BUbiNG
Massive Crawling for the Masses

Paolo Boldi, Andrea Marino,
Massimo Santini, Sebastiano Vigna

Dipartimento di Informatica
Università degli Studi di Milano
Italy

Once upon a time UbiCrawler

- UbiCrawler was a scalable, fault-tolerant and fully distributed web crawler (*Software: Practice & Experience*, 34(8):711-726, 2004)
- LAW (Laboratory for Web Algorithmics) used it many times in the mid-2000s, to download portions of the web (.it, .uk, .eu, Arabic countries...)
- Based on this experience, LAW decided to write a new crawler, **10 years later**!

BUbiNG

Why a new crawler?

- **OPEN SOURCE!**
- Not so many **open-source** crawlers
  - *Heritrix* (Internet Archive; used for ClueWeb12)
  - *Nutch* (used for ClueWeb09)
- Not suitable to collect really **big** datasets
- Not so easily **extensible**
- **Distributed**? (Heritrix is not distributed; Nutch uses Hadoop)

Challenges

- **Pushing hardware to the limit**: Use massive memory and multiple cores efficiently
- Fill **bandwidth in spite of politeness** (both at host and IP level) => coherent time frame
- Producing **big** datasets in spite of limited hardware
- Making crawling and analysis **consistent**

- Completely configurable
- Extensible will little eff
- Integrated with spam detection (Hungarian Academy of Sciences)

High Parallelism

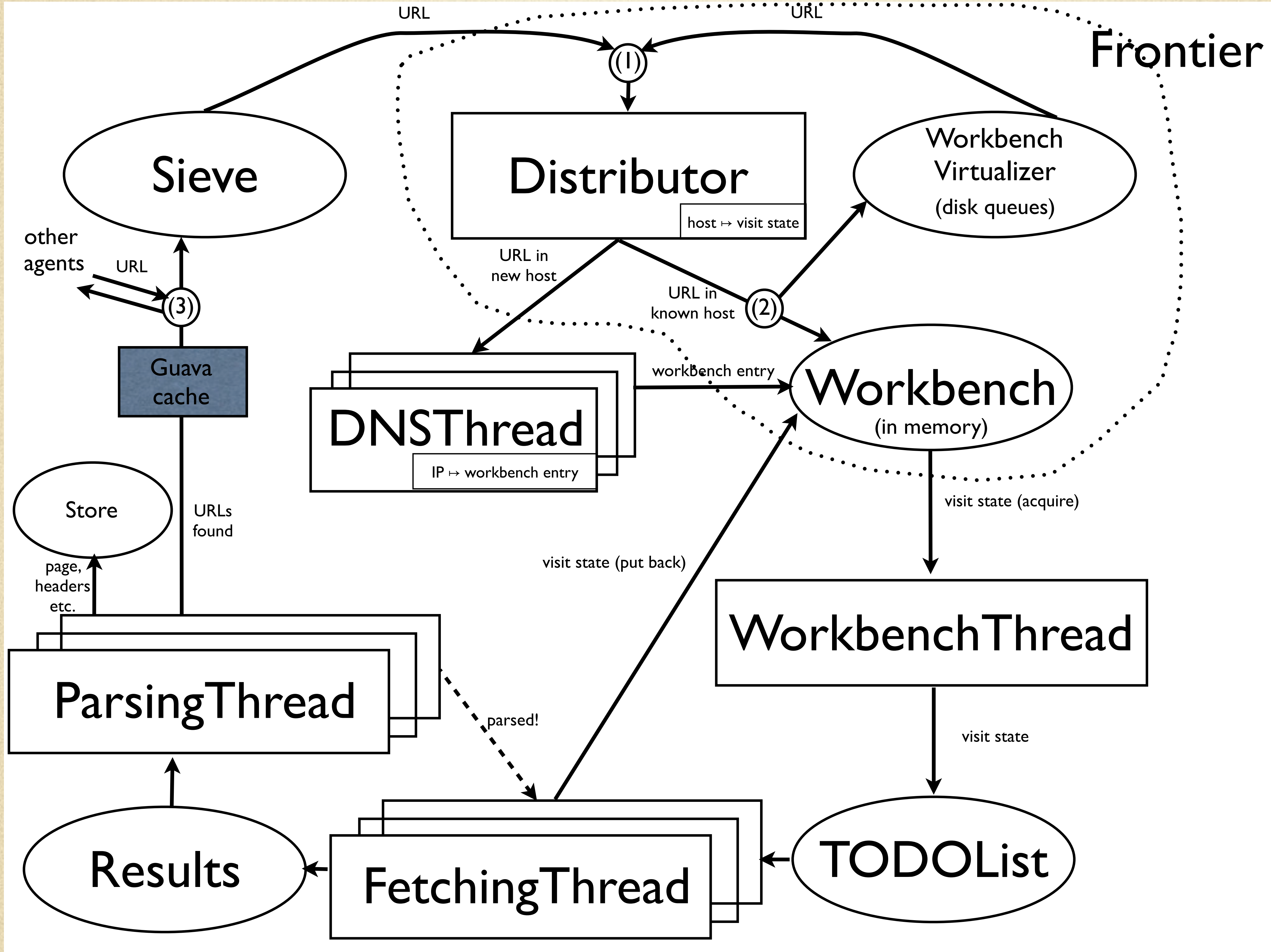
- We use **massively** multiple (like 5000) **fetching** threads
- Every thread handles a request and is I/O bound
- Parallel threads parse and store pages
- Slow data structures are sandwiched between **lock-free queues**

