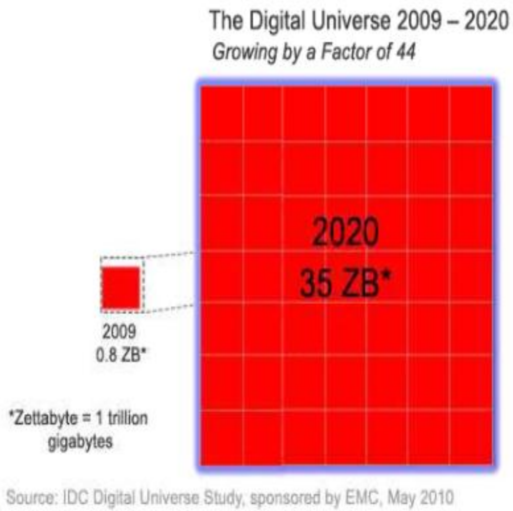
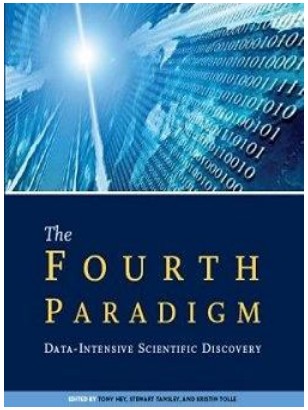
A Comparative Study of Data Processing Approaches for Text Processing Workflows

Ting Chen, Kenjiro Taura
The University of Tokyo
2012/11/12 @ MTAGS

Data Intensive Text Processing

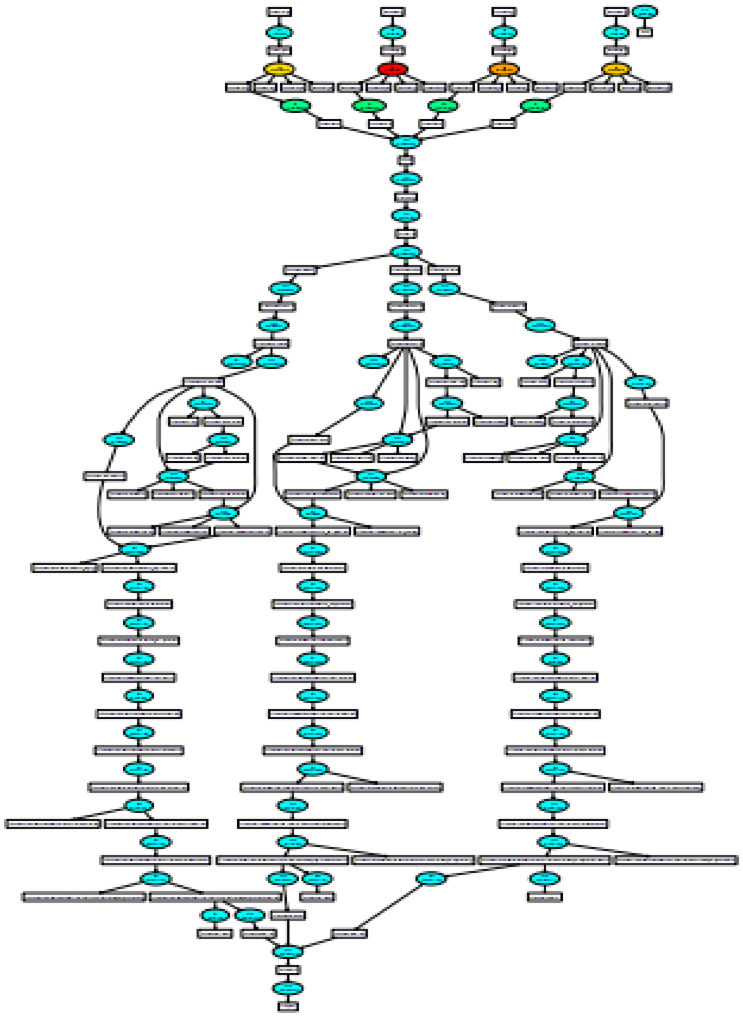
- The fourth paradigm of science: Data-intensive computing
- Data-intensive text processing (NLP: Nature Language Processing and IR: Information Retrieval) faces big challenge
- Workflows are widely used to solve text processing applications





Workflow

- A DAG of coarse-grained jobs and their dependency
- Each job is typically an existing binary or executable (e.g. *sentence splitters, parsers* and *named entity recognizers* in NLP)
- Data are normally stored in and transferred via files
- Many workflow systems: GXP make, Swift, Dryad…



Problems in workflow with files

- Low-level description
  - workflow is very complex with many steps
  - a large number of intermediate files
- Inflexible selection of data
  - tedious and inefficient to select a subset of data
- workflow engine-dependent job execution

MapReduce-enabled workflows

- get wide interests
  - a heavy task can be expressed as Map and Reduce jobs or a whole workflow composition is created as MapReduce style
- provide simple programming model and good scalability across hundreds of nodes
- However, MapReduce model has some shortcomings
  - low-level expression (use algorithm to state the requirement)
  - integrating third-party executables is not straightforward and flexible

Database-based Workflows

- simplify description of workflows by completing simple data processing entirely within a SQL query
- allow flexible selection of data
- have better performance in data selection, join and aggregation [Andrew Pavlo et al.2009]
- However, databases have a limited support for
  - integrating external executable into data processing pipeline
  - optimizing data transfers between data nodes and parallel clients that process large query results

