A Web Service for Scholarly Big Data Information Extraction

Kyle Williams, Lichi Li, Madian Khabsa, Jian Wu, Patrick C. Shih and C. Lee Giles

Information Sciences and Technology
Computer Science and Engineering
The Pennsylvania State University

Scholarly Big Data

- Scholarly Big Data includes all academic research output
  - Journal and conference publications
  - Books
  - Theses
  - Slides, data, course materials…

- Often found in data repositories
  - Google Scholar, Arxiv, Microsoft Academic, CiteSeerX, PubMed, University Libraries, etc.

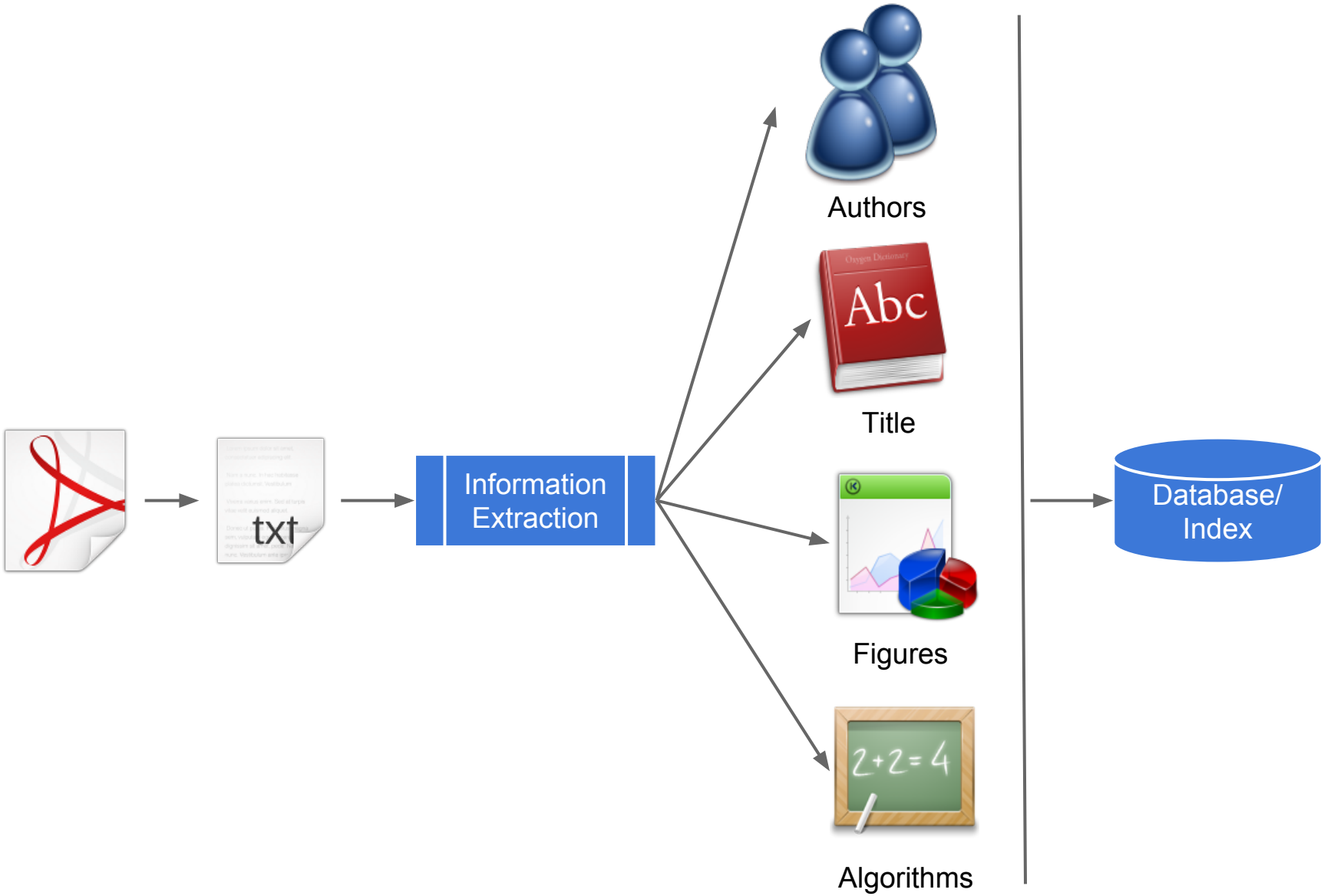
Why is it Useful?

