A stream computing approach towards scalable NLP

Xabier Artola, Zuhaitz Beloki, Aitor Soroa
<zuhaitz.beloki@ehu.es>

IXA group. University of the Basque Country.

LREC, Reykjavík 2014

Table of contents

1. Introduction
   - NewsReader project
   - Goals
   - Storm
2. The NLP Annotation Format
3. Experiment and results
   - Experiment setting
   - 1. experiment
   - 2. experiment
4. Conclusion and future work
   - Conclusion
   - Future work

Introduction

- Overwhelming flow of textual data available (Big Data)
- Computational power needs increased
- Solution: **distributed computing**

NewsReader project

- Base project for our experiments
- NewsReader project goals:
  - Perform real-time event detection
  - Extract from text what happened to whom, when, where...
- Estimation of 2 million news items per day to process

Requirements

- NLP modules distributed across a cluster
- Distribution of data
- **Use of a stream computing framework**
    - Synchronisation between nodes
    - Load balancing
    - Fault tolerance

Goals

Create a distributed system to process large amount of documents in parallel

Goals

Create a distributed system to process large amount of documents in parallel

- Use of third party NLP modules

Goals

Create a distributed system to process large amount of documents in parallel

- Use of third party NLP modules
- Several possible levels of parallelisation:

Goals

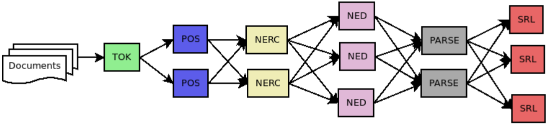
Create a distributed system to process large amount of documents in parallel

- Use of third party NLP modules
- Several possible levels of parallelisation:
  - Redesign and reimplementation of the core algorithms of each NLP module (out of scope)

Goals

Create a distributed system to process large amount of documents in parallel

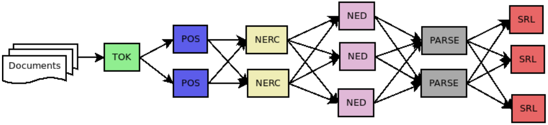
- Use of third party NLP modules
- Several possible levels of parallelisation:
  - Redesign and reimplementation of the core algorithms of each NLP module (out of scope)
  - **Deploy multiple instances of each NLP module**



Goals

Create a distributed system to process large amount of documents in parallel

- Use of third party NLP modules
- Several possible levels of parallelisation:
  - Redesign and reimplementation of the core algorithms of each NLP module (out of scope)
  - **Deploy multiple instances of each NLP module**



- **Goal**: test and measure performance improvements with this approach

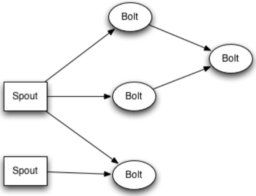
Apache Storm

- Distributed stream computing system
- Open source
- Horizontal scalability
- Fault-tolerant
- Guarantees all data will be processed
- Large and active user community

Apache Storm

Storm concepts

- **Topology**: a graph of computation, composed by spouts and bolts
- **Spout**: input processing modules
- **Bolt**: rest of processing modules
- **Tuple**: structure of the data to transfer between processing modules



The NLP Annotation Format (NAF)

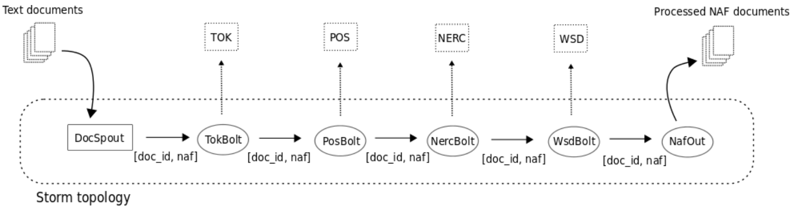
- Common annotation format: NAF (Fokkens et al., 2014)
- Used in NewsReader project to integrate all the NLP modules
- Layered, stand-off format
- References between annotations in different layers
- Specifically designed to work on distributed environments
- +10 layers: raw, text (tokens), terms, chunks, entities...

