FP7-ICT Strategic Targeted Research Project TrendMiner (No. 287863)
Large-scale, Cross-lingual Trend Mining and Summarisation of Real-time Media Streams

D5.1.2 Architecture for Distributed Text Annotation and Trend Mining Over Media Streams (v.1)

Marin Dimitrov (Ontotext)

Alex Simov (Ontotext)

Abstract
FP7-ICT Strategic Targeted Research Project TrendMiner (No. 287863)
Deliverable D5.1.2 (WP 5)

Keyword list: text mining, big data, Hadoop, MapReduce, machine learning, sentiment mining, NoSQL

Nature: Report Dissemination: PU
Contractual date of delivery: M12 Actual date of delivery: 02.11.2012
Reviewed By: DFKI, USFD, IMR
Web links: (none)

CHANGES

Version	Date	Author	Changes
0.1	04.10.2012	Marin Dimitrov	Initial draft
0.2	24.10.2012	Marin Dimitrov	Draft version ready for internal review
0.3	26.10.2012	Thierry Declerck	1st Internal review
0.4	27.10.2012	Trevor Cohn	2nd internal review
0.5	29.10.2012	Alex Simov	2nd draft with improvements according to internal review recommendations
0.6	30.10.2012	Philippe Rigaux	Additional Comments
0.7	31.10.2012	Marin Dimitrov	Final draft to be submitted to the EC
0.8	02.11.2012	Thierry Declerck	Last Check by the Co-ordinator

TrendMiner Consortium

This document is part of the TrendMiner research project (No. 287863), partially funded by the FP7-ICT Programme.

DFKI GmbH
Language Technology Lab
Stuhlsatzenhausweg 3
D-66123 Saarbrcken
Germany
Contact person: Thierry Declerck
E-mail: declerck@dfki.de

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Tel: +44 114 222 1930
Fax: +44 114 222 1810
Contact person: Kalina Bontcheva
E-mail: K.Bontcheva@dcs.shef.ac.uk

University of Southampton
Southampton SO17 1BJ
UK
Contact person: Mahensan Niranjan
E-mail: mn@ecs.soton.ac.uk

Ontotext AD
Polygraphia Office Center fl.4,
47A Tsarigradsko Shosse,
Sofia 1504, Bulgaria
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Internet Memory Research
45 ter rue de la Révolution
F-93100 Montreuil
France
Contact person: France Lafarges
E-mail: contact@internetmemory.org

Sora Ogris and Hofinger GmbH
Bennogassee 8/2/16
1080 Wien
Austria
Contact person: Christoph Hofinger
E-mail: ch@sora.at

Hardik Fintrade Pvt Ltd.
227, Shree Ram Cloth Market,
Opposite Manilal Mansion,
Revdi Bazar, Ahmedabad 380002
India
Contact person: Suresh Aswani
E-mail: m.aswani@hardikgroup.com

Eurokleis S.R.L.
Via Giorgio Baglivi, 3
Roma RM
00161 Italia
Contact person: Francesco Bellini
E-mail: info@eurokleis.com

Executive Summary

This deliverable describes the initial version of the architecture for distributed text annotation and trend mining. The deliverable is comprised of three main parts: (1) an overview of the main data processing tasks that will be performed within TrendMiner and the scalability challenges associated with each task; (2) an overview of the most prominent approaches to scalable Big Data processing available at the moment; (3) recommendations of how to apply various technologies for scalable Big Data processing in the context of TrendMiner. The architecture will be further refined and implemented by the prototypes in two subsequent deliverables in WP5.

Contents

List of Acronyms

API	Application Programming Interface
DFS	Distributed File System
ECL	Enterprise Control Language
ETL	Extract, Transform Load
HDFS	Hadoop Distributed File System
HPCC	High Performance Computing Cluster
P2P	Peer-to-peer
RDBMS	Relational database management system
REST	REpresentational State Transfer
SPARQL	SPARQL Protocol and RDF Query Language

List of Tables and Figures

Introduction

The main objectives of WP5 in the TrendMiner project are:

- Providing a scalable architecture for real-time stream processing
- Research of new approaches to large-scale distributed text mining
- Implementation of the architecture and the distributed text mining approaches into an integrated stream processing platform

This deliverable focuses on the first of the WP5 objectives, namely the architecture for distributed text annotation and trend mining from social streams. The deliverable is organised as follows:

- Chapter 1,"Data Processing in TrendMiner" provides a summary of the data processing tasks that will be performed within TrendMiner and the scalability challenges associated with each of the tasks.
- Chapter 2,"Distributed Approaches to Processing Big Data" provides an overview of the most prominent current approaches to scalable Big Data processing
- Chapter 3,"Architecture for Distributed Trend Mining" provides recommendations for the distributed TrendMiner architecture, based on the analysis made in Chapters 1 and 2
- Finally, chapter 4, "Future Work" outlines the follow-up work of this deliverable that will be done in the period M13-M24.

The main outcomes of this deliverable are:

- Analysis and recommendations of the suitability of the available approaches to scalable Big Data processing to the data processing needs and specifics of TrendMiner
- An initial version of the distributed text annotation and trend mining architecture for TrendMiner, which will be refined, implemented and improved in two subsequent deliverables: D5.3.1 (due M24) and D5.3.2 (due M36)

Data Processing in TrendMiner

TrendMiner will provide a platform for distributed and real-time processing over social streams (implemented by two prototypes in WP5). The platform will cover all the phases from the social stream processing lifecycle: large scale data collection[1], multilingual information extraction and entity linking, sentiment extraction, trend detection, summarization and visualisation (Figure 1).

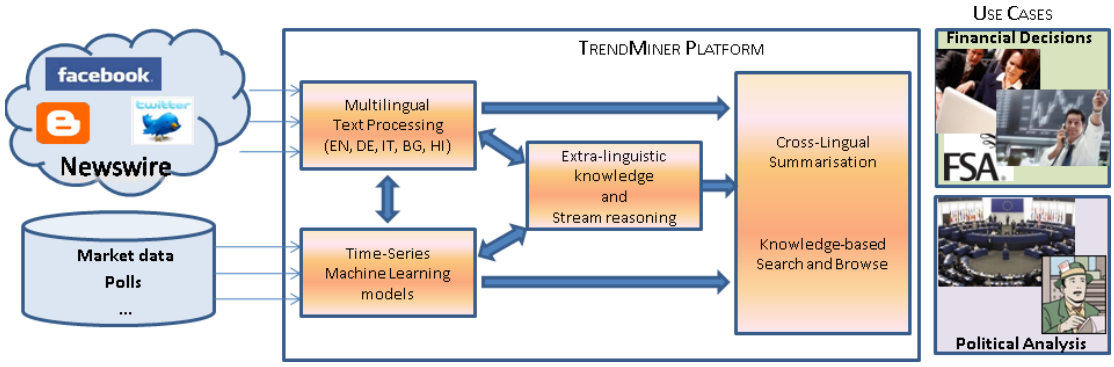


Figure 1 TrendMiner platform

Tasks

This section provides more details about the data processing tasks on the TrendMiner platform and the challenges (regarding scalability) associated with the tasks.

Data Collection

This task provides large scale, real-time collection, aggregation and storage of data from social media streams (blogs, Twitter, newswires, etc.)

The major challenges for this task are:

- Data **volume** – terabytes of data need to be collected, aggregated and stored for further processing
- Data **velocity** – new data is generated (or existing data updated at a high rate) makes it challenging to collect the new data in a timely manner
- Data **variety** – heterogeneous data sources, from semi-structured to unstructured need to be processed and analysed

Data Collection and storage in TrendMiner is the focus of T5.1, and described in D5.1.1 "Real-time Stream Media Collection, v.1"

Pre-processing

[1] Data Collection and storage in TrendMiner is the focus of T5.1, and described in D5.1.1 "Real-time Stream Media Collection, v.1"

Various tasks from WP2 (Multilingual Ontology-Based Information Extraction and Knowledge Modelling), WP3 (Machine Learning models for Mining Trends from Streaming Media) and WP5 (Platform for Real-time Stream Media Collection, Analysis, and Storage) need to pre-process data in order to improve the quality and reduce the time of the subsequent data processing. Such pre-processing tasks include data cleaning, data de-duplication, some basic extraction and pattern matching tasks, formatting tasks (e.g. processing of HTML/XML structures for example).

