FP7-ICT Strategic Targeted Research Project TrendMiner (No. 287863)

Large-scale, Cross-lingual Trend Mining and Summarisation of Real-time Media Streams

D5.3.1 Architecture for distributed text annotation - v2; Real-time Stream Media Processing Platform - v1

Alex Simov (Ontotext)

Petar Kostov (Ontotext)

Abstract

FP7-ICT Strategic Targeted Research Project TrendMiner (No. 287863)
Deliverable D5.3.1 (WP 5)

This deliverable provides an updated architecture as well as the first integrated implementation of the TrendMiner platform. It focuses on the integration aspects of the separate components (APIs and data exchange formats) as well as on the overall system workflow. The document content reveals the implementation of all phases of the data processing lifecycle, from data harvesting to end user representation.

Keyword list: Integration, REST, scalability, cloud computing, GPU

Nature: Report Dissemination: PU
Contractual date of delivery: M24 Actual date of delivery: 31 Oct 2013
Reviewed By: Nick Gibbins
Web links: n/a

CHANGES

Version	**Date**	**Author**	**Changes**
0.1	23.10.2013	Alex Simov	Initial version, content structure
0.2	25.10.2013	Alex Simov	Integration implementation sections
0.3	30.10.2013	Alex Simov	Components description
0.4	30.10.2013	Petar Kostov	GPU sections
1.0	31.10.2013	Alex Simov	Proof reading and final editing

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 Rvolution
F-93100 Montreuil
France
Contact person: France Lafarges
E-mail: contact@internetmemory.org

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

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

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

Executive Summary

This deliverable provides an updated architecture as well as the first integrated implementation of the TrendMiner platform. It focuses on the integration aspects of the separate components (APIs and data formats) and the overall system workflow. The document content focuses on the implementation of all phases of the data processing lifecycle, from data harvesting, through various data processing phases to end user representation.

The structure of the deliverable is as follows:

- high level architecture overview identifying the various tasks and challenges;
- overview of the first version of the integrated TrendMiner platform, providing high level description of the implemented workflow;
- description of the APIs of the separate components and their input/output data specification;
- more detailed description of the integration implementation itself, focusing on the important interaction points;
- finally, we present a research work on applicability of massively parallel processors for accelerating workloads within the TrendMiner system.

Contents

List of abbreviations

UX	User Experience
REST	REpresentational State Transfer
HDFS	Hadoop Distributed File System
RDF	Resource Description Framework
GPU	Graphics Processing Unit
CPU	Central Processing Unit
APU	Accelerated Processing Unit
GPGPU	General Purpose GPU
API	Application Programming Interface
I/O	Input/output
EU	Execution Unit
PU	Processing Unit
SIMD	Single Instruction Multiple Data
LLC	Last Level Cache

List of figures

1 Architecture Overview

TrendMiner will provide a platform for distributed and real-time processing over social streams. 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).

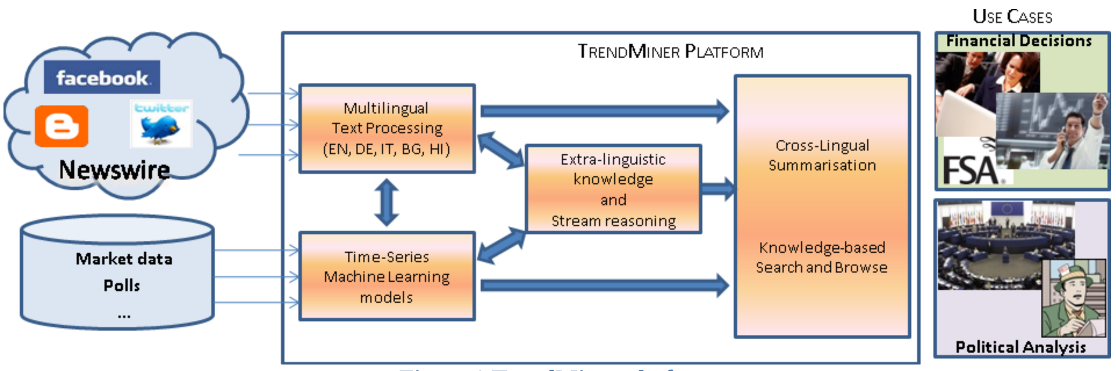
facebook
twitter
Newswire
Market data
Polls
...
TRENDMINER PLATFORM
Multilingual Text Processing (EN, DE, IT, BG, HI)
Time-Series Machine Learning models
Extra-linguistic knowledge and Stream reasoning
Cross-Lingual Summarisation
Knowledge-based Search and Browse
USE CASES
Financial Decisions
FSA
Political Analysis

Figure 1 TrendMiner platform

1.1 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. The major challenges for this task are: data volume, velocity (timely new data harvesting) and variety (heterogeneity of sources and formats)

Pre-processing

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.

[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”

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.

Data volume and data velocity are the main challenges for this task. Processing large volumes of data means that there will be lots of 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.

2 Integrated Platform Overview

The following Figure 2 provides an overview of the first version on the integrated TrendMiner platform. It identifies the major functional blocks of the system covering all the phases of the social media data processing chain: data collection; applying different analysis and transformations; summarization and visualization. The platform integrates all relevant components available in M24 by the partners from the technical work packages.
The data collection is supported by two complementary components responsible for monitoring different social media sources. The processing services are classified in two groups depending on their usage in the processing lifecycle. The first group performs pre-processing of the resources (metadata extraction) regardless any context of usage - named entities detection, language identification, geo-location detection. The results produced by these components are stored in the data repository and serve as a base for subsequent data searching and browsing. The second group of services computes collective analytical information based on user defined context and resources selection. The outcome of these services is intended for direct visualization and is not persisted in any way. Finally the role of the UX Data service is to provide an abstraction over the actual data prior to its presentation to the end user (analyst).

