FP7 Project ASAP
Adaptable Scalable Analytics Platform

ASAP D8.4 Results of the performance and scalability study

WP 8 – Applications: Web Content Analytics

Nature: Report

Dissemination: Private

Version History

Version	Date	Author	Comments
0.1	January 10, 2017	Stan Barton	Initial Version
0.2	January 22, 2017	Philippe Rigaux	Sections 1-3
0.3	February 08, 2017	Philippe Rigaux	Usability report
0.4	February 13, 2017	Stanislav Barton	Draft performance study
0.5	February 14, 2017	Philippe Rigaux	Concluding remarks, proofreading
0.6	February 23, 2017	Stanislav Barton	Experimental results
0.7	February 24, 2017	Philippe Rigaux	Final revision
0.8	February 24, 2017	Arno Sharl	Dashboard section
0.9	February 26, 2017	Katerina Doka	IReS clarifications
1.0	February 28, 2017	Philippe Rigaux	Final revision

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

We report our final evaluation of the ASAP platform over our main data processing workflow. The workflow takes Web pages as input, and supplies as output structured and classified descriptions of product offers extracted from eCommerce sites. It goes through several steps of data extraction, clean-up, features production and on-line classification. These steps have been implemented as ASAP operators and we experimented their combined execution as ASAP workflows, testing the features of the platform, its usability and evaluating performance aspects for a large distributed dataset.

We describe the workflow, the business services it supports, the ASAP implementation, and report usability and performance results. The deliverable finally summarizes our conclusions regarding the current state of ASAP, and its ability to fulfill our requirements and manage our processes in the specific context of our web data management business activities.

1 Introduction

We propose in the present deliverable an evaluation of the ASAP system on the full operational workflow used at Internet Memory Research (IMR) for collecting, extracting and classifying products offers. This workflow is a central part of our business activities, since it produces the core information (classified products collected from thousands of web sites) that supports our services. It features several operators with various properties, costs and behaviors, and is therefore a concrete and representative candidate to assess the usability of the various ASAP components.

In order to conduct this evaluation, we first modeled our product management functionalities as ASAP abstract operators. This gives a high-level view over the sequence of steps that takes raw web pages at the initial step, and delivers a database of product offer descriptions, classified in categories and sub-categories, at the final step. Each step is abstract in the sense that it disregards the specific implementation (as an operator) or data source and sink (operators input and output). This modeling thus offers a flexible framework where a high-level perspective on an operator or a sub-workflow can be refined with an effective implementation (called *materialization*in ASAP) addressing a specific method and a specific data storage configuration.

As a second step, we implemented some variants for a few core operators, each supplying a particular behavior in a particular context. All these implementations are compliant with the materialized operator concept of ASAP. Some are simply a wrapping over pre-existing functionalities,

already used by IMR as part of its processes, and packaged with an API to be plugable in the ASAP framework; others have been implemented as a useful alternative in some specific cases which are further explained in the deliverable.

The ability of ASAP to run an abstract workflow as a context-dependent materialization has then been exploited to evaluate our general process in a range of various situations. We experimented several possibilities of the platform, interacting with our partners whenever necessary to better understand its current possibilities, limitations, and potential usages beyond its ability to optimize abstract workflows by estimating the costs and behavior of operators in a catalog of alternative implementations.

Finally, we installed an evaluation environment with a large dataset in a dedicated cluster, and ran several experiments to assess the robustness, usability and performance of the workflow controlled by ASAP.

The deliverable reports in details these evaluation tasks and their results, under the general goal of identifying the usages and benefits of the abstract, high-level approach promoted by ASAP. Its organization is as follows:

- Section 2 is devoted to the functional aspects (data acquisition, processing and services) that we chose to use as a basis for the evaluation of ASAP. The data processing workflow aims at maintaining and expanding a database of *product offers* collected from eMarketPlaces. The workflow is an extension of the one already investigated in year 2, and covered in Deliverable 8.3 [1]. We recall the essential and spend more time to present two of the services that rely of the database of eMarketPlaces products, since they provide a concrete and practical illustration of our requirements business-wise.

- In Section 3, we report our work on modeling and implementing the workflow with the ASAP system. We present the full abstract workflow, introduce the abstract operators, describe interesting materializations for each of them, and give a some examples of specific, materialized workflows that we need to run in particular situations.

- Section 4 describes the experimental setting, and our usability and performance conclusions. We detail the ASAP functionalities involved in the management of complex analytic workflows, showcase interactions and feedback of the platform from a user perspective, and report the main performance. results.

- Section 5 shows how the classified data can be visualized by the ASAP dashboard, and enriched with contextual information.

- Finally, Section 6 summarizes our work and proposes a list of perspectives to extend ASAP.

2 Overview of our IMR data processing and services

IMR collects, cleans and classifies data from Web, and use the result as a support for some of its on-line services. The present section give gives a brief summary of the IMR data processing steps, which is essentially identical to the one presented in depth in D8.3 deliverable [1]. The subsequent parts of the section introduce two services that exploit the datasets of products extracted from the web, as an illustration of the practical needs and usages of the acquisition wokflow.

2.1 Use case summary

The general goal of our data collection/extraction/classification process is to build and maintain a *catalog* of product references, and to discover *product offers* related to this catalog on public marketplaces.

1. A *Catalog* is a tree of categories and sub-categories. An example of category is *Coffee machines*, and a sub-category is *Expresso machine*. Any product that belongs to a sub-category also belongs to its parent category. We aim at classifying at the sub-category level for a better accuracy, and simply refer to it as "category" in the following.

2. A *product offer*, abbreviated POFFER, is an on-line proposal to sale one or several items of a product, with specific conditions (price, delivery, etc.).

If, for instance, some eMarketplace proposes 100 items of the coffee machine xxP34, at a given price YY, this constitutes a product offer for product xxP34.

Internet Memory Research maintains a "Wep map" of classified sites that references hundreds of thousands of marketplaces (see Table1 for a list of indicators related to our datasets and data acquisition). Our crawler scans the pages and identifies those that contain lists of products. We then, thanks to a semi-supervised approach, analyse the product page structure and produce a wrapper apt at supporting structured data extraction to obtain a product offer record. This mixes all kinds of product-related information, brand, type, price, textual description, user comments.

Once a site has been crawled, we revisit it periodically. The so-called refreshment policy depends on several factors and is decided by our crawl engineers. To state it briefly, what we consider as the "golden" sites are continuously crawled, whereas less top-ranked market places are visited with a varying periodicity, ranging from a week to a few months.

Once product information has been extracted from the page, we run a classification process to predict the category of the product. The *product matching* operation, denoted PMATCH, consists in associating a POFFER with a product category in the catalog, given the description of an offer extracted from some e-marketplace,.

IMR maintains and expands a large database of classified POFFERs. This ProductDB database supports several services which can be split in two main categories depending on their target users. B2B services, for brands and eCommerce companies, propose a range of market analysis reports,

Indicator	Value
Average HTML page size	25,000 bytes
Nb of known MarketPlaces	426,410,735
Nb of wrappers	234,126
Nb pages collected per day	2,305,056
Batch size	1,000,000
Nb of unique products	83,623,000
Size pages collected per day	57,6 GBs
Size pages collected per month	1,7 TBs
Size pages collected per year	20 TBs
Batch size	25 GB
Nb POFFER per moth	69 M
Nb POFFER per year	0,8 B

Table 1: Main parameters of the performance study

statistics, and monitoring, whereas B2C services propose marketplaces comparison, and an agnostic shopping cart. Those services are introduced in what follows.

2.2 The PRESENCE service

PRESENCE is a a B2B service that takes advantage of the ProductDB database to provide competitive intelligence. Product selling companies can get specific data on their brand and analyse their main competitors.

The service offers a public interface (see Fig 1) which gives, for a given brand, the list of eCommerce sites where the brand can be found. This interface is essentially a demonstrator, and more sophisticated functionalities are proposed to the PRESENCE customers to help them make strategic decisions.

- *eMarket overview*. Thanks to the availability of a large database of classified products, PRESENCE shows an eSeller where its products are sold, where its competitors are, and which retailers are relevant
- *Segmentation*. PRESENCE identifies platforms that are the most valuable for specific products categories. Analysis of the ProductDB allows to obtain targeted details on each category, including what are the leaders in each category and market place.
- *Coverage*. PRESENCE analyses the 5 main websites that are selling a brand online, giving key information, such as traffic, country or world rank, helping to decide whether these retailers are the right ones for you or not. Moreover, PRESENCE recommends 5 retailers an eSeller should work with based on its competitors distribution and their relevance.

