Managing Broken URLs in Federated Metadata

Tien-Dung Le and Elena Shulman
September 2010

Disclaimer

- The work presented here is partially supported by the European Community eContent*plus* programme - project ASPECT: Adopting Standards and Specifications for Educational Content (Grant agreement number ECP-2007-EDU-417008).
- The authors are solely responsible for the content of this presentation. It does not represent the opinion of the European Community and the European Community is not responsible for any use that might be made of information contained therein.

LRE and Broken URLs

- Issues and solutions in managing federations of learning object repositories
- LRE Federation
  - Content providers expose metadata
  - LRE provides unified access by compiling a digital catalog
    - Metadata contains URLs where learning objects can be retrieved
  - LRE does not host the objects
  - Outdated metadata exposed to the LRE can lead to broken URLs

Objective

- Effectively detect broken URLs
- Automate communication with content providers
- Allow for greater flexibility for LRE to insure quality user experience

Presentation Roadmap

- Background of problem
- Broken URL handlings system and heuristic algorithm
- Guidelines to support collaboration and communication in a federation
- Outstanding issues and future plans

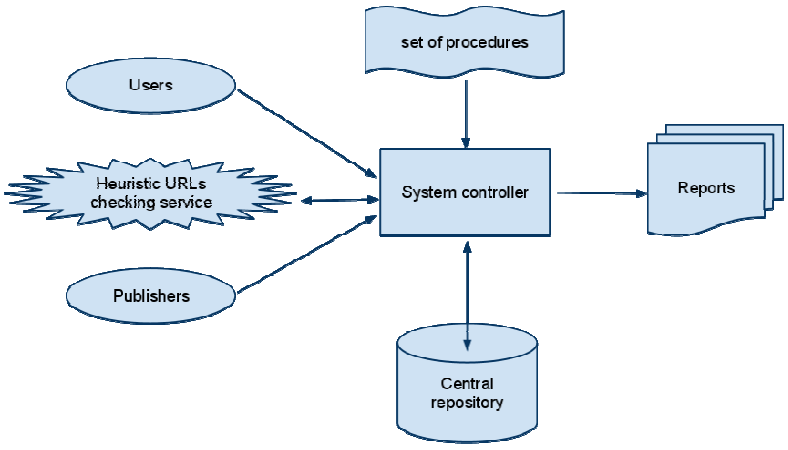
Background

- Broken URLs can become significant problems
  - User satisfaction impacted by broken URLs
- Other methods of addressing problem
  - User reports of broken links
    - Individualized solution
  - Regular harvesting of metadata
    - Relies on the content providers to update their metadata
  - Systematically check all URLs in catalog
    - Full check of more than 200,000 records taxes system for extended period of time (2 days)
    - Appears as 'unfriendly' on content provider systems

Solution: Broken URL Handling System

- Feasibility in detecting broken URLs demonstrated
- Mechanisms to trigger ameliorative actions
- Avoids detection techniques that can appear as denial of service attacks

System Architecture



Broken URL Detection (1)

- Simple check
  - Check all URLs one by one
  - Long time (more than 48 hours)
  - 'Denial-of-service attack'

Broken URL Detection (2)

- Solving the speed performance by
  - Cloud computing?
  - Better network bandwidth?
  - Heuristic algorithm?
  - What else?

Broken URL Detection (3)

- Cloud computing
  - JPPF with 4 machines in a local network
  - More than 24 hours
    - Internal processing takes less than 30 minutes
    - Network communication (between system and learning objects' network hosts) takes all the rest
    - It took almost same amount of time using only 1 machine
    - Cloud computing is faster mainly because URLs are checked in a random order

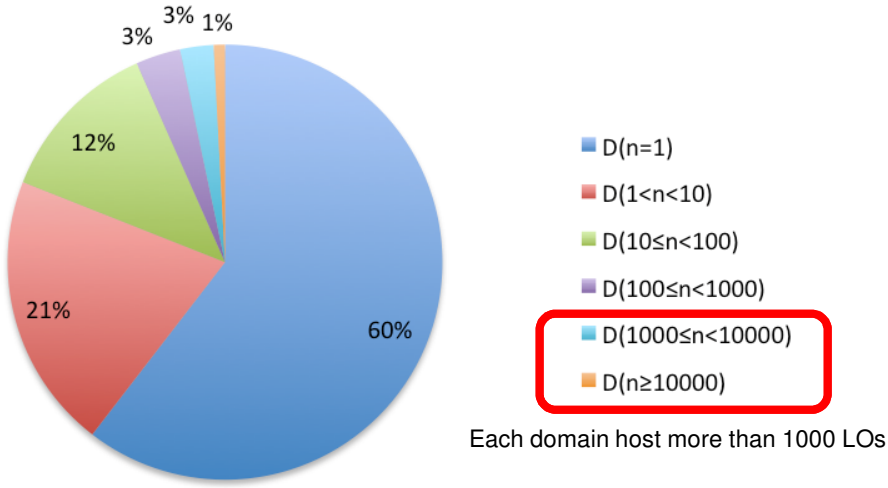
Broken URL Detection (4)

- But why random order made a difference?
  - Maximum HTTP requests per second
- Better network bandwidth?
  - Expensive
  - Might not work well (see Learning Object Location Distribution)
- Heuristic algorithm

