Distributed Dual Iterative Pattern Relation Expansion (D-DIPRE)

Anup Chalamalla  Mina Farid
Outline

• DIPRE
  • Introduction
  • Problem Definition
  • Algorithm

• Distributed DIPRE
  • Basic idea
  • Implementing different modules
  • Data flow
  • Iterative MapReduce

• Conclusions and Future Work
DIPRE- Introduction and Problem Definition

- Large amount of information on the Web
- Extract structured information from unstructured documents
- $D$ large set of documents (WWW)
- Looking for occurrences of $R$
- $R$ is a binary relation, e.g., (author, title), (person_name, email)
DIPRE- Algorithm

1. Feed some seeds
2. Find Occurrences
3. Generate Patterns
4. Find patterns matches
5. If enough tuples are found exit, else repeat 2
DIPRE- Algorithm (cont’d)

1. Feed some seeds
2. Find Occurrences
3. Generate Patterns
4. Find patterns matches
5. If enough tuples are found exit, else repeat 2
DIPRE- Algorithm (cont’d)

1. Feed some seeds

Provide some sample instances of the relation

For example,

<table>
<thead>
<tr>
<th>Author</th>
<th>Book Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isaac Asimov</td>
<td>The Robots of Dawn</td>
</tr>
<tr>
<td>David Brin</td>
<td>Startide Rising</td>
</tr>
<tr>
<td>James Gleick</td>
<td>Chaos: Making a New Science</td>
</tr>
<tr>
<td>Charles Dickens</td>
<td>Great Expectations</td>
</tr>
<tr>
<td>William Shakespeare</td>
<td>The Comedy of Errors</td>
</tr>
</tbody>
</table>
1. Feed some seeds

2. **Find Occurrences**

3. Generate Patterns

4. Find patterns matches

5. If enough tuples are found exit, else repeat 2
DIPRE- Algorithm (cont’d)

2. Find Occurrences of Seeds

- Occurrence Structure: (prefix, author, middle, book, suffix, order, url)
- Example:
  - look for (Charles Dickens, Great Expectations) in the domain www.books.com
    www.books.com/TopRated
    “The famous writer Charles Dickens wrote Great Expectations book”
  - Extracted Occurrence:
    (The famous writer, Charles Dickens, wrote, Great Expectations, book, true, www.books.com/TopRated)

- Repeat for all seeds in all documents
- Result: A set of occurrences of seeds in the documents
DIPRE- Algorithm (cont’d)

1. Feed some seeds
2. Find Occurrences
3. **Generate Patterns**
4. Find patterns matches
5. If enough tuples are found exit, else repeat 2
3. Generate Patterns

Pattern Structure: (order, urlprefix, prefix, middle, suffix)

• Group occurrences having similar “order” and “middle”


(The great writer, Nicholas Sparks, \textit{wrote}, The Last Song, book, \textit{true}, www.books.com/BestSellers)

Generate a pattern as general as possible to match all occurrences.

\texttt{writer .*? wrote .*? book}

Prefix = writer \hspace{1cm} Suffix = book \hspace{1cm} order = true

Middle = wrote \hspace{1cm} urlprefix = www.books.com
DIPRE- Algorithm (cont’d)

1. Feed some seeds
2. Find Occurrences
3. Generate Patterns
4. Find patterns matches
5. If enough tuples are found exit, else repeat 2
4. Find patterns matches and extract relations

   writer  .*?  wrote  .*?  book

......The writer Mario Puzo wrote The Godfather book.....

Extract relation (Mario Puzo, The Godfather)
5. If enough tuples are found exit, else repeat repeat again, having new tuples as seeds

New Seed: (Mario Puzo, The Godfather)

.... the book The Godfather was written by Mario Puzo....

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Author</th>
<th>Middle</th>
<th>Book</th>
<th>Suffix</th>
<th>Order</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(the book, Mario Puzo, was written by, The Godfather, NULL, false, <a href="http://www.library.com">www.library.com</a>)</td>
</tr>
</tbody>
</table>

Pattern: (False, library.com, The book, was written by, NULL)

the book .*? was written by .*?

Match patterns ....
DIPRE- End Result

Output:

1. Tuples of Relation R extracted (set of authors and book titles)

2. List of Patterns to extract books

- Patterns are used to extract relations from new documents added to the database
Distributed DIPRE System

- $\rightarrow \{\text{Instances of a Relation}\} \rightarrow_D \{\text{Occurrences in the documents}\} \rightarrow \{\text{RegEx. Patterns}\} \rightarrow_D \{\text{New instances}\}$

- All documents and seed instances reside on Hadoop’s HDFS

- Hadoop’s MapReduce framework is used to process instances and patterns over documents local to each map and reduce worker machines on HDFS

- New instances and patterns generated in each iteration are stored on HDFS
Modules in the implementation

- **Document Loading** – Hadoop loads the documents’ corpus and places the documents on HDFS on the name node and data nodes in the cluster.

- **Occurrence Extractor** – Finding occurrences of seed instances in the documents and extracting their context \{prefix, suffix, middle, order\}
  - Map function:\[(key=docID, value=documentText) \rightarrow R (key={middle, order}, value=Occurrence)\]

- **Pattern Extractor** – Extracts patterns on to HDFS from re-grouped occurrences
  - Reduce function:\[(key={middle, order}, value=List<Occurrence>) \rightarrow List<Patterns>\]

- **Pattern Matcher** — Match extracted patterns against documents and output more instances which are again fed as seeds to the occurrence extractor
  - Map function:\[(key=docID, value=documentText) \rightarrow P List<R>\]

- **Iteration Controller** – Stops the iterative process of pattern-relation extraction when it finds that number of new seed instances generated in an iteration is very small.
Data flow

ITERATION CONTROLLER

SEED INSTANCES

GenOccurrences(s())

Occurrence

Hash Partitioner

PARTITIONED OCCURRENCES

OCCURRENCES

Input: (Key=<middle, order>, List<Values>= List<Occurrences>)

GenPatterns(s())

PATTERNS

PATTERN Extractor

(Reduce Function)

PATTERN Matcher

(Map Function)

NEW INSTANCES

Reduce

HDFS

Documents

Occurrence Extractor

(Map function)
Status of Implementation

- Implemented the DIPRE system with map and reduce functions for various stages in the processing.

- Need to configure Hadoop to run the map reduce jobs iteratively.

- Need to configure Hadoop to run on Amazon EC2 and produce scalability results.
Iterative Map Reduce

- Additional Combine phase to collect all the final output instances of an iteration

- Distinction on static and variable data

- Remove or rename the instances file and load new instances for next iteration

- Hadoop supports iteratively running Map Reduce tasks
User program’s process space

configureMaps()

configureReduce()

while(condition){

runMapReduce()

Combine() operation

updateCondition()

} //end while

close()

Two configuration options
1. Using local disks
2. Using pub-sub bus

Worker Nodes

Cacheable map/reduce tasks

Can send <Key,Value> pairs directly

Map()

Reduce()

All communications/data transfers happen via the pub-sub broker network

Source: http://www.iterativemapreduce.org/
Conclusion

- A distributed information extraction system, which is logically same as DIPRE
  - Can extract instances of an arbitrary relation from documents of any arbitrary domain

- Experiments: Scalability results on
  - Number of nodes used (vs) Time taken
  - Number of documents processed (vs) Time taken,

- Systems Compared: DIPRE, D-DIPRE
Future Work

- Use inverted index to process documents for generating occurrences and patterns

- Katta: Distributed Lucene system which can create a distributed index on Hadoop
References