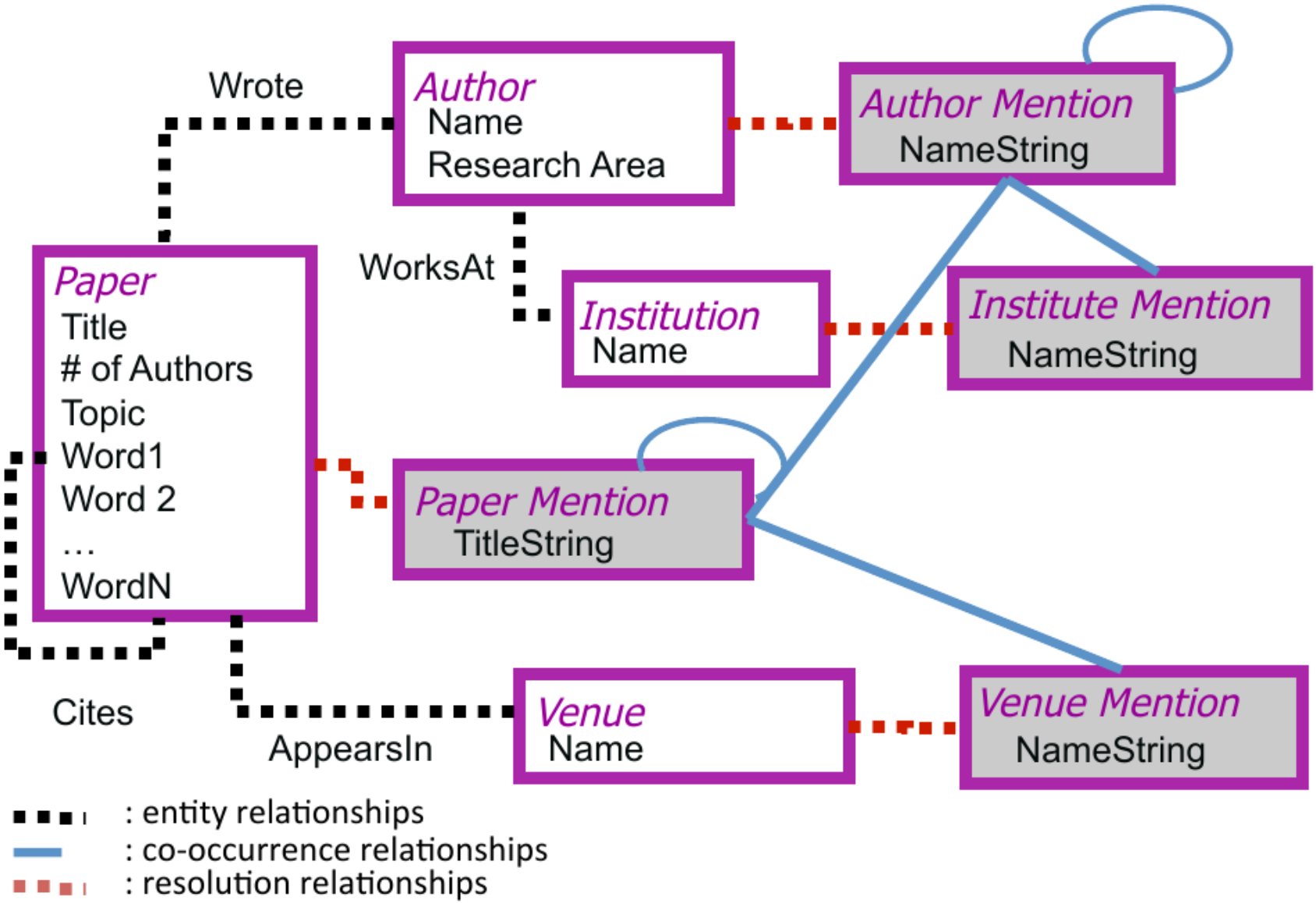
- Analyze scholarly and research trends
- Evaluation of investments in science and scholarship
- Identify opportunities for collaboration
- Evaluate individual scientist, groups and organizations

The Role of Metadata

- Metadata plays a crucial role in accessing, organizing and evaluating scholarly data

- Manual extraction of metadata is tedious and time consuming, thus motivating automatic methods


txt
Information Extraction
Authors
Abc
Title
Figures
2+2=4
Algorithms
Database/ Index



Source: Getoor & Machanavajjhala, KDD'13

Duplication in Scholarly Big Data

- Duplication is common among scholarly data found on the Web
  - Co-authors storing versions of the same paper on their personal websites
  - Updated versions of papers
  - Published papers and preprints
- For a Web service for information extraction, users may submit the same paper
- At Big Data scale, we want to avoid redundant information extraction

CiteSeerExtractor

- A Web service from CiteSeerX for scholarly information extraction
- Performs automatic information extraction
- Deals with duplication by including a near duplicate matching backend

CiteSeerExtractor

A RESTful API for extracting information from scholarly documents...