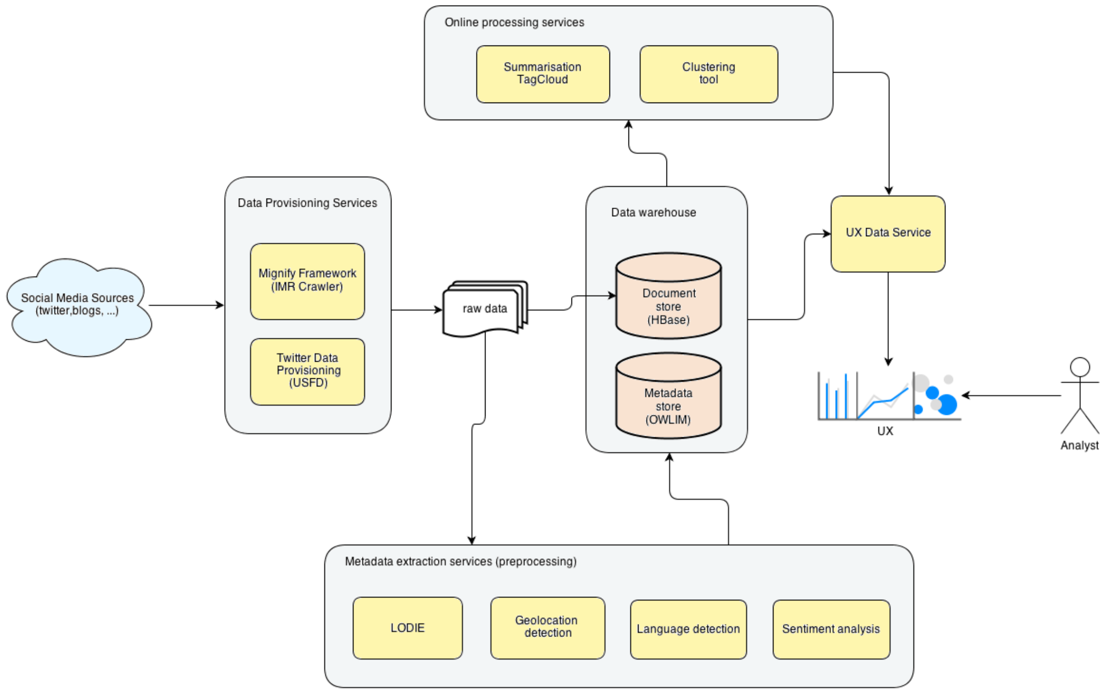
Online processing services
Summarisation TagCloud
Clustering tool
Data Provisioning Services
Mignify Framework (IMR Crawler)
Twitter Data Provisioning (USFD)
Social Media Sources (twitter,blogs, ...)
raw data
Data warehouse
Document store (HBase)
Metadata store (OWLIM)
UX Data Service
UX
Analyst
Metadata extraction services (preprocessing)
LODIE
Geolocation detection
Language detection
Sentiment analysis

Figure 2. Integrated Platform overview

3 Component API specification

3.1 Data Model

In order to manage the complex process of data integration between heterogeneous resources and processing components, a common data model is defined. It covers all essential features required by the system and is flexible enough to accommodate data resources of different origin. The data model is an abstract description and is not dependent on concrete serialization format.

3.1.1 Social Media Resource

The following table presents the key features a single social media resource could have in the TrendMiner platform. This includes the results from the application of various processing tools involved in the metadata extraction process. Instantiations of such enriched resources can be single twitter messages (the example in the table), blog posts or any other resource originating from social media stream.

Property	Description	Example
id	Unique (global) identifier of the resource	*tm:tweet_11050918246*
url	Original source location (URL) of the resource if available	*https://api.twitter.com/1.1/statuses/show.json?id=11050918246*
author	Author name (and URI if it can be resolved)	David MacLean (*http://dbpedia.org/resource/David_Maclean*)
topics	Related keywords	Labour, Budget
sentiment	sentiment/polarity value	2.0
text	text content of the resource	Budget 2010: Labour is stealing from our children's future to buy votes http://bit.ly/aVjtqC #Politics #Telegraph
time	time of issuing of the resource	2010-03-25T22:05:40
source	resource is retrieved from	*twitter.com*
image	image URL it applicable	
mentions	Named entities detected in the text	*http://dbpedia.org/resource/Labour_Party*, *http://dbpedia.org/resource/Budget*
location	geo-location related to the resource (location of origin)	*http://dbpedia.org/resource/London*
hashtag	Hashtags contained in the text	#Politics #Telegraph

3.1.2 Tracks (persistent queries)

These entities are persistent queries which the user creates to monitor certain type of events or entities. The queries represent complex sets of filters and constrains which the user defines and uses across multiple working sessions with the system. The definition and lifecycle management of the tracks are controlled by the users of the system in the UX frontend.

Parameter	Description	Example
id	Track name, used to identify different predefined tracks	"AustrianElections2012"
topics	Keywords relevant for the resources of interest	"Labour", "Budget", "Vienna"
matchAllTopics	match ALL or ANY of the topics	
languages	filter by language	"en", "de"

regions	filter by originating regions	"Austria"(http://dbpedia.org/page/Austria)
from/to	time range	"2010-03-25T22:06:18"/ "2010-03-26T22:06:18"
period	relative time interval until present. Example: past 3 days, last week, etc.. This parameter is exclusive to 'from'/'to' and has a higher priority (if both are present, the pair 'from'/'to' is ignored)	"3h", "1w", "2m", ...
sources	filter by source	"twitter.com"

3.2 Data Provisioning Component

3.2.1 Mignify framework[2]

Mignify is a scalable platform for collecting, storing and analyzing Big Data harvested from the web. It aims at providing an easy access to focused and structured information extracted from Web data flows.

