IRLbot: Scaling to 6 Billion Pages and Beyond

Presented by Xiaoming Wang

Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov

Internet Research Lab
Computer Science Department
Texas A&M University

April 25, 2008

Agenda

- Introduction and challenges — we are here
- Background
- Overcoming the challenges
  - Scalability
  - Spam and reputation
  - Politeness
- Experiments
- Conclusion

Introduction

- WWW has evolved from a handful of pages to billions of diverse objects
  - In January 2008, Google reported indexing 30 billion documents and Yahoo 37 billion (see paper for details)
- Search engines consist of two fundamental parts
  - Web crawlers
  - Data miners

our focus in this paper

- Challenges
  - Scalability
  - Spam avoidance
  - Politeness

Challenges – Scalability

- Inherent tradeoff between scalability, performance, and resource usage
  - Scalability: number of pages N the crawler can handle
  - Performance: speed S at which the crawler discovers the web as a function of N
  - Resource usage: CPU and RAM requirements Σ needed to download N pages at average speed S
- Previous research can satisfy any two objectives (i.e., large slow crawls, fast small crawls, or fast large crawls with unlimited resources)
- Our goal: achieve large N (trillions of pages) with fixed S $(1000+$ pages/sec) and modest Σ (single server)

Challenges – Spam Avoidance

- Experience shows that BFS eventually becomes trapped in useless content
  - The queue of pending URLs is filled with spam links and infinite auto-generated webs
  - The DNS resolver is overloaded with new hostnames that are dynamically created within a single domain
  - Crawler is "bogged down" in synchronized delay attacks of certain spammers
- Prior research has not attempted to avoid spam or even document its effect on the collected data
- Our goal: prevent low-ranked (spam) domains from impacting crawler performance

Challenges – Politeness

- Web crawlers often get in trouble with webmasters for slowing down their servers
  - Sometimes even reading the robots.txt file generates a complaint
- Per-website and per-IP hit limits are simple
  - However, even with spam avoidance, the entire RAM eventually gets filled with URLs from a small set of hosts (e.g., ebay, blogspot) and the crawler simply chokes on its politeness
- Previous algorithms do not address this issue
- Our goal: crawl large legitimate sites without stalling

Agenda

- Introduction and challenges
- Background

we are here

- Overcoming the challenges
  - Scalability
  - Spam and reputation
  - Politeness
- Experiments
- Conclusion

Computer Science, Texas A&M University

Background – Crawler Objectives

- Ideal task is to start from a set of seed URLs Ω_0 and eventually crawl the set of all pages Ω_∞
- Crawler may reorder URLs to achieve some “good” coverage of useful pages $\Omega_U \subseteq \Omega_\infty$ in some finite amount of time
- We call an algorithm non-scalable if it
  1) imposes hard limits on any of the following metrics
     - Max. # pages per host, #hosts per domain, #domains in the Internet, #pages in the crawl

  2) is unable to maintain crawling speed when these metrics become arbitrarily large
- Scalability is an important objective

Background – Crawler Operation

- For each URL u in the queue Q of pending pages, the crawler downloads u's HTML page and extracts new URLs u_1, u_2, …, u_k
- For each u_i, the crawler verifies its uniqueness using URLseen and checks compliance with robots.txt using RobotsCache
- It then adds the passing URLs to Q and URLseen
- Last, it updates RobotsCache if necessary
- The crawler may also maintain a DNScache structure to reduce the load on the local DNS server

Related Work

- Largest crawl with a disclosed implementation: 473M HTML pages
- Fastest: 816 pages/sec

Crawler	Year	Crawl size (HTML pages)
WebCrawler [25]	1994	50K
Internet Archive [6]	1997	–
Mercator-A [14]	1999	41M
Mercator-B [23]	2001	473M
Polybot [27]	2001	120M
WebBase [7]	2001	125M
UbiCrawler [2]	2002	45M

Crawler	URLseen		RobotsCache		DNScache
	RAM	Disk	RAM	Disk	
WebCrawler [25]	database		–		–
Internet Archive [6]	site-based	–	site-based	–	site-based
Mercator-A [14]	LRU	seek	LRU	–	–
Mercator-B [23]	LRU	batch	LRU	–	–
Polybot [27]	tree	batch	database		database
WebBase [7]	site-based	–	site-based	–	site-based
UbiCrawler [2]	site-based	–	site-based	–	site-based