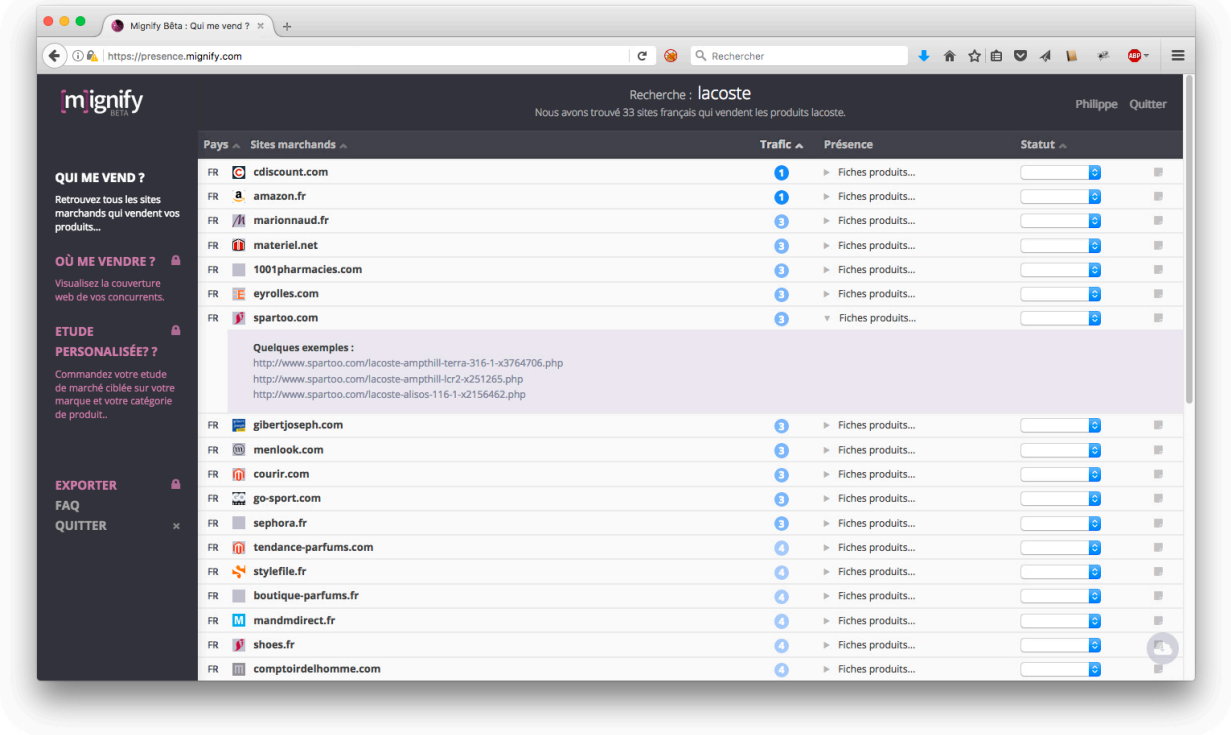

Figure 1: Interface of the PRESENCE service

- *Competitors*. PRESENCE helps to uncover every retailer website in which competitors are selling, and allows to better understand the competition distribution strategy, eventually discovering new opportunities on websites your competitors sell.

Finally, PRESENCE can also provide market studies in countries in which an eSeller is not selling yet, finding out who are the main actors, which retailers are relevant to the brands or if the domestic competitors are present as well.

2.3 The BOMERCE service

BOMERCE, (beta version launched in January 2017) is a web and mobile application that aims at addressing the main shortcomings of online shopping. Many people, when trying to explore the list of on-line offers for a product or a service, are confronted to the silo curse of proprietary eCommerce sites. A customer who wishes to compare several offers for a same product must open several tabs in the browser, and independently access in each tab to a specific site. This makes the comparison of offers difficult, and often requires to use additional tools such as a spreadsheet that summarizes the offers. Moreover, some useful and natural inspections cannot be achieved with this

clumsy mechanism: asking for an advice to a friend or third-party; finding similar and competitive offers in other eCommerce sites; being notified of promotions for the chased product or similar ones; or even checking the reputation and trustworthy of an eSite (some eCommerce platform tend to increase their price when the motivation of a visitor is know to be high...).

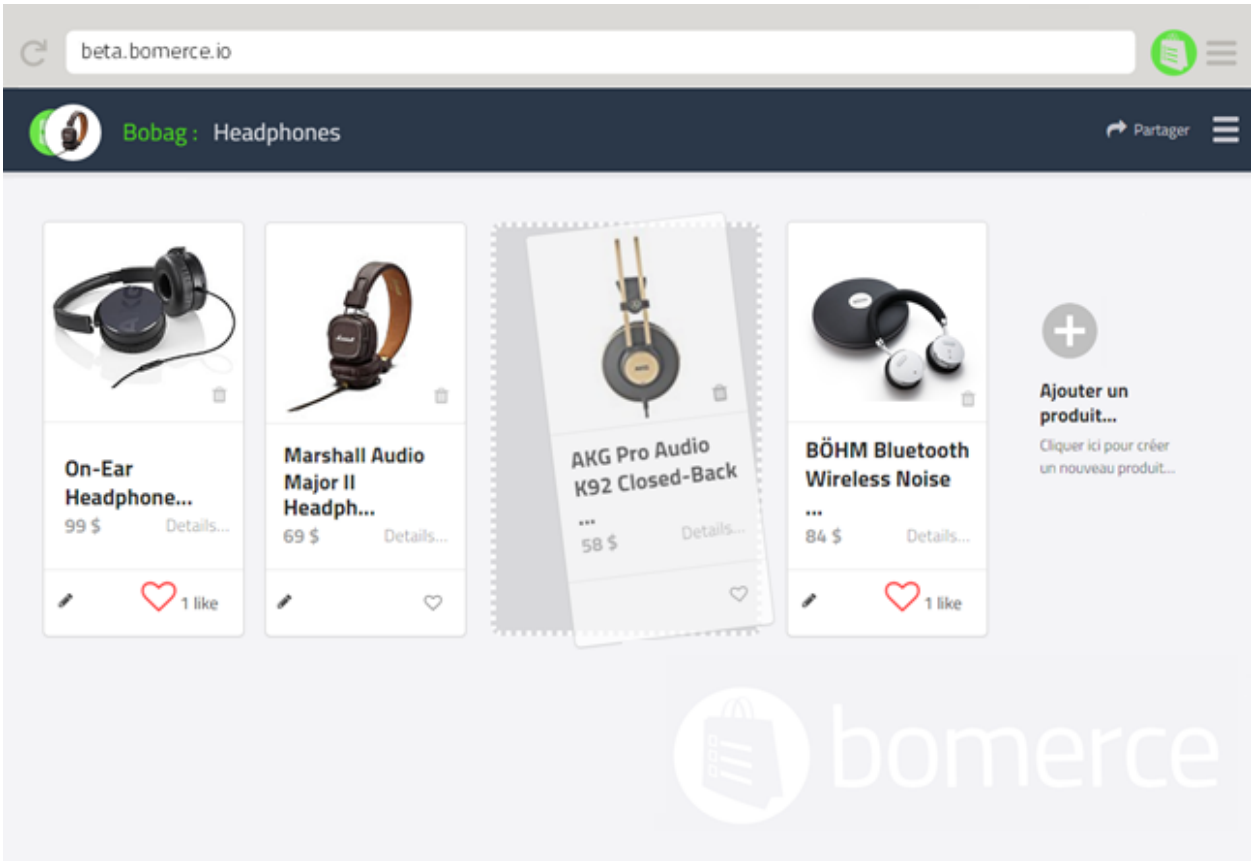

Figure 2: An agnostic shopping cart in BOMERCE

BOMERCE essentially acts as an "agnostic" shopping cart, independent from eCommerce platform, offering to customers a neutral place where product offers can be stored, compared and discovered. The BOMERCE customer experience can be described as follows:

1. The customer explores her/his favorite eSites, in search of a product matching her/his current interest; standard search/navigation operations lead her/him to a product page on an eSite A.

2. Thanks to the BOMERCE add-on in its browser (see the grren button in the top-right corner of Fig 2), the customer then requires the insertion of the product in a BOMERCE "bag". BOMERCE then :

(a) extracts on-the-fly the description of POFFER from the Web page, and identifies the product in the ProductDB thanks to the product offer matching mechanism.

(b) inserts the product offer found in A in the customer's "bobag", along with additional information that have been collected and stored in the ProductDB relatively to the product: customers's comments, reviews, demonstration videos, etc.

3. The customer can inspect her/his bobag, compare the offers, explore all the contextual information collected from the Web on the product, ask her/his friends for advices, and, in summary, disposes of all the relevant information to calmly reach a purchase decision.

Fig 2 shows a bobag for headphones.

This brief description of the two main current services developed by IMR illustrate the importance of an up-to-date, accurate, and complete information about the eMarket ecosystem. Timeliness is obtained through a continuous refreshment of the database content, thanks to the regular activity of ouw MemoryBot crawler. Accuracy is a result of our web data extraction semi-supervised technology, which allows to access to fine-grained information in complex HTML pages, and of our classification of extracted data as known and well-described product offers. Completeness refers to the coverage of eMarketPlace identification and extraction. Although our activity currently focuses on European countries, we carry out an in-depth exploration of theses countries's domains to discover eMarketPlaces, whatever their size, and achieves a representation overview on eSelling activities.