Resource Oriented Architecture

- Design is based on the resource oriented architecture
  - Defined by resources, identifiers, representations, addressability, statelessness, connectedness and uniformity

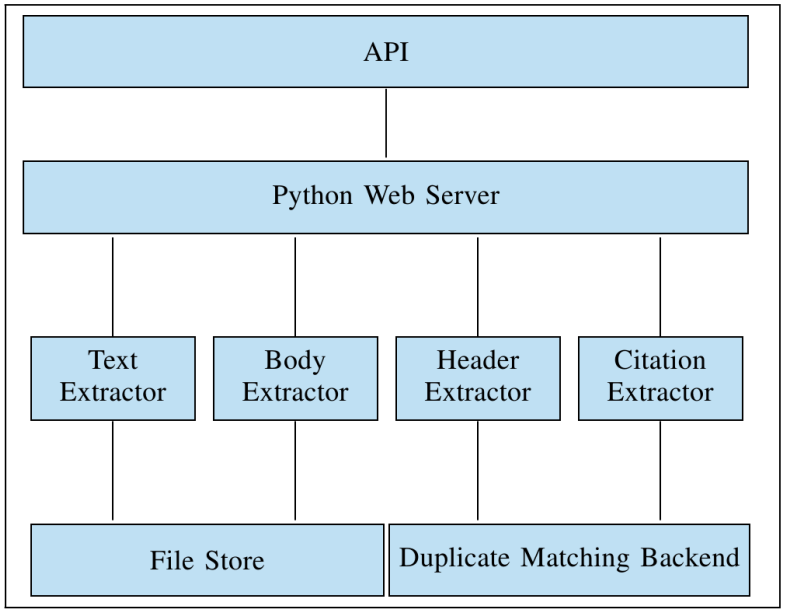
Resources are documents submitted to the Web service, which are **identified** by a random identifier, **represented** by the original file, header, citations, body or text and **addressable** through their IDs. CiteSeerExtractor is **stateless** as each request happens independently, **connected** through links to resources and makes use of the **uniform** HTTP protocol.

Table I
HTTP METHODS SUPPORTED BY CITESEERXTRACTOR

Method	URL	Description	Returns	Options
POST	/	Uploads a new PDF document either via a form or via bytestream	XML document with URIs to resource, resource_id	myfile=@filename (required for form POST)
GET	/resource_id/file	Used to download original document for resource	Document for resource	N/A
GET	/resource_id/header	Extracts the header information (authors, title, etc) from the resource	Representation of header information	output=xml (default) | json
GET	/resource_id/citations	Extracts the citations from the resource	Representation of the citations	output=xml (default) | json
GET	/resource_id/body	Extracts the body (text excluding header and citations) from the resource	Representation of the body	output=xml (default) | json
GET	/resource_id/text	Extracts the full text from the resource	Full text of resource	N/A
DELETE	/resource_id	Deletes the resource	Confirmation	N/A

Architecture

- Python Web server handles requests for submitting documents and retrieving representation
- Extractors extract various components from resources
  - Header, citations, etc.