Figure 3**Error! Reference source not found.** outlines the three main objects which represent three successive phases during the data acquisition and annotation process:

- Crawls consist of raw data harvested from some external source (the Web, some OSN).
- Data from crawls can be ingested into a collection and available for further processing by baseline extractors. The original raw data in a crawl will be annotated with various metadata features produced by the baseline extractors (such as language, MIME type, keywords, etc.), which will be utilized during the crawled data search & discovery process.
- Finally, a set of registered queries (called pipes) make it possible for user applications and processes to retrieve the annotated data. The definition of a pipe is created and modified via the *MyMignify* user interface.

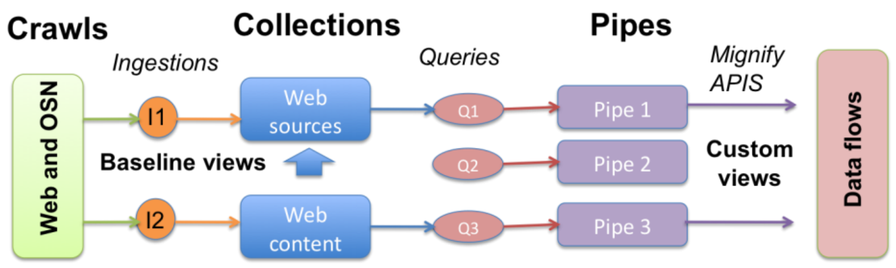

Figure 3 Mignify Framework Overview

Mignify supports two distinct REST APIs for accessing the data being collected: synchronous and asynchronous. The former one is useful for small results. It sends the annotated resources as part of the answer to the HTTP request.

[2] Detailed description available is available in *D5.1.1 Real-time Stream Media Collection - v1*

The asynchronous API is more appropriate where larger amounts of data are expected as result. When the request is sent by the client application, a request id is returned and Mignify registers internally a job to process the request. The job consists in executing the query in a distributed environment where the result is partitioned in small segments. Whenever the result is complete, a notification is sent to the application agent, along with persistent call-back URL from which corresponding files can be retrieved. This API is utilized by the TrendMiner platform and the usage is described in details in the integration sections.

3.2.3 Twitter data provisioning

This is a set of Python scripts that performs some usual data collection tasks from Twitter. This includes continuous crawling of tweets of a list of users or getting all the tweets from a user's timeline. For the purpose of the project, SORA has provided a list of 2000 politically interesting Twitter users which are continuously monitored and the results are delivered to the TrendMiner platform.
Concerning the integration approach with the platform, the component implements a *push* notification mechanism which periodically notifies the rest of the system for any new tweets being collected.

3.3 Pre-processing Services

This collection of tools was created for efficient text processing and analysis of social media text. It works in an online setting or over batches of data. It has pipeline architecture, with the user having control over the modules that he wishes to run, each adding additional information to the input. The tool has 2 variants: a single node tool and a distributed tool that runs in the Hadoop Map-Reduce framework

3.3.1 Tokenization

A Twitter-specific tokenizer based on a chainable set of regular expressions. The tokenizer is capable to handle: URLs, strange usage of punctuation, emoticons, hashtags, retweets, @-mentions, abbreviations and dates.

3.3.2 Language Detection

The module detects language automatically (assume one language/tweet) and doesn't rely on user's self-reported profile language. It is a reimplementation of Lui and Baldwin’s (2011) language detector[3] - fast, standalone, pre-trained, support for 97 languages.

3.3.3 Sentiment Mining

Sentiment is computed on the base of term lists for each language. A single collective value is calculated per document/tweet.

[3] Lui, M., and Baldwin, T. 2011. Cross-domain Feature Selection for Language Identification. In Proc. IJCNLP ’11, 553–561

The pre-processing set of services is accessible via HTTP based service interface, supporting various options for processing customisations:

URL	http://trendminer.ecs.soton.ac.uk/process/input.output
Method	POST
Input parameters	*Input format*: USMF[4], Twitter JSON, Raw text, Ruby T, RDF *Output format*: USMF, Twitter JSON, Raw text, Ruby T, RDF *Mode*: one of TOKENISE, LANG_ID, SENTIMENT Input text
Response	Enriched document description, depending on the mode selected

3.4 Geo-location Detection Service

Geo-location detection based on the location field of the tweet's author (where it is provided). The location field is parsed, the mentioned locations are identified and disambiguated to actual inhabited places, regions and countries. The disambiguation process uses a domain-specific, high precision method, using regular expressions to look up the content of the location field (ignoring region, county and country names and casing) against a gazetteer list of place names and their corresponding URIs, extracted from DBpedia. Where there are several locations with the same name and in the absence of other context, the city with the largest population is selected.
The response from the service is: official location name; DBpedia URI; type of location and geo coordinates. The service supports the following RESTful API:

URL	/resolve_loc
Method	GET
Input	's' - the location description field
Response	Geo location information in JSON format

3.5 LODIE[5] - Named-entities detection service

The goal of this component is to identify named entities in text and attach the correct DBpedia URI to each one of them. For the former, the ANNIE Information Extraction system from GATE it used. It combines some small lists of names (e.g. days of the week, months) and rule-based grammars, to processes text and produce NE types such as *Organization*, *Location* and *Person*. ANNIE also resolves co-reference so that entities with the same meaning are linked. Another component of GATE - Large Knowledge Gazetteer (LKB) performs lookup and assigns URIs to words/phrases in the text thus linking named entities with the corresponding URIs from DBpedia. This process might introduce ambiguities in the URIs assignment so an additional processing step is performed to resolve this problem. The disambiguation algorithm uses context in which the particular entity appears and combined weight measure of string, structural and contextual similarity.
The functionality of this component is accessible via RESTful web service interface:

[4] https://github.com/Tawlk/hyve/wiki/Unified-Social-Media-Format-(USMF)
[5] Details in *D2.2.1 Multilingual, Ontology-Based IE from Stream Media - v1*

URL	http://demos.gate.ac.uk/trendminer/lodie/service/annotate
Method	POST
Content-type	Application/xml
Input	<request> <text>*...text to be analyzed ...*</text> <lang>en</lang> </request>
Response	Original text content with the detected named entities + instance URI and optionally the corresponding ontological class URI. Example: ...<Menton inst="http://dbpedia.org/resource/Iran" class="http://dbpedia.org/ontology/Country">Iran</Mention>

3.6 Clustering Service

Spectral clustering component developed and described in D3.2.1 *Clustering models for discovery of regional and demographic variation - v1* (to be delivered at M24). It builds hard clusters of words that co-occur in the same resource/document.