The major challenge for the timely execution of this task is related to the data volume and the data velocity.

Multilingual Information Extraction

The ontology based multi-lingual information extraction tasks within WP2 aims at extracting structured information from unstructured sources. Such information may include:

- entity mentions (of people, organisations, places, etc.)
- relations between two entities
- more complex structures involving (related) entities, like events in various periods.

Since information extraction traditionally relies on pattern matching (via some complex state automata) or on machine learning approaches, it tends to be computationally expensive and time consuming. The major challenges are again related to data volume (the need to process huge data volumes), data velocity (the need to perform the processing in real time) and to some extent data variety (dealing with heterogeneous data sources which may require expensive pre-processing).

Entity Disambiguation and Linking

Entity disambiguation aims at identifying the concrete class and the unique entity corresponding to a text label in the input. Depending on the particular approach employed – string similarity, structural similarity or context similarity techniques (or even a combination of the above), the disambiguation process may become computationally expensive and may require access to background information in the knowledge base (the TrendMiner domain specific ontologies as established in WP2).

Data volume and data velocity are the main challenges for this task. Processing large volumes of data means that there will be many relevant facts extracted, thus more and more entities with the same labels that need to be disambiguated. Data velocity requires that the disambiguation algorithm is scalable and can cope with real-time data.

Sentiment Extraction and Trend Detection

Sentiment extraction and trend detection in WP2 aim at extracting sentiments from real-time streams and detecting trends based on various statistical approaches (regression and clustering models) and rule-based approaches (relying on the TrendMiner knowledge base). Data volume and velocity are again the major challenges for these tasks, since the machine leaning and statistical modelling

approaches are quite computationally expensive and may require multiple iterations of processing in order to achieve a high quality model. This is even more critical for rule-based approaches, which therefore in TrendMiner will apply only to small sets of documents, or to data that do not need to be processed in real-time. Rule-based methods will also apply to summaries generated by WP4, for consolidating the knowledge base.

It is important to note that the Machine Learning tasks have two different phases: a *training* phase and a *prediction* phase and that the volume and velocity of the data is different for the two phases. The training phase is performed offline at certain intervals (daily, in the beginning) and is done over the batch of data for the interval (day). The training phase itself may require multiple iterations over the data and may be very expensive computationally, thus it cannot be performed in real time. The prediction phase on the other hand will be performed in real time, as new data enters the system, or in relatively small batches for short time periods. This phase is not computationally expensive, but requires a distributed processing approach in order to scale to the amount of new data in real time.

Summarization

The summarization task within WP4 aims at identifying representative text fragments for trends and events within a time period. Data volume, velocity and variety pose challenges for the summarisation task and require scalable algorithms to be employed or developed, so that summarization can be performed over real time streams.

Stream reasoning

Stream reasoning aims at deriving new knowledge from existing knowledge. Unlike traditional reasoning approaches which aim at deriving the complete closure of a set of logical statements (via forward or backward chaining approaches), stream reasoning works by deriving only a partial closure of the statements, with the added benefit that it can scale to cope with real-time input streams which will otherwise be impractical to process with traditional approaches.

Data volume and velocity are again the main challenges, since they pose very strict requirements about the scalability of the reasoning implementation.

Querying over annotated streams

The data generated by all the processing tasks in the TrendMiner pipeline is persistently stored (by WP2)for further access, aggregation and visualisation (by components within WP4). The storage and query answering tasks are constrained by the data volume and velocity, since the storage and query engine must be able to efficiently handle storage and indexing of huge volumes of data, as well as efficient querying over it.

Distributed Approaches to Processing Big Data

This section provides a brief overview of various approaches to processing large volumes of data on distributed clusters. The overview does not cover every platform or tool available, but instead focuses on the most popular, widely used and mature ones.

In particular, the sections in this chapter cover the following platforms, due to their potential relevance to the TrendMiner project:

- Hadoop, Storm and HPCC as generic platforms for distributed data processing,
- several NoSQL datastores,
- Pig (distributed aggregations & transformations of data)
- and Mahout (distributed machine learning)

Hadoop

Hadoop[2] is an open source[3] platform for distributed and scalable data processing and storage. Since its inception around 2007 (initially as an internal data processing platform within Yahoo!), Hadoop has attracted a vibrant community of developers, users and contributors, so by 2012 Hadoop is not a single platform anymore, but an ecosystem of tools and platform for distributed data processing:

- *Hadoop Common* – common tools for all other modules and platforms
- *Hadoop Distributed File System* (HDFS) – a distributed, high-throughput and high-reliability file system
- *YARN* – tools for distributed job scheduling and cluster management
- *MapReduce* – the generic platform for MapReduce-based processing of large datasets.

Additionally, numerous projects and platforms have emerged from Hadoop over the years and are now independent Apache projects:

- *Avro*[4], the data serialization and de-serialization system used by various Hadoop related tools
- *HBase*[5], a scalable, distributed NoSQL database built on top of HDFS
- *Hive*[6], a distributed data warehouse system on top of Hadoop
- *Pig*[7], the data-flow and transformation language based on Hadoop and MapReduce
- *Chukwa*[8], a generic data platform system for distributed systems
- *Mahout*[9], distributed machine learning platform based on Hadoop and MapReduce
- *ZooKeeper*[10], a high-performance coordination service for distributed systems

[2] https://hadoop.apache.org/
[3] Distributed under an Apache Software License 2.0, https://www.apache.org/licenses/
[4] http://avro.apache.org/
[5] http://hbase.apache.org/
[6] https://hive.apache.org/
[7] http://pig.apache.org/
[8] http://incubator.apache.org/chukwa/
[9] http://mahout.apache.org/

MapReduce

MapReduce[11] (Dean & Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, 2004) is a generic programming model for distributed processing on clusters of nodes. While the MapReduce model was popularised (and later patented) by Google, it is inspired and influenced by the *map* and *reduce* programming paradigms coming from the functional programming approach.

The MapReduce based processing consists of two steps:

- *Map* step, where the input data is divided into small subsets (usually by a special node in the cluster, called "master" node) and these subsets are distributed for actual processing by the "worker" nodes. When a worker node is finished with the processing it sends the results back to the master node
- *Reduce* step, where the aggregated results from processing of the small data subsets by each node are being combined to form the answer of the original problem being solved

Hadoop implements the MapReduce approach to distributed data processing.

Architecture

The architecture of Hadoop is based on a classic Master/Slave paradigm, where one or more master nodes are responsible for the general task scheduling, coordination and workflow, while the Worker nodes perform a specific task over a subset of the input data.

In Hadoop (Figure 2), the master node is called *JobTracker* and it is responsible for scheduling, coordinating, monitoring and failure handling of the whole cluster. Each worker node is comprised of a *TaskTracker* (responsible for setting up tasks sent by the master and communication with it) and one or more *TaskRunners* (which execute the actual Map or Reduce tasks scheduled for this worker node). The worker nodes (TaskTrackers) in the Hadoop cluster are configured to send progress updates to the master node (JobTracker) every 3-5 seconds. If the master does not receive an update for an extended period from a particular worker, then this node will be considered down and the tasks that were allocated to it will be re-distributed to the other workers by the master.