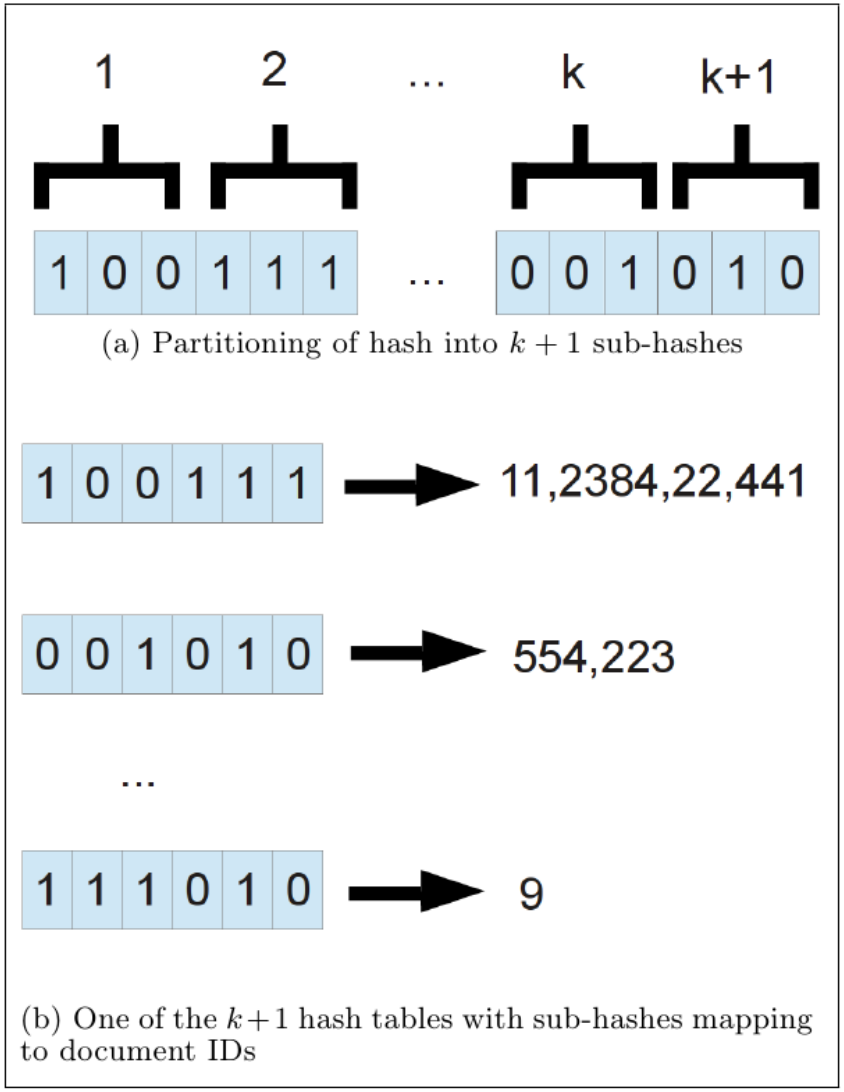
- File store stores resources on disk for extraction
- Duplicate matching backend stores extracted metadata and matches incoming documents to previously processed documents

Duplicate Matching Backend

- Duplication among scholarly documents on the Web and documents submitted to an extraction service
- Store previously extracted metadata
- Match incoming documents to previously extracted documents
  - Return already extracted metadata if match exists
  - Or extract metadata and store

Simhash Near Duplicate Matching

- Documents represented by 64-bit hash
- If Hamming distance between 2 documents < k, we say the documents are near duplicates
- To find near duplicates, partition each hash into k+1 sub-hashes and store doc ids indexed by subhash (Manku et al, 2007)
- For a query document, lookup each sub-hash in tables to get ids



(a) Partitioning of hash into $k+1$ sub-hashes

(b) One of the $k+1$ hash tables with sub-hashes mapping to document IDs

Calculate the hamming distance between each matched document and the query document

Implementation in CiteSeerExtractor

- Redis NoSQL key-value store used for storing extracted metadata
  - Metadata can be given a time to live (TTL) afterwhich it is removed from Redis database
- Algorithm:
  - Calculate simhash of input document
  - Test for exact match
  - If found, return exact metadata
  - Else, partition into k=3 subhashes and lookup subhash matches
  - If subhash matches found: calculate Hamming distance and return any matches with distance <= 3
  - If no matches, extract, store and return metadata

Algorithm 1 Duplicate matching algorithm

```
1:  procedure MatchDuplicates(doc, metadata)
2:      simhash ← CalculateSimhash(doc)
3:      data ← Lookup(simhash, metadata)
4:      if data ≠ NULL then
5:          return data
6:      end if
7:      subhashes ← GetSubhashes(subhashes)
8:      dupes ← GetMatches(simhash, subhashes, k)
9:      if dupes ≠ NULL then
10:         data ← Lookup(dupe[0], metadata)
11:         if data ≠ NULL then
12:             return data
13:         end if
14:     else
15:         AddSubhashes(subhashes, simhash)
16:         data ← extract(metadata)
17:         SaveMetadata(simhash, metadata)
18:         return data
19:     end if
20: end procedure
```