The clustering component exposes 3 RESTful service endpoints, all returning JSON objects:

GET /cluster?c=cid&t=topw

returns the cluster with *cid* as its label, coherence score and the top *topw* words and their importance for the topic.

GET /words?w=word

returns the id of the cluster where the *word* belongs.

POST /topics?t=notop&coh=c

receives a tweet file, 1 json/line and returns the top *notop* topics in the file and their importance score. *coh* filters to include only topics with at least c coherence rating (default 3).

3.7 Data Warehouse

To meet the requirements for storage and querying huge amounts of data we deployed two different types of scalable big data stores, each responsible for different aspects of the data. For mass storage of textual data we use Apache HBase[6] - distributed data store on top of Hadoop and HDFS. The locality of the data to be stored (plain text + simple meta data properties) do not require any kind of relational database system.

[6] http://hbase.apache.org/

To host the linkage data between resources and entities from the LOD (e.g. DBpedia) we use the OWLIM[7] RDF database capable to handle billions of facts. It's extremely fast and flexible query evaluation capabilities will support the analytical exploration of the resources from the user interface.

3.8 UX Frontend & UX Data Service

The representation layer of the platform (The UX frontend) plays central role in the user interaction with the system. It provides the analytical view over the aggregated content and the added value of the machinery delivered by WP3 and WP4. Described in D4.2.1[8], here we specify its location in the overall system architecture and the interactions with the other subcomponents. While the user interface serves the analyst to make queries and observations on the content, the data service is the bridging component which mediates the communication flow with the backend components, hiding the technical particularity and complexity. The service plays a role of data aggregator and in the same time orchestrates the online processing components invocation based on the current user's request.
The UX frontend is loosely coupled with the data service by communicating entirely through RESTful web service interfaces. All data is collected from the backend via AJAX calls, transferred in JSON format.

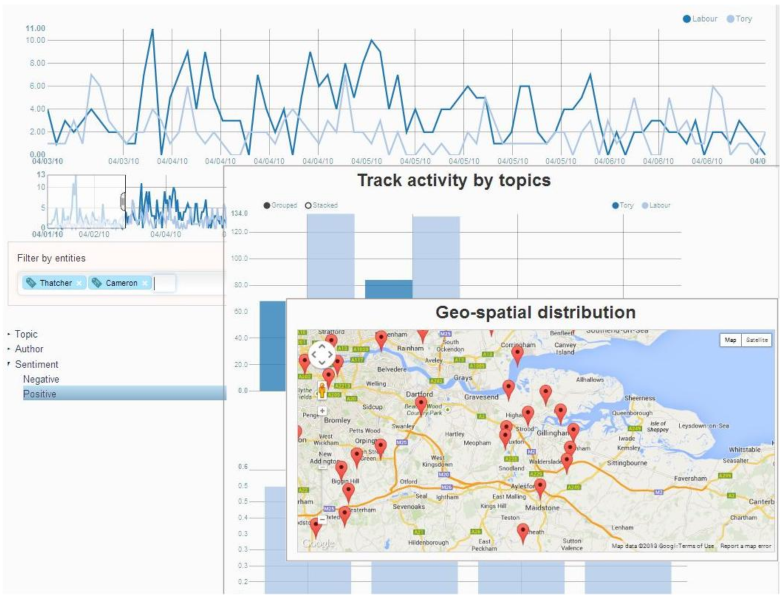

Figure 4. UX Frontend

4 Integration implementation

The following sub-sections detail different integration points of the platform (recall Figure 2).

4.1 Data provisioning integration

[7] http://www.ontotext.com/owlim
[8] *D4.2.1Multi-paradigm search software – v.1; Cross-lingual web UI for trends/sentiments in streaming media*

Figure 5 represents simplified workflow diagram of the integration approach of the two data provisioning services. Though there are minimal differences in the communication protocols they both follow the same pattern. In the two cases the crawling/monitoring jobs are long running processes and the only way to have 'fresh' data in the system is to have regular and asynchronous data delivery. This happens in four major steps:

1. The provisioning services send notifications to the platform when new content is available. In general the content might include huge amount of data so it is partitioned into multiple files and only call-back URLs are sent in the notification message.
2. The platform is responsible to fetch (in asynchronous manner) all the files supplied by the provisioning components.
3. On its arrival in the platform, each new file is scheduled for pre-processing and enrichment with meta-information.
4. Having completed the pre-processing step, the content is automatically loaded into the data warehouse and thus it is immediately visible and searchable from the UX interface.

