FP7 Project ASAP
Adaptable Scalable Analytics Platform

ASAP D8.2
Use Case Requirements

WP 8 – Applications: Web Content Analytics

Nature: Report

Dissemination: Public

Version History

Version	Date	Author	Comments
0.1	January 16, 2015	Philippe Rigaux	Initial Version
0.2	January 24, 2015	Tsuyoshi Sugibuchi	Revised Version
0.3	February 4, 2015	Philippe Rigaux	First complete draft
0.4	February 16, 2015	Philippe Rigaux	Revised draft
0.5	February 27, 2015	Philippe Rigaux	Final revision based on internal reviewing

Acknowledgement This project has received funding from the European Union's 7th Framework Programme for research, technological development and demonstration under grant agreement number 619706.

Contents

List of Figures

Abstract

This deliverable presents a detailed list of the Web Data Use case requirements. It provides a focused description of the MIGNIFY platform proposed by Internet Memory Research, with emphasis on a set of services called *pipes* that allow to specify and execute workflows of web data processing operators. Based on this description, the deliverable develops three types of requirements: monitoring and performance evaluation, multi pipes execution and continuous pipe execution. A testbed has been set up by IMR with the necessary software components to evaluate the capacity of the ASAP platform to support these requirements.

1 Introduction

The present deliverable covers the requirements of the Web content use case. It is centered on the set of services proposed by Internet Memory Research as part of the MIGNIFY platform (`http://mignify.com`). They provide access to a large collection of contents extracted from the Web, cleaned, annotated and indexed in a distributed infrastructure based on Hadoop components: HDFS (storage), HBase (primary indexing on URLs), MapReduce (Hadoop 1) and ElasticSearch (secondary indexing). The infrastructure and ingestion workflow have been described in Deliverable 8.1, along with the main characteristics of the Web dataset.

We expose a list of requirements all focused on extending and enriching the public workflow interface supplied by MIGNIFY, called *pipes*. MIGNIFY relies on Hadoop V1 to run MapReduce jobs, and this severely limits the workflows that can be expressed by pipes. In particular, iterative workflows, typical of data mining algorithms, cannot be obtained with MapReduce without a heavy programming effort. During the project we aim at exploring the capacities of recent data processing engines, namely Flink (`http://flink.apache.org`) and Spark (`http://spark.apache.org`) to extend pipes with iteration and fixpoint operators.

Based on this setting, the requirements can be divided in three main groups which can be organized to fit in the ASAP project roadmap as follows:

- **(R1) Requirement 1: monitoring the cost of pipes execution.**

  MIGNIFY proposes a public interface that lets customers specify and execute pipes on the IMR Web collections. All pipes have to run concurrently in a single distributed infrastructure. It is essential to obtain a reliable estimation on the time frame of a pipe execution, in order to report to our customer the expected delay to obtain the result.

- **(R2) Requirement 2: resource sharing and optimization for multi-pipes execution.**

  Internet Memory Research uses its own infrastructure to store data and execute pipes. Many distributed softwares share the resources of this infrastructure, and even if we decide to allocate a dedicated cluster to pipe execution, the problem of sharing the resources between multiple pipes running concurrently remain. Based on the monitoring results (R1), we need to come up with a scheduling module apt at allocating resources for pipes execution based on the services constraints.

- **(R3) Requirement 3: continuous pipe execution.**

  In the current setting, a customer explicitly requires the execution of a pipe. In many scenarios (e.g.; extraction of indicators from social sources) the pipe should run almost continuously on incoming new contents. We therefore need to study how this continuous subscription mechanism can be implemented in the context of a large set of concurrent workflows execution.

These requirements can/should be examined in sequence and addressed one after the other throughout the ASAP project development. Requirements R2 and R3 are somewhat independent from one another, but both depend on R1.

The organization of the Deliverable is as follows. Section 2 first develops the concept of pipe in MIGNIFY, and introduces the current user interface. Section 3 exposes the principles of pipe extension that will be experimented during the project with modern distributed processing engines. Suggested solutions based on either Flink or Spark are briefly introduced. Section 4 is devoted to the requirements of our use case in the context of ASAP. Finally, Section 5 describes the test platform which has been set up for ASAP experiments and cooperation with our partners.

2 Pipes in MIGNIFY

Figure 1 shows the current architecture of MIGNIFY, with emphasis on pipes definition and processing. Data is collected from the Web by a crawler, MemoryBot, designed to operate at large scale, and stored in HBase, a persistent distributed data structure with indexing capabilities, apt at managing very large datasets (TeraBytes or even PetaBytes).

2.1 Storing and indexing data

The ingestion process gets the raw documents supplied by the crawler, annotate documents with extracted features, and insert them in HBase. Feature extraction operations are applied either to a single document or to group of documents. Features (e.g., language of the document, MIME type, keywords) are stored as meta-data in HBase along with the document itself.