This paper targets to

- built three real-world text-processing workflows on top of MapReduce (Hadoop, Hive), database system (ParaLite) and general Files
- discuss their strength/weaknesses both in terms of programmability and performance for the workflows
- reveal the trade-offs that all these systems entail for workflows and provide a guiding information to users

Outline

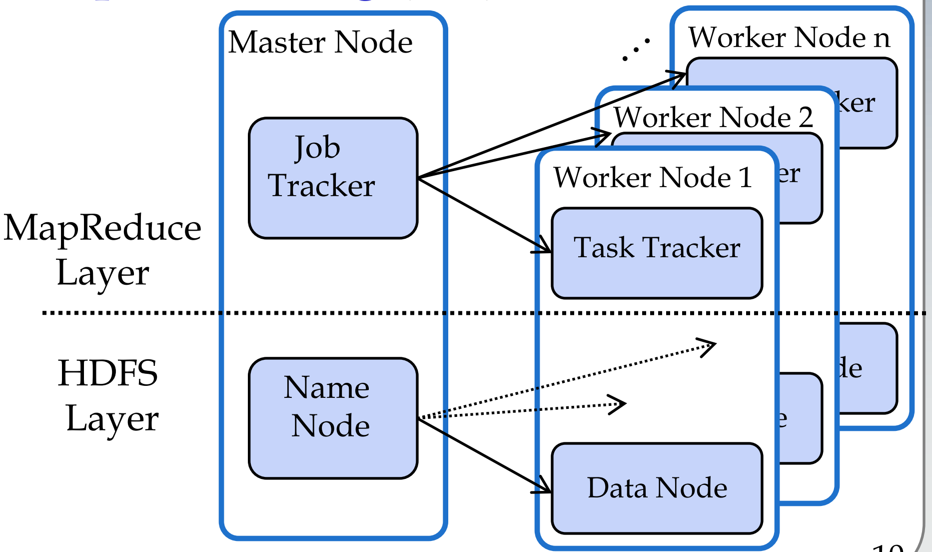
- Background
- Motivation
- Review of Several Approaches
    - Hadoop, Hive and ParaLite
- Real-World Text-Processing Workflows
- Evaluation
- Conclusion

Outline

- Background
- Motivation
- Review of Several Approaches
    - Hadoop, Hive and ParaLite
- Real-World Text-Processing Workflows
- Evaluation
- Conclusion

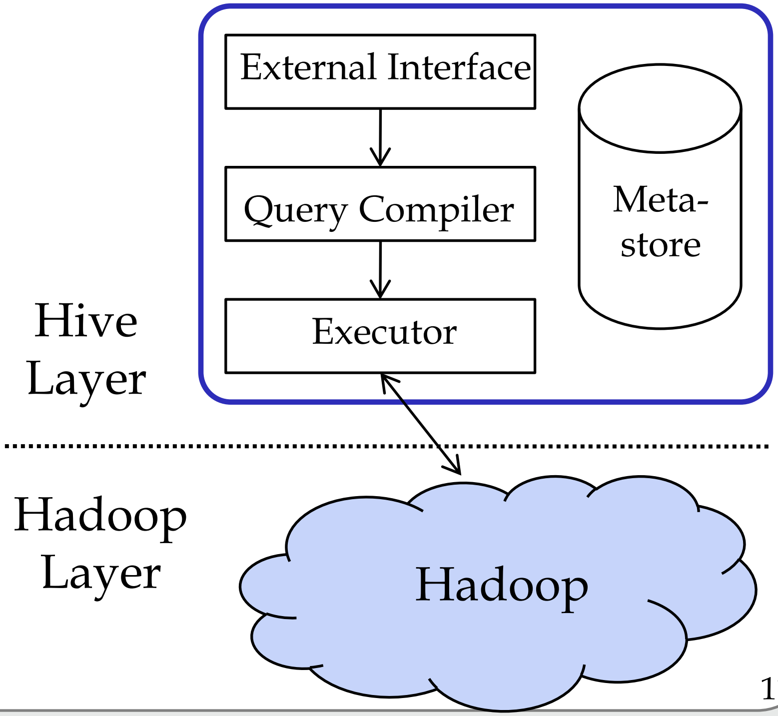
Hadoop [http://hadoop.apache.org/]

- an open-source incarnation of MapReduce model
  - provides users easy programming model with *Map* and *Reduce* functions
  - uses HDFS as the data storage layer
  - takes MapReduce as the data processing layer
- to reuse *map/reduce* function, Hadoop Streaming (HS) is developed
  - allows you to create and run map/reduce jobs with any executable or script as the mapper and/or the reducer



Hive [A. Thusoo et al. 2009]