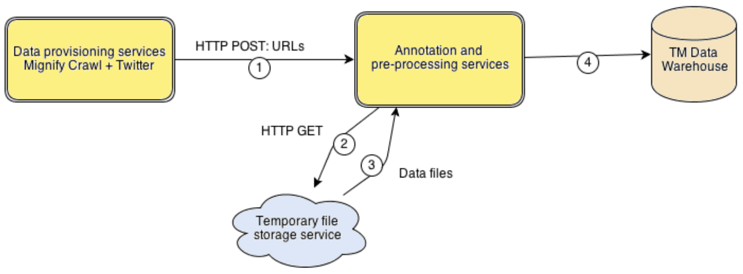

Figure 5. Data provisioning integration

4.2 Pre-processing services integration

For implementing the integration of the various content processing components we needed a powerful solution which provides scalable and flexible means for data integration, data management and application integration. For the first version of the platform we chose to use the Talend Open Studio[9] which covers the requirements we mentioned above. For the sake of brevity we'll skip any detailed description of the framework itself. In essence it's a visual workflow modelling environment which produces highly optimized executable program (in Java). Equipped with rich variety of data processing and application integration components, composing the complex integration process is quite straightforward.

Figure 6 is a representative part of the workflow diagram responsible for the processing of the raw data coming from the provisioning component. The services integration is handled by simple HTTP based client adapters which receive the input data in proper format, send request to the corresponding service and output the response to the following components in the workflow. The terminal nodes in the workflow output the various results in temporary files which are directly loaded in the data warehouse by final post-processing job (not shown on the diagram).

[9] http://www.talend.com/

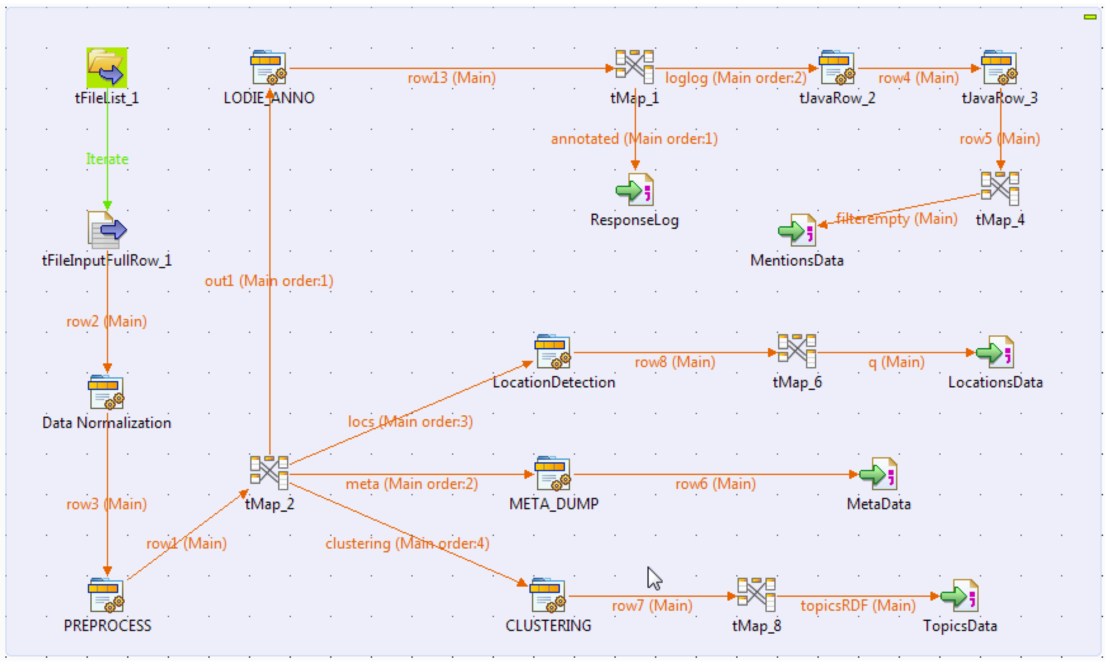

Figure 6. Pre-processing workflow in Talend

5 Deployment

The following table summarizes the deployment prerequisites the separate components require. All of them are portable to the most popular operating systems, however the preferred one is Linux.

Component	Dependencies	Hardware requirements
OWLIM	Java 1.6, Tomcat	64GB RAM (including LODIE requirements)
HBase	Hadoop, HDFS	4 GB RAM
LODIE	GATE[10], OWLIM, Tomcat	10 GB RAM
Twitter data provisioner	Python	<2 GB RAM
Pre-processing service	Python, CherryPy[11]	2 GB RAM
Geo-location detection	Python,CherryPy	<2 GB RAM
Clustering service	Python, CherryPy, NumPy[12]	<2 GB RAM
Integration backbone	Java 1.6, Tomcat	8GB RAM

By the time of delivering this document, a live system is deployed on a cluster of machines at the Amazon Elastic Compute Cloud[13].

[10] http://gate.ac.uk/
[11] http://www.cherrypy.org/
[12] http://www.numpy.org/
[13] http://aws.amazon.com/ec2/

6 Applicability of massively parallel processors for accelerating workloads within the TrendMiner system

6.1 Motivation

The data processing flow within the TrendMiner system happens in two distinct phases. First, the documents are retrieved, annotated and all relevant information is extracted entirely in the backend. Then, upon a user request, a subset of this data is loaded according to some user-specified filtering criteria and aggregated in a form which is comprehensible for the analyst.

This separation implies different requirements for the two workflows. Whereas the pre-processing of the data can happen offline and in batch mode, it is essential to design filtering and aggregation services which are as responsive as possible.

In order to achieve this goal, we add the step of a preliminary data set reduction where the analyst can configure what subset of data he/she is willing to investigate. This configuration is persistent and can be reloaded at a later moment. Within the TrendMiner terminology this subset of all documents available within the system is called a *track (*refer 3.1.2 Tracks (persistent queries)*).*

However, even the reduced dataset can easily grow too large to be filtered and aggregated in real time. Because of this, we research the applicability of GPU accelerated algorithms in order to make this step more scalable.

6.2 GPU Architecture

This section describes the high level architectures of modern massively parallel processors.

Discrete GPU architecture

A typical discrete GPU device is constituted by a number of Processing Units (PUs) or cores each of which has access to some amount of fast memory (*Shared memory*).
Each of these cores has one or more vector processors which execute the same instruction in lock-step over a vector of values (i.e. SIMD – *single instruction multiple data* approach).