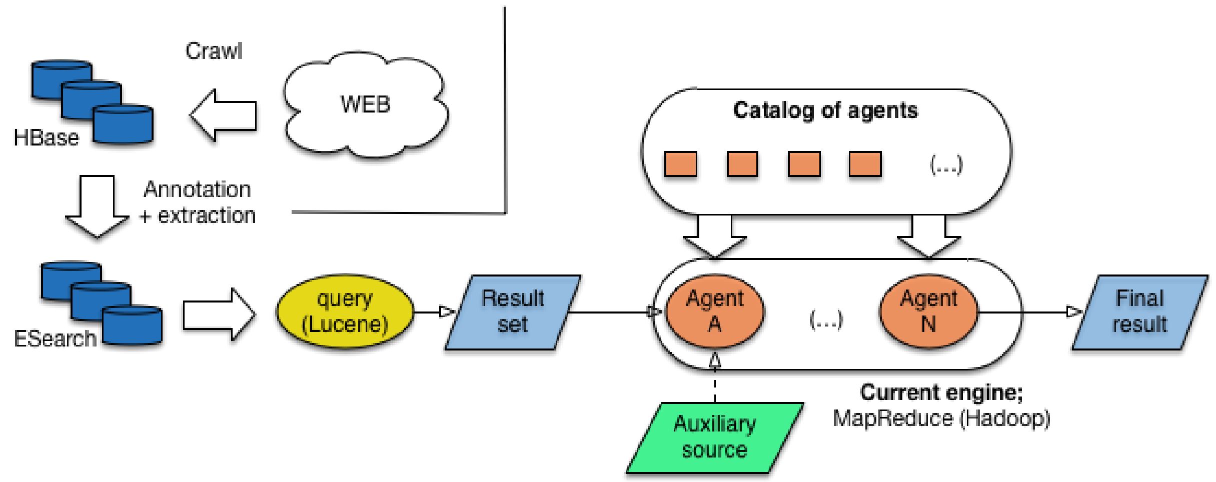


Figure 1: Current architecture of MIGNIFY (with focus on pipes)

We use those features to build a secondary index on the collection with ElasticSearch[1]. Until a recent version, the ElasticSearch index was used to obtain the list of documents ids, and an additional access-by-id request was necessary to retrieve the document from HBase. For simplicity and efficiency, we now also store in ElasticSearch the document itself. This means that all the information necessary to index and process web content can now be obtained from ElasticSearch without involving the other back-end components (HDFS, HBase) anymore. As a side effect, this makes the architecture much simpler and understandable for our partners.

Thanks to ElasticSearch (which is essentially a distributed version of Lucene), resources can be accessed via their unique identifier (combination of a URL and a timestamp), or through complex queries expressing criteria over the features produced during the ingestion process. It is possible for instance to query a collection by specifying a combination of language code, MIME type and time range ("Give me all PDF documents in French collected during *this* period").

2.2 Agents, queries, and pipes

The main interface proposed by MIGNIFY to access our Web collections is called *pipe*. A pipe (Figure 1) consists of :

1. a *query* executed over the ElasticSearch index;
2. a workflow of so-called *agents* which take the query result set as input, and progressively transform/annotate the elements of the result set.

[1]http://www.elasticsearch.org

Agents are, together with the ability to access a lage repository of cleaned Web content, the main assets of the platform. They implement text mining operators such as:

- keyword extraction (most significant terms of the content);
- entity recognition (locations, events, names, etc.);
- entity disambiguation (link to an ontology such as Yago or DBpedia);
- sentiment analysis.

Many of these agents are implemented by encapsulating text mining functions from some well-know open source library, e.g., Gate[2] or LingPipe[3]. An agent may also require some auxiliary source (Figure 1) such as, for instance, an in-memory representation of an ontology for entity disambiguation.

Pipes can be defined thanks to the MIGNIFY Web interface which is illustrated by Figures 2 and 3. Figure 2 shows the state of the UI after the definition of a query on the *Forum* collection. It contains user generated contents published in the hundreds of thousands of forums crawled by MIGNIFY. Several parameters can be set from the UI, including the maximal number of documents taken from the full ranked list supplied by ElasticSearch, as well as the features that must be incorporated in the result set (content, but also user, date, etc.)

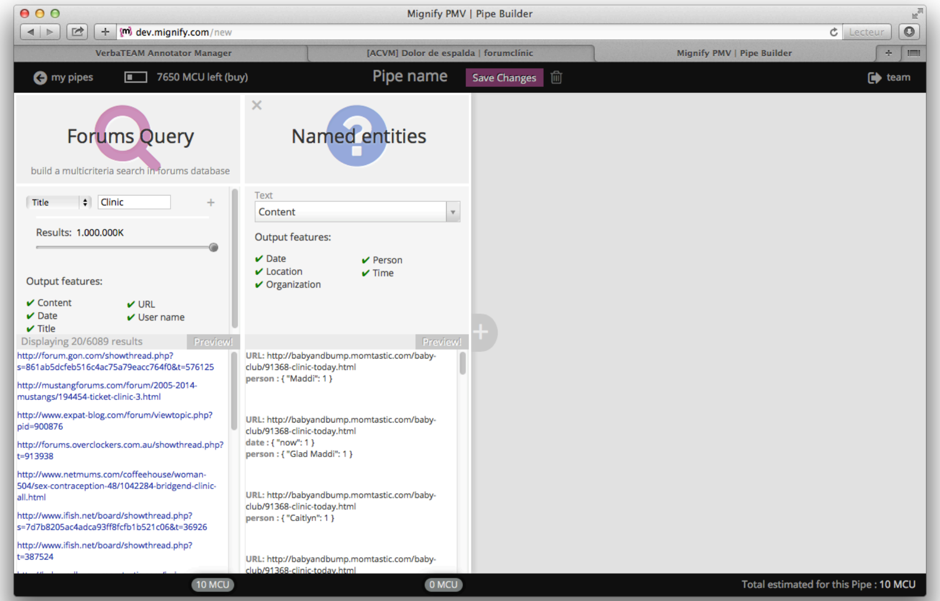
The pipe also feature an initial agent which extracts named entities from the contents of the result set. The workflow can progressively be extended by adding more agents. Figure 3 shows the final required step when building a pipe: specifying the output channel. As a general principle, the pipe is saved with a name and a unique pipe URI is produced. The pipe execution produces a set of files (say, in JSON or XML), and the customer can access to these files via REST services associated to the pipe URI.