Our Beliefs

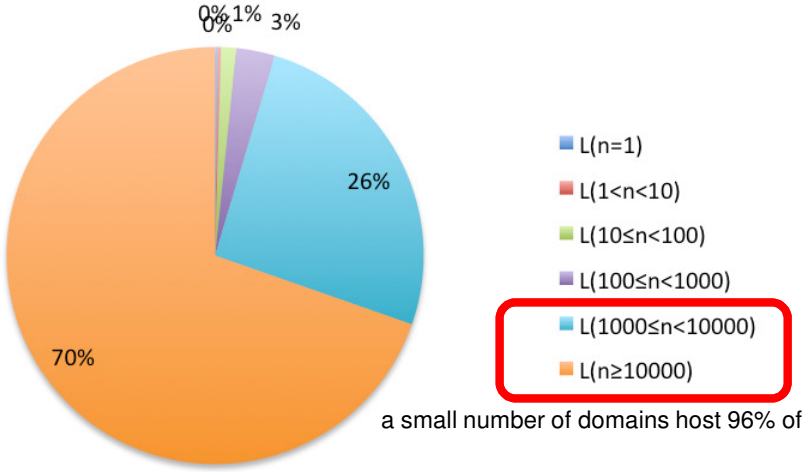
- Probability of a broken URL
  - url_1, $url_2 \in domain_1$, $url_3 \in domain_2$
  - A, B, C are events of url_1, url_2, url_3 are broken
  - Probability(B | A) ≥ Probability(C | A)
- Reasons
  - Domain is unavailable
  - Folder on the domain is moved or deleted

Learning Object Domain Distribution



Each domain host more than 1000 LOs

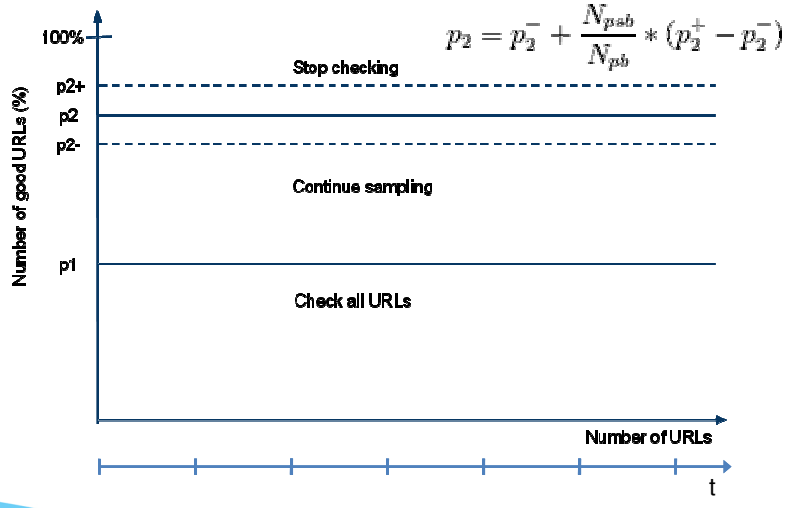
Learning Object Location Distribution



a small number of domains host 96% of all LOs

Sampling Plan

$$p_2 = p_2^- + \frac{N_{psb}}{N_{pb}} * (p_2^+ - p_2^-)$$



Algorithm 1 Heuristic checking algorithm

```
1: procedure CHECK
2:     for all domain ∈ domains do
3:         Re-check all broken URLs on domain
4:         Calculate p2 using equation 2
5:         sampling = true
6:         while (sampling and (domain has unchecked URLs)) do
7:             Take a group of URLs for checking
8:             Check this group
9:             Update p using equation 3
10:            if p < p1 then
11:                Check all other URLs
12:                sampling = false
13:            else
14:                if p2 ≤ p then
15:                    sampling = false        ▷ assume that all other URLs are good
16:                end if
17:            end if
18:        end while
19:    end for
20: end procedure
```

All selected URLs at a "time stamp" are checked in batch mode

Experiment (1)

- June 23rd, 2010
  - 45711 / 236763 broken URLs
  - Params: $G = 100,\ p_1 = 50\%, p_2^- = 90\%, p_2^+ = 95\%$

Experiment (2)

Table 1. Results

Run	Selected URLs (in number and in percentage of the total number of URLs)	Broken URLs (in number and in percentage of the total number of broken URLs)	Rate (number of broken URLs /number of selected URLs)
1	41102 = 17,36%	33587 = 73,48%	81,72%
2	105126 = 44,40%	43388 = 94,92%	41,27%
3	115005 = 48,57%	45101 = 98,67%	39,22%
Overall	710289 = 36,78% (in average)	45101 = 98,67% (max)	54,07% (in average)

Procedures to Correct Broken URLs

- Heuristic checking algorithm deployed to test discrete domains
  - This allows for multiple-recheck without taxing systems
- Possible to automate communication with providers using a scheduling sequence
  - Automate initial report of broken URLs found
  - Seven days after report, a re-check is possible
    - If not corrected – initiate attempts to communicate between LRE service managers and repository managers
  - Thirty days after initial discovery of broken URL another re-check
    - Records with broken URL are removed from search
    - Automated notification to repository

Related Works

- Broken link detection
  - Based on the relationship between the resources
- Proactive solutions
  - Permanent identifiers (such as PURL)
  - Local copies

Future Works

- Broken URL filter integrated in
  - OAI-PMH harvester
  - SPI service
- Even better Broken URL detection algorithm
  - Based on URL string similarity / distance
    - When a folder is moved, deleted

Thank you

Broken URL detection using Adaptive Sampling Plan

- Adaptive plan
- URL string similarity
- A result

– First run with an assumption that all URLs are good.
  - Number of selected URLs = 118442
  - Number of broken URLs = 44925

– Second run using knowledge from the first run
  - Number of selected URLs = 57077
  - Number of broken URLs = 45053

– Third run using knowledge from the second run
  - Number of selected URLs = 56107
  - Number of broken URLs = 45094