Agenda

- Introduction and challenges
- Background
- Overcoming the challenges
  - Scalability
  - Spam and reputation
  - Politeness
- Experiments
- Conclusion

we are here

Scalability

- One of the bottlenecks in web crawling is the disk I/O in checking URLseen
- A 6B page crawl requires verification of 394B URLs
  - Disk structures must be optimized to support the crawling rate
- Our solution is called Disk Repository with Update Management (DRUM)
- The purpose of DRUM is to allow for efficient storage of large collections of <key, value> pairs
  - Key is a unique identifier of some data (e.g., URL hash)
  - Value is arbitrary information attached to keys (e.g., URL text)

Scalability 2

- Three operations are supported
  - Check: verifies uniqueness of a key
  - Update: overrides the value if the key exists and otherwise adds a new entry
  - Check + update: performs both check and update in one pass through the disk cache
- Unlike prior methods, DRUM is based on disk bucket sort rather than variations of merge/insertion sort
  - For certain values of RAM size, DRUM achieves a linear number of disk reads rather than quadratic as in prior work
  - See paper for details

Scalability 3

- Overhead metric ω: # of bytes written to/read from disk during uniqueness checks of lN URLs
  - l is the average # of links per downloaded page
  - N is the # of crawled pages
- Assume that the average URL length is b bytes and R is the size of RAM allocated to URL checks
- Then ω can be split into a product two terms: $\alpha \times blN$
  - blN is the number of bytes in all parsed URLs
  - α is the number of times they are written to/read from disk
- Various methods differ in the first term only
  - Metric α may be a constant or a linear function of N

Scalability 4

- Theorem 1: The overhead of URLseen batch disk check is $\omega = \alpha b l N$ bytes, where for Mercator:

$$\alpha = \frac{2(2UH + pHlN)(H+P)}{bR} + 2 + p$$

  - and for Polybot:

$$\alpha = \frac{2(2Ubq + pbqlN)(b+4P)}{bR} + p$$

Both have $\alpha \sim N/R$

- Theorem 2: DRUM's URLseen overhead is $\omega = \alpha b l N$ bytes, where:

$$\alpha = \frac{8M(H+P)(2UH + pHlN)}{bR^2} + 2 + p + \frac{2H}{p}$$

Notice that $\alpha \sim N/R^2$

Scalability 5

- For certain values of R, DRUM achieves $N/R^2 \approx 0$ and thus α remains almost constant as N increases
- Overhead α for dynamically scaling disk size:

N	$R = 4$ GB		$R = 8$ GB	
	Mercator-B	DRUM	Mercator-B	DRUM
800M	4.48	2.30	3.29	2.30
8B	25	2.7	13.5	2.7
80B	231	3.3	116	3.3
800B	2,290	3.3	1,146	3.3
8T	22,887	8.1	11,444	3.7