Both PRESENCE and BOMERCE depend on the quality of the ProductDB database, and therefore on the data acquisition, extraction and classification workflow. The next section details this workflow and its integration in ASAP.

3 Workflow Modelling and Implementation with ASAP

We now present how we modeled and implemented our data processing workflow with ASAP. First, a high-level modelization as an *abstract workflow* has been defined. It gives a full, high-level view of the various steps involved in the transformation that takes raw web pages collected from the web on one side, and delivers structured and classified description of product offers on the other side. This abstract workflow is then materialized thanks a catalog of concrete, alternative implementation of its operators. Finally, we identify situations where a specific combination of materialized operators is of interest.

3.1 The abstract workflow of IMR

Figure 3 shows the abstract ASAP workflow of our use case. Following the design of the ASAP system, it alternates datasets and operators.

The leftmost node of the workfow is a dataset, called "Web source", which is external to the ASAP workflow management scope. In principle, the source is simply the Web, explored by our crawler. However, the astract modeling gives us an opportunity to adopt a higher perspective where the source is actually any repository (including the Web itself) from which raw web pages can be obtained.

The *Raw Data Acquisition* operators (abbreviated RDA in the following) is in charge of feeding the workfow with web pages, whatever the actual web source is. Although this task remains the same for all materializations, the implementation depends on the exact nature of the Web source. Depending on the context, we might prefer to use one materialization or the other, and this adds an appreciated flexibility to the specification and execution of the workflow. This idea, which turned out to be useful for our internal processes, is further developed below in the part devoted to materialization. In all cases, the RDA output consists of a "Web Pages" dataset, explicitly managed by ASAP.

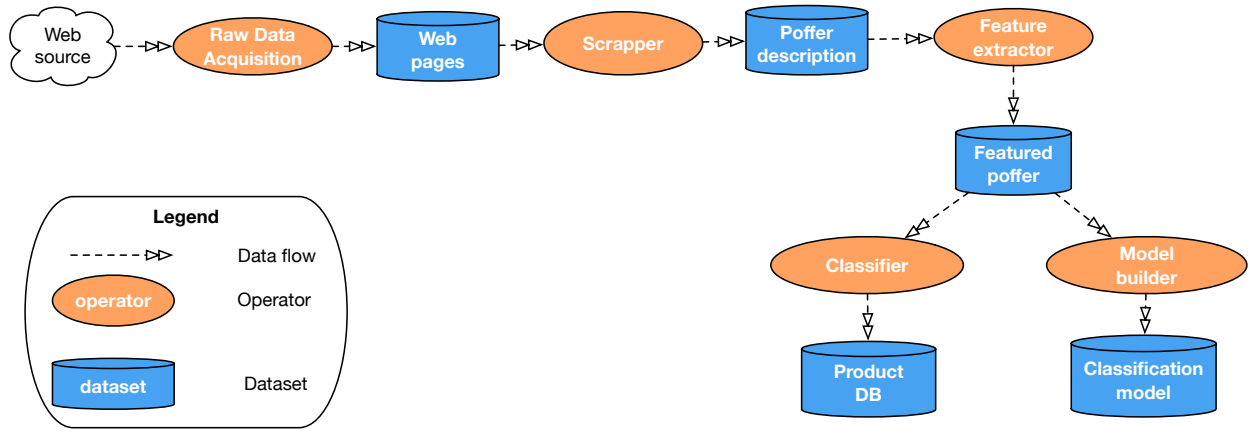

Figure 3: The full abstract workflow

The *scrapper* is a abstract operator in charge of extracting structured content from a Web page. The scrapper relies on an analysis of each web source to obtain a set of labelled locations of interest in the source's pages, called a wrapper. This analysis is carried out by an independent process, not decribed here. The wrapper for a given site (which takes the form of a set of XPath expressions) is stored in a Postgres database (no shown in the figure) which is queried at run time by the scrapper.

When applied to a relevant web page, a wrapper extracts a set of textual values, each associated to a label. As a typical example, in the case of product offers taken from eMarketPlaces, the labels are the product's title, the product's description, the price, the brand, etc. The scrappers takes as input a raw HTML page, aplies the relevant wrapper, and stores the result as a JSON object.

The set of JSON documents constitutes the *Product offer* dataset on Figure 3. An example of a record is this dataset is given below.


```
{"key":"http://www.ditech.at/shop/details.php?art=136349",
  "crawlDate":1478688898000,
```

```
  "Brand":"SAMSUNG Akku fr Galaxy S3",
  "Price":"375,99 E",
  "Image":"http://www.ditech.at/shop/ProductPics/ice120/136349.jpg",
  "LastCategorie":"Startseite Mobile Gerte Handyzubehr Handyzubehr Samsung",
   "ProductName":"SAMSUNG Akku fr Galaxy S3",
   "ProductLink":"http://www.ditech.at/shop/details.php?art=136349"
}
```


Next, the *feature extractor* operator is responsible for preprocessing the product offer description and produce a vector of features that serves as input for the classification process. The abstract definition of the operator encapsulates textual data pre-processing (typically, stemming and stop word removal) and the feature production (typically, tf/idf or Word2Vec). Although it could be refined in several sub-operators, we did not consider it as useful in the perspective of modeling the workflow with ASAP since the choice of a set of preprocessing operations and of a feature vector are often interrelated. This means that the decision for materialization of these sub-steps are not independent from one another, and that merging them at an abstract level simplifies the whole specification and avoid an intermediate materialization step at execution time.

We obtain a dataset called "featured poffers" in the following. This dataset can be used as input of either the Model Builder operator (which yields a model file) or the Classifier operator which produces a product offer enriched with the predicted category. At point, the result is ready to be inserted in the "product DB" dataset, and further used as a support for the PRESENCE and BOMERCE services as explained above.

3.2 Materialized operators

Given the abstract workflow of Figure 3, we implemented a catalog of materialized operators that can be used during the instanciation of the workflow as a materialized one, ready to be executed. Figure 4 summarizes the state of this catalog at the time of writing. Green nodes represent the materialization of abstract operators (orange nodes).

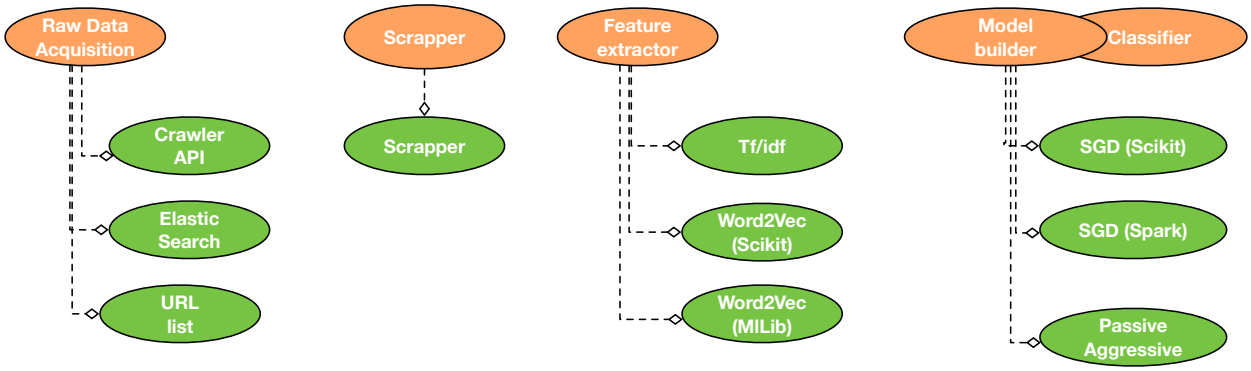

Figure 4: The catalog of materialized operators

Most of these operators are already exploited by the IMR processes, independently from one another, and in distinct execution contexts. ASAP allows to integrate them as steps in the global data transformation workflow. This integration as ASAP materialized operator relies in many cases on the following pattern:

1. The existing IMR functionalities is made accessible via a Web service,
2. The ASAP operator takes the form of a script that calls the service.

Although this method is not optimal from a performance point of view, it gives an easy way to plug an existing function in the ASAP execution context, and to test their respective behaviors and compatibility.

3.2.1 Data acquisition

The Raw Data Acquisition (RDA) abstract operator, in charge of feeding the workflow with web page collected from eCommerce sites, appears with three alternative implementations.

- **Crawler API** implements a call to our crawler API. Our crawler (MemoryBot) executes in parallel several "campaigns" whose scope is defined by a so-called "crawl specification". One such campaign is for instance the harvesting of all the products of a given category in a big eCommerce site. MemoryBot stores the collected pages in a staging area as Web Archive files (WARC). It also supplies an API to control the progress of the campaign, and to stream the web pages of the WARCs to a consumer application.