Fully Distributed

- We use JGroups to set up a view on a set of agents
- We use JAI4J, a thin layer over JGroups that handles job assignment.
- Hosts are assigned to agent using **consistent hashing**
- URLs for which an agent is not responsible are quickly delivered to the right agent

Near-Duplicates

- We detect (presently) near-duplicates using a fingerprint of a stripped page (stored in a *Bloom filter*)
- The stripping includes eliminating almost all tag attributes and numbers from text


Frontier
URL
URL
(1)
Sieve
Distributor
host ↦ visit state
Workbench
Virtualizer
(disk queues)
other
agents
URL
(3)
URL in
new host
URL in
known host
(2)
Guava
cache
DNSThread
IP ↦ workbench entry
workbench entry
Workbench
(in memory)
visit state (acquire)
Store
URLs
found
page,
headers
etc.
visit state (put back)
WorkbenchThread
ParsingThread
parsed!
visit state
Results
FetchingThread
TODOList

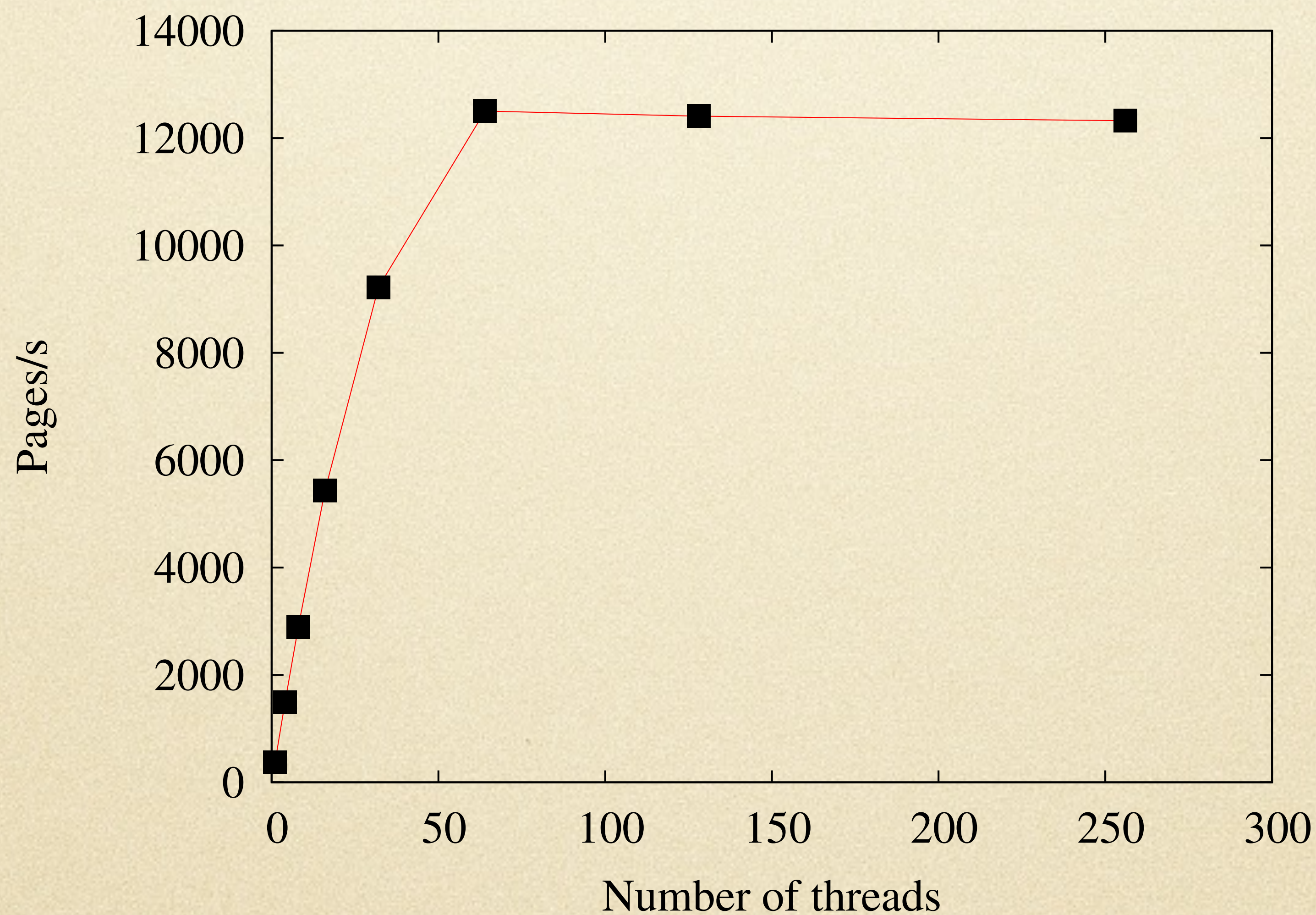
Highlight: The workbench

- A double priority queue of *visit states* (the state of visit of a host)
- Organized by next-fetch per host & per IP
- Works like a delay queue: wait until a host is ready to be visited