2.3 Pipe execution

The user can request the execution of a pipe. A request id is returned and MIGNIFY, in the current version, initiates a MapReduce job which is run asynchronously. The job consists of executing the query as a distributed process running on ElasticSearch shards as input, followed by the application of the workflow of agents to the documents extracted from the shards. This distributed execution produces the pipe result (a large set of annotated resources) which is then packaged in container files. Whenever the result is complete, a notification is sent to the user, along with the URL from which container files can be retrieved.

[2]`http://gate.ac.uk`

[3]`http://ir.exp.sis.pitt.edu/ne/lingpipe-2.4.0/`


Mignify PMV | Pipe Builder
dev.mignify.com/new
VerbaTEAM Annotator Manager
[ACVM] Dolor de espalda | forumclinic
Mignify PMV | Pipe Builder
my pipes
7650 MCU left (buy)
Pipe name
Save Changes
team
Forums Query
build a multicriteria search in forums database
Named entities
Preview!
Preview!
10 MCU
0 MCU

Title: Clinic

Results: 1.000.000K

Output features:
✓ Content ✓ URL
✓ Date ✓ User name
✓ Title

Displaying 20/6089 results

```
http://forum.gon.com/showthread.php?s=861ab5dcfeb516c4ac75a79eacc764f0&t=576125
http://mustangforums.com/forum/2005-2014-mustangs/194454-ticket-clinic-3.html
http://www.expat-blog.com/forum/viewtopic.php?pid=900876
http://forums.overclockers.com.au/showthread.php?t=913938
http://www.netmums.com/coffeehouse/woman-504/sex-contraception-48/1042284-bridgend-clinic-all.html
http://www.ifish.net/board/showthread.php?s=7d7b8205ac4adca93ff8fcfb1b521c06&t=36926
http://www.ifish.net/board/showthread.php?t=387524
```

Text: Content

Output features:
✓ Date ✓ Person
✓ Location ✓ Time
✓ Organization

```
URL: http://babyandbump.momtastic.com/baby-club/91368-clinic-today.html
person : { "Maddi": 1 }

URL: http://babyandbump.momtastic.com/baby-club/91368-clinic-today.html
date : { "now": 1 }
person : { "Glad Maddi": 1 }

URL: http://babyandbump.momtastic.com/baby-club/91368-clinic-today.html
person : { "Caitlyn": 1 }

URL: http://babyandbump.momtastic.com/baby-club/91368-clinic-today.html
```

Total estimated for this Pipe : 10 MCU

Figure 2: Public interface of MIGNIFY (1)

3 Extending the expressivity of pipes

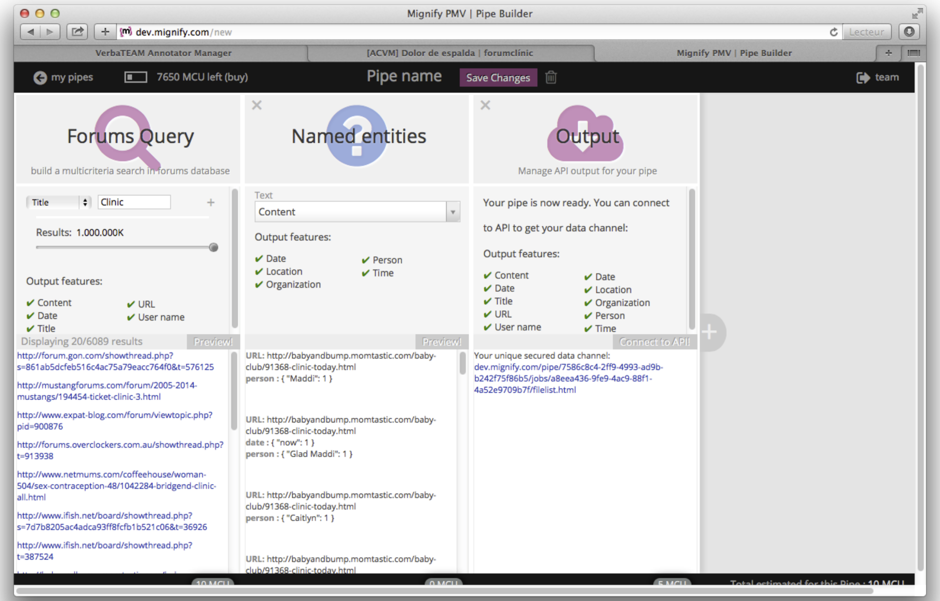
A pipe workflow appears to be quite limited in the set of operations that it can express. Basically, only direct acyclic graphs of data flows can be represented. This motivates efforts to extend the operators to a richer set, including in particular *loops* to allow the iterative construction of some solution. We examine in this section how this can be achieved with new processing engines that go far beyond the mere Hadoop's V1 MapReduce operator.