  This operator simply calls the MemoryBot API to obtain the result of a crawl campaign. A campaign may cover a period of several days (or even several weeks), and MemoryBot continuously supplies a stream of new resources during this period. At the time of writing, we collect approximately 2.3 millions of POFFER per day. Since there is no support for continuous workfow execution in ASAP, we decided to split this input in batches, and the ASAP operator (simply implemented as a loop on `curl` commands calling the service) stops whenever a given number of pages has been collected. The default value for the number of pages in a batch is 1 million, and it takes a few hours for MemoryBot to collect them.

- **Elastic Search API**. A separate process (called the ingestion process, not described here) co-operates with MemoryBot to get the crawled pages and store them in a distributed, queryable repository. We currently use Elastic Search as our main storage engine, due to its capacity to act both as a reliable, fault-tolerant strorage system and as a search engine.

  Keeping all the collected pages in a repository is costly, but gives us two benefits. First we obtain a temporal perspective over the on-line distribution of products, how their are covered by eMarketPlaces, and possibly some additional information useful for trend analysis (price, rank). This is an important asset for the Market Analysis service offered by PRESENCE, and

detect e-commerce websites that use the IP address and cookies to increase the price of items a customer has already been looking at. Second, if it turns out that an extraction/classification process failed for some reason (for instance the wrapper was outdated) we can make another round of product offer processing by using Elastic Search as an input instead of MemoryBot.

This ASAP operator simply queries the ElasticSearch engine (via its REST API) and retrieves a collection of pages based on some search criterias. The criteria might be a crawl domain identifier, a brand name (e.g., "Lacoste" or "Samsung"), a period, or a combination.

- **List of URLs**. The final materialization alternative simply takes as input a local file containing a set of URLs that can be used as keys to search for web pages stored in ElasticSearch. The motivation for this possibility is the ability to test a materialized workfow with a sample of web pages without having to launch the whole machinery that involves the crawler, the ingestion process, and the storage in ElasticSearch.

  This testing feature did not exist at IMR and is made possible by the abstract definition of the ASAP workflow and the flexibility it brings regarding the specific behaviour of a materialized operator. A quite useful value of this option is to let our content manager check the robustness of the workflow when confronted to the pages of a new site. By listing a few URLs in a text file, the behaviour of the wrapper for instance can be quickly evaluated.

  The ASAP materialized operator is a simple script that takes each URL in the file, one by one, and get their content with `curl`, either by accessing the Elastic Search repository through its REST API, or by directly retrieving the page from the Web.

3.2.2 Scrapper

There exists only one materialization for the scrapper. Given its simple task (applying a set of XPath extractions to a web page) and the existence of a single data source, we do not see here any opportunity to provide alternative implementations.

3.2.3 Feature extraction

Several feature extraction methods have been added to our catalog of materialized operators. They rely on a modelization of the vocabulary obtained by a pre-processing step (not described here) which supplies statistical properties on a corpus of product offers. In particular, the pre-processing of a corpus yields the following global, corpus-dependent indicators.

- **Inverse document frequency dataset**. For each term in the vocabulary, we can obtain how rarely it occurs in the corpus.
- **Word2Vec model**. The W2V model generator is responsible for generating the learned model that maps each discrete word id (0 through the number of words in the vocabulary)

into a low-dimensional continuous vector-space from their distributional properties observed in the input text corpus.

The corpus models are stored as files and loaded on demand by the following operators.

- **Tf/idf**. The operator takes a textual document and produces a vector of the tf/idf features. In incorporates some pre-processing functions, notably stemming and stop-word removal.
- **Word2Vec model (Gensim)**. This operator derives word-vector representations for the input data. It analyses the input text, tokenizes it to find the terms, and derives the feature vectors for each terms based on the W2V model. Once the feature vectors for all the words in a text are derived, their values are aggregated and averaged to derive the feature vector. It essentially calls the GENSIM routine (`http://radimrehurek.com/gensim/models/word2vec.html`) which itself is a port from the C Google package.
- **Word2Vec model (Spark/MlLib)**. Similar to the previous one, except that it relies on the Spark/MlLib implementation.

3.2.4 Model building and classification

Three pairs of (model/builder, classifier) are implemented in our catalog. Two of them were already used in the work conducted during year 2, and are part of the standard ASAP distribution.

- **Stochastic Gradient Descent (SGD), Scikit**[1]. In the classification model builder, the SGDClassifier model from scikit-learn permits us to train the logistic regression classifier in an incremental manner. The "loss" parameter in SGDClassifier is set as "log", the number of iteration n iter is 100, and the regularization parameter alpha is 0.0001.
- **Logistic Regression (Spark)** MlLib has a rich feature-set with regards to classification. MLlib uses a general Logistic Regression Model that can be trained with a number of algorithms. The one we have opted for is an implementation of L-BFGS (Limited-memory BFGS). BFGS is an iterative optimization algorithm that falls into the of Quasi-Newton family of methods.
- **Passive Aggressive Classifier**. This method is not part of the ASAP library and has been introduced in the IMR workflow recently due to the high precision that it delivers. The PA classification algorithm belongs to the family of online learning. The algorithm takes each instance and predicts a category. If the instance belogs to a golden set for which the exact category is known, the prediction mechanism can be improved to improve the chances of making an accurate prediction on subsequent rounds [2]. Online algorithms are important for

[1] `http://scikit-learn.org/stable/modules/sgd.html#stochastic-gradient-descent`

IMR because new products are constantly proposed, and the classifier therefore constantly needs to be updated in order to accurately reflect the state of the market.

To the best of our knowledge, there does not exists a distributed version of the PA algorithm, and we therefore rely on the Scikit implementation [2]. An initial model is built offline from a stored golden set, and the classifier is made available via a Web service. The ASAP operator is then simply implemented as a call to the Web service, combined with an `Update` request sent to the ProductDB database to record the classification of the product offer.

We are currently working on a distributed implementation with partners from the Data Mining group of the Sztaki University (Budapest). This distributed implementation will use the Flink engine, and when available, can be introduced as an alternative for our workflow.

3.3 Materialized workflows

In principle, all the possible combinations of materialized operators can be examined by ASAP, and executed if there are deemed efficient enough regarding a particular optimization objective. However, some specific combinations are known to be more relevant to achieve satisfying results, and we therefore chose to focus on them for the evaluation study. There of them are now discussed.

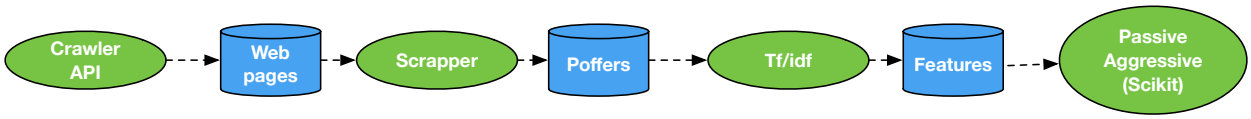

Figure 5: The main option for the full workflow materialization

The main option for materializing our full workflow is illustrated on Fig. 5. It corresponds to the sequence of steps currently implemented at Internet Memory (although not modeled as a regular workflow). It is worth noting that the combination of choice for the classification is the Tf/Idf feature extraction combined with the Passive/aggressive classifier.

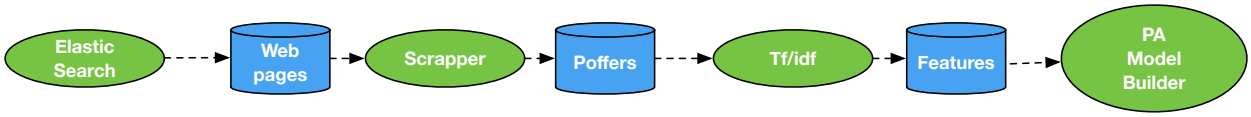

Figure 6: Building the classification model with Tf/Idf and the PassiveAggressive

We also consider two equivalent workflows that build the classification model. The first one (which corresponds to our main choice at the moment) combines the Tf/Idf extraction and the PassiveAggressive classifier and is illustrated by Fig. 6. An alternative is the workflow of Fig. 7 that

[2] `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PassiveAggressiveClassifier.html`

combines the Word2Vec feature extractor, and the SGD classifier. Recall that those options can be either run with a centralized implementation based on Scikit, and a distributed one based on Spark/MlLib. We evaluated the Scikit implementation for fairness with the alternative implementation of Fig. 6, also based on a Scikit/python3 implementation.

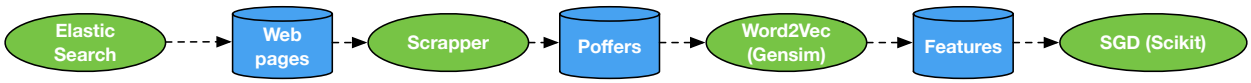

Figure 7: Building the classification model with W2Vec/SGD

Finally, a last work aims at exploring the ability of the ASAP framework to quickly set up a testing environment by running "local" materializations of a workflow to inspect the result of a specific operator. The workflow of Fig. 8 is intended to check the behavior and accuracy of the scrapper on a well-chosen and controled set of URLs. We therefore conbine the URL list implementation as a Web page provider, and run the scrapper on this list, the scrapper output being the main result that we want to investigate. Essentially, this permits to our content management team to zoom on a specific part of the workflow in a situation that matches as closely as possible the actual one.