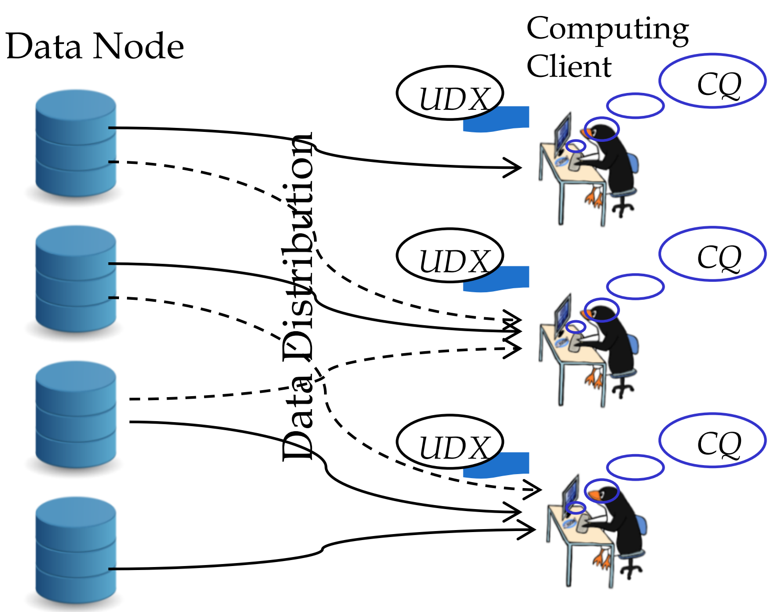
- a data warehouse system built on top of Hadoop
- projects structured data files to relational database tables and supports queries on the data
- use a SQL-like language HiveQL to express queries and compiles them into MapReduce jobs
- allows users' own mappers and reducers (executables written in any language) to be plugged in the query



ParaLite [Ting Chen et al. 2012]

A Workflow-oriented parallel database system

- Basic idea
  - Provides a coordinate layer to connect single-node database systems (SQLite) and parallelize SQL query across them
- New features for workflows
  - Extension of SQL syntax to embed an arbitrary command line (User-Defined Executables or UDX)
  - Parallelization of UDX across multiple computing clients by collective query (CQ)



WordCount Task

Table: data

text
This is a test! It is sunny today.
I am a student. I am working now.

word	count
It	1
is	2
am	2
test	1
...	...

Hadoop Streaming

Hadoop jar hadoop-streaming.jar
-input myInputDirs
-output myOutputDir
-mapper wc_mapper.py
-reducer wc_reducer.py

select word, count(*) from(
select F(text) as word from data
with F= "wc_mapper")
group by word

ParaLite

Hive

select mapout.word, count(*)
from (
map text using 'wc_mapper.py' as word from data
) mapout
group by mapout.word

Outline

- Background
- Motivation
- Review of Several Approaches
  - Hadoop, Hive and ParaLite
- Real-World Text-Processing Workflows
- Evaluation
- Conclusion

Text-Processing Workflows

- Natural Language Processing
  - Japanese Word Count
  - Sentence-Chunking Problem
  - Event-Recognition Application
- GXP Make [Kenjiro Taura et al. 2010]
  - uses make to describe the whole workflow and provides the parallelization of jobs across clusters
  - performs each single job by the four different systems

Text-Processing Workflows

- Japanese Word Count
- Sentence-Chunking Problem
- Event-Recognition Application

Japanese Word Count

→Calculate the occurrence of Japanese words from crawled Japanese web pages.

YOMIURI ONLINE 価格.com 朝日新聞 楽天 kobo YAHOO! JAPAN 日本経済新聞 ………………… ***input***: *web pages in Japanese*

1 ………………… ***html2sf***: *crawled data → standard format*

2 ………………… ***sf2rs***: *extraction of plain text*

3 ………………… ***juman***: *a morphological analyzer for Japanese*

4 ………………… ***word count***: *calculation of occurrences of words*

五輪日本	91
民主党	27
地震	1874
…	

………………… ***output***: word, count

Discussion of JAWC Workflow

- This workflow is a simple pipeline style
- Hadoop use a HS script to express each job since it cannot pipe multiple mappers/reducers
- Hive performs the workflow by only one query
- ParaLite uses a single query to perform the first three jobs followed by another aggregation query
- With file-based systems, split/merge files for parallelization is required

```
select tokens.word, count(*) as count  from (
   map rst.rs using ‘juman’ as word from (
      map sft.sf using ’sf2rs’ as rs from (
           map html.con using ’html2sf_wrap’ as sf from
               html) sft) rst) tokens
group by tokens.word;
```

```
create table tokens as
select T(S(H(con))) as word from html
with H="html2sf  html_file" input ’html_file’
        S="sf2rs"
        T="juman"
partition by word ;
select word, count(*) from token group by word;
```

Discussion of JAWC Workflow (Cont.)

- Two difficulties
  - File-based executable : *html2sf* (which can only takes file as the input)
  - Input data with complicated format, e.g. multiple lines per record