Scalability 6

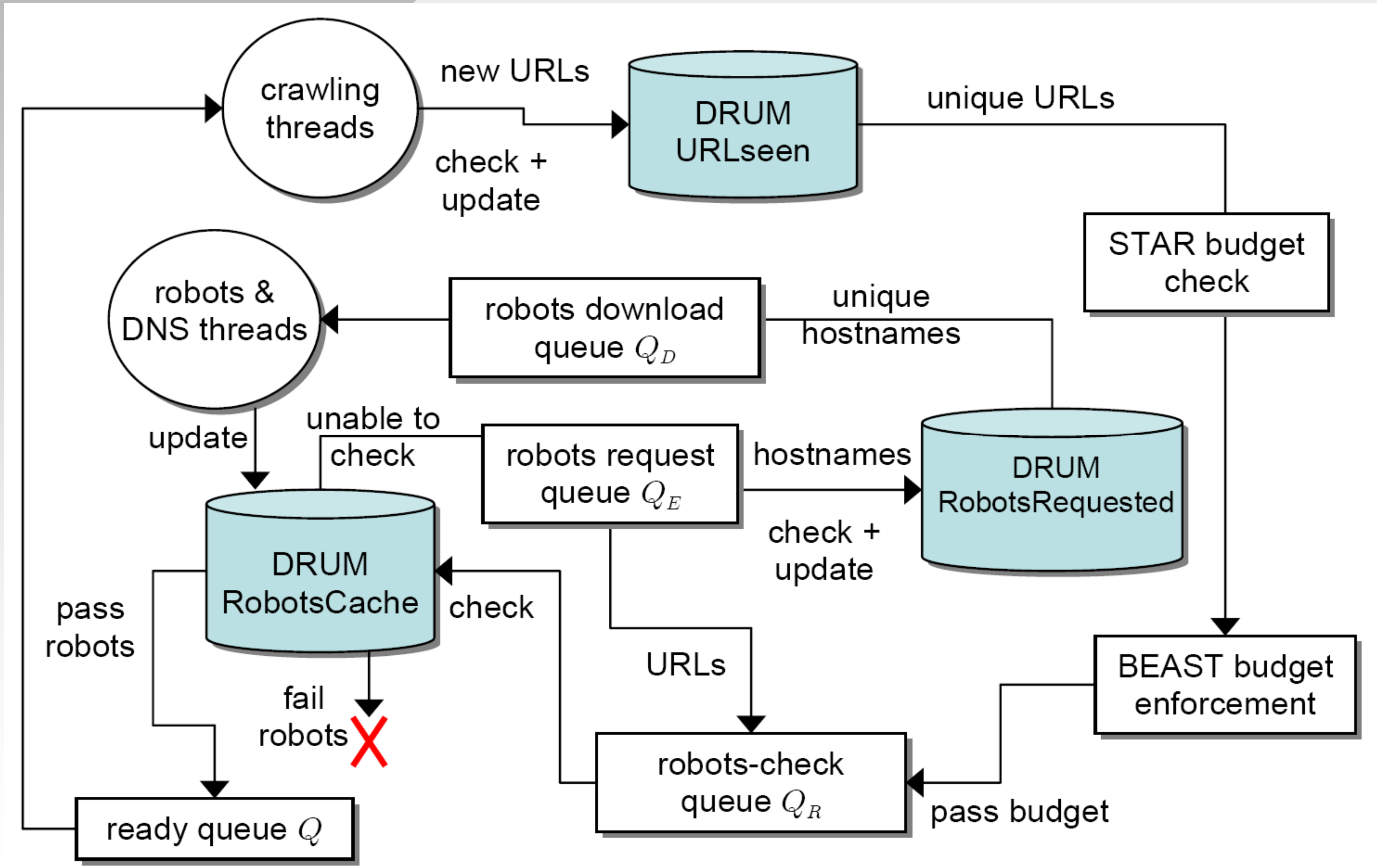
- Theorem 3: Maximum download rate (in pages/s) supported by the disk portion of URL uniqueness checks is

$$S = \frac{W}{\alpha b l}$$

average disk I/O speed

- For IRLbot, we set disk speed $W = 101.25$ MB/s and RAM size $R = 8$ GB
- Assuming $N = 8$ trillion pages, DRUM yields a sustained download rate of $S = 4{,}192$ pages/s
  - 10 DRUM servers and 10-gb/s link could give 100 billion pages per month
- For $N = 8$T, Mercator achieves an average rate of only 1.4 pages/s and Polybot 0.2 pages/s

IRLbot Organization



Agenda

- Introduction and challenges
- Background
- Overcoming the challenges
  - Scalability
  - Spam and reputation
  - Politeness
- Experiments
- Conclusion

we are here

Spam and Reputation

- In our experience, BFS is a poor technique in the presence of spam farms and infinite webs
- Quickly branching sites (1000+ links per page) are potential traps
  - Dominate the queue after 3 levels of BFS with 10^9 pages (exact impact depends on the seed URLs and crawls size)
- Simply restricting the branching factor or the maximum number of pages/hosts per domain is not a viable solution
  - A number of legitimate sites contain over 100 million pages and over 10 million virtual hosts
  - Yahoo reports 1.2B objects within its own domain