3.1 Iterative workflows

If we compare with the operators from Pig latin [5] [4] for instance (Table 1), workflows in MIGNIFY are built with the subset that excludes binary operators (i.e., `join`, `cross`, and `cogroup`). This makes sense if we view pipes in MIGNIFY as specialized workflows for applying text mining operators, but also shows that we cannot capture some very useful operators. It turns out that, in particular, *iteration* is the key for covering many data mining tasks that would be quite relevant and valuable.

One of the expectations of the MIGNIFY customers is indeed to be able to build an interpretation model for the result set of a query, and then to apply on-the-fly the model to the very same result

[4] `http://pig.apache.org`



http://forum.gon.com/showthread.php?s=861ab5dcfeb516c4ac75a79eacc764f0&t=576125

http://mustangforums.com/forum/2005-2014-mustangs/194454-ticket-clinic-3.html

http://www.expat-blog.com/forum/viewtopic.php?pid=900876

http://forums.overclockers.com.au/showthread.php?t=913938

http://www.netmums.com/coffeehouse/woman-504/sex-contraception-48/1042284-bridgend-clinic-all.html

http://www.ifish.net/board/showthread.php?s=7d7b8205ac4adca93ff8fcfb1b521c06&t=36926

http://www.ifish.net/board/showthread.php?t=387524

```
URL: http://babyandbump.momtastic.com/baby-club/91368-clinic-today.html
person : { "Maddi": 1 }

URL: http://babyandbump.momtastic.com/baby-club/91368-clinic-today.html
date : { "now": 1 }
person : { "Glad Maddi": 1 }

URL: http://babyandbump.momtastic.com/baby-club/91368-clinic-today.html
person : { "Caitlyn": 1 }

URL: http://babyandbump.momtastic.com/baby-club/91368-clinic-today.html
```

Your pipe is now ready. You can connect to API to get your data channel:

Your unique secured data channel:
dev.mignify.com/pipe/7586c8c4-2ff9-4993-ad9b-b242f75f86b5/jobs/a8eea436-9fe9-4ac9-88f1-4a52e9709b7f/filelist.html

Figure 3: Public interface of MIGNIFY (2)

set.

This is illustrated by Figure 4. It shows a pipe that retrieves a result set, applies some document-level agents (Agent A on the Figure), and reaches a point where it becomes possible to mine the annotated result set to learn some semantic information (say, the main topics in the result set, or the identification of some graph structure, etc.). This requires the execution of a machine learning agent (ML op on the Figure) which derives an interpretation model from the result set, and stores this model as an auxiliary structure. The pipe can then proceed by interpreting the documents of the result set with respect to the model. The whole machine learning process (model creation, and model application) is integrated in a single workflow.

Such a pipe would be much more powerful than the current simple linear annotation workflows currently handled by MIGNIFY. It would produce quite valuable derived information, enabling MIGNIFY customers to run on-line machine learning on Web datasets, with large-scale execution as a service

It is important to point out that building the model on-the-fly on the result set, and not as a pre-processing step on a whole, stored, collection, is a key to an accurate interpretation of the data mining result. Consider for instance a pipe that classifies forums content based on keywords extraction. Identifying the relevant keywords implies the construction of a language model or topic model. This cannot be done on a general collection of world-wide forums which mix contents in all languages and related to all kinds of topics, whereas the pipe focuses on a forum subset defined

Operator	**Description**
foreach	Apply one or several expression(s) to each of the input tuples.
filter	Filter the input tuples with some criteria.
order	Order an input.
distinct	Remove duplicates from an input.
cogroup	Associate two related groups from distinct inputs.
cross	Cross product of two inputs.
join	Join of two inputs.

Table 1: List of Pig latin operators (excerpt)

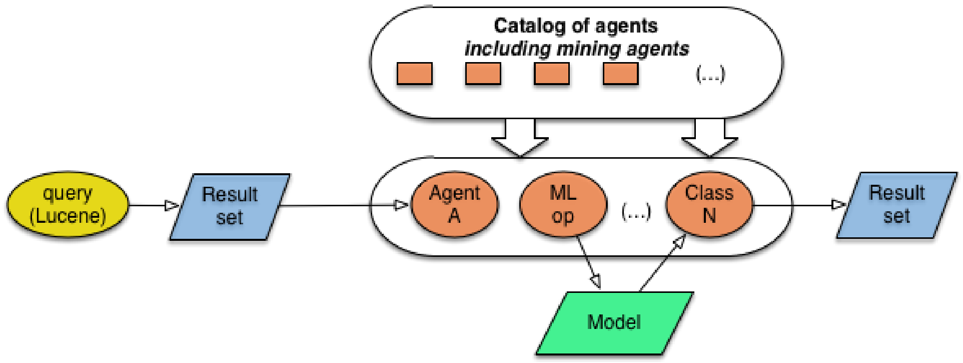


Figure 4: Adding machine learning capabilities to pipes

by the initial query, which restricts for instance the language to French, and only gets content from medicine-related sites. The integrated evaluation therefore leverages the value of the machine-learning process by focusing on the subset of interest to the customer.

3.2 Example: running a k-means clustering on result sets

We propose to design and evaluate the pipe extension with the concrete case of k-means clustering based on salient keywords identified on-the-fly during the pipe execution. The workflow is depicted in Figure 5 at a generic level. We will later on propose possible implementations with two modern processing engines.