[10] http://zookeeper.apache.org/
[11] https://en.wikipedia.org/wiki/MapReduce

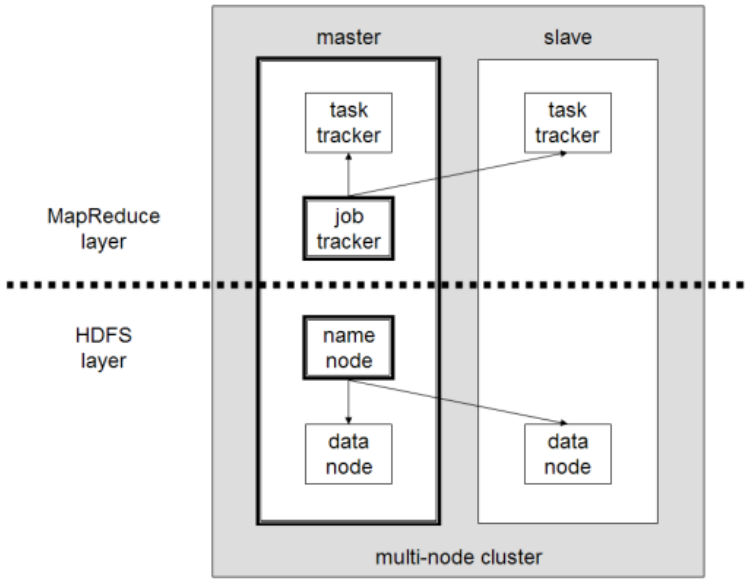


Figure 2 Hadoop architecture, (c) Wikipedia

HDFS

HDFS is the distributed file system of the Hadoop platform. It is based on the same Master/Slave architecture as Hadoop (Figure 3), with a single master node called *NameNode* which is responsible for storing the metadata (filesystem tree for all the directories and files stored on the cluster) and cluster node coordination, and several *DataNodes* which store individual blocks of data sent by client applications using the HDFS cluster.

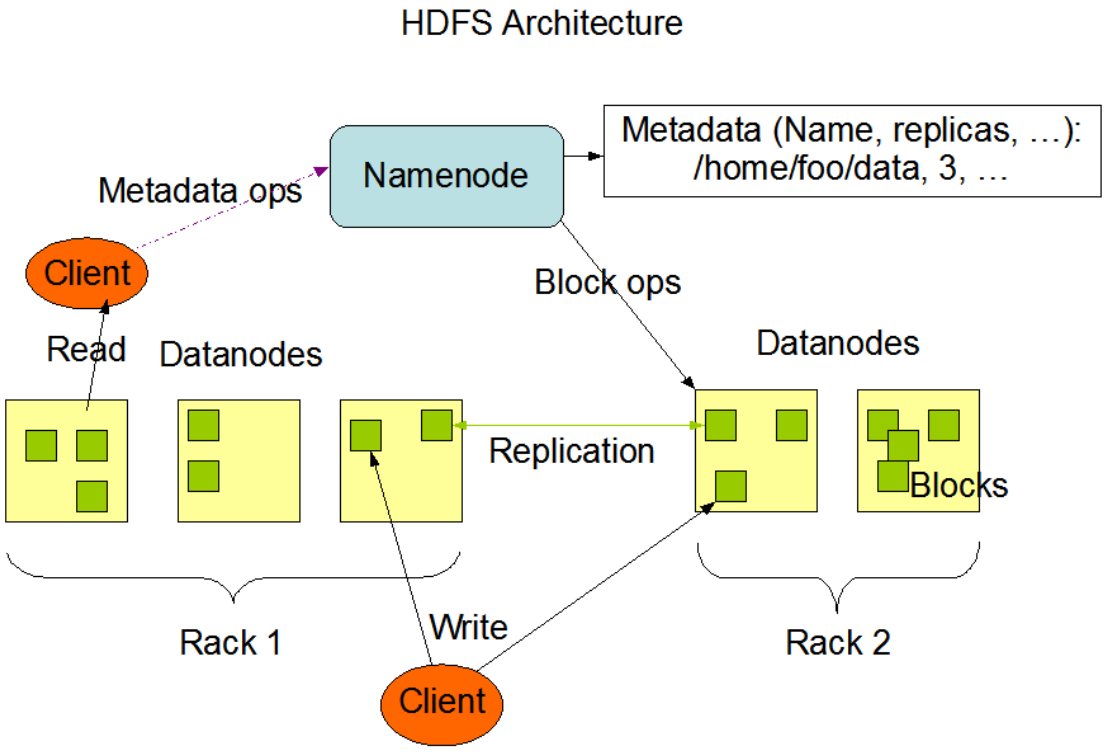


Figure 3 HDFS Architecture, (c) Apache Hadoop

In order to avoid the NameNode becoming a bottleneck for the data throughput of the cluster, data which is read/written on HDFS is never moved through the NameNode itself, but is exchanged directly between the client applications (or nodes) and the DataNodes of the cluster.

In order to guarantee the availability of the data, all data stored on the HDFS cluster is replicated (usually on three DataNodes). The DataNodes send periodical "hearbeats" to the NameNode every few seconds, and if the NameNode does not receive a heartbeat from a particular DataNode for an extended period, then that node will be considered non-functioning and all the subsequent requests for pieces of data located on the failed DataNode will be re-routed to and served by one of its replicas.

Storm

Storm[12] is an open-source[13] platform for distributed real-time computation. Initially Storm was developed and used internally within Twitter for its real-time stream processing infrastructure, but in 2011 Twitter decided to open source and release the Storm code to the general public.

Architecture

The architecture of a Storm cluster is similar to that of Hadoop and based on the classic Master/Slave architecture. The master node in a Storm cluster is called *Numbus* and it corresponds to the Hadoop's *JobTracker*. It is responsible for distributing the tasks to the worker nodes, monitoring and management of failed nodes in the cluster. The worker nodes (called *Supervisors*) receive tasks from the Master and execute the actual processes to complete the tasks. The actual coordination between the master (Numbus) and the workers (Supervisors) is not direct, but via one or more ZooKeeper[14] agents, which provide a generic coordination service for distributed systems.

An important difference between Storm and Hadoop is that while a MapReduce job in Hadoop will eventually finish when all the Map and Reduce tasks are complete, a Storm job (called a "topology") is designed so that it never finishes, but processes messages from an input stream, and generates output messages as long as the Storm cluster is running.

[12] http://storm-project.net/
[13] Distributed under an Eclipse Public License, http://www.eclipse.org/legal/epl-v10.html
[14] http://zookeeper.apache.org/

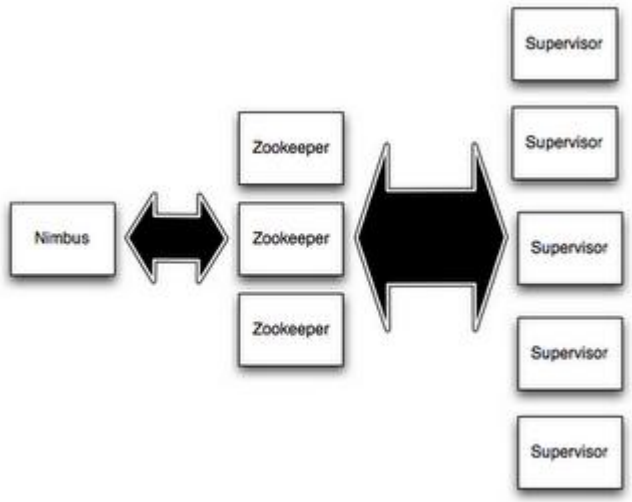


Figure 4 Storm Cluster, (c) Storm Project

Concepts

The main Storm concepts are:

- **Topologies** – analogous to a MapReduce job, the topology describes the processing tasks that should be performed over the input data and the output that will be generated
- **Tuples** – a tuple is the main data structure in Storm, it is comprised by a set of key/value pairs. The values can be of arbitrary type, including general container types (list, set, map, etc.).
- **Streams** – a stream represents and unbounded sequence of tuples that will be processed. Each stream has a unique identifier and is associated with a schema defining the structure of its tuples.
- **Spouts** – provide an abstraction for a data source. A spout reads data from some external system or datastore and writes tuples into one or more streams. Note that the tuples are read in a sequential fashion, e.g. spouts support only one method for reading data, called *nextTuple()* which will return the next tuple available in the data source (or an empty result if there is no more data to be processed at this point in time).
- **Bolts** – provide an abstraction for a processing task. Any kind of data processing (transformation, filtering, aggregation, joins, etc) is represented by means of a *bolt*. Bolts read input data from one or more input streams, and emit output data into one or more output streams.
- **Tasks** – represent concurrent threads, which execute the same data processing steps as defined by a single bolt. The level of parallelism of a bolt (e.g. the number of concurrent tasks executed at each moment) is specified during the topology initialisation process. The Storm framework takes care of sending the data generated from the parallel task threads into the respective output streams associated with the bolt of the task.

Figure 5 shows a simple Storm topology with two spouts emitting data into four streams, which are then processed by four bolts.