	# of intermediate file	# of wrappers
Hadoop	No	3
Hive	No	1
ParaLite	No	0
File	A lot!	0

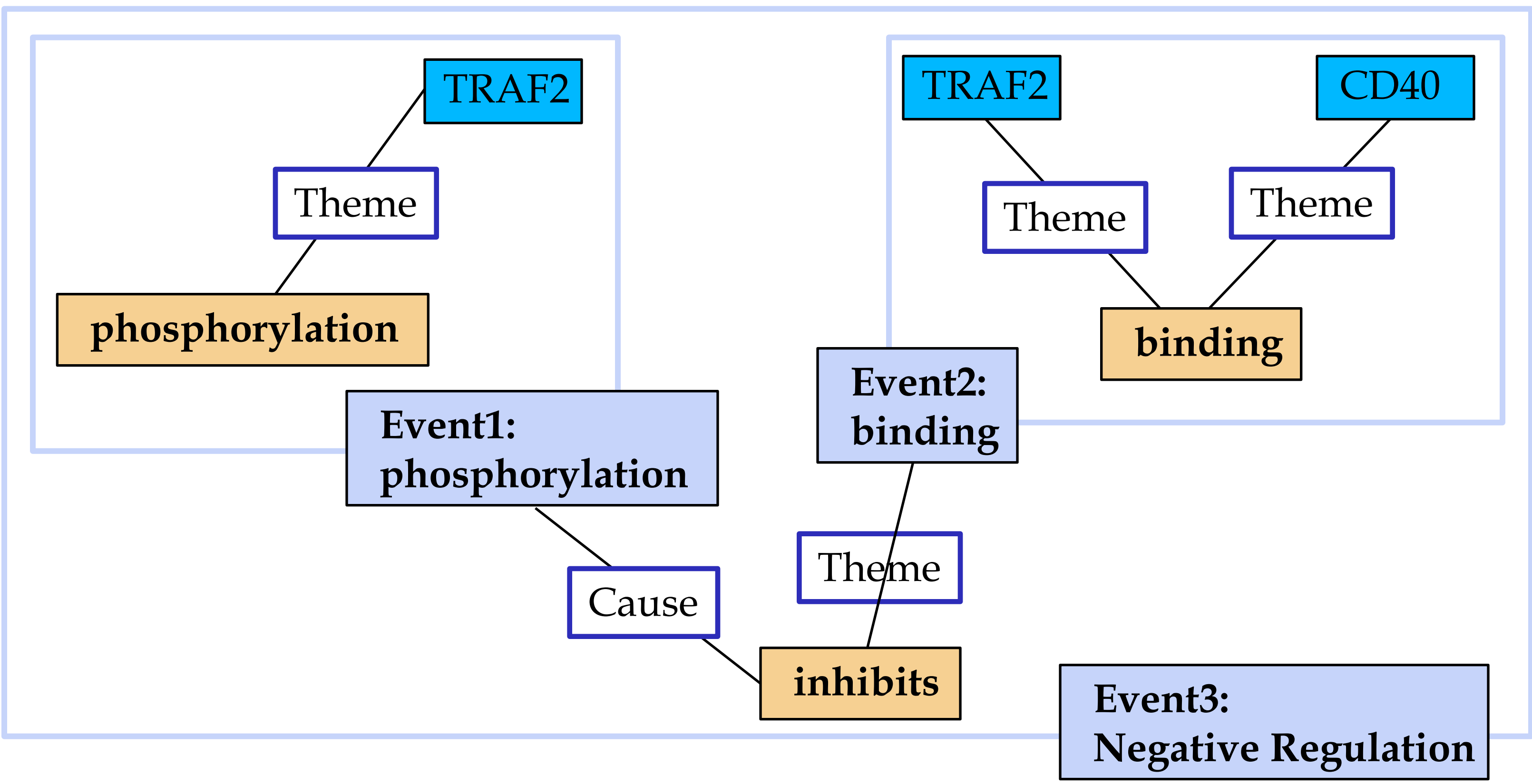
Text-Processing Workflows

- Japanese Word Count
- Event-Recognition Application
- Sentence-Chunking Problem

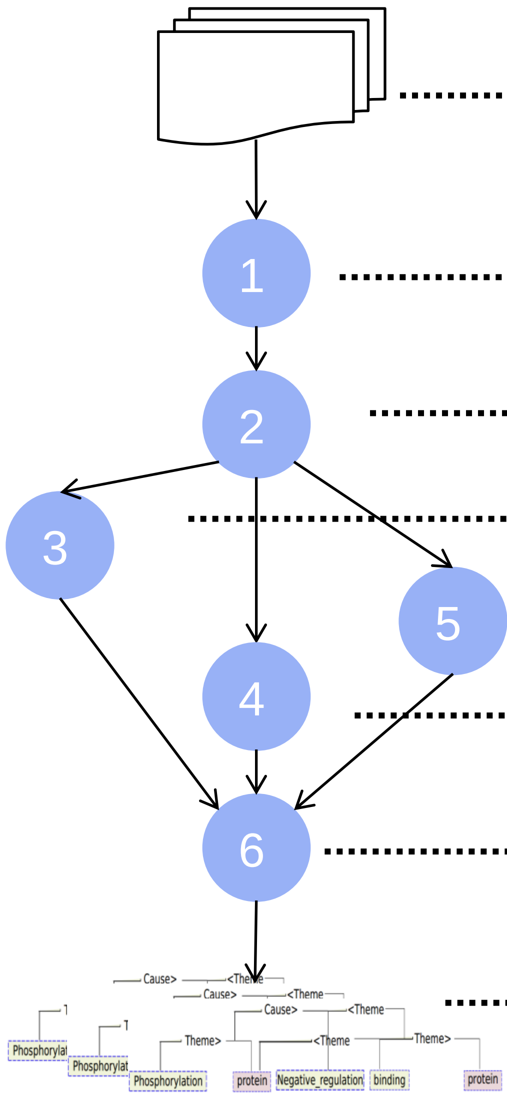
Event Recognition Application [M. Miwa, et al. 2010]

→To recognize complex bimolecular relations (bio-events) among biomedical entities (i.e. proteins and genes)

The phosphorylation of TRAF2 inhibits binding to the CD40 domain.