The pipe starts, as usual, with a result set extracted from the Web collections with an Elastic-Search query. One of the MIGNIFY agents extracts the term frequency (tf) from documents and produces an annotated data set RS1. From RS1, another agent creates an auxiliary structure containing the inverse document frequency (idf). We are then ready to build the clustering model by applying a k-means algorithm. It takes as input RS1, the idf table, and iteratively adjusts the set of centers. This requires to repeatedly scan RS1, as well as a fast access to the idf table. Once the clusters centers are determined, a final ML agent (the classifier) produces clusters of the documents in RS1 and outputs the final result.

Some comments are noteworthy.

1. Such a process, featuring repeated scans of an intermediate result, is definitely not supported by a processing engine based on the MapReduce operator. It would be both cumbersome to implement, and quite slow to execute.

2. The k-means example is just an instance of a more generic class of problems. We can mention for instance *topic modeling* [2]: one of the MIGNIFY agents is an implementation of the Latent Dirichlet Allocation [3] which produces, from a set of documents, a list of ”topics” represented by a term distribution. Classifying the documents by topics with a pipe complies to the same process as the one illustrated above for k-means clustering

Implementation of iterations with MapReduce is a burden for the developer who must take care of properly storing the result, and design a mechanism that takes as input, at each iteration, the result produced during the previous phase.

Moreover, MapReduce relies on a checkpoint-based fault tolerance. This implies that all intermediate steps are materialized on disks. In this case (and in the case of iterative algorithms in general), each iteration results in a write/read phase of the result, plus repeated disk reads of the whole result set.

We aim at exploring the ability of modern distributed processing engines to support the execution of such extended pipes. Candidates of choice are Flink (`http://flink.apache.org`, formerly Stratosphere [1]) and Spark [8] (`http://spark.apache.org`), two top-level Apache projects.

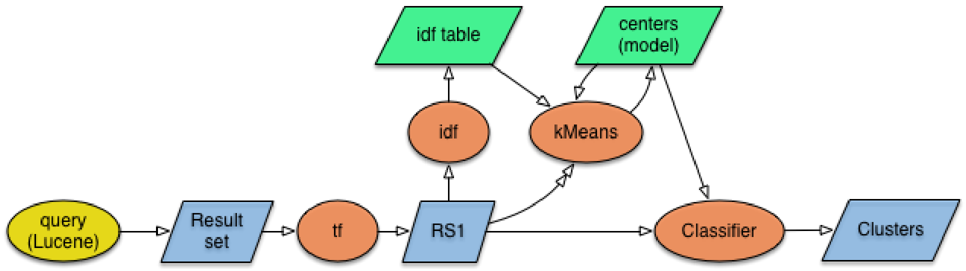


Figure 5: A concrete example: a k-Means workflow

3.3 Distributed processing engines: Flink / Spark

Both Flink and Spark support a set of high-level operators similar to those of Pig latin (see Table 1). Most of these operators are second-order function that apply some User Defined Function (UDF) to a data flow in a distributed environment. In our context, UDF are the MIGNIFY agents which run text mining, document-level operations on textual content. A nice additional feature of Flink is that it features two operators that greatly simplify the specification of iterative workflows [5].

1. The ITERATE operator runs a loop over an input dataset, maintaining a partial solution at each step.
2. The DELTA ITERATE operator covers the case of incremental iterations which progressively refine a solution. The situation is typical in Machine Learning algorithms.

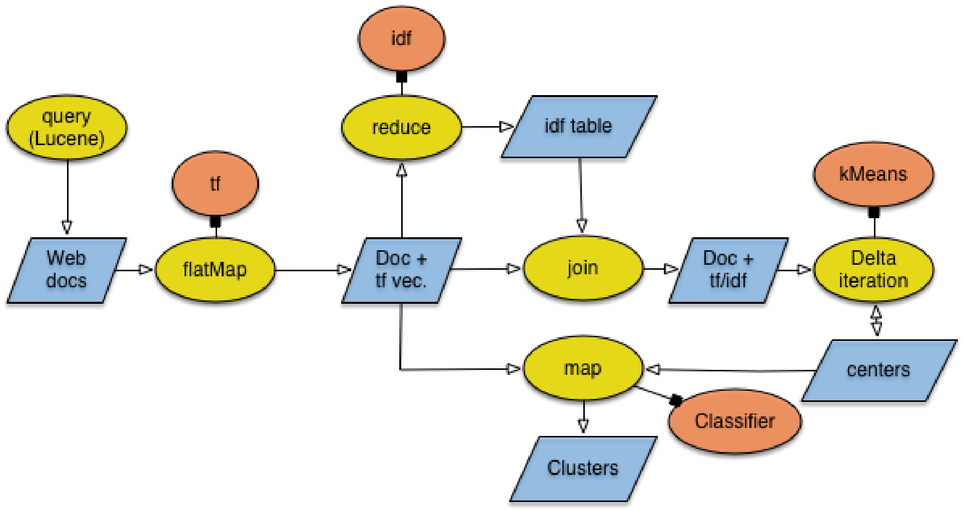


Figure 6: Running the k-means example with Flink

As a result, the k-means pipe execution with Flink is illustrated by Figure 6. Yellow ovals correspond to Flink operators, and brown ovals to MIGNIFY agents. Intermediate result sets are shown as blue rectangles. The salient part is the k-means computation: it can simply be implemented with an application of the DELTA ITERATE operator to the tf/idf annotated documents, and produces the solution (a list of cluster centers) used to classify the documents.

Applying the classifier is just a matter of scanning a second time the web documents, with a MAP operator that runs the classifier function for each document.