Figure 6 shows a more complex example, where the spouts and the bolts are distributed among several nodes in a cluster so that the data processing is parallelised.

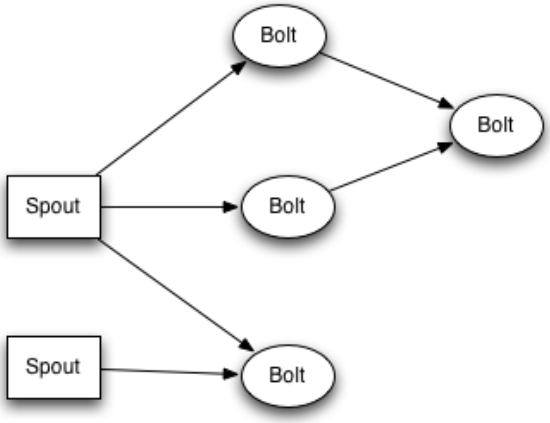


Figure 5 Storm topology, (c) Storm Project

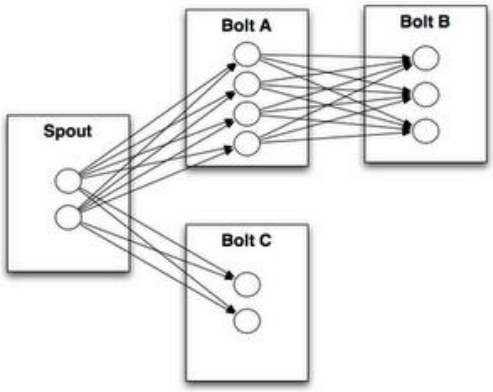


Figure 6 Distributed Storm topology, (c) Storm Project

HPCC

High Performance Computing Cluster[15] (HPCC) is an open source[16] parallel processing platform for Big Data, initially developed by LexisNexis[17] (Middleton, 2011).

Architecture

HPCC is comprised of two platforms, Thor (Data Refinery) and Roxie (Data Delivery Engine) which target different data processing tasks and can be additionally customised according to the specific needs of a use case or a domain (Figure 7). A particular HPCC deployment may include only a Thor cluster or both Thor and Roxie.

[15] http://hpccsystems.com/
[16] Distributed under an Apache Software License, http://www.apache.org/licenses/LICENSE-2.0
[17] http://www.lexisnexis.com/risk

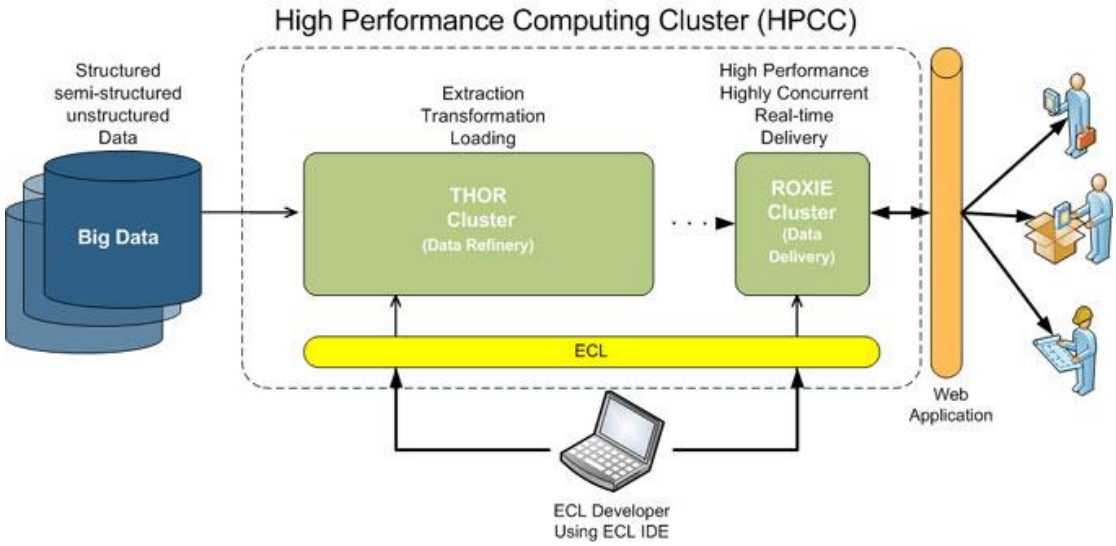


Figure 7 HPCC architecture, (c) HPCC Systems

Data Refinery (Thor)

Thor provides distributed ETL[18], storage and indexing functionality within the HPCC platform. The storage functionality in Thor is based on the Thor Distributed File System (DFS), which is built on top of the Linux file system and is based on a Master/Slave architecture similar to HDFS. There are several additional similarities between HDFS and the Thor DFS:

- each of the slave nodes in a Thor cluster also serves as a data node, just like the slave nodes in a Hadoop cluster are also HDFS data nodes
- there is a single master node in Thor, called *Dali*. It corresponds to the *NameNode* in a HDFS cluster
- data stored on a Thor DFS is replicated among several physical nodes in the cluster in order to improve the reliability and availability of the cluster.

The batch jobs executed on the Thor cluster are defined in ECL (mode details are available in section ECL)

Data Delivery Engine (Roxie)

Roxie provides distributed storage, querying and data warehousing functionality within the HPCC platform.

Roxie also provides a DFS[19] that is designed for efficient indexing and querying and utilizes a distributed B+Tree index. The Roxie DFS roughly corresponds to the HBase[20] and Hive[21] components of the Hadoop platform.

[18] http://en.wikipedia.org/wiki/Extract,_transform,_load
[19] Note that the Thor and Roxie DFS are different
[20] http://hbase.apache.org/
[21] https://hive.apache.org/

ECL

Enterprise Control Language (ECL) is the dataflow programming language for distributed querying and manipulation of Big Data on the HPCC platform. ECL can be further extended by custom 3rd party libraries (in C++) and is compiled into C++ code as well.

ECL provides high-level programming primitives for data transformation, aggregation and querying, statistical operators, workflow operators, as well as the option to extend the language with 3rd party functionality (HPCC Systems, 2012) & (HPCC Systems, 2012).

ECL roughly corresponds to the Pig[22] data processing language from the Hadoop platform.

NoSQL datastores

NoSQL datastores[23] represent a class of databases, which are not based on the relational model of traditional databases. The NoSQL family includes a broad range of different databases such as key/value datastores, column databases, document databases, graph/RDF databases.

NoSQL databases are designed with different goals than traditional RDBMS and the focus is on:

- low and predictable response time
- scalability and elasticity
- high availability
- flexible schema management

In order for NoSQL databases to satisfy the above requirements, they sacrifice some traditional features for RDBMS, such as transactions, strong consistency and complex query support.

This section will provide a brief overview of two of the most popular NoSQL databases, namely HBase and Cassandra. Other popular open source NoSQL databases, which fall outside the scope of this deliverable, include CouchDB[24], MongoDB[25], Voldemort[26] and Redis[27].

HBase

[22] http://pig.apache.org/
[23] We provied the generally accepted definition of a "datastore" from Wikipedia: *A data store is a data repository of a set of integrated objects. These objects are modeled using classes defined in database schemas. Datastores include not only data repositories like databases, but also more general concepts like flat files that can store data*
[24] http://couchdb.apache.org/
[25] http://www.mongodb.org/
[26] http://project-voldemort.com/
[27] http://code.google.com/p/redis/

HBase[28] (George, 2011) is an open source[29] distributed key-value datastore. It is based on the design of Google's BigTable datastore (Dean, et al., 2006).

The architecture of HBase is based on a Master/Slave architecture, with data stored on the Hadoop filesystem (HDFS) and ZooKeeper coordinating the nodes in an HBase cluster. The master node of the cluster is called *Master Node*, and the slaves, which are responsible for managing the data and handling user requests, are called *Region Servers*. The HBase deployment can be replicated in order to provide higher availability and reliability (Figure 8).