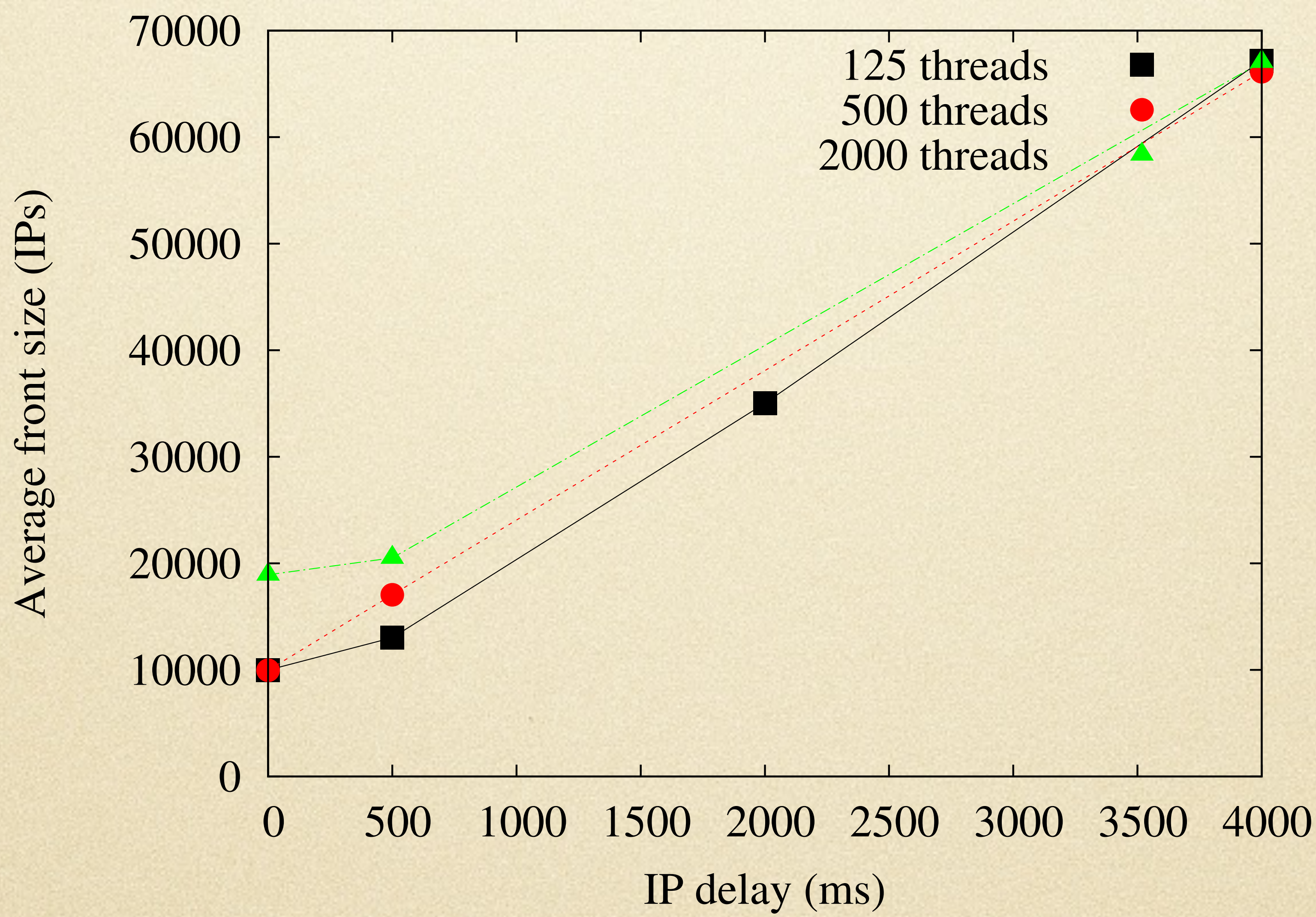
Highlight: the workbench virtualizer

- Visit states keep track of URLs that are to be visited for a given host (those already been output from the sieve)
- How to reconcile this with *constant memory*?
- Keeping only the tip of each queue and using on-disk refill queues for the rest...