[5] See http://flink.apache.org/docs/0.6-incubating/iterations.html for details.

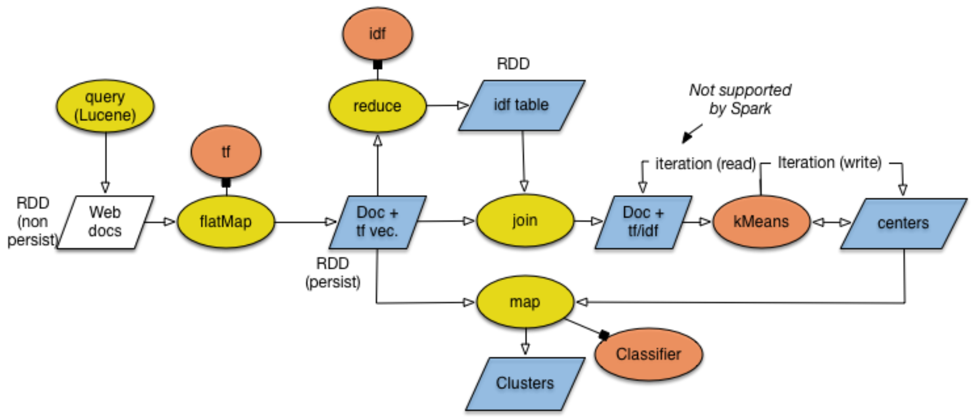


Figure 7: Running the k-means example with Spark

If we now turn to Spark (Figure 7), a quite similar execution workflow can be defined, with a few notable differences. Spark is based on the concept of *Resilient Data Set* (RDD), which are essentially intermediate results pinned in memory, with a fault tolerance method based on partial re-computation. RDD can be made *persistent* (in memory as much as possible), or *transient*. Figure 7 shows our k-means workflow with one transient RDD (the initial query result), all other RDD being persistent, at least during a part of the workflow execution timeframe (the idf table for instance can be flushed once the k-means operators is finished.)

Spark does not natively support iteration. This means that the burden of implementing a fix-point computation for machine learning algorithms remains. In our context, this is a limitation, although we can expect that the current community activity around Spark will eventually result in additional functionalities that would match those of Flink in this respect.

This means that we have to manually encode the iteration step for the k-means solution, and may result in a sub-optimal implementation for the management of intermediate results, as suggested with some preliminary studies conducted with the TUB team that leads the development of Flink. Ease of implementation, and thus cost of maintenance, is an important criteria to assess the industrial strength of a solution. Conducting a comparison of those engines as part of ASAP would probably prove to be an interesting study, fully in line with the project's objectives.

4 Use Case Requirements

We now detail the requirements of the use case. We first expose functional requirements and then devote a short sub-section to non-functional requirements (scalability and usability aspects).

4.1 Pipes monitoring

At the moment we can hardly estimate the resources needed by a pipe, nor can we anticipate the time it will take until completion. This is a major concern because this greatly impacts our ability to faithfully obtain the cost of a pipe execution, in order to expose this cost to our customers. The problem involves several aspects which must all be considered together.

- **Cost of queries**. Each pipe starts with a query evaluation with ElasticSearch. The pipe interface lets the customer define the number of documents that must be processed by a pipe (i.e., "process only the Top-10000 documents of the ranked result"). Although this part does not seem to be significant with respect to the rest of the workflow evaluation, this remains to be confirmed.
- **Cost of workflows**. A pipe is executed as a graph of second-order operators. We should be able to evaluate, at a global (workflow) level, the behavior of these operators. This is particularly important for iteration based operations.
- **Cost of agents**. Finally, an agent (MIGNIFY) operates at the document level, and mostly apply some text mining method. This turns out to be the most costly part in some cases, and even sometimes prevent the agent to be applied at large scale due to its lack of efficiency.

We made a study to evaluate the resources (time and space) consumption of our main agents:

(A) Text normalization

(B) Tf extraction

(C) Topic extraction (based on language model)

(D) Microformats (MF) and micro-data (MD),

(E) Entity recognition (Agent Aida, based on Yago[6])

The following table shows the average cost of processing each agent per document, and the overall cost of running a pipe that linearly applies agents A to D (entity disambiguation is discussed separately). It should be underlined that the figures result from important optimization efforts in order to achieve the best possible performance, as each additional millisecond has a strong impact when processing millions of documents. The whole workflow takes on average less than 100ms for a single resource.

Agent	(A)	(B)	(C)	(D)	Total
Cost	18 ms	17 ms	40 ms	16 ms	91 ms

[6]http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/aida/

This translates to processing, on a single processor, 10 documents per sec., 864,000 per day and 6M per week. Distributed over several processors and computers, this scales linearly. Entity disambiguation is an issue. At the time of writing, optimizations are still in progress. The performance of the entity extraction is currently a few seconds (2-3), with a high standard variance: complex and large documents are much more costly to process.

Those figures constitute the baseline upon which we can build a reliable cost model. We need a sound methodology to produce performance indicators for each agent, so as to create and maintain a reference table.