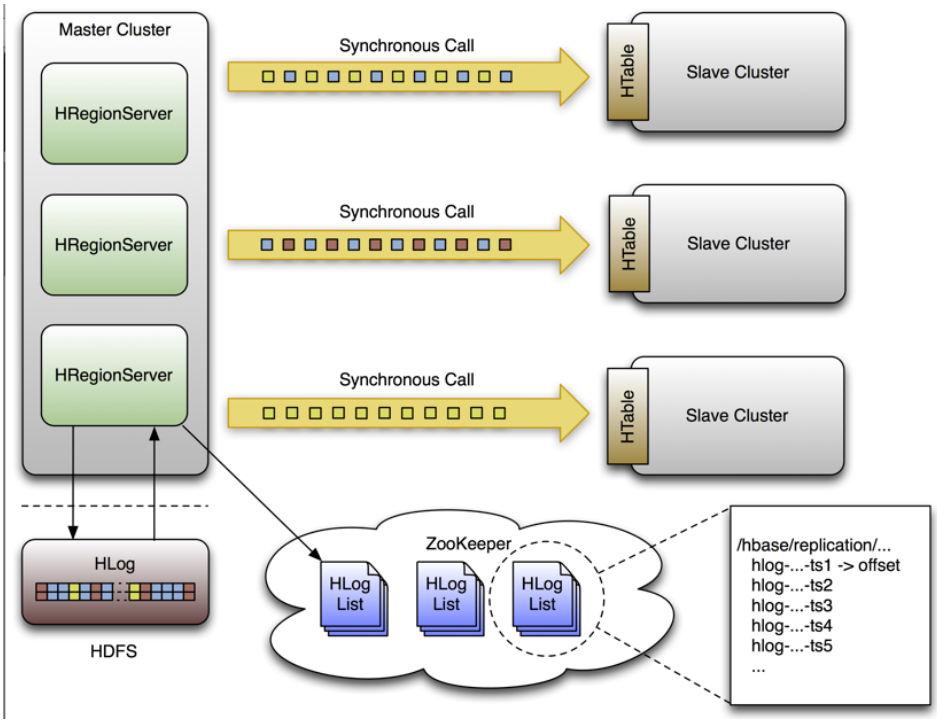


Figure 8 HBase replicated cluster, (c) HBase/Apache

The features of HBase include:

- almost linear scalability
- consistent reads and writes
- automated partitioning and re-balancing of data in the cluster
- automated failover and fault tolerance

Cassandra

Cassandra[30] (Hewitt, 2010) is an open source[31] distributed column store, originally developed by Facebook. It is based on the design of Google's BigTable datastore (Dean, et al., 2006) and Amazon's Dynamo datastore (DeCandia, et al., 2007).

[28] http://hbase.apache.org/
[29] Distributed under an Apache Software License, http://hbase.apache.org/license.html
[30] http://cassandra.apache.org/
[31] Distributed under an Apache Software License, http://www.apache.org/licenses/

The design goals of Cassandra include high availability (with *eventual consistency*), incremental scalability and elasticity, and optimistic replication of data. An important characteristic of the Cassandra architecture is that it is P2P based and there is no distinction of Master and Slave nodes. Such architecture removes the "single point of failure" risk present in Master/Slave architecture and greatly improves the elasticity of the cluster since throughput can dynamically increase or decrease as new nodes are added to / removed from the cluster.

Since there is no Master node, which can route read/write requests to the proper Slave data node, in the Cassandra cluster the read and write operations are performed in the following manner (Figure 9):

- write requests are sent to a *random* node in the cluster. The node considers which node in the cluster is responsible for storing the data (based on the consistent hashing strategy employed by Cassandra) and the data to be stored is re-routed to the appropriate node. Additionally, data is replicated on *N* other nodes (where *N* is specified at cluster configuration)
- read requests are sent to a *random* node in the cluster. The node considers which are the N nodes storing replicas of the data and re-routes the request to **all** the N nodes. Based on the read strategy, the value return to the client is either read from the first node to respond to the request, or is based on a "quorum read" strategy where the value that the majority of the nodes agree on is returned as valid[32].

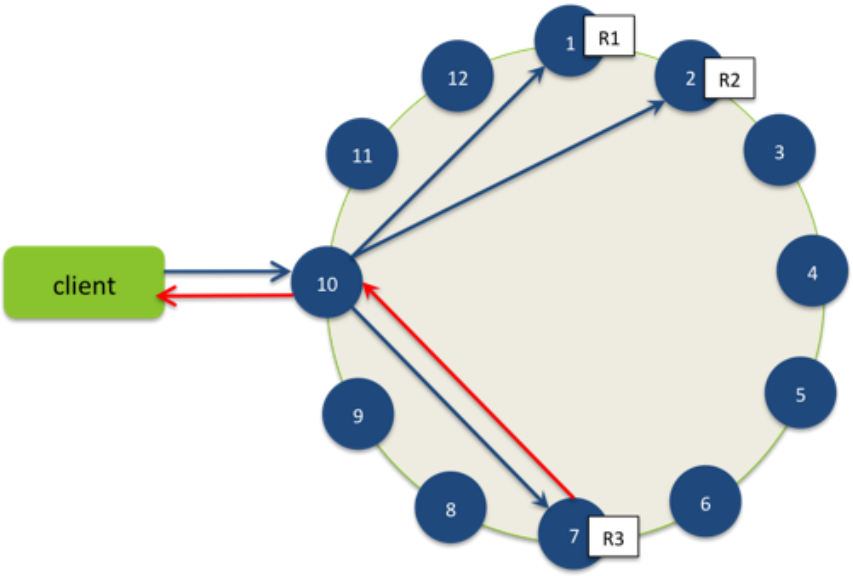


Figure 9 Cassandra P2P cluster, (c) Datastax

Pig

[32] Keep in mind that Cassandra employs *eventual consistency*, so data changes may take some time to propagate across the P2P cluster and there may be nodes with inconistent (out of date) data.

Pig[33] (Gates., 2011) and (Olson, Reed, Srivastava, Kumar, & Tomkins, 2008) is the high-level data manipulation and querying language part of the Hadoop platform. Pig scripts are compiled into a sequence of MapReduce programs that can be executed on a Hadoop cluster. The Pig language itself aims at providing an easy abstraction for expressing highly parallel data manipulation and querying tasks where the software developer focuses on expressing easy to understand data flow sequences, while the Pig compiler and optimiser take care of generating an optimal MapReduce code and execution plan for it.

Architecture

A Pig deployment is comprised of a set of client tools (command line shell or programming APIs) and a Pig server, which is responsible for executing Pig scripts. Execution of any Pig script involves the following steps (Figure 10):

- *parsing*, which generates an initial logical execution plan for the tasks described in the Pig script
- *optimisation*, which applies various optimisation techniques in order to generate an optimal logical execution plan for the script
- *compilation*, which translates the original Pig script into a set of MapReduce programs in Java, which can be executed on any Hadoop cluster
- actual *execution* and aggregation of results, based on the output of the MapReduce tasks executed in parallel on the Hadoop cluster

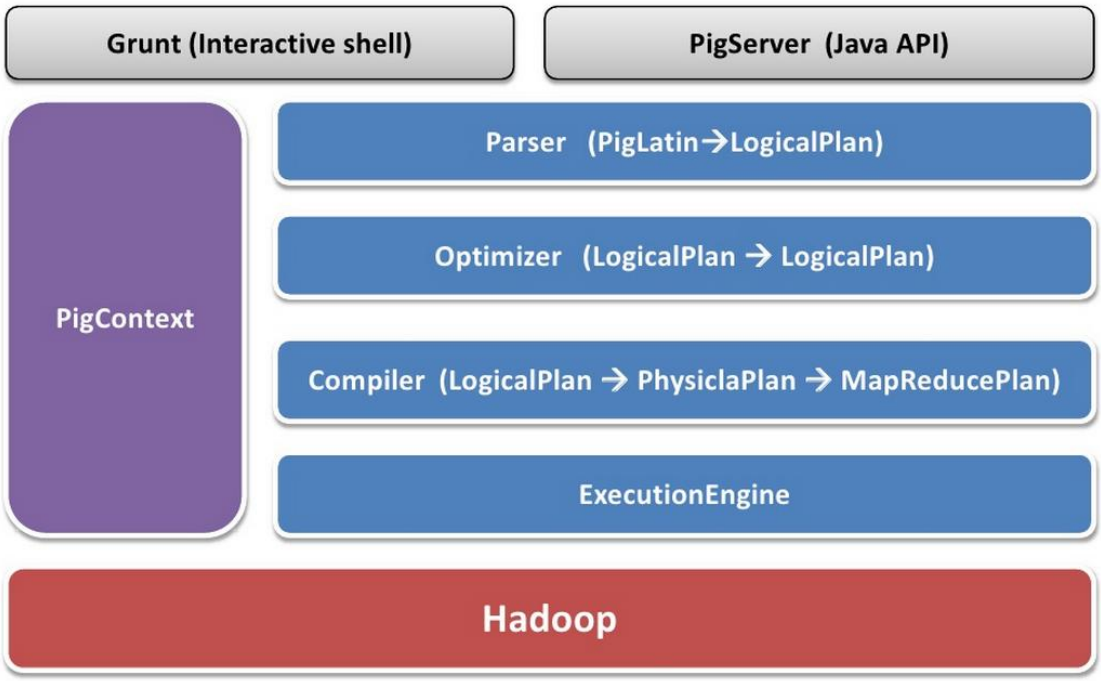


Figure 10 Pig Architecture, (c) J. Zhang / Apache Software Foundation

Figure 11 provides a sample Pig program, which provides a solution to the problem of finding the top 5 most visited pages on a website by its customers in the 18-25 age