Workflow of Event-Recognition



input: articles from MEDLINE database

xml2text: extraction of abstract for articles

geniass: split abstract into sentecnes

ner: recognition for bio-medical entities

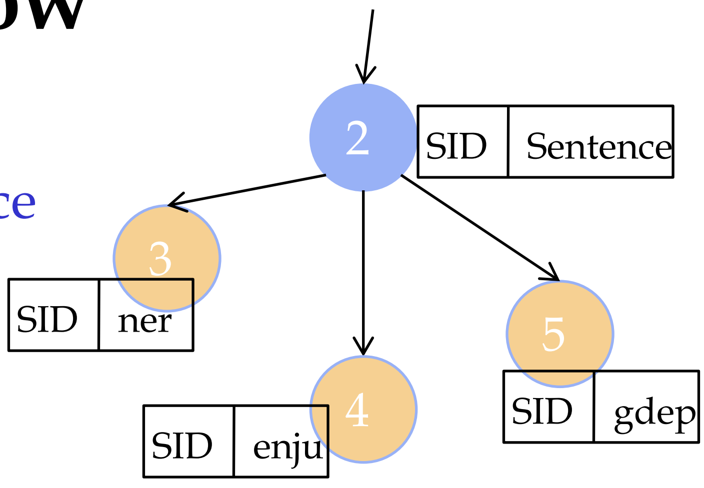
enju: syntactic/semantic parser for sentences

gdep: : dependency parser for biomedical text.

event-recog: recognition for complex events

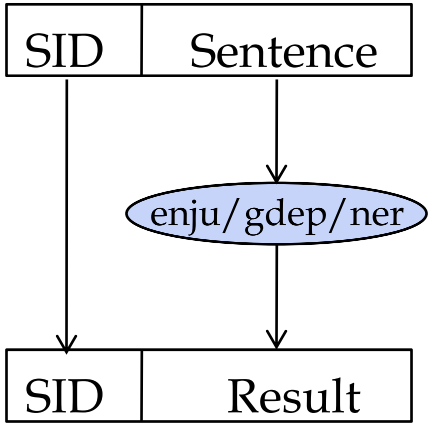
output: event structure

Discussion of ER Workflow



- It is a typical NLP workflow with both data access patterns of pipeline and reduce
- It firstly applies several existing tools to each document/sentence
- With files, Hadoop or Hive , it would be tedious to track the association between input and output
- With ParaLite, it is easy to trace the association using the SQL query:

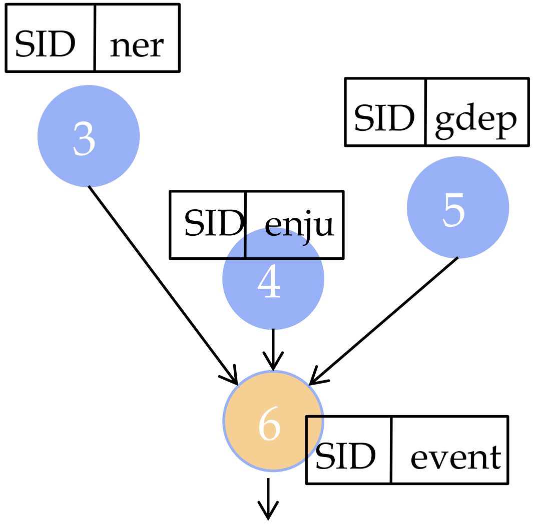
select SID, X(sentence) from ...



	Hadoop	Hive	ParaLite	File
# of wrappers	12	10	5	10

Discussion of ER Workflow (Cont.)

- Then the workflow joins the three results for event detection
- With files or Hadoop, it is not straightforward to join several files
- With Hive and ParaLite, it is easy to join several tables by SQL query:




```
select out.SID, out.event
from (map abst.SID, abst.sentence, enju_so.enju,
            ksdep_so.ksdep, gene_so.gene
      using 'event-detector' as (SID, event)
      from abst
      join enju_so on (abst.SID = enju_so.SID)
      join ksdep_so on (abst.SID = ksdep_so.SID)
      join gene_so on (abst.SID = gene_so.SID)
) out
```

```
select F(abst.SID, abst.sentence, enju_so.enju,
         ksdep_so.ksdep, gene_so.gene) as (SID, event)
from abst, enju_so, ksdep_so, gene_so
where abst.SID = enju_so.SID
      and abst.SID = ksdep_so.SID
      and abst.SID = gene_so.SID
with F="event-detector"
     output_row_delimiter EMPTY_LINE
```

Text-Processing Workflows

- Japanese Word Count
- Event-Recognition Application
- Sentence-Chunking Problem

Sentence Chunking Problem [A. S. Balkir et al. 2011]

- To find a best way to chunk a sentence to get meaningful chunks, e.g. technical term, named entities and relations.

MapReduce | and | Parallel database system | may | be | good | choices | for | text processing | workflows.