Requirement 1.1. *Define and implement a methodology to obtain a reliable estimation of the CPU cost (per document, possibly parameterized by document size and type) of agents.*

The memory consumption is another concern that needs to be addressed. In general, agent process documents, and the memory usage is roughly proportional to the document size which can be known from the repository statistics. Here again, entity disambiguation distinguishes by the need to maintain an in-memory representation of an ontology. In the case of Yago for instance, this requires a PostgreSQL database containing Yago whose footprint is almost 4 GB. For this specific case, we use SSDs to avoid allocating a large part of our servers memory to the disambiguation agents. Being able to know in advance the memory requirements, and to model memory usage, is therefore part of the monitoring requirements.

Requirement 1.2. *Define and implement a methodology to model memory usage (per document, possibly parameterized by document size and type) for the* MIGNIFY *agents.*

Regarding the workflow second-order operators, we are mostly interested by iterative algorithms in general, and machine-learning model construction in particular. We need to obtain performance assessments for the following techniques

1. k-means based on tf.idf;
2. LDA topic model.

IMR already implemented the agents to execute these functions. We must incorporate them in a distributed processing engine and test their performance. We plan to proceed with Flink, but a comparison with Spark in the context of ASAP would be quite valuable.

Requirement 1.3. *Evaluate iterative machine learning algorithms (k-means and LDA) with Flink, and possibly Spark.*

4.2 Multi-pipes execution

Hadoop V1 is a monolithic platform which is able to run one MapReduce job at a time. This constitutes a quite severe limitation in a context where many concurrent pipes can be submitted to the system. Moreover, we do not dispose in the moment of a mean to determine a scheduling of pipes based on their expected performance (e.g., a pipe that would complete in 10 mns may have to wait for 2 days before being assigned an execution slot).

We plan to design a solution to set-up a general scheduler that will distribute and optimize a *multi-pipes* portfolio. We expect to benefit from the meta-services of the ASAP platform to estimate the costs and resources usages of a pipe. Based on this estimation we should be able to schedule the execution of multiple pipes concurrently by allocating bounded resources to each pipe.

The design of such a scheduler should be inspired by some existing cloud management platforms like Mesos [4] and Yarn [7]. We investigated both solutions, and it seems that Yarn is a resource sharing system which only manages softwares belonging to the Hadoop ecosystem. Mesos does not seem to present this limitation. The ASAP scheduler should not be limited to a single execution platform (e.g., Hadoop) and should allow the specification of a software ecosystem sharing the resources of a cloud infrastructure. At the very least, we should be able to declare the distributed indexing mechanism that we use in the moment (ElasticSearch) and the distributed crawler developed and used by Internet Memory (MemoryBot).

Requirement 2.1. *ASAP must include a resource management and scheduling platform for sharing our cluster's resources. Must take in consideration distributed applications beyond the Hadoop ecosystem (e.g., ElasticSearch, MemoryBot).*

Based on the expected performance and resource utilization of pipes (see the requirements above), we need to come up with a scheduling that ensures an optimal or near-optimal solution to execute a portfolio of pipes with respect to a list of constraints which together define a Service Level Agreement (SLA) for MIGNIFY.

The cost of running a pipe in MIGNIFY is expressed in MCUs (*Mignify Computing Units*). This cost is currently the sum of a fixed cost based on the number of documents processed by the pipe and a variable cost which depends on the "price" of each agent involved in the pipe. This price in turn is determined, more or less accurately, by considering both the resource consumption of the agent and the added value it brings to the information set built by the pipe. Informally speaking, an agent that merely finds the language of some content has less added value than an agent that finds and disambiguates all the named entities.

So far, a pipe runs as a MapReduce job in the cluster, and all pipes benefit from the same computing and storage resources. If we can estimate in advance the pipe resource consumption and completion time, it becomes possible to propose to our customers several possible SLAs, each associated with a specific cost. The lower the cost of a SLA, the lower the priority of a pipe.

Requirement 2.2. *Design a scheduling algorithm for pipes which automatically determines an allocation plan that satisfies the* MIGNIFY *SLA.*

Note that Requirement 2.2 implies an automatic evaluation of the resources needed by a pipe given the constraints associated to the MCU cost. This is not possible with state-of-the-art solutions (e.g., Yarn or Mesos, previously mentioned) which both require a manual input of this information.

We cannot afford a manual estimation for each pipe, and thus current solutions do no scale to the expected submission rate of pipes in MIGNIFY (see Subsection 4.3). If the requirement can be satisfied by ASAP, this will constitute a distinctive advantage for MIGNIFY in particular, and subscription platforms similar to MIGNIFY in general.

4.3 Continuous pipe execution

The final set of requirements pertains to *continuous* execution of pipes. The current state of MIGNIFY requires an explicit triggering of a pipe execution. Although pipes are able to run a delta-computation that only considers the documents newly collected with respect to the previous execution, they do not self-decide on the appropriate moment to run themselves. Relying on a human decision jeopardizes the ability of the system to provide timely results.

We therefore target an evolution of the pipe mechanism to enable a *publication/subscription* system. A publication event in this context is the insertion of a new document matching the pipe's query; a subscription is the production of the pipe output when applied to the delta result. This gives rise to the following challenges:

1. Detecting that a new document matches one or several pipe queries.
2. Delta computation of the result.
3. Execution strategy for delta computation: distributed or local?

The first issue is well known in the context of pub/sub system, and essentially relies on an indexing of queries to match incoming events against an efficient structure. The second issue is easy for document-level agents, but intricate when it comes to maintain a result based on some aggregation step. Typically, *models* produced for machine learning tasks evolve as new data is fed to the system. Maintaining k-means centroids, or a topic-model built by LDA, as the dataset continuously evolves, is a challenging problem.