Behavior on a slow connection

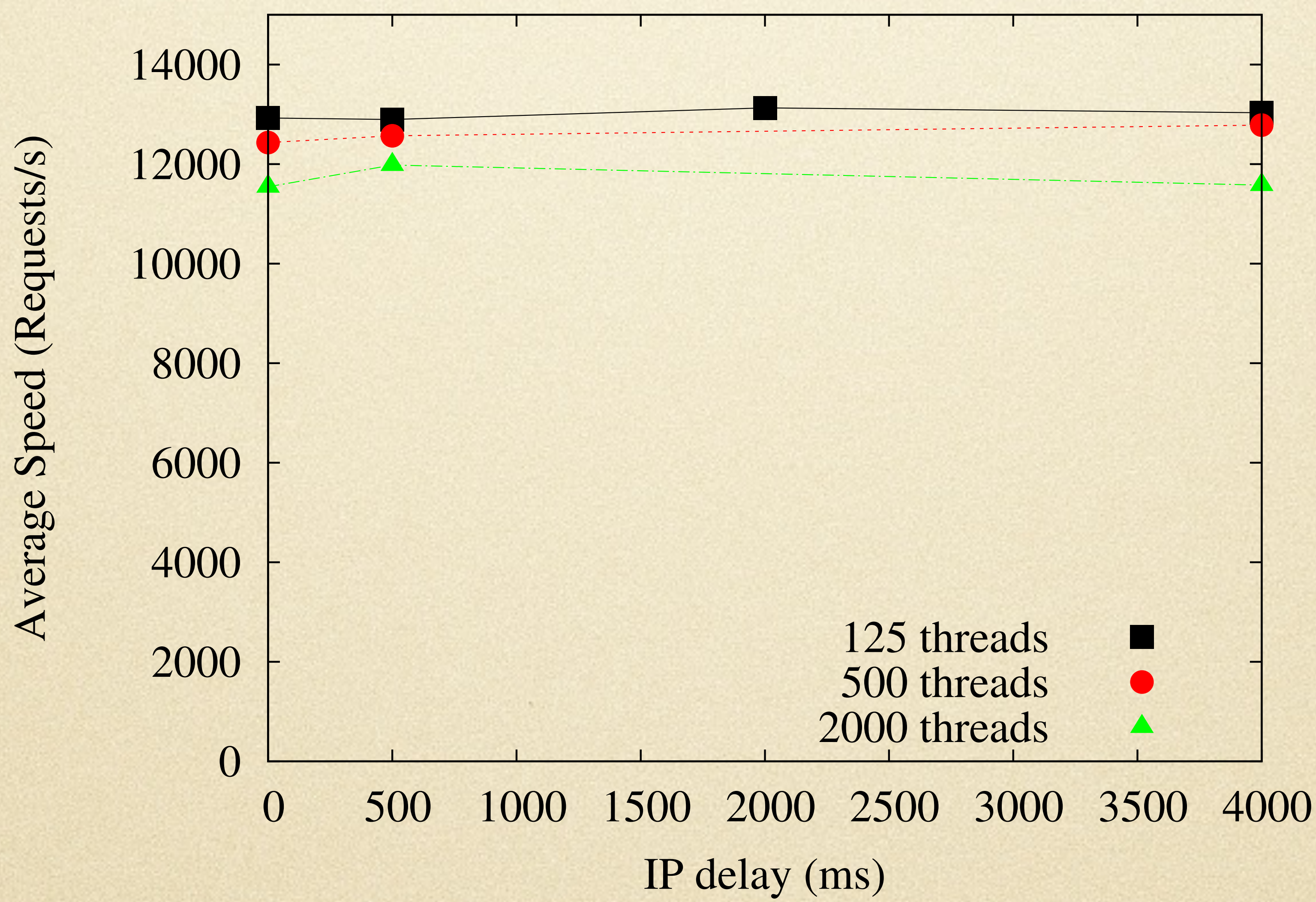


Front size



125 threads
500 threads
2000 threads
Average front size (IPs)
70000
60000
50000
40000
30000
20000
10000
0
0
500
1000
1500
2000
2500
3000
3500
4000
IP delay (ms)

Average speed



Comparisons

	Machines	Speed / agent (MB / s)
Nutch (ClueWeb09)	100	0,1
Heritrix (ClueWeb12)	5	4
Heritrix (*in vitro*)	1	4,5
IRLBot	1	40
BUbiNG (*in vivo*)	**1**	**154**
BUbiNG (*in vitro*)	**4**	**160**

Fast?

- *In vitro*: >9000 pages/s average, peaks at 18000 pages/s
- *In vivo* (@iStella): >3500 pages/s average (single crawler), steady download speed of 1.2Gb/s
- ClueWeb09 (Nutch): 4.3 pages/s
- ClueWeb12 (Heritrix): 60 pages/s
- IRLbot: 1790 pages/s (unverifiable)

We broke down almost everything!

- **Hardware** broke down: €40,000 server replaced for no charge with a €60,000 server
- **OS** broke down: Linux kernel's bug 862758
- **JVM** broke down: try opening 5000 random-access files
- Dozens of bug reports and improvements to a number of open-source projects, including the Jericho HTML parser, Apache Software Foundation's HTTP Client, etc.

Vital importance in open-source development

Future works

- Download@ `http://law.di.unimi.it/`
- Using other prioritizations for URL
- But first of all: making crawling technology more and more accessible to the masses

Thanks!