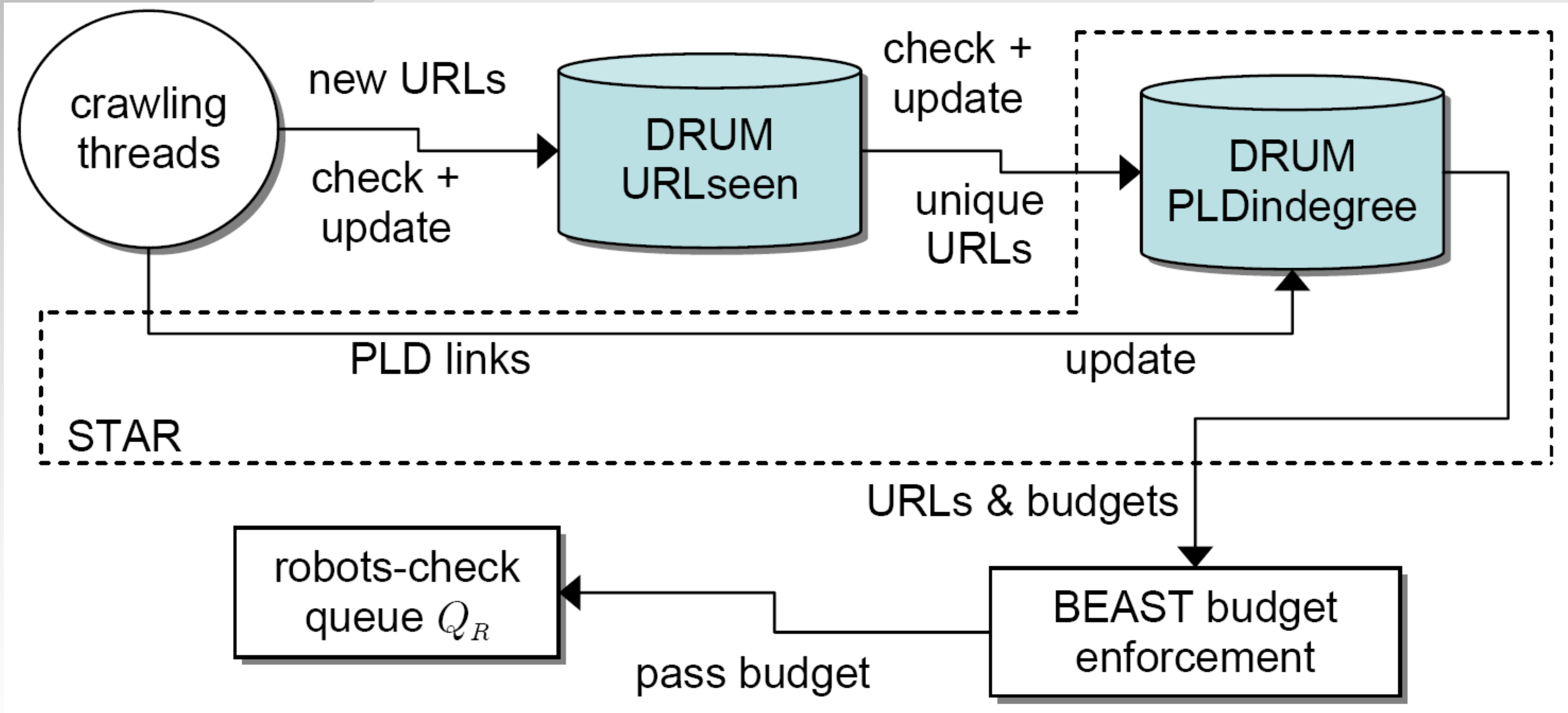
Spam and Reputation 2

- Computing traditional PageRank for each page could be prohibitively expensive in large crawls
  - In our case, over 41B pages in the webgraph
- However, the goal here is not to classify each page, but rather to understand which domains should be allowed to massively branch
- Spam can be effectively deterred by budgeting the number of allowed pages per pay-level domain (PLD)
  - PLDs are domains that one must pay for at some registrar
- PLD reputation is determined by its in-degree from resources that spammers must pay for, which in our case are other PLDs

Spam and Reputation 3

- This solution we call Spam Tracking and Avoidance through Reputation (STAR)
- Each PLD x has budget B_x that represents the number of pages that are allowed to pass from x (including all subdomains) to crawling threads every T time units
- Each PLD x starts with a default budget B_0, which is then dynamically adjusted as its in-degree d_x changes over time
  - Diverse strategies can be achieved by varying the adjustment function (e.g., more preference for popular/unpopular domains, equal preference, linear/non-linear scaling of B_x)
- DRUM is used to store PLD budgets and aggregate PLD-PLD link information

Operation of STAR



Agenda

- Introduction and challenges
- Background
- Overcoming the challenges
  - Scalability
  - Spam and reputation
  - Politeness
- Experiments
- Conclusion

we are here

Politeness

- Prior work has only enforced a certain per-host access delay τ_h seconds
  - Easy to crash servers that co-locate 1000s of virtual hosts
  - Thus, a per-IP limit τ_s is needed as well
- Low-ranked PLDs ($B_x = B_0$)
  - Keep $\tau_h = 40$ seconds and $\tau_s = 1$ second
- High-ranked domains ($B_x > B_0$)
  - Both τ_h and τ_s are scaled proportional to B_x to crawl URLs no slower than the rate at which they are admitted into RAM
- By controlling the coupling between PLD budgets and their crawl rate, we can avoid memory backlog