In the case of NVIDIA’s high-end GTX580 GPU the number of cores is 16 each of which has 2 SIMD array processors, each of which is 16 – lane wide. This makes up for a total of 512 computational units.

The different PUs have access to a global device memory space. The access speed to that memory is considerably slower than the access to the local memory. Memory can be copied from system memory to the global device memory when required. This happens through the PCI Express bus at even slower speed.

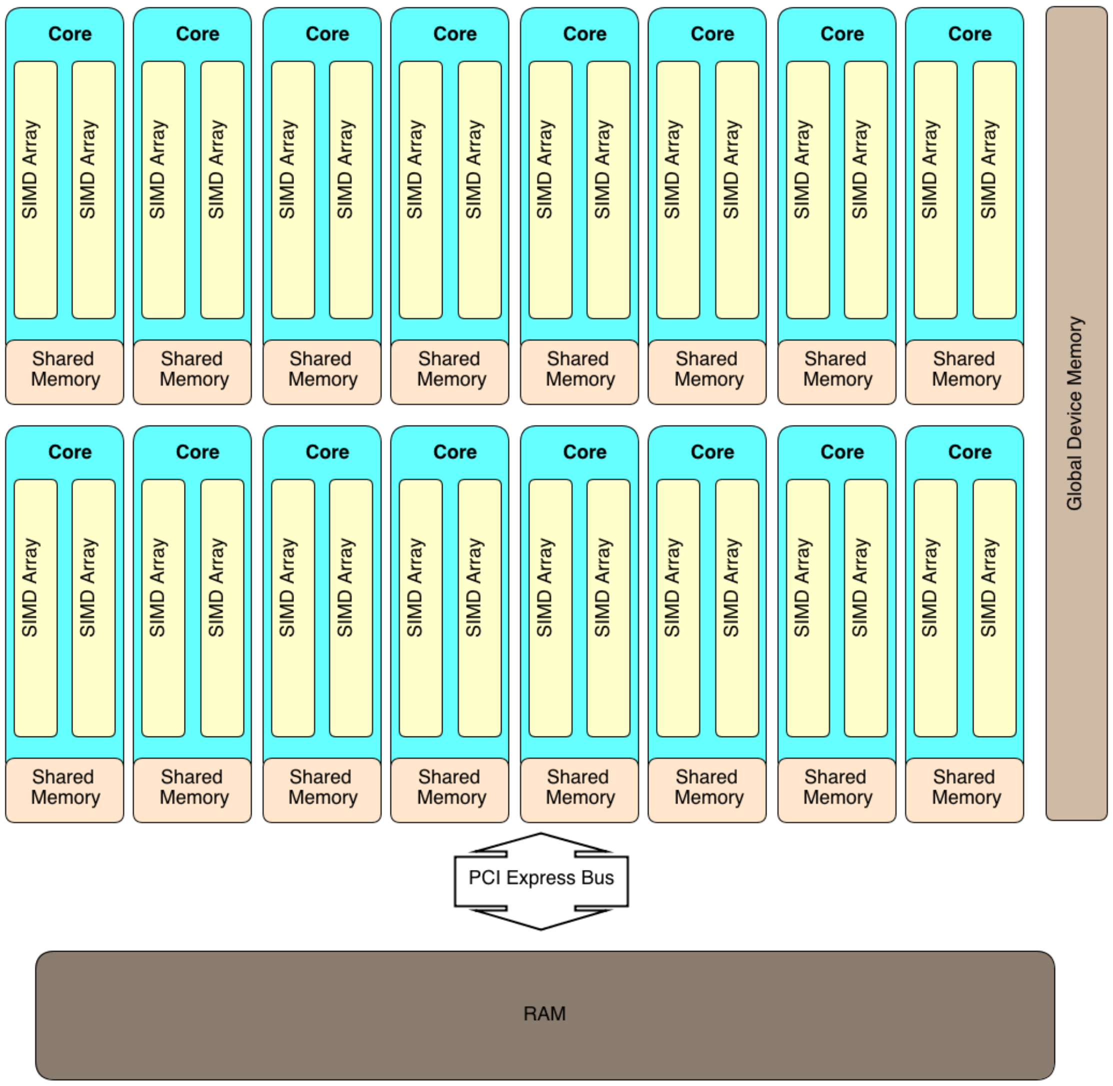

Figure 7. NVIDIA GTX580 High level architecture

APU architecture

Some of the recent processor architectures include an integrated GPU module on the same chip as the CPU. Such chips are referred to as *Accelerated Processing Units* (APUs). The processing unit hierarchy is similar to that of the discrete GPU.

However, no separate global device memory exists on the chip but rather the GPU uses part of the main memory as device memory. This has the consequence of not having to copy data between main memory and device memory before and after execution. Another implication is that we have to deal with the larger latency of the main memory compared to a dedicated global device memory. To mitigate this to some extent, some APUs have shared Last Level Cache (LLC) between the CPU cores and the GPU.