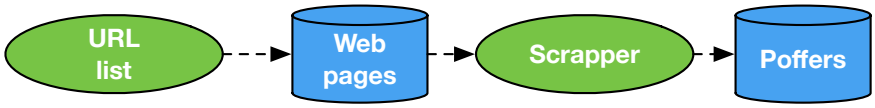

Figure 8: Testing the scrapper accuracy

4 Evaluation study

The ASAP platform as been installed on a dedicated IMR cluster. Note that this cluster is distinct from the one used by our partners for their own experiments. Details on this dedicated cluster are given below. It is connected to the main IMR repository and can therefore directly access and exploit our actual datasets and the crawler production.

ASAP has been installed from the github repository and following the documentation instructions. Some exchanges have been necessary to fix some issues, which led to several improvements of the installation procedure and of the documentation.

4.1 Modeling and running workflows with ASAP

4.1.1 Abstract operators

The core concept in ASAP is that of *abstract operators*. An abstract operator is basically defined by a name and the degree of its input/output dataflows. Our `ImrRDA` operator for instance represents an abstraction of the Web data acquisition process, and is simply modeled as:

```
Constraints.Input.number=0
Constraints.OpSpecification.Algorithm.name=ImrRDA
Constraints.Output.number=1
```

Note that the data source lies outside the scope of the ASAP, platform, hence the 0-degree input. Each operator materializing `ImrRDA` will internally take care of connecting to the exterior data source in order to supply web pages. The `ImrScrapper` abstract operator on the other hand accepts one input flow (web pages) and produces one output flow of structured product offers.

```
Constraints.Output.number=1
Constraints.OpSpecification.Algorithm.name=ImrScrapper
Constraints.Input.number=1
```

Once a catalog of abstract operators is in place, we can specify abstract workfows. They take the form of a DAG whose branches alternate datasets and (abstract) operators. The graph can be either created from the Workflow Management Tool or directly as a text file that encodes the edges for the IReS components. We used the second approach since the communication between WMT and IReS was still under implementation when we conducted the study. The graph is pretty easy to encode, as shown by the following example which represents our full workflow.

```
ImrRDA,WebPages
WebPages,ImrScrapper
ImrScrapper,Poffers
Poffers,ImrFeatureExtractor
ImrFeatureExtractor,features
features,ImrClassifier
ImrClassifier,output
output,$$target
```

Note that the operators do not communicate directly but via an dataset whose main purpose is to isolate the specific execution environment of each operator. We can distinguish:

1. *Materialized datasets*, already stored in HDFS and exploited as such. Those datasets (HDFS path, and meta-descriptors useful for profiling the operators) must be desribed in the `datasets` folder of the IReS library.

2. *Abstract datasets* are just aliases for intermediate files produced at run time as output of an operators and input of a subsequent one.

In our case, datasets are mostly abstract ones. Indeed, none of our dataset directly resides in HDFS. We therefore introduce as a first operator (the abstract `ImrRDA`) in charge of copying the data to be processed fro an external source (MemoryBot or Elastic Search) to HDFS for further processing by the ASAP operators. Likewise, our output mostly consists of a storage of the extraction/classification result in our ProductDB database (managed by ElasticSearch as well). The last operator in the workflow (abstract `ImrClassifier`) takes care of this storage. The `output` (materialized) file referred to in the graph description above contains messages reporting the status of the workflow execution.

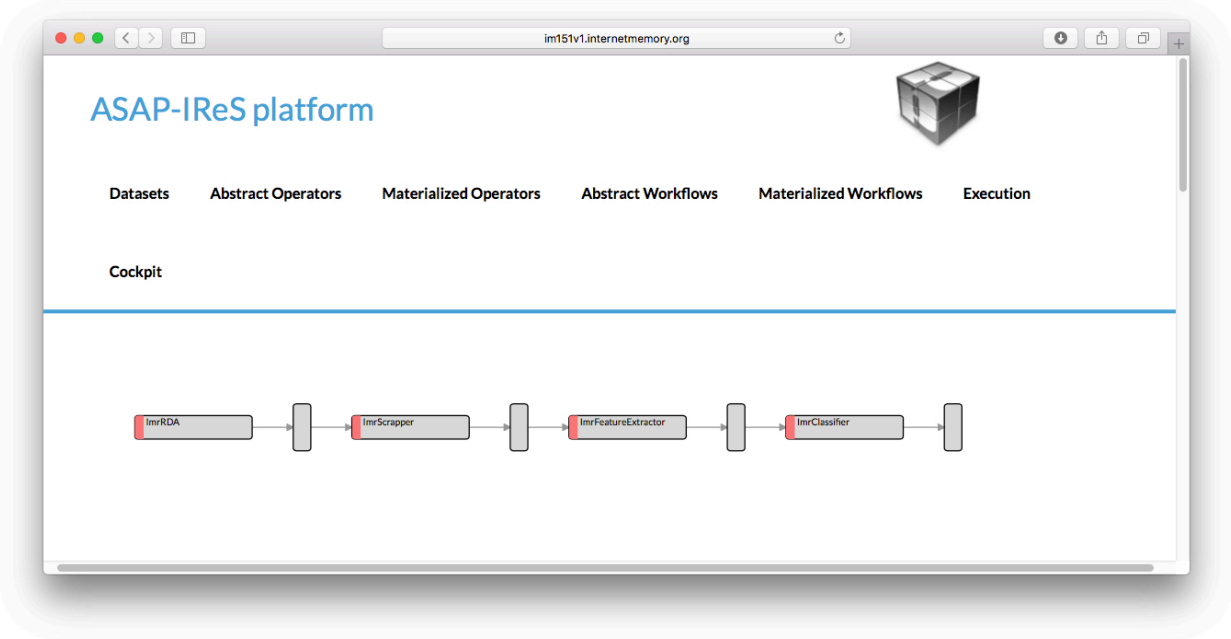

Figure 9: ASAP abstract model of the full workflow

Storing intermediate results in HDFS files leverages the possibility to alternate several execution engines in a same workflow, such as for instance a Spark operator followed by a Flink operator. It also provides a quite convenient mechanism to inspect the output of a specific operator and check that it matches our expectations. A downside is that the materialization is a penalty when confronted to large datasets. Note that we could combine two or more of the functional operators of our workflow as a single ASAP operator, loosing the benefits just mentioned, but avoiding

a costly HDFS checkpointing operation. The graph given above deliberately targets a maximal decomposition of the workflow in order to study the behavior of each operator and the workflow management features of IReS.

The Web interface of IReS then shows the full workflow, as illustrated by Fig. 9. It also proposes several actions to inspect the workflow definition, update it by adding/removing nodes, and finally materialize the workflow with materialized operators, to be presented next.

4.1.2 Materialized operators and materialized workfows

For each abstract operator, we can define one or more materialized operators. A materialized operator comes with an implementation, a LUA file to inform Yarn of the execution settings, and a description file that contains meta-data about the operator. Here is an excerpt of the description file for the materialized operator `Crawler` that calls the crawler API.

```
Constraints.Output.number=1
Constraints.Input.number=0
Constraints.OpSpecification.Algorithm.name=ImrRDA
Constraints.OpSpecification.Algorithm.impl=CrawlerRDA
Optimization.inputSpace.In0.documents=Integer,100,10000000,10000
Optimization.inputSpace.In0.lines=Integer,1,40000,100
Optimization.inputSource.type=mongodb
Optimization.inputSource.host=localhost
Optimization.inputSource.db=metrics
Optimization.model.execTime=gr.ntua.ece.cslab.panic.core.models.UserFunction
Optimization.model.cost=gr.ntua.ece.cslab.panic.core.models.UserFunction
Optimization.outputSpace.execTime=Double
Execution.Arguments.number=1
Execution.Argument0=crawlresults
Execution.Output0.path=$HDFS_OP_DIR/crawlresults
Execution.copyFromLocal=crawlresults
```

The first three lines (`Constraints.*`) are used to declare that `Crawler` is a valid materialization of the abstract operator `ImrRDA` and the fourth specifies the exact algorithm implementation, which in this case is the `CrawlerRDA`. The matching is done not only on the algorithm name and the input/output degrees but on any other field specified in an abstract operator. Thus, to specifically select the RDA algorithm implementation of Crawler the user needs to enhance the abstract operator with the extra field `Constraints.OpSpecification.Algorithm.impl`.

We implemented and describe our materialized operators as explained in Section 3. IReS then considers, for each node of the abstract workfow, a materialized operator. Fig. 10 shows the materialization obtained for our workflow.