- Method: statistical model

For example, a sentence S with 3 words (A B C)

(1) , get f_i the probability of phrase i based on its frequency

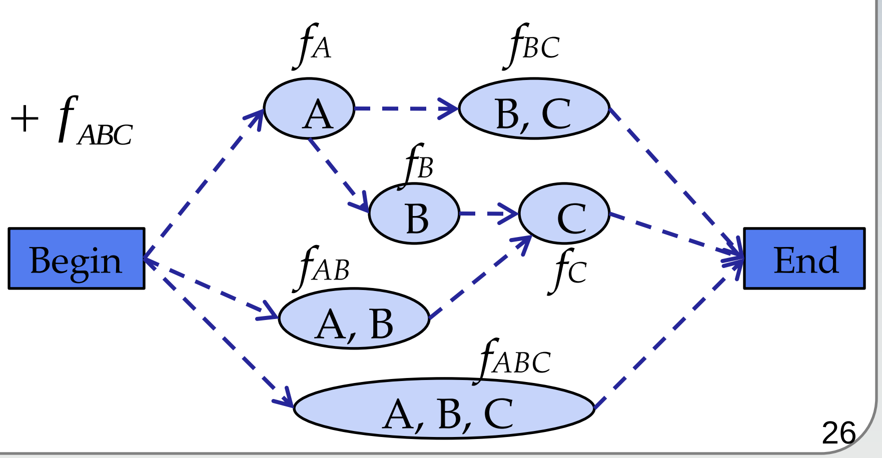
(2) , calculate the likelihood of each sentence

$$L(S) = \sum_{\sigma \in \Psi} \prod_{i \in \sigma} f_i$$

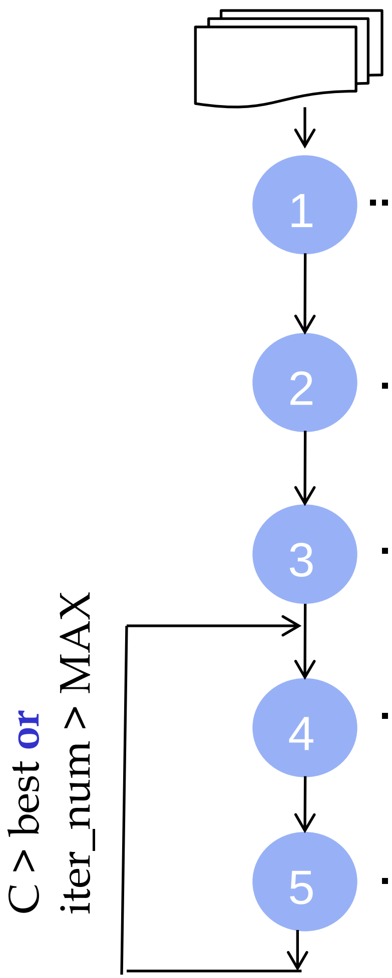
$$= f_A \cdot f_{BC} + f_A \cdot f_B \cdot f_C + f_{AB} \cdot f_C + f_{ABC}$$

(3) , train the whole corpus and maximize its likelihood

$$L(C) = \prod_s L(S) \qquad f = \arg\max_f L(C)$$



Workflow of Sentence-Chunking



input: *articles from MEDLINE database*

senSplit: *split abstract into sentecnes*

This is a test sentence.
I am a student!
Today is very hot!

freqGen: *calculation the occurrences of all phrases*

This	*100*
This is	*56*
test sentence	*1*

filter: *filter phrases with frequencies greater than one*

This	*100*
This is	*56*

probGen: *calculation of probability of phrases based on their frequencies*

100	*0.1892*
56	*0.08183*
183	*0.17384*

likelihoodCal: *calculation of likelihood of the whole corpus*

sentence1	*0.9183*
sentence2	*0.9293*
sentence3	*0.1938*

$$L(C) = \prod_{s} L(S)$$

Discussion of SC Workflow

- One iteration of this workflow is simple pipeline style as JAWC workflow, but aggregate jobs appears alternately with general jobs
- This workflow is easily expressed by Hadoop, Hive and ParaLite
- But to perform data selection job (filter) and aggregation jobs Hadoop still requires more efforts (an extra mapper or reducer) than Hive and ParaLite
- File-based method is not appropriate for such workflow in which most jobs perform aggregations to all data

Outline

- Background
- Motivation
- Review of Several Approaches
    - Hadoop, Hive and ParaLite
- Real-World Text-Processing Workflows
- Evaluation
- Conclusion

Environment

- a 32-node cluster
- 2.40 GHz Intel Xeon processor with 8 cores
- 24GB RAM
- HDD: 500GB, SATA 3Gbps

System Configurations

- Hadoop v1.0.3 on Java 1.6.0
  - the maximum number of mappers/reducers on each node : 6
  - allow JVM to be reused
  - # of mappers and reducers
    - for time-consuming jobs, make sure that the execution time of each job is no more than 10 or 30 minutes.
  - replica = 1
- Hive 0.8.1 : same configuration as Hadoop
- ParaLite
  - SQLite 3.7.3
  - # of computing clients / node: <=6
- File system: NFS3