Politeness 2

- To admit URLs into RAM we have a method called Budget Enforcement with Anti-Spam Tactics (BEAST)
- BEAST does not discard URLs, but rather delays their download until it knows more about the PLD they belong to
- A naïve implementation is to maintain two queues
  - Q contains URLs that passed the budget check
  - Q_F contains those that failed
- After Q is emptied, Q_F is read and again split into two queues – Q and Q_F

Politeness 3

- Theorem 4: Lowest disk I/O speed (in bytes/s) that allows the naïve budget-enforcement approach to download N pages at fixed rate S is:

$$\lambda = 2Sb(L-1)\alpha_N$$

  - where

$$\alpha_N = \max\left(1, \frac{N}{E[B_x]V}\right)$$

- This theorem shows that $\lambda \sim \alpha_N = \Theta(N)$
- For IRLbot, $\lambda = 3.8$ MB/s for $N = 100$ million, $\lambda = 83$ MB/s for $N = 8$ billion, and $\lambda = 826$ MB/s for $N = 80$ billion

Politeness 4

- The correct implementation of BEAST rechecks Q_F at exponentially increasing intervals
- Suppose the crawler works with j queues Q_1, …, Q_j
  - Old URLs are read from Q_1 and sent to robots check and later to the crawling threads
  - New URLs are written to Q_2, …, Q_j based on their remaining budget (B_x URLs per queue)

wrap-around

- After Q_1 is emptied, the crawler moves to reading Q_2 and spreads new URLs between Q_3, …, Q_j, Q_1
- After it finally empties Q_j, the crawler re-scans Q_F and splits it into j additional queues Q_{j+1}, …, Q_{2j}
  - URLs violating the budget of Q_{2j} are placed into new Q_F