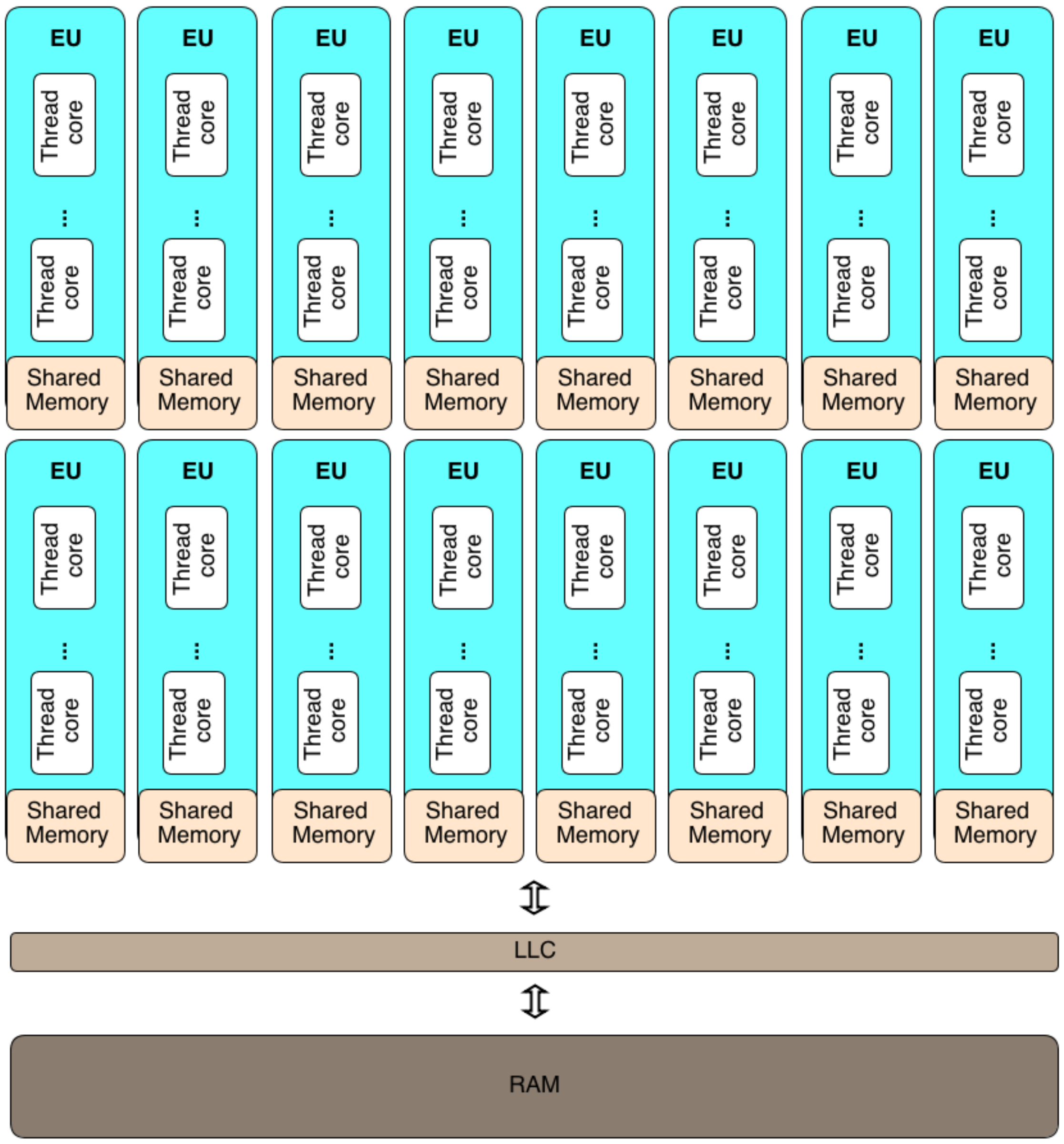

Figure 8. Intel HD Graphics 4000 high level architecture. The APU unit has 16 EUs each of which is 8-lane wide SIMD processor. The total number of computational cores is 128

6.3 GPU Programmability

As seen from the previous section, the GPU's architecture is significantly different from that of a CPU. Furthermore, GPU's instruction set is much more restricted than that of a CPU.

GPUs were originally developed in order to accelerate the highly data-parallel problems found in computer graphics. There have been attempts to map general computation problems on the graphics pipeline implemented in the GPU architectures.

In the recent years a number of technologies have appeared which provide the opportunity for easier development of GPU algorithms for tasks not related to computer graphics. Such technologies are collectively referred as *General Purpose GPU* (GPGPU). The first such technology was CUDA developed by GPU manufacturer NVIDIA only for their products. Later, an open standard – OpenCL was developed by the Khronos Group consortium. This standard defines a unified API for programming various types of processors – CPUs, GPUs and APUs from various manufacturers.

We chose to use OpenCL for our experiment because of its cross-platform capabilities.

OpenCL programming concepts

The core concept of an OpenCL program is a *kernel*, which is a piece of machine code which executes multiple times on the device. An instance of a kernel is called a *work item.* OpenCL defines the concept of global *work space* which is a 1, 2 or 3 dimensional grid of all work items. Each work item has a mechanism to detect its position within the work space and to perform a different task depending on its position. The work space is divided in *work groups.* The OpenCL standard only defines synchronisation mechanisms within a work group. No synchronisation between work groups is possible. This relaxed synchronisation constraints allow the execution to be mapped efficiently to various architectures at the price of increased programming complexity.

OpenCL standard does not define how the work items will map on a device. On a GPU, many work items may be executed in the same moment, whereas on a CPU the same work space may be executed sequentially.

Aparapi

The integration of GPU-accelerated algorithms within the TrendMiner system is further simplified by the usage of the Aparapi API initially developed by chip manufacturer AMD. Aparapi is a Java based API which allows fragments of Java bytecode to be converted to OpenCL code at runtime thus allowing the developer to utilise GPUs by writing only in the Java language.

Algorithms suitable for GPU acceleration

Despite the modern GPU’s enormous processing power, not all kinds of algorithms can be accelerated on a GPU. In order for an algorithm to take advantage of a massively parallel processor, generally speaking, it has to do more of the type of workloads a GPU is good at and less of the type of workloads a GPU is not good at. Examples of workloads which will not be accelerated on a GPU are:

- Algorithm which is data and/or I/O bound. If an algorithm requires a lot of data transfer between the host memory and the device memory, it can not be accelerated on a GPU since data transfers are very expensive.
- Intrinsically sequential algorithms. If an algorithm cannot be parallelised, it will be slower on a GPU, since a single processing unit in a GPU is much slower than a CPU.
- Very simple algorithms. An algorithm has to be of a certain complexity in order to outweigh the housekeeping overhead of enqueuing kernels and data on the GPU.