Data Preparation

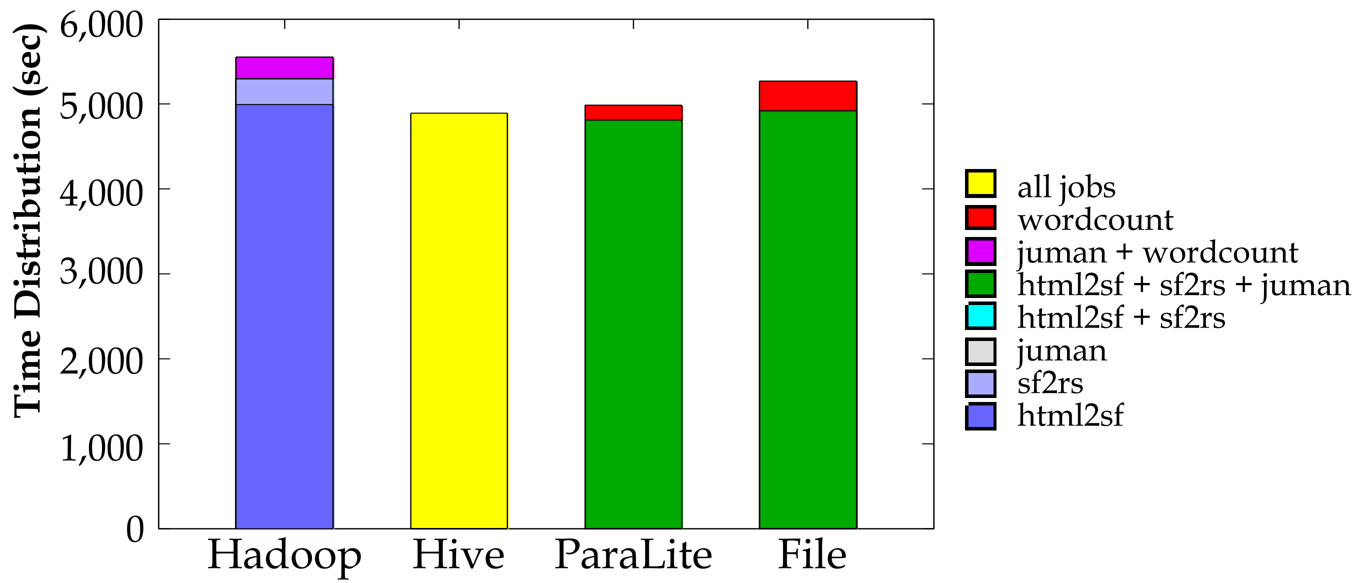
- Hadoop
  - directly loads a big input file by Hadoop command line
    $ hadoop fs –put input_file input_dir_on_hdfs
  - Splits the input file into sub-files distributed on all data nodes and runs the above command in parallel
- Hive
  - loads data to table from either local disk or HDFS by Hive Data Definition Language (DDL): $ *load data …*
- ParaLite
  - provides the same API with SQLite and loads data to the database by the "*.import …*" command line
- File
  - splits the input file into a number of sub-files

JAWC

- 104 GB crawled data → 62 GB useful information

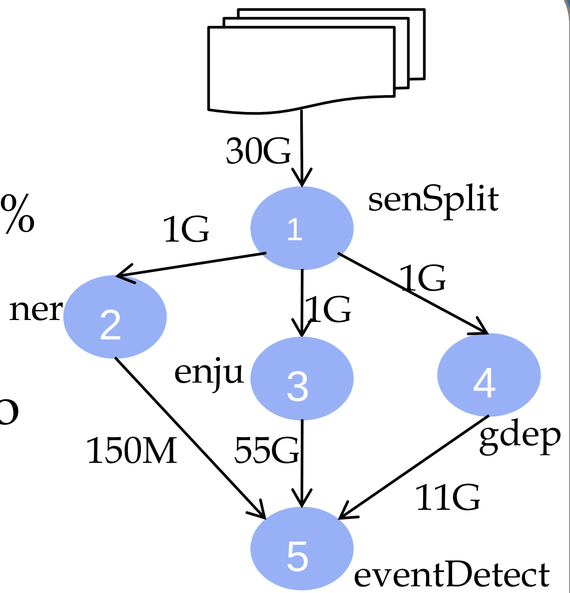
	Hadoop	Hadoop (parallel)	Hive	Hive (parallel)	ParaLite	File
Data Preparation Time(sec)	1280	126	1310	131	432	980

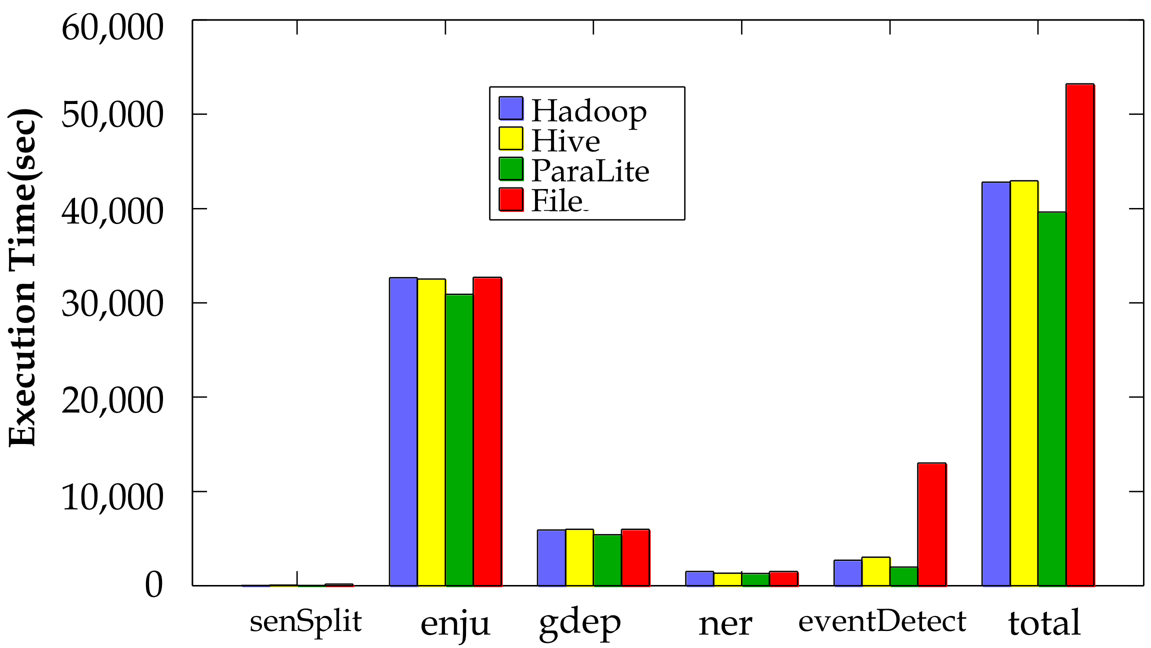
- Hadoop is about 15% slower than Hive and ParaLite