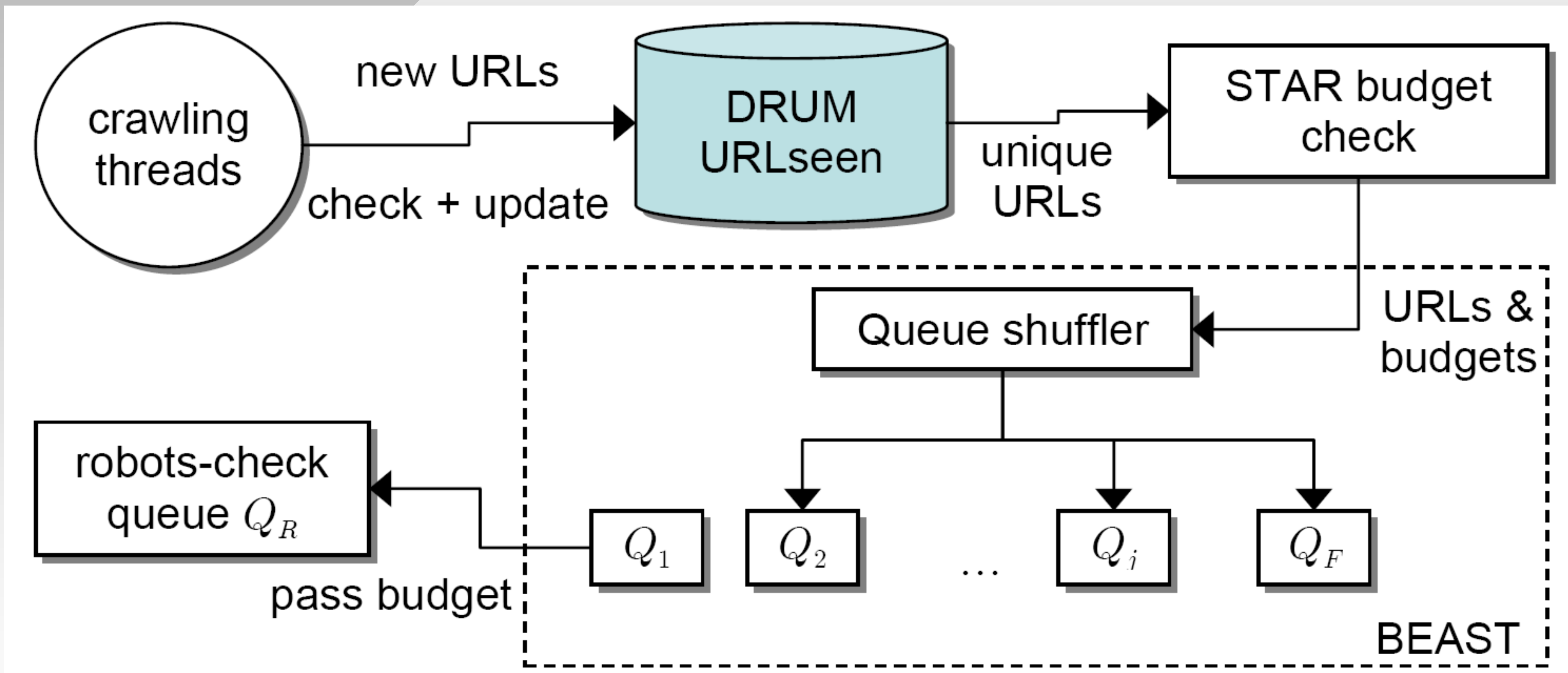
Politeness 5

- Theorem 5: Lowest disk I/O speed (in bytes/s) that allows BEAST to download N pages at fixed rate S is:

$$\lambda = 2Sb\left[\frac{2\alpha_N}{1+\alpha_N}(L-1)+1\right] \leq 2Sb(2L-1)$$

- For $N \to \infty$, disk speed $\lambda \to 2Sb(2L-1)$ = constant
  - It is roughly four times the speed needed to write all unique URLs to disk as they are discovered during the crawl
- For the example used in Theorem 4, BEAST requires $\lambda \leq 8.2$ MB/s regardless of crawl size N

Operation of BEAST



Agenda

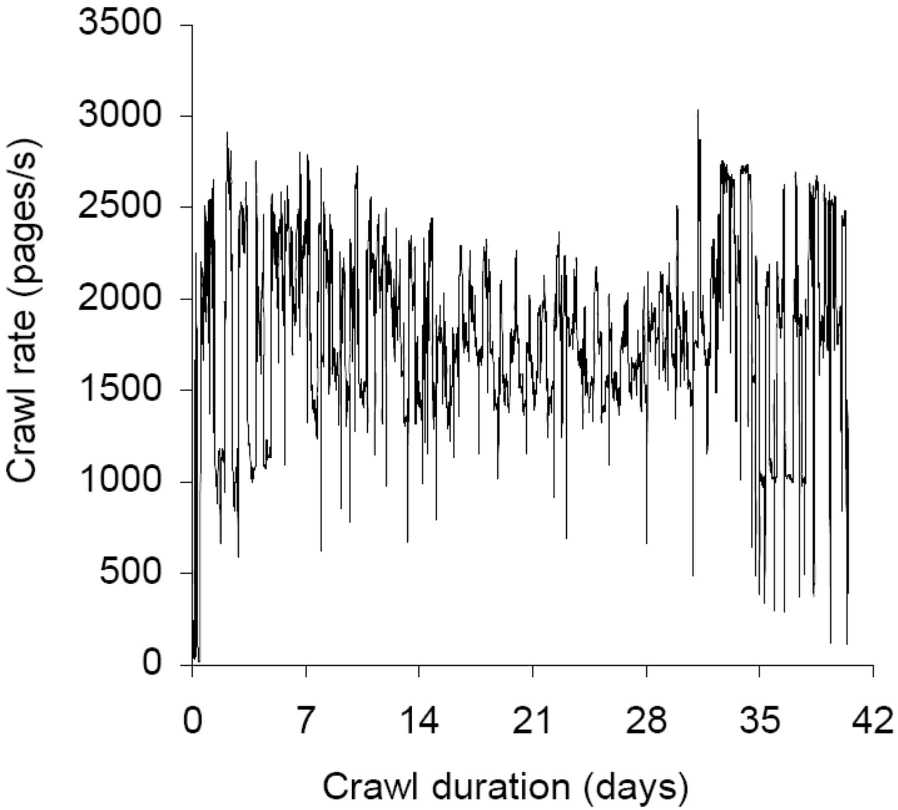
- Introduction and challenges
- Background
- Overcoming the challenges
  - Scalability
  - Spam and reputation
  - Politeness
- Experiments
- Conclusion