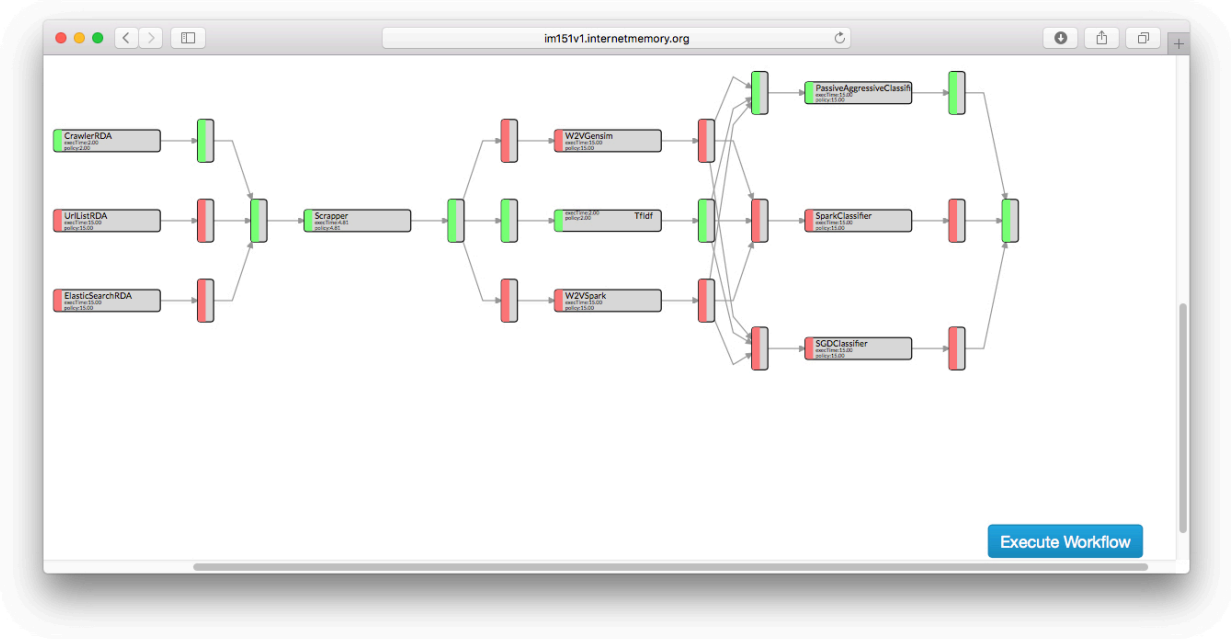

Figure 10: ASAP materialization of the full workflow

The green nodes represent those that have been chosen by the IReS optimizer. In principle, to take full advantage of the ASAP system capabilities, this choice would be dictated by the performance estimation made by IReS. If two operators are strictly iso-functional (they apply exactly the same algorithm, but in different contexts), this makes sense, and this indeed corresponds to the core design of IReS. Our workflow exhibits another potential case, where the materializations of an operator satisfy the same high-level needs, but with totally distinct algorithms. The raw data acquisition for instance (`ImrRDA` abstract operator) is in charge of supplying web page to the workflow. The way this data is obtained and transferred may vary a lot, and in our situation it is represented either by streams of pages continuously collected by the crawler, by archived pages taken from a storage engine, or from a short list of URLs which are retrieved on-demand for testing purposes.

All three acquisition methods represent a valid implementation of the abstract workflow node. This perspective is reminiscent of the specialisation/inheritance concept of object-oriented frameworks. Thus, we need to be able to choose distinct `ImrRDA` implementations through the materialization mechanism of IReS.

This requires to specify the exact algorithm implementation required by the analyst in the abstract metadata description (field `Constraints.OpSpecification.Algorithm.impl`). Changing the abstract operator description is as simple as updating a text file, and this allows with minimal effort to switch to a specific combination of materialized operators in a workflow.

4.1.3 Profiling and testing

Profiling is a standard functionality of IReS, and testing the behavior of a specific operator (beyond performance aspects) independently from its adjacent nodes in a workflow is an opportunity offered by the framework. We present our study of both characteristics of the *testing workflow*. We recall that this workflow focuses on the `Scrapper` operator so that it becomes possible to investigate it in isolation. Fig. 11 shows the materialized worklow in IReS.

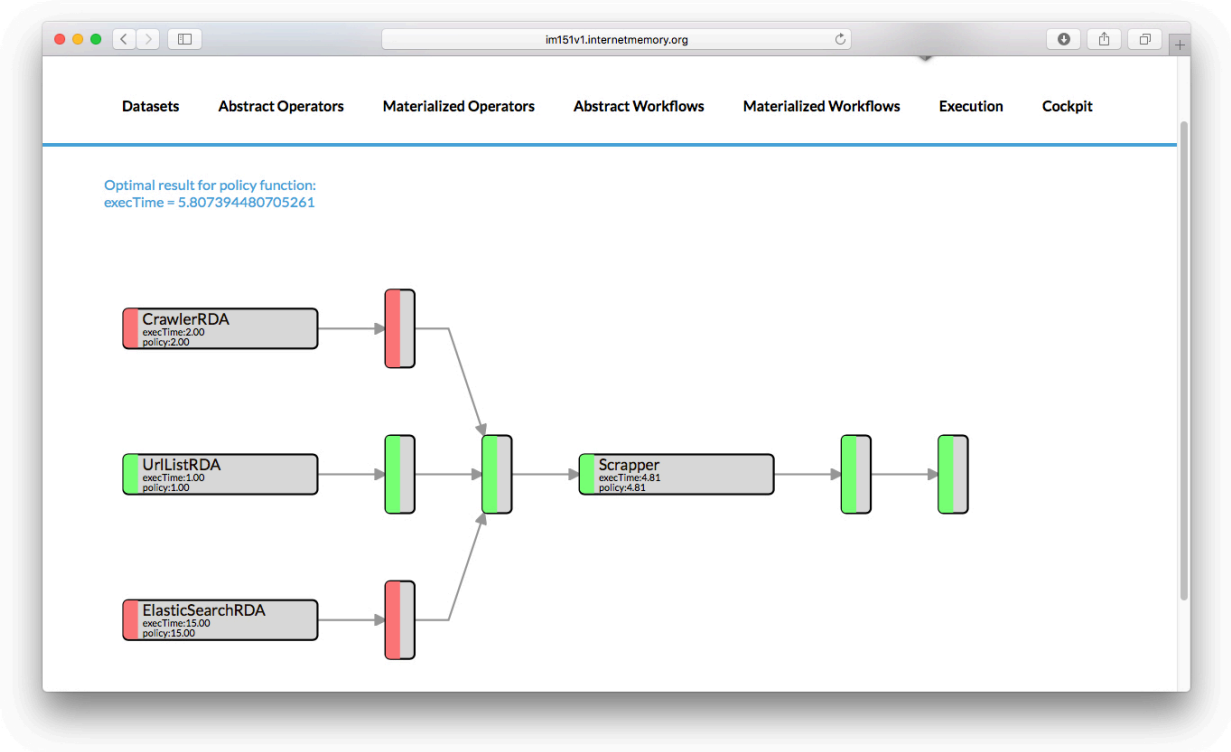

Figure 11: Running the scrapper profiling workflow

Unlike the full workflow, we did not set the explicit performance of the `Scrapper` in it description file, but let IReS evaluate it with respect to the following parameters:

1. Nb of documents (web pages) poffers are extracted fom.
2. Size (number of lines) of these documents.

We expect the cost of the scrapper to increase linearly with respect to the number of documents (assuming a constant size). The cost should also be linearly dependent on the size of the document, since this parameter affects the parsing to obtain a DOM representation, and the evaluation of

XPath expressions. Another parameter of choice would have been the number of features extracted from a document but we did not investigate this aspect.

In order to declare to IReS that those two parameters define the optimization space, we add the following lines to the workflow description.

```
Optimization.inputSpace.In0.documents=Integer,100,10000000,10000
Optimization.inputSpace.In0.size=Integer,0,40000,8000
```

The `size` parameter ranges from 100 to 40,000 bytes, with a step equals to 8,000 (note that, on average, the size of the web documents that we managed is 25,000 bytes). For the `documents` parameter, the maximum number is set to 1M documents, which is much lower than the cardinality of our datastets, but deemed sufficient to confirm the linear scalability and obtain an average cost per document.

Whenever a workflow is executed, the dataset description features the values of these parameters, such as for instance:

```
Optimization.documents=100000
Optimization.size=16000
```

This gives the coordinates of the point in the optimization space where the metrics collected by IReS will be positionned. In order to test the operator, we therefore generated 500 lists of URLs, combining cardinalities (100 values ranging from 10,000 to 1 million, step 10,000) and document size (5 values ranging from 8,000 to 40,000, step 8,000). We extracted those URLs from our web repository, and used the `UrlList` materialization of `ImrRDA` to enable a simulation of a data source supplying this flow of documents. For each URL, the operator gets the document from our Elastic Search storage, and calls the scrapping web service.