Key properties which make an algorithm suitable candidate for GPU acceleration are:

- Compute-bound. The only thing a GPU is good at is performing a lot of calculations.
- Highly data-parallel. GPUs are optimised to perform the same operations on large amounts of data.
- Large amounts of work. It is important to have a lot of work items waiting to be executed as they will hide any latency caused by data access.
- Limited need to access external data
- Data locality. GPUs provide mechanisms for caching frequently used data. Thus if an algorithm accesses a limited amount of data multiple times, it can reduce the data access latencies.

6.4 Accelerating TrendMiner workloads

The filtering and aggregation stage of the TrendMiner platform consists of the following steps:

0. Prerequisites: A list of documents matching the track definition together with the information extracted is retrieved from the data layer. This step is not necessarily in real time as it happens not more than once in the beginning of the user session.
1. Filtering: for each document it is checked if it matches the filtering criteria configured by the user. Only the matching documents are considered in the next step. The filters are:
   - 1.1. Topic filter: does the document match a certain topic
   - 1.2. Mentions filter: does the document mention a certain entity
   - 1.3. Time filter: is the document within a specified time period
   - 1.4. Authors filter: is the document created by a certain author
   - 1.5. Sentiment filter: is the document’s sentiment index positive or negative
2. Aggregation: the filtered document are aggregated in several ways:
   - 2.1. Entity co-occurrence: calculates a co-occurrence matrix for the entities mentioned in the documents
   - 2.2. Topic mentions by sub-periods. Calculates the total count for each topic for each sub-period.
   - 2.3. Sentiment by sub-periods. Calculates the average sentiment for each sub-period.

It is more important to optimise the filtering step of the service as it is a reduction step and as such operates on the entire document set. This step can be formulated as a stream compaction operation: *Given an array of documents, output only those matching certain criteria.*

Input array:

A	B	C	D	E
n	y	n	y	y

Output array:

B	D	E

We can reformulate this as follows: *Given an array of documents, calculate the indices for the documents matching the filters in an output array.*

Input array:

A	B	C	D	E
n	y	n	y	y

Output array:

A	B	C	D	E
	0		1	2

This, however, can be reformulated as a well-studied problem – *prefix sum* or *scan* problem: *Given an array of documents, for each document matching the filters calculate the number of documents before it which match the filters.*

A parallel algorithm for this problem is presented in (Blelloch, 1990).
We experimented with the algorithm using Aparapi OpenCL wrapper. The comparison between the parallel GPU accelerated algorithm and a linear scan algorithm are shown in the following diagram.

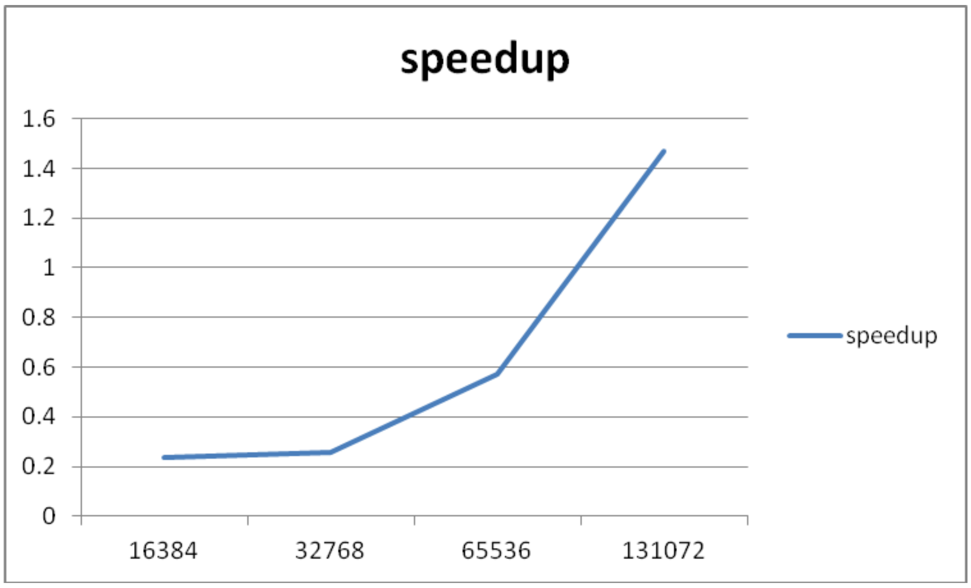

Figure 9. Parallel Scan acceleration

As shown some level of speedup is achieved with datasets of realistic sizes so we are encouraged to investigate further the integration of the algorithm in the TrendMiner platform.

Conclusions and Future Work

In this document we have presented the first implementation of the TrendMiner integrated platform including all components available by M24 of the project. The whole platform has been deployed on the Amazon EC2 Cloud and it is accessible for all consortium members.
The next version of the platform will include:

- more processing components to be produced by M36;
- an improved version of the UX frontend based on feedback collected;
- more social media sources to be monitored.

Bibliography and references

Aparapi. (n.d.). Retrieved from https://code.google.com/p/aparapi/

Blelloch, G. E. (1990). Prefix Sums and Their Application.

CUDA. (n.d.). Retrieved from http://www.nvidia.com/object/cuda_home_new.html

Intel® SDK for OpenCL Applications 2013 - Optimization Guide for Windows* OS*. (n.d.). Retrieved from http://software.intel.com/sites/products/documentation/ioclsdk/2013/OG/index.htm

NVIDIA GTX580 Architecture. (n.d.). Retrieved from http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580/architecture

OpenCL. (n.d.). Retrieved from http://www.khronos.org/opencl/