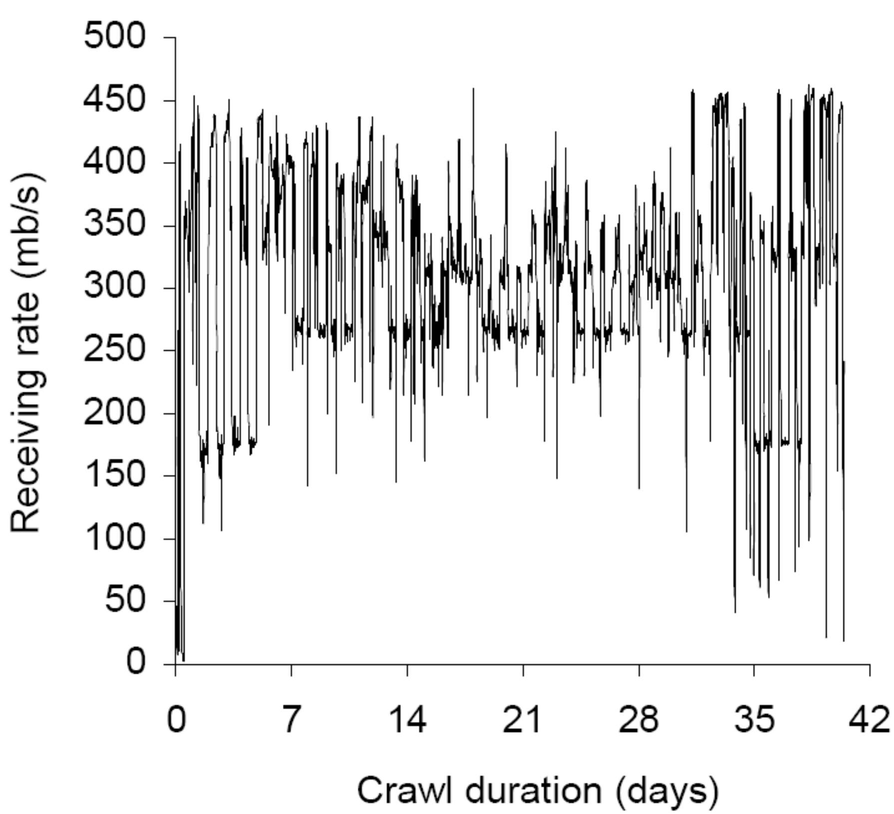
we are here

Experiments – Summary

- One quad-CPU AMD Opteron 2.6 GHz sever with 16 GB RAM, 24-disk RAID-5, and 1-gb/s link
- Active crawling period of 41 days in summer 2007
- IRLbot attempted 7.6 billion connections and received 7.4 billion valid HTTP replies
  - 6.3 billion responses with status code 200 and content-type text/html (964M errors and redirects, 92M non-HTML)
- Average download rate 319 mb/s ($1{,}789$ pages/s)
- Crawler received 143 TB of data (254 GB of robots.txt files) and sent 1.8 TB of HTTP requests
  - After decompression, 161 TB of HTML code went through the parser

Experiments – Summary 2





Crawl rate (pages/s)

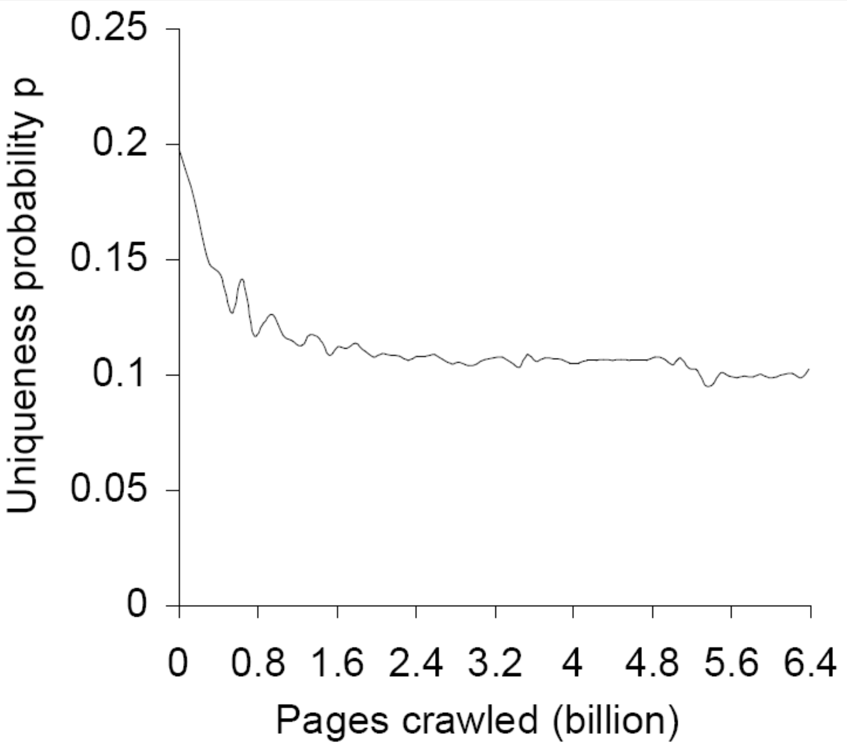
Receiving rate (mb/s)