[33] http://pig.apache.org/

range. The input data is comprised of two data sets (user profiles and a web site log file) and the solution involves filtering, merging, aggregation and sorting of data from datasets distributed on a HDFS cluster.

Pig Latin Script	Work Plan

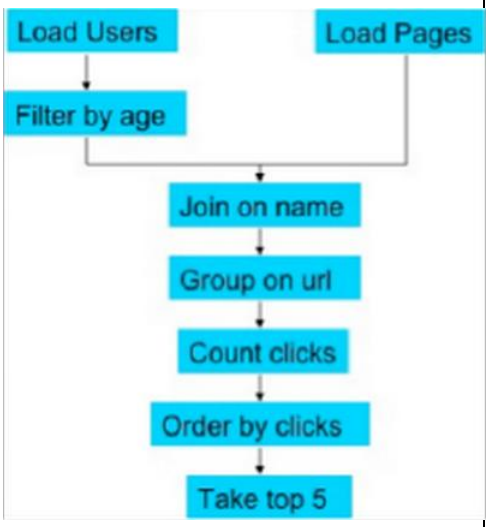
users = LOAD 'users' AS (name, age);
users = FILTER users BY age >= 18 AND age <= 25;
pages = LOAD 'pages' AS (user, url);
joined = JOIN users BY name, pages BY user;
grouped = GROUP joined BY url;
summed = FOREACH grouped
GENERATE group,
COUNT(joined) AS clicks;
sorted = ORDER summed BY clicks DESC;
top5 = LIMIT sorted 5;
STORE top5 INTO 'top5sites';
	

Figure 11 Sample Pig program

Mahout

Mahout[34] (Owen, Anil, Dunning, & Friedman, 2011) is an open source[35] scalable platform for distributed machine learning, built on top of Hadoop.

Mahout provides scalable implementations[36] of various machine learning algorithms:

- classification
- clustering
- pattern mining
- regression
- dimension reduction
- collaborative filtering
- and various genetic algorithms

[34] https://mahout.apache.org/
[35] Distributed under an Apache Lincese, https://www.apache.org/licenses/
[36] The complete list of algorithms implemented on top of Mahout is available at https://cwiki.apache.org/confluence/display/MAHOUT/Algorithms

In addition to the core components providing distributed machine learning algorithms, Mahout provides a set of utilities for converting common input data (such as various log files), tools for visualisation (Figure 12), training data collections for testing and tuning algorithms, as well as performance benchmarking tools.

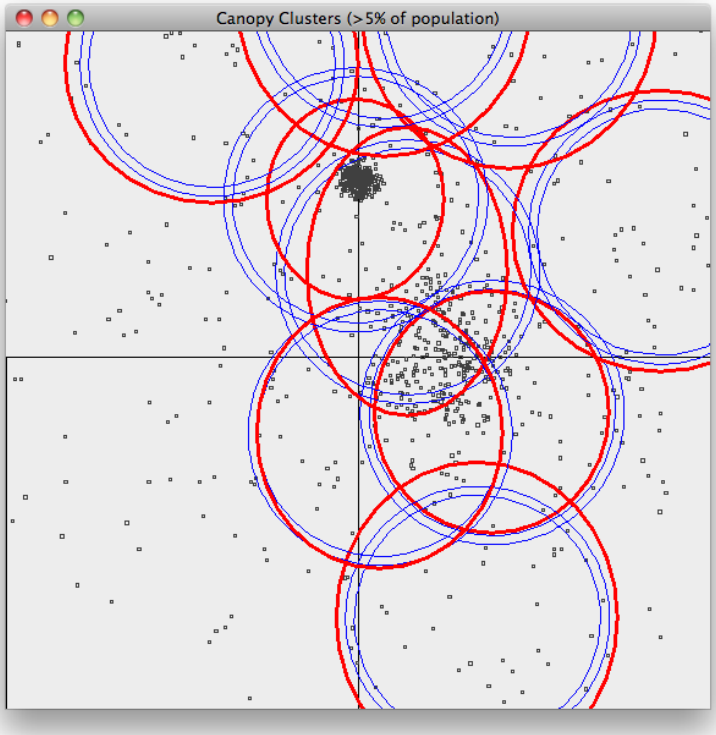


Figure 12 Clustering Visualisations, (c) Apache Mahout

Memcached

Memcached[37] is an open source[38] distributed memory cache. It serves as a very fast in-memory key-value store, which can be distributed among hundreds or thousands of machines in a cluster. The Memcached server can be accessed by various client APIs in Java, C/C++, Python, Perl and others[39].

The main operations supported by the caching server are:

- *storage commands* – set, check-and-set, add, replace, append
- *retrieval commands* (get)
- *deletion* (delete)
- *statistics* – various commands returning runtime information about the cache server
- *cache invalidation* (flush_all)

[37] http://memcached.org/
[38] Distributed under a BDS license, http://opensource.org/licenses/BSD-3-Clause
[39] https://code.google.com/p/memcached/wiki/Clients

Memcached provides an easy and scalable solution for cases when a lot of small data objects are frequently accessed by many simultaneous clients. For example, Facebook announced[40] that their 800-node Memcached cluster supports around 300,000 object requests per second, at an average latency of only 170 microseconds.

[40] https://www.facebook.com/note.php?note_id=39391378919

Architecture for Distributed Trend Mining

Based on the requirements and challenges for scalable text and trend mining outlined in section "Data Processing in TrendMiner" and the various technologies for scalable Big Data processing summarised in section "Distributed Approaches to Processing Big Data", this section provides an initial recommendation for a scalable architecture for distributed text and trend mining over social media streams.

General Recommendations

From the list of approaches for scalable Big Data processing the following two will NOT be considered for use within TrendMiner:

- *HPCC* – it offers powerful capabilities for scalable big data processing but within the context of TrendMiner Hadoop is a better technological choice for several reasons
  - most technical partners already have expertise with Hadoop
  - some of the components to be used in TrendMiner already have initial or partial prototypes based on Hadoop
  - Hadoop is Java based (HPCC requires components to be implemented in C++) and it makes it easier to port to it existing code, such as GATE which has big and complex existing codebase
  - The Hadoop ecosystem is richer than HPCC and offers various tools for data processing (see section "Hadoop" for more details)
- *Cassandra* – it offers a very scalable distributed data storage solution and in some scenarios it may provide a definite advantage over HBase, but for the needs of TrendMiner HBase is an adequate choice[41] and it is already in use within the focussed data crawling architecture of one of the TrendMiner partners (IMR)

A second important clarification regarding the architecture for distributed text and trend mining is that the list of approaches for Big Data processing is *not prescriptive*, e.g. these approaches provide potential solutions but the exact suitability in the context of TrendMiner is yet to be evaluated, benchmarked and verified[42]. There may be minor deviations in the final list of technologies which will be employed by WP2-5 for the first integrated prototype of the TrendMiner platform (D5.3.1, due M24) based on the results of the aforementioned evaluation and verification process.