When the operator is executed, IReS measures its cost, and stores it a MongoDB database. This database is used at optimization time to determine the best operator when several alternatives are present. It also more directly serves as a valuable information source to inspect the behaviour of an operator. Fig. 12 gives a view of the IReS user interface which proposes to explore the optimization as a set of 2D projections. We can also directly query the MongoDB database, as illustrated below.

```
> db.Scrapper.find({},{"_id":null,"In0@documents" : 1,
...         "In0@size" :1, "time":1}).limit(10)
{ "time" : 3750, "In0@documents" : 10000, "In0@size" : 4000 }
{ "time" : 8247, "In0@documents" : 20000, "In0@size" : 4000 }
{ "time" : 10783, "In0@documents" : 30000, "In0@size" : 4000 }
{ "time" : 13851, "In0@documents" : 40000, "In0@size" : 4000 }
{ "time" : 19448, "In0@documents" : 50000, "In0@size" : 4000 }
```

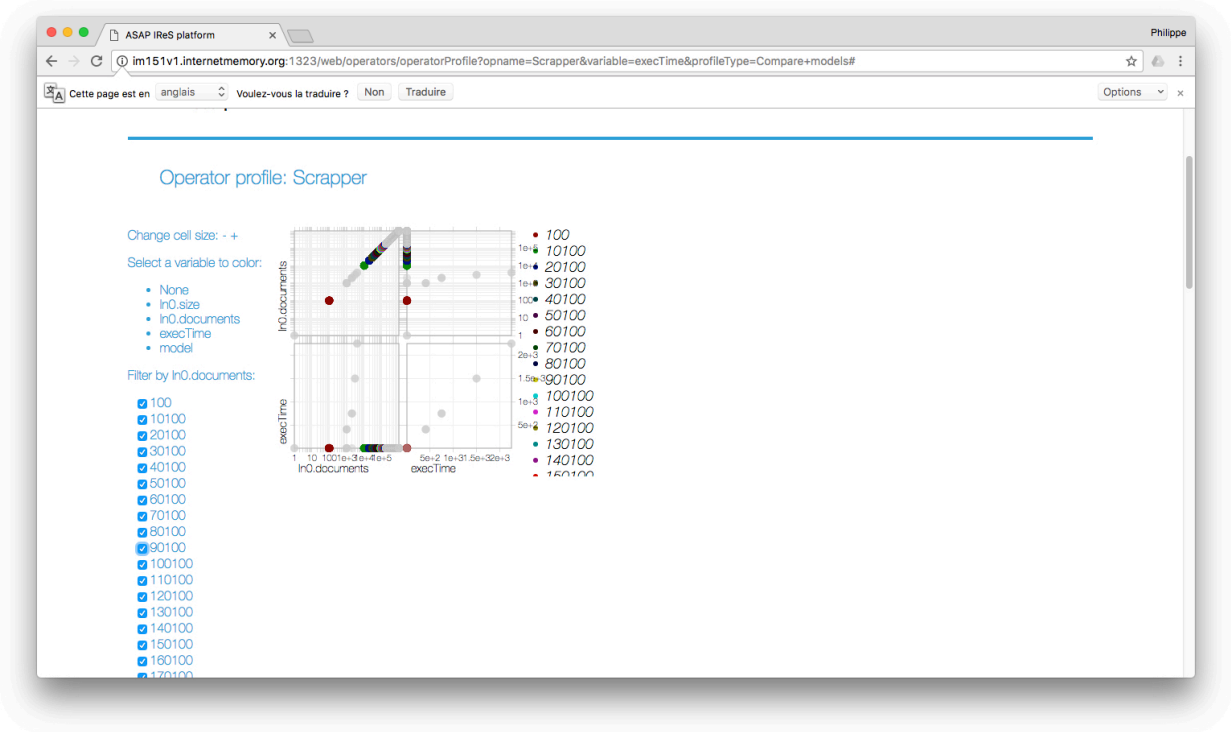

Figure 12: Profiling operators

```
{ "time" : 4587, "In0@documents" : 10000, "In0@size" : 8000 }
{ "time" : 5035, "In0@documents" : 10000, "In0@size" : 12000 }
{ "time" : 5262, "In0@documents" : 10000, "In0@size" : 16000 }
{ "time" : 5381, "In0@documents" : 10000, "In0@size" : 24000 }
{ "time" : 5501, "In0@documents" : 10000, "In0@size" : 32000 }
```

The query searches the `Srapper` collection, and projects the parameter values (`documents` and `size`) and the `time` measured by IReS.

Beyond profiling, this experiment illustrates a valuable features offered by the multi-level modelling of the workflows in ASAP. By choosing a specific materialization of a workflow, we can choose to focus on a particular operator, and evaluate this operator both under a performance perspective (as explained above) and under a functional perspective. In our case, the combination of the `UrlList` operator with the scrapper gives to our content management team a means to inspect the result of the latter, and therefore to debug some faulty parts of the full workflow (a typical example is that of a wrapper that becomes obsolete and unable to extract relevant data from a page whose DOM structure has changed). This intermediate storage of each operator in an HDFS file, which on the one hand is a performance penalty, on the other hand enables this operator monitoring possibility.

4.1.4 Performance-driven operator choice

The choice of an operator can also be dictated by the estimated cost of this operator, and this actually constitutes the default optimization choice of IReS. In order to evaluate this feature we implemented two equivalent workflows that train the classification model and run the classifier. The first one combines the Tf/Idf extraction and the PassiveAggressive classifier and is illustrated by Fig. 6, page 17, the second one is the workflow of Fig. 7, page 18 that combines the Word2Vec feature extractor, and the SGD classifier. Both are implemented in Python and rely on the gensim/scikit libraries.

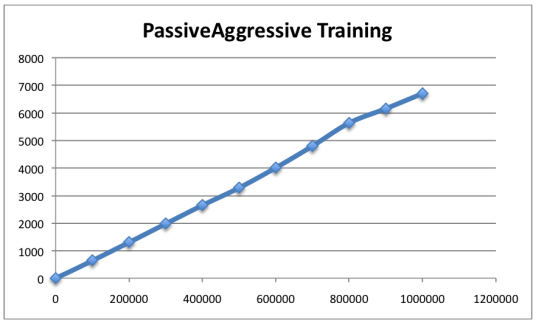

Figure 13: Model building (PA)

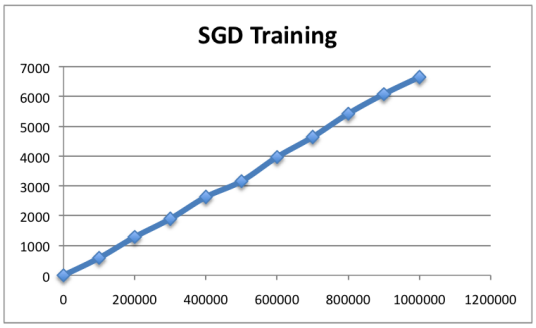

Figure 14: Model building (SGD)

We profiled the cost of these solutions over datasets ranging from 0 to 1,000,000 documents (i.e., the size of a batch datasets currently to simulate continuous classification). Results are summarized respectively on Fig. 13 (PA model training) and Fig. 14 (SGD model training). It turns out that SGD is constantly, but slightly better than the PassiveAggressive solution. In both cases, it takes less than 2 hours to build the model. Given that, the main metrics that we want to consider is not the building cost but the precision obtained by the classifier. We measured a precision of 75% with the PassiveAggressive solution, and 62% with SGD. The former is therefore our main option. Figure 15 shows the respective costs of the PassiveAggressive and SGD classifiers once the model is built. The costs are again essentially similar (though the PA now runs a bit faster), but are not deemed essential in our current perspective since we can roughly classify 4M porudcts per hour which is sufficient to our needs.

It is worth noting that in the case of a training algorithm, the materialization of a dataset prior to the construction of a model is a necessity, and thus the IReS workflow inter-operators materialization does not here constitute a penalty.

The above results are interesting to illustrate some exploitation aspects of ASAP.

Tuning a machine learning operator is not a simple task. A lot of parameters have to be chosen to obtain an acceptable robutsness with respect to the training set statistical distribution and to achieve a good balance between the ability of the model to process highly variable input and the

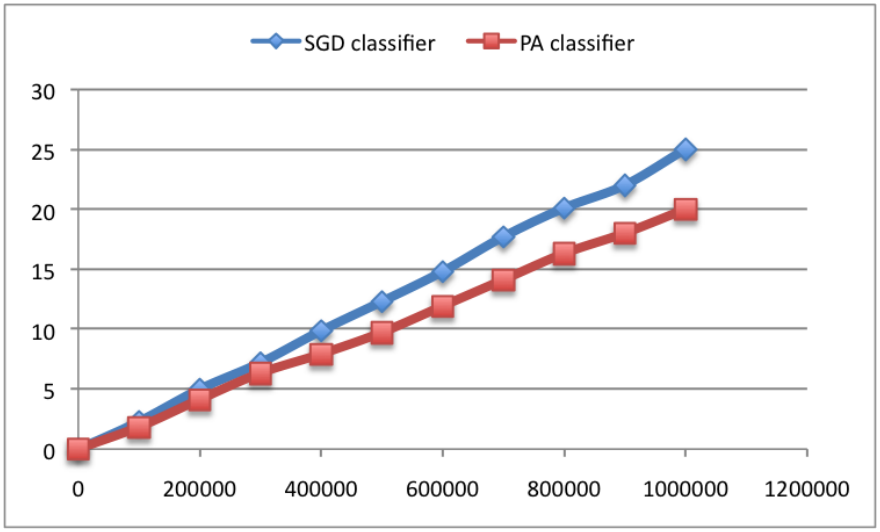

Figure 15: Performance of SGD and PassiveAggressive classifiers, in docs/sec.

necessity to preserve a satisfying precision. Several choices not directly related to the tuning of the classifier (such as the pre-processing steps) also play a significant role.