Experiments – Summary 3

- IRLbot parsed out 394 billion links
  - Removing invalid URLs, this translates to 59 links/page
- URLseen contained 41 billion unique URLs
  - On average, 6.5 unique links per crawled page
  - Pages hosted by 641 million websites
- Discovered 89 million PLDs
  - PLD-PLD graph contained 1.8B edges
- Received responses from 117 million sites
  - Belonged to 33M PLDs
  - Were hosted on 4.2 million IPs

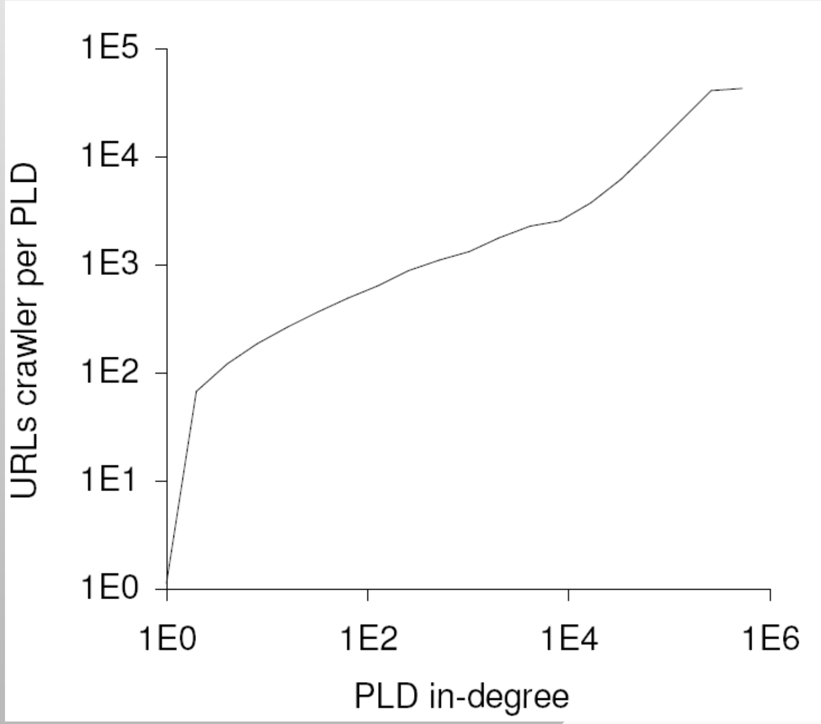
Experiments – Uniqueness Probability

- The probability of uniqueness p stabilized around 0.11 after the first billion pages were downloaded
  - p was bounded away from 0 even at $N = 6.3$ billion
- We certainly know there are ≥ 41 billion pages
  - The fraction of them with useful content and the number of additional pages not seen by the crawler are a mystery at this stage



Experiments – Effectiveness of STAR

- Top 10K ranked domains were given budget B_x linearly interpolated between 10 and 10K
  - All other PLDs received the default budget 10
- Figure shows that IRLbot succeeded at correlating PLD bandwidth allocation with their in-degree
- Manual examination reveals only 14 spam sites in the top 1000 PLDs



Experiments – Top Ranked PLDs

Rank	Domain	In-degree	PageRank	Pages
1	microsoft.com	2, 948, 085	9	37, 755
2	google.com	2, 224, 297	10	18, 878
3	yahoo.com	1, 998, 266	9	70, 143
4	adobe.com	1, 287, 798	10	13, 160
5	blogspot.com	1, 195, 991	9	347, 613
7	wikipedia.org	1, 032, 881	8	76, 322
6	w3.org	933, 720	10	9, 817
8	geocities.com	932, 987	8	26, 673
9	msn.com	804, 494	8	10, 802
10	amazon.com	745, 763	9	13, 157

Agenda

- Introduction and challenges
- Background
- Overcoming the challenges
  - Scalability
  - Spam and reputation
  - Politeness
- Experiments
- Conclusion

almost done

Conclusion

- This paper tackled the issue of scaling web crawlers to billions and even trillions of pages
  - Single server with constant CPU, disk, and memory speed
- We identified several bottlenecks in building an efficient large-scale crawler and presented our solution to these problems
  - Low-overhead disk-based data structures
  - Non-BFS crawling order
  - Real-time reputation to guide the crawling rate
- Future work
  - Refining reputation algorithms, accessing their performance
  - Mining the collected data