Experiments

- High end server
  - CPU: 24 x Intel(R) Xeon(R) CPU X5650 @ 2.67 GHz
  - RAM: 48GB
  - OS: Red Hat Enterprise Linux (RHEL) Server 5.9; Python: 2.7; Redis: 2.4.10 with 44GB memory limit
- 3.6 million documents from the CiteSeerX collection
- 24 threads used for submitting documents to the Web service

Duplicate Matching Overhead

- Duplicate matching backend should not have a negative effect on performance

- Timed the processing of 100 files while extracting even near duplicates were found
  - 4.26 seconds (sd. 1.24) without duplicate matching
  - 4.35 seconds (sd. 1.25) with duplicate matching

- No large overhead by including near duplicate matching backend

Timing and Storage

- Compared the time and disk usage for header and citation extraction for 100 docs

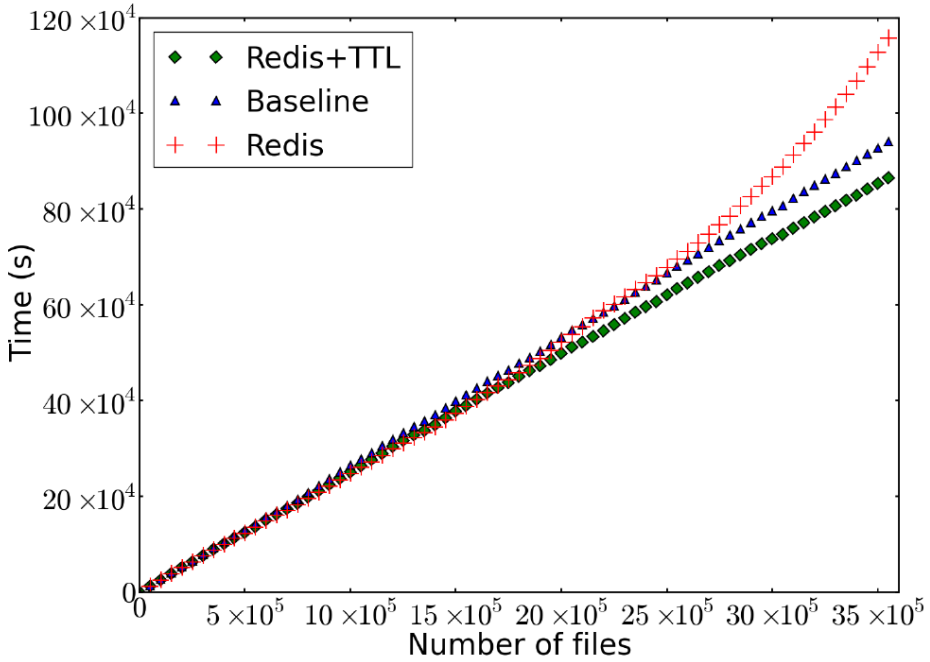
EXTRACTION TIMES AND DATA SIZE FOR CITATIONS AND HEADERS EXTRACTED FROM 100 DOCUMENTS

	Citations	Header
Mean Time (std. dev.)	01.11 (0.29) seconds	2.86 (1.18) seconds
Total time	111.31 seconds	286.40 seconds
Size	1.4 MB	152 KB

- Citations are faster to extract and use more disk space
  - Use this information to set a TTL on citations so as to free up memory consumption of Redis database

Extractor Performance

- Baseline with no duplicate matching
- Redis with storing all metadata and citations
- Redis+TTL sets a TTL of 6 hours on citations and uses compression



- Redis initially performs well until memory becomes full
- Redis+TTL solves this by expiring citations to free memory
- Performance increases as more files submitted
- 8.46% improvement in extraction time after ~3.6 million files (~21 hours saved with total running time of 10 days)

Verifying Results

- Number of documents processed:
  - No duplicate matching: 3,490,791
  - Redis: 3,484,213
  - Redis+TTL: 3,490,799
  - Using Redis+TTL does not lead to more failures

- Recorded first 100 near duplicate matches
  - 92% accuracy
  - False positives had large amount of mathematical notation, which can be fixed by filtering numeric characters

Conclusions

- Designed a RESTful Web service for scholarly big data information extraction
  - Deals with the issue of duplication
  - Improved performance by matching duplicates
  - Accuracy of matching can be improved by strengthening the criteria for 2 documents to be considered near duplicates

- Highly modular and can be extended with additional extractors

Demo

http://citeseerextractor.ist.psu.edu

Thanks

- Partial support by the National Science Foundation