Event-Recognition

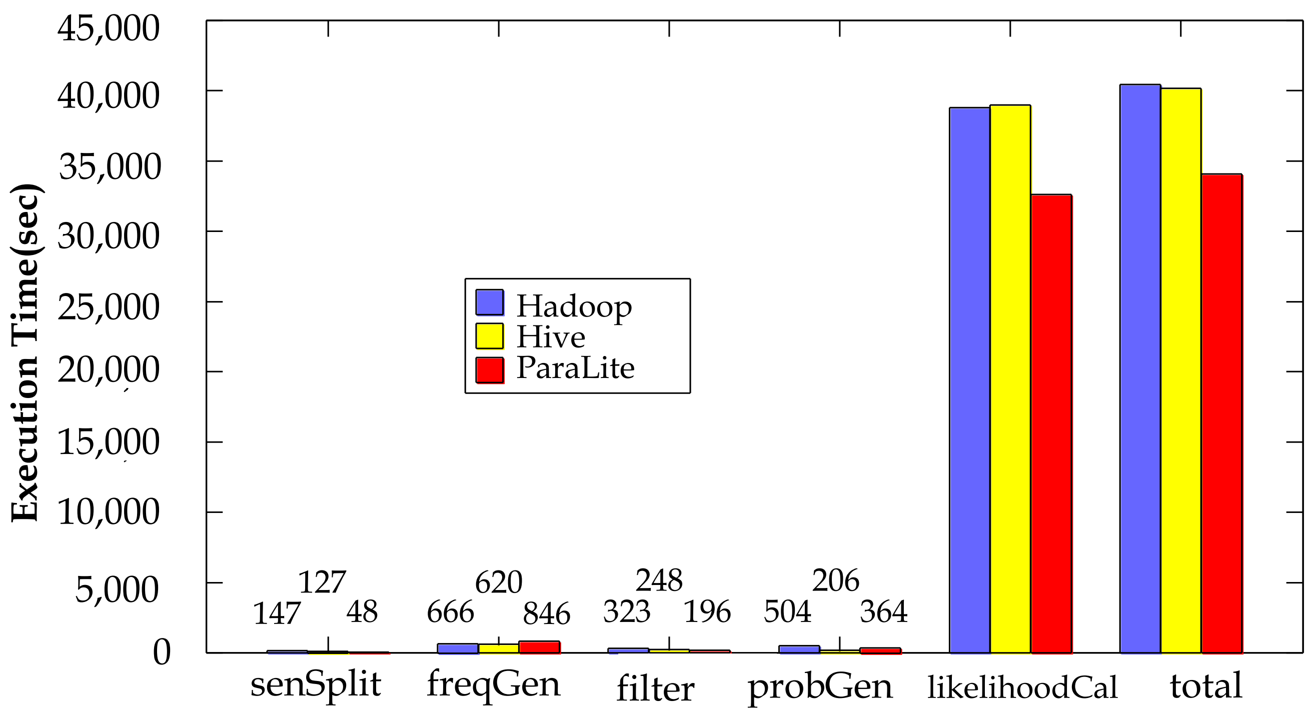
- ParaLite outperforms Hadoop and Hive about 10%
  - less data parsing operations
  - better performance on join operation due to data partitioning





Sentence-Chunking

- 60GB data from MEDLINE database produces 145GB phrases
- ParaLite outperforms Hadoop and Hive about 18%



Outline

- Background
- Motivation
- Review of Several Approaches
  - Hadoop, Hive and ParaLite
- Real-World Text-Processing Workflows
- Evaluation
- Conclusion

Conclusion

- We studied three real-world text processing workflows and developed them on top of Hadoop, Hive, ParaLite and Files.
- We compared the programmability and performance of these workflows
    - high-level query languages (SQL of ParaLite, HiveQL of Hive) are helpful for expressing the workflows elegantly
    - ParaLite is especially useful in the reuse of existing NLP tools
    - Each system has similar performance in the execution of overall workflows but ParaLite shows some potential superiority on typical SQL tasks (e.g. aggregation and join)

Thank you!