The final issue can be related to the volume for which a distributed execution becomes relevant. We expect that in many cases, a local computation of pipes applied to the delta result will do the job. This can be clarified by considering the cost model for pipes developed to answer earlier requirements.

Requirement 3.1. *Design a pub/sub extension of* MIGNIFY *based on continuous pipe execution.*

4.4 Non functional requirements

We conclude with a list of non-functional requirement which mostly address the usability and scalability of the use case.

Visualization

The visualization components currently under development in WP 6 constitute important potential assets for the use case. Of particular interest are the charts devoted to time series data [6] that could be coupled with continuous pipe execution to monitor the evolution of topics and display the sentiment associated with a given entity in social web exchanges.

Monitoring

There is no means for end user now inspect the progress of pipes as they are running. Given that a pipe execution can span, in the worse case, several days, this leaves customers in the dark during a significant period regarding the scope of the data which is processed and the accuracy/quality of the result. The platform would therefore strongly benefit from a continuous monitoring, yielding indicators and clues on the forthcoming result. A promising approach is to provides such in-advance estimations based on samples drawn from the input datasets.

Scalability

At the time of writing, MIGNIFY maintains a catalog of more than 700,000 active sources from which web pages are constantly crawled. Wrappers associated to these sources extract structured content and publish this content, indexed, in ElasticSearch as input to pipes execution. The crawler collects approximately 5M of web ressources per month for these sources (feeds, web content referred to by feeds, and linked web pages up to a small number of hops for discovery purposes). Since we constantly enrich the catalog with new sources, we expect to reach, during the course of ASAP, in the order of 10 millions of sources and about 50 millions of web content crawled and indexed every month. ASAP should therefore be apt at dealing efficiently with several Billions of indexed ressources.

Regarding scalability requirements for pipes and multi-pipes execution, the following figures can be anticipated. Each pipe is allowed to retrieve up to 1M resources from ElasticSearch (this limit might change in the future). IMR agents are designed to process one resource in less than one second on average. A rule of thumb for this choice is that, by allocating 10 machines to execute a pipe in parallel, the result should be available in less than one day. The customer might choose to process less resources from ranked ElasticSearch result. S/he might also choose to allocate more or less machines to the job, depending on the expected business value of the result.

Our goal is to be able to evaluate the capacity of ASAP to manage concurrently up to 100 pipes running in parallel, and processing a number of resources on the order of 100 M per day. The number of machines that can be allocated to the pipe execution cluster in this scenario is 120. This corresponds to a linear scalability of the current performance, with the strong additional constraint that pipes should run in parallel and exploit optimally the resources.

Experiments at a smaller scale have to be carried out. The testbed described in the next section provides a small but representative infrastructure similar to the large scale setting envisaged above.

5 Test platform

We briefly describe in this section the test platform which has been installed on a dedicated cluster at Internet memory, and opened to our partners.

5.1 The test cluster

We installed a small cluster for tests and experimentations, with a 1TB collection. The cluster will be extended during the last year of the project to test for scalability. The cluster initially consists of 5 machines, with 4 cores CPU Intel(R) 2.90GHz processor, 2 x 4TB SSHD of storage, and 16GB of RAM.

We installed the following softwares:

- OS: Debian 6.0.10 (”squeeze”)
- Java: 1.7.0_55
- Hadoop: V2.0.0 in CDH4.6.0.
- HBase: V0.94.15 in CDH4.6.0
- Elasticsearch: V1.4.1
- Flink, V0.8
- Spark, V1.2.0

The collection consists of a large of European News site, periodically crawled from the Web. It contains approximately 28,913,436 documents, 500 GB of data at the beginning of the project, constantly increasing, indexed with ElasticSearch.

Access to the cluster is granted via SSH; we provide SSH keys to our partners whenever required.

5.2 MIGNIFY agents and workflow

A simplifier version of MIGNIFY has been created and installed on the cluster for testing purposes. We also provide sample code packaged as a Maven project. All the dependencies for agents and text mining libraries are defined in the maven specification file, and access to the maven IMR repository to download the necessary Java files. in particular:

- `TermVectorExtractorSample` demonstrates the whole procedure to instantiate, initialize, execute and finalize agents by using term vector extractor as example.

- `NamedEntityRecognitionSample` is another sample demonstrating named entity extraction from plain texts.

Since each MIGNIFY agent is packaged as a Maven module, partners can easily embed an agent in Java code (for example, to execute agents within a Flink or Spark workflow) by adding Maven modules of agent as dependencies of a new project.

References

[1] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix Naumann, Mathias Peters, Astrid Rheinländer, Matthias J. Sax, Sebastian Schelter, Mareike Höger, Kostas Tzoumas, and Daniel Warneke. The stratosphere platform for big data analytics. *VLDB J.*, 23(6):939–964, 2014.

[2] David M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, 2012.

[3] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[4] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, 2011.

[5] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1099–1110, 2008.

[6] A. Scharl, A. Weichselbraun, A Hubmann-Haidvogel, and W Rafelsberger. Deliverable 6.1 - ASAP InfoViz Services Early Design . Technical Report MSU-CSE-00-2, ASAP, February 2015.

[7] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, 2013.

[8] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A

fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, 2012.

FP7 Project ASAP
Adaptable Scalable Analytics Platform

End of ASAP D8.2
Use Case Requirements

WP 8 – Applications: Web Content Analytics

Nature: Report

Dissemination: Public