Scaling out the TrendMiner Data Processing Tasks

Table 1 provides a summary of the applicability of the various Big Data processing approaches to the specific data processing tasks to be performed in TrendMiner. We use a very simple notation, e.g.:

[41] The features of HBase, which are most important for TrendMiner, include: (1) automated partitioning; (2) automated failover; (3) replication; (4) good integration with the Hadoop ecosystem. Advanced Cassandra features such as: no single point of failure (P2P architecture) are not differentiating for the needs of TrendMiner.

[42] For example, there is an ongoing work in evaluating and benchmarking Pig in the context of TrendMiner related data processing

- **3 stars** means that the technology is *highly applicable* for the needs of the particular data processing task and it WILL BE used throughout the project
- **2 stars** means that the technology is *generally applicable* and WILL BE used at least for some of the sub-tasks within a particular data processing task
- **1 star** means that the technology is somewhat applicable to the data processing task and it MAY BE used for the actual implementation (further research is needed)
- Technologies, which are not directly relevant or which require significant adaptations in order to be used for some data processing task, WILL NOT be used.

Table 1 Approaches to scaling TrendMining data processing

Task / Approach	Hadoop / MapReduce	Hadoop / HDFS	Storm	HBase	Pig	Mahout	Memcached
Data Collection	***	**	*	***	*	-	-
Pre-processing	***	**	*	*	**	-	-
Multilingual Information Extraction	***	**	*	*	*	*	**
Entity Disambiguation and Linking	**	-	*	-	-	-	**
Sentiment Extraction / Trend Detection	***	**	*	**	*	**	*
Summarization	***	**	*	**	-	*	*
Stream reasoning	*	*	***	**	-	-	**
Querying over annotated streams	*	-	*	*	-	-	**

Data Collection

The data collection infrastructure being developed by IMR (Figure 13) already employs distributed processing based on Hadoop and distributed storage with HBase.

Since this task does not require real-time processing, but is mostly batch oriented, Storm is unlikely to provide any advantage and will most probably not be employed for this task within the TrendMiner platform. Tools such as Mahout and Memcached are not applicable to this task.

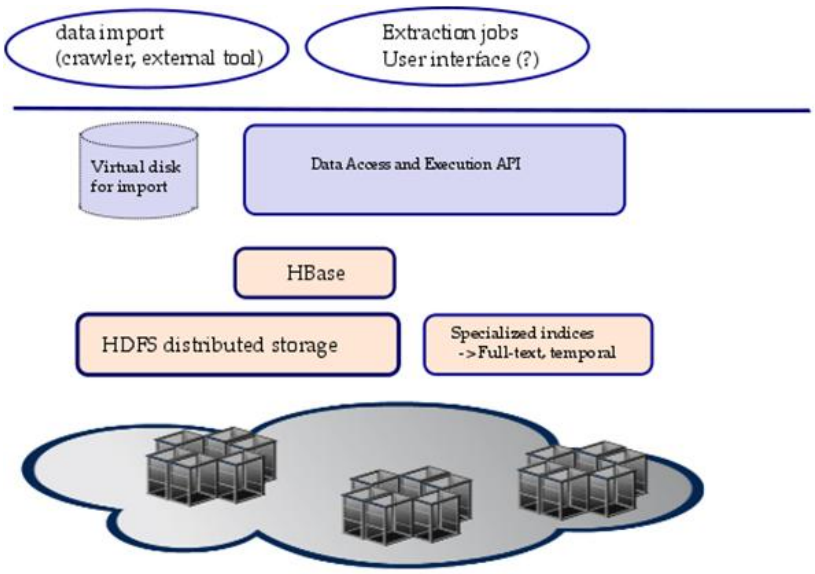


Figure 13 IMR distributed crawling architecture (WP5)

Pre-processing

Pig is suitable for distributing and simplifying the data cleanup and pre-processing tasks (implemented in Java or custom scripts at present). It relies on Hadoop and HDFS, and so these technologies will also be part of the solution. An additional area where Pig may be applicable is simple ETL tasks related to data transformation between components in the processing pipeline (to guarantee that the output of one component is formatted or transformed in a way which makes is suitable for input for the next component in the workflow).

Distributed storage based on HBase and real-time processing with Storm does not provide a significant advantage for this task, so these technologies will most probably not be part of the final solution. Tools such as Mahout and Memcached are not applicable to this task.

Multilingual Information Extraction, Sentiment Extraction and Trend Detection, Summarization

These three tasks have very similar profiles and requirements for distributed data processing:

- Hadoop and HDFS provide the general framework for distributed data processing and temporary data storage
- HBase provides access to original data (social streams) for processing
- Mahout can speed up the machine learning and statistical algorithms employed within these tasks (the prediction phase, not the initial training phase)
- Memcached is useful for speeding up read access to commonly accessed data (static resources, results of repeatedly executed structured queries, etc)
- Pig and Storm do not provide a significant advantage, so they will not play a major role in the implementation of this task

As already mentioned in section "Sentiment Extraction and Trend Detection" the Machine Learning task has two distinct phases: training and prediction, which have different profiles in terms of the data velocity and volume, real-time processing and computational overhead. The training phase will be performed with tools such as MATLAB. The prediction phase over the real time streams may be based on Mahout, though as of the time of writing this deliverable the subset of Machine Learning methods available for Mahout is not sufficient for the complex models that will be developed within TrendMiner. Mahout's suitability for the real-time prediction in TrendMiner is yet to be verified and for the first platform prototype (due M24) we may consider not using Mahout at all and porting the Machine Learning algorithms directly on Hadoop and MapReduce.

Entity Disambiguation and Linking

For entity disambiguation, Hadoop and Memcached provide the means for distributed processing and caching of commonly accessed data (static resources, results of repeatedly executed structured queries, etc). The other Big Data technologies do not provide a significant advantage

Stream reasoning

For stream reasoning the major advantage is provided by Storm and its real-time data processing capabilities (Figure 14). Memcached additionally improved query speed for commonly executed queries over static data. The rest of the technologies do not provide significant advantages (Hadoop/Pig are more suitable for batch oriented processing)

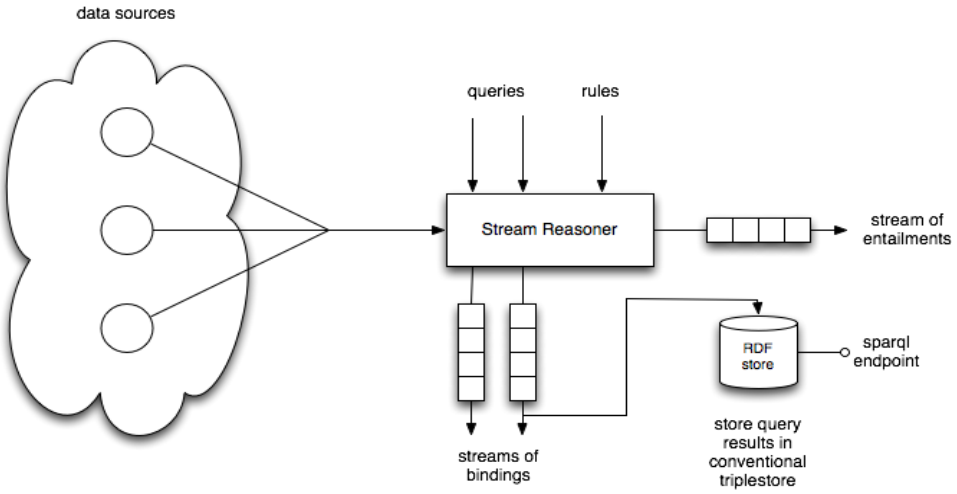


Figure 14 Stream reasoning with Storm (WP2)

Within WP2 there is already work in progress related to the implementation of a scalable real-time reasoning based on Storm and the Rete algorithm[43]. Figure 15 provides an example about implementing RDF triple patterns as Storm streams and bolts.

