster.pdf

Prune the Trees, Not the Forest

Reduce storage overhead: Remove non-functional code

No back-end origin server	Insights	Solution
Remove code that interacts with back-end server	❶ Compartmentalized into files Header.js, footer.js, login.js ❷ Hosted on third-party domains www.forbes.com <script src="profiles.com/login.js"> </script>	*Filter list to identify URLs of non-functional scripts* Rules: - "cdn.onesignal.com" - "cdn.pushly.com" - "cdn.parsley.com" - ...

Reduce storage overhead: Remove unused code

Remove (unused) alternate code flows	Preserve code for event handlers	
	Insight	Solution
Insight: Absent sources of non-determinism: - Server-side state - Client APIs return values - Client-side state *Solution: Dynamically identify used code (during page load)*	Order of execution, and inputs to events do not impact code coverage	*Trigger each event once with default inputs* + *Track executed lines of code*

https://goelayu.github.io/files/jawa-poster.pdf

Testing

- ArchiveBox used as basis for testing
  - Proprietary Internet Archive code and headless Chrome engine
- Custom crawler to inject blocking and code analysis capabilities
- Decreased I/O calls due to filtering
- No performance impact from DRP APIs

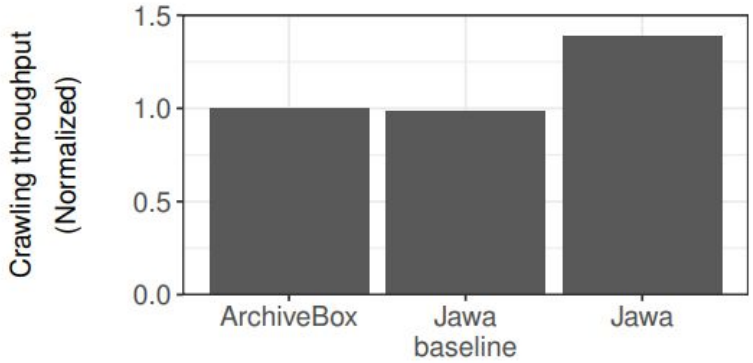
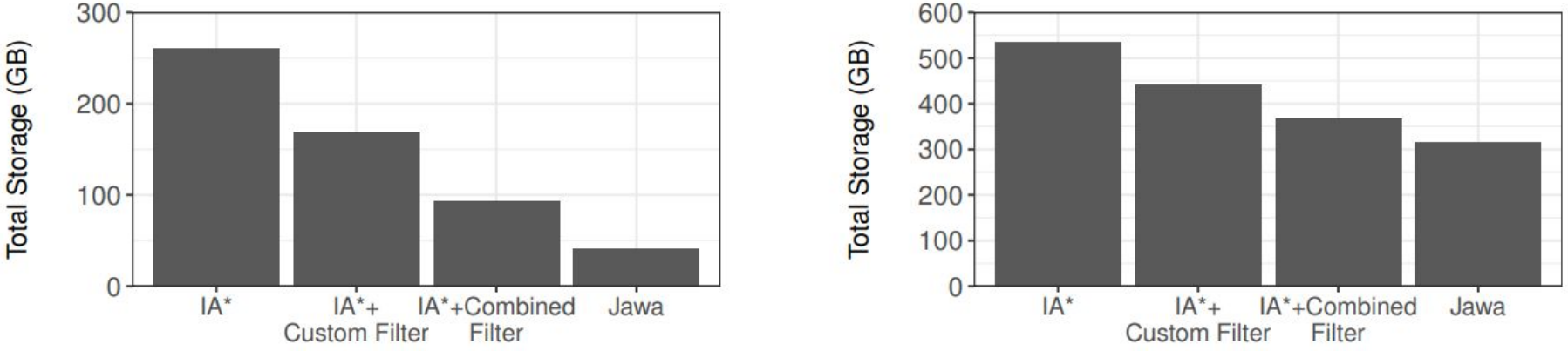


Figure 9: **Comparison of crawling throughput, normalized to that offered by ArchiveBox.**

https://www.usenix.org/system/files/osdi22-goel.pdf

Code Reduction



(a) JavaScript resources

(b) All resources

Figure 7: **Total storage necessary to store corpus of 1 million page snapshots.**

https://www.usenix.org/system/files/osdi22-goel.pdf

Pros and Cons

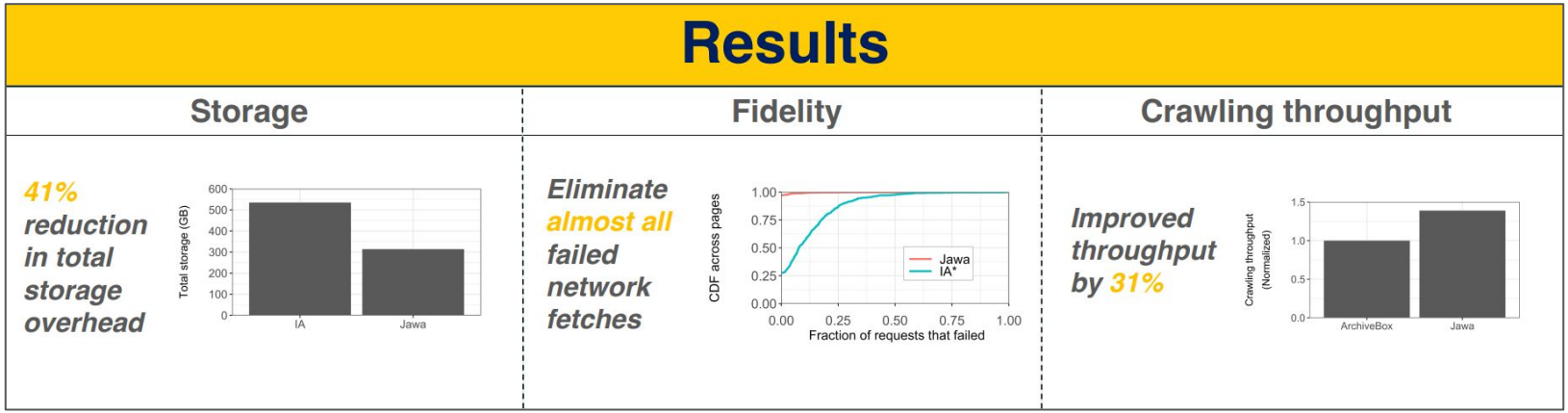
- Amazing promise
- Great presentation and ephemera
- Modular design
- Large sampling
- Reproducible results
- Open code and data

- Adoption
- Integration
- Small sampling
- No improvements with regard to assets
- Static filter list
- Shifting landscape

- Link rot on project page

Takeaways

- JavaScript must ultimately be addressed in web archives
  - Largest issues related to non-deterministic nature
- Non-breaking improvements can presently be implemented
- Huge potential for cost and overhead savings



https://goelayu.github.io/files/jawa-poster.pdf