NAF example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<NAF xml:lang="en" version="v3">
  <nafHeader>
    <public publicId="3_3012" uri="http://casa400.com/docs/test.pdf"/>
    <linguisticProcessors layer="text">
      <lp name="ixa-pipe-tok-en" timestamp="2013-06-26 14:15:18" version="1.0"/>
    </linguisticProcessors>
  </nafHeader>
  <text>
    <wf id="w1" sent="1" offset="0" length="9">Followers</wf>
    <wf id="w2" sent="1" offset="10" length="2">of</wf>
  </text>
  <terms>
    <!--Followers-->
    <term id="t1" type="open" lemma="follower" pos="N" morphofeat="NNS">
      <span>
        <target id="w1"/>
      </span>
      <externalReferences>
        <externalRef resource="wn30g" reference="eng-30-10099375-n" confidence="0.525004"/>
        <externalRef resource="wn30g" reference="eng-30-10100124-n" confidence="0.474996"/>
      </externalReferences>
    </term>
    <!--of-->
    <term id="t2" type="close" lemma="of" pos="P" morphofeat="IN">
      <span>
        <target id="w2"/>
      </span>
    </term>
  </terms>
  <deps>
    <!--nsubj(clashed-5, Followers-1)-->
    <dep from="t5" to="t1" rfunc="nsubj"/>
  </deps>
  <entities>
    <entity id="e1" type="misc">
      <references>
        <!--British-->
        <span>
          <target id="t7"/>
        </span>
      </references>
    </entity>
  </entities>
```

Experiment setting

- Storm concepts in our system:
  - Spout: reads text documents and sends to the first bolt
  - Bolts: wrappers of NLP modules (tokenizer, POS tagger...)
  - Tuple: <doc_id, NAF_doc>
- Small pipeline (4 modules):
  - Tokenizer => POS tagger => NERC => WSD



Experiment setting

Input document sets

- 1. set: 10 documents (16.208 words, 682 sentences)
- 2. set: 100 documents (138.803 words, 5.416 sentences)
- 3. set: 1000 documents (1.185.933 words, 48.746 sentences)

- Hardware for testing: single commodity PC (Linux), Intel Core i5-3570, 3.4GHz (x4), 4GB RAM

1. experiment

- Baseline system: the four modules sequentially processed
- Experiment: Storm topology implementation
- Pipeline approach, **but**:
  - When a module finishes processing a document, starts with the next one
- Parallelisation level: number of NLP modules

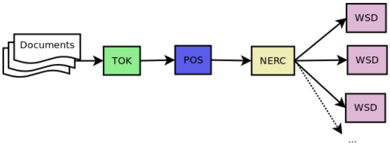
1. experiment results

	Total time	words/s	sent/s	Gain
	10 documents			
pipeline	2m42s	99.8	4.2	-
Storm	2m25s	111.5	4.7	**%10.4**
	100 documents			
pipeline	21m16s	108.8	4.2	-
Storm	18m43s	123.5	4.8	**%12.0**
	1000 documents			
pipeline	3h15m16s	101.2	4.2	-
Storm	2h50m21s	116.0	4.8	**%12.8**

- Little performance gain
  - 96% of the processing time was spent in the WSD module

2. experiment

- Multiple instances of the WSD module
- Parallelisation level: configurable
  - Only 4 CPU cores available: test with 2x, 3x, 4x, 5x and 6x WSD instances



2. experiment results

	Total time	words/s	sent/s	Gain
	10 documents			
pipeline	2m42s	99.8	4.2	-
Storm	2m25s	111.5	4.7	%10.4
$Storm_{2xWSD}$	1m29s	182.9	7.7	%45.4
$Storm_{5xWSD}$	1m28s	182.5	7.7	%45.3
$Storm_{6xWSD}$	1m22s	195.4	8.2	**%49.0**
	100 documents			
pipeline	21m16s	108.8	4.2	-
Storm	18m43s	123.5	4.8	%12.0
$Storm_{2xWSD}$	10m48s	214.3	8.4	%49.3
$Storm_{5xWSD}$	7m44s	299.1	11.7	**%63.7**
$Storm_{6xWSD}$	7m48s	296.1	11.6	%63.3
	1000 documents			
pipeline	3h15m16s	101.2	4.2	-
Storm	2h50m21s	116.0	4.8	%12.8
$Storm_{2xWSD}$	1h40m37	196.5	8.1	%48.5
$Storm_{5xWSD}$	1h10m45s	279.3	11.5	**%63.8**
$Storm_{6xWSD}$	1h11m37s	276.0	11.3	%63.3

Conclusion

- We proposed a new approach for scalable distributed NLP using Storm
- Performance gain of 63% with a single commodity PC
- Significant boost in performance expected with large clusters
- **Big room for improvements in overall NLP performance**

Future work

- Test with a multi-node cluster
  - More real scenario
  - Much larger input set
- Enhance general system architecture
  - Distributed message queue system (Kafka)
  - Use of a NoSQL database to store/retrieve data (MongoDB)
- Topology design improvements
  - Non-linear topologies
  - Granularity-based splitting of documents

A stream computing approach towards scalable NLP

Xabier Artola, Zuhaitz Beloki, Aitor Soroa
<zuhaitz.beloki@ehu.es>

IXA group. University of the Basque Country.

LREC, Reykjavík 2014