[43] http://en.wikipedia.org/wiki/Rete_algorithm

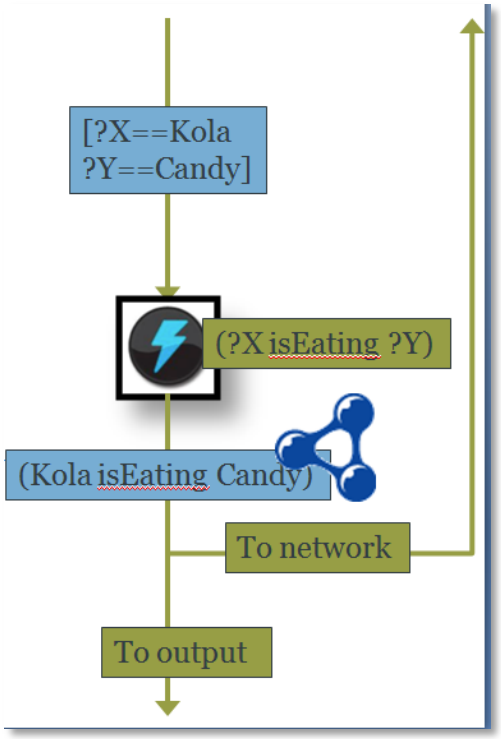


Figure 15 RDF rules implemented in Storm

Querying over annotated streams

For the final task of the TrendMiner data processing pipeline, a combination of a RDF database cluster (OWLIM) and a distributed key-value store (HBase) will be employed. The RDF database will be responsible for answering complex structured queries over the annotated data, while HBase will provide fast access to the original content from social media streams. Memcached will also be employed in order to reduce the load of the databases by caching most commonly executed queries and results.

Distributed Trend Mining Architecture

In the previous sections we have ranked the applicability of various technologies for scalable Big Data processing to the data processing tasks in TrendMiner (Table 1). As already noted, these technologies need to be further evaluated and benchmarked so that their suitability is verified in the context of TrendMiner.

Even though some changes to the concrete list of technologies used may be necessary for the final architecture and platform prototype (due M24), the initial TrendMiner architecture is already defined and is not going to undergo major changes for the duration of the project (Figure 16).

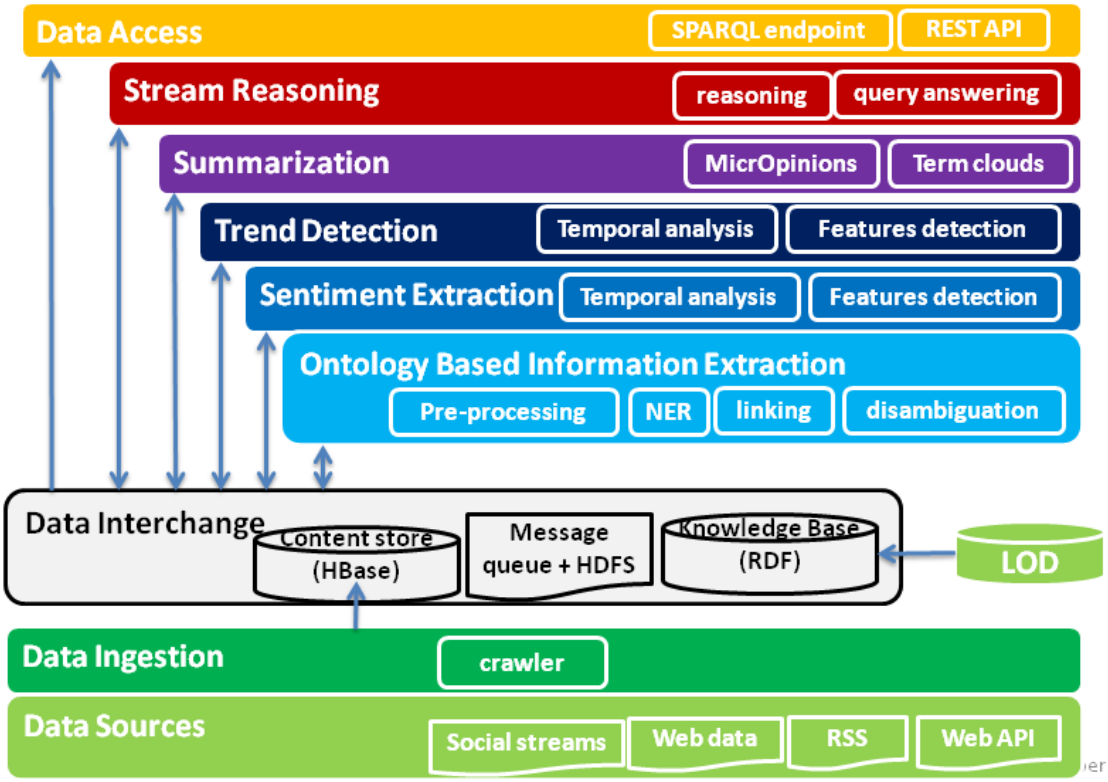


Figure 16 TrendMiner Architecture

We present a layered architecture capturing the various data processing tasks in separate layers. It is important to note that the dataflow is not necessarily linear (from bottom to top) and it is possible for the upper layers to provide input which can be further used in the data processing tasks of the lower layers.

General clarifications and notes regarding the architecture:

- HBase is used as a distributed **content store** (for data ingested from input data sources). Only the Data Ingestion layer components add data to the content store, all other components and layers only read data from it.
- An RDF Database (OWLIM) is used as a **Knowledge Base**, storing various annotations generated from data processing tasks in TrendMiner in RDF
- **Data exchange** between various components is based on a loosely coupled approach via distributed message queues and distributed filesystem (HDFS)
- **Linked Open Data** providing external knowledge from various sources is imported and used throughout the Ontology Based IE, Sentiment Extraction and Summarisation phases
- **GATE** is used as the general platform for most of the text processing related tasks. We also use NooJ[44] for processing of German and Italian texts.
- For the purpose of simplifying the diagram various **pre-processing tasks** responsible for data transformations between the layers are omitted.
- **Hadoop** is the platform used for distributed text processing and machine learning.

[44] http://www.nooj4nlp.net/pages/nooj.html

- **Storm** is the platform used for stream reasoning. The results of the stream reasoning are stored persistently in the Knowledge Base (OWLIM) for further use.
- The **Data Access Channels** provide access to the data generated by all the text mining and trend mining processing tasks to the User Interface layer (omitted from this diagram). The data is accessed (in a read-only manner) from the Content Repository (HBase) or the Knowledge Base (OWLIM). The two data access channels envisioned at present are: (1) a SPARQL endpoint; and (2) a RESTful service

Future Work

This deliverable provides two main outcomes for WP5 and the project in general:

- Analysis and recommendations of the relevance of various approaches to scalable Big Data processing to the data processing needs of the project
- An initial version of the distributed text annotation and trend mining architecture for the project

Within the second year of the project the outcomes of the deliverable will be used as a basis for D5.3.1 (due M24) which has the goals of providing a refined 2nd version of the TrendMiner architecture as well as the initial prototype of the integrated platform for real-time stream processing. In particular, concrete outcomes of D5.3.1 will be:

- Refined architecture for distributed text annotation and trend mining
- Recommendations for scaling various text mining and machine learning algorithms on GPU platforms
- Initial implementations of scalable text mining & machine learning alrogirhms
- Fully integrated initial set of components that cover the various apsects of stream data processing in TrendMiner

Bibliography

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. *OSDI'04: Sixth Symposium on Operating System Design and Implementation.* San Francisco.

Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., et al. (2006). Bigtable: A Distributed Storage System for Structured Data. *Seventh Symposium on Operating System Design and Implementation.* Seattle, WA.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., et al. (2007). Dynamo: Amazon's Highly Available Key-value Store. *SOSP'07.*

Gates., A. (2011). *Programming Pig.* O'Reilly.

George, L. (2011). *HBase: The Definitive Guide.* O'Reilly Media.

Hewitt, E. (2010). *Cassandra: The Definitive Guide.* O'Reilly Media.

HPCC Systems. (2012). *ECL Language Reference.*

HPCC Systems. (2012). *ECL Programmers Guide.*

Middleton, A. (2011). *HPCC Systems: Introduction to HPCC.*

Olson, C., Reed, B., Srivastava, U., Kumar, R., & Tomkins, A. (2008). Pig Latin: A Not-So-Foreign Language for Data Processing. SIGMOD.

Owen, S., Anil, R., Dunning, T., & Friedman, E. (2011). *Mahout in Action.* Manning.