Our choice has recently moved from the SGD classification algorithm to the PassiveAggressive one, due to the superior precision that we managed to obtain. Since the performances of both solutions are comparable, the optimization parameter that should be privileged here is precision. This can simply be done in IReS by setting:

- A new optimization parameter in the description file of all materialized classifier operators (Optimization.precision). Since the profiler module of IReS is not able to measure precision, the field value is specified by the analyst as a constant.
- Maximizing precision as the optimization objective in the workflow definition.

Indeed, maintaining the option to switch quite easily from a materialized operator to the other is a quite interesting engineering feature in a context where we constantly have to adapt to an evolving situation. In the present setting for instance, our wrappers extracts a small set of features (title, description, brand, price) that we plan to extend in the future, for instance by capturing the comments of the customer on a product. Changes in the product description might impact the cost and precision of the classifier, and the ability to backtrack to a former solution, or to evaluate a completely new one at low integration cost is quite valuable.

4.2 Performance evaluation

We evaluated the performance of the other materialized operators. The reader is referred to Table 1, page 8 for a list of the main indicators describing our datasets. In the following, all the performances are reported on a batch of 1 million pages. Basically, each operator exhibits a linear behavior, and the results can therefore be extrapolated to larger datasets.

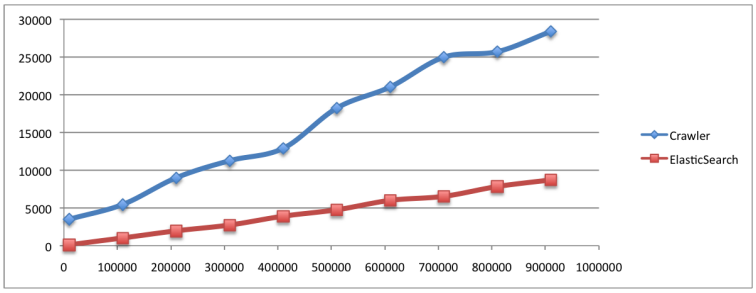

Figure 16: Performance of the `CrawlerRDA` and `ElasticSearchRDA` operators

Fig 16 compares the performances of the two main RDA materializations (we did not evaluate the `UrlList` operator which is designed for small testing datasets). The crawler curve shows a highly varyiable throughput, due to its dependency to a bunch of contextual parameters, including the politness rules, sites latencies, and network delays. The configuration of the crawler itself is essential: memoryBot is a distributed crawler, and its acquisition rate is proportional to the number of nodes. The figures of Fig 16 are obtained with a 4-nodes setting. We obtain a troughput of about 10 pages per second, which means that a batch of pages is retrieved in about 6 hours.

ElasticSearch supplies a more steady flow of pages, and its throughput is much better, with about 100 docs/sec, and a complete batch retrieved in approximately 2 hours. Recall that those operators are not equivalent and are used in quite distinct situations. The crawler is the initial supplier of web pages, whereas ElasticSearch is used when we need to re-execute the worflow on a part of the collection which is misclassified for some reason.

Figure 17 reports the performance of the only implementation of the `Scrapper` operator. The average time to process one page is 0.016 seconds, and we can proces approximately 62 pages per second (in a centralized environement). This is quite sufficient when the source is the crawler, but the scrapper becomes a bottleneck when the data comes from ElasticSearch.

These results were ignored by our exploitation team, and the ability to get and maitain them easily with ASAP has revealed their importance for the management of our workflow, regarding in particular an accurante allocation of resources to the operators based on their local performance on the one hand, and the global throughput of the workflow on the other hand.

For instance, the following guidelines can be deduced from the above results:

1. whenever the workflow input is supplied by MemoryBot, crawling (i.e., the retrieval of content from the web) is an inherent limitation, and there is no need to allocate to the other

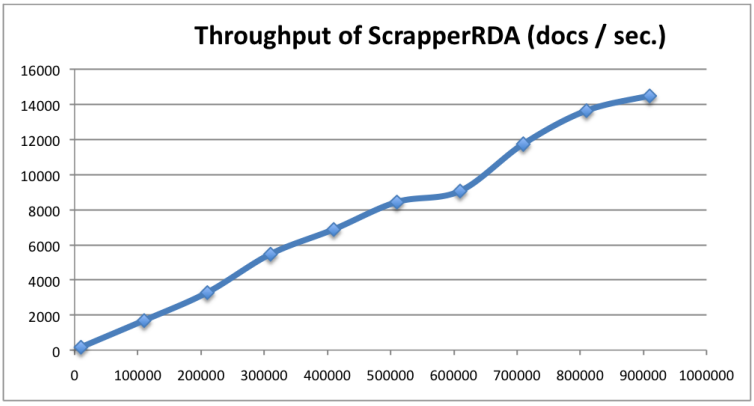

Figure 17: Performance of the `Scrapper` operator

operators more resources than needed to cope with its relatively modest throughput.

2. on the other hand, a more efficient data source calls for an appropriate configuration of the sensible operatots, for instance by choosing a distributed evaluation of the scrapper on several nodes.

More generally, the individual performance indicators supply a valuable input to balance the resources of the differents steps of a workflow. Ideally, this balance would be automatically determined and chosen by the workflow management system.

Finally, Fig. 18 shows the cost of the features extractors. They are all one order of magnitude more efficient than the scrapper. This makes sense if we recall that the scrapper deals with large HTML pages, 25K on average, that must be parsed as a large DOM tree, whereas the input of a feature extractor is a small textual description.

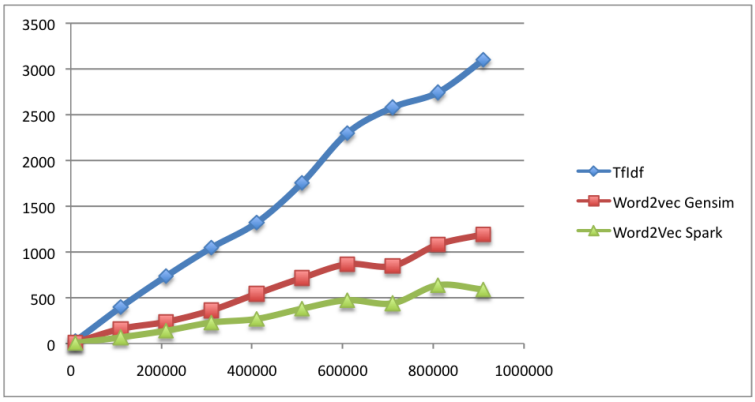

Figure 18: Performance of the features extraction operators

5 Visualizing Results with the ASAP Dashboard

To extract and visualize context information contained in the collected POFFERs, the Document API developed in WP6 and first reported in D6.3 was used to ingest their textual descriptions - the PDF documentation of the extended REST API was converted to a more user-friendly online format[3] using the Swagger open source framework. The integration into the ASAP dashboard of WP6 extends price comparisons by:

1. Visualizing aggregated keywords computed from noisy textual descriptions contained in the POFFERs collected from e-commerce and price comparison sites such as Amazon.com, Cdiscount.com and Geizhals.at.

2. Identifying the leading sources of POFFERs among these sites, including an analysis of which keywords each source associates with either a specific product or an entire product category.

3. Exploring specific product features that impact the perception of a product in online media coverage (news channels vs. social media vs. product offers), creating additional value for sales and marketing decision makers as an important target group of business intelligence tools. This allows not only to better understand which attributes the various e-commerce sites place most emphasis on, but also to gain additional insight into the social perceptions of specific mediators (variables with an interfering effect) or moderators (variables changing the magnitude) that were identified in consumer satisfaction-loyalty relationships[4].

4. Adding metadata dimensions such as sentiment, which serves as an indicator as to whether a feature is mainly perceived to cause satisfaction (= unique selling proposition) or dissatisfaction (= hygiene factor). This distinction can have an important impact on strategic resource allocation for marketing communications[5].

Fig. 19 shows a screenshot of the ASAP dashboard, comparing offers for the Samsung Galaxy, Apple iPhone and Google Nexus product lines as an example of high-impact consumer goods. The screenshot is based on a query for the Samsung Galaxy product series, returning 51,006 matches between 15 Sep and 15 Dec 2016. Associated keywords are shown in the form of a list sorted in decreasing order by co-occurrence significance (lower left), a tag cloud sorted alphabetically, and a graph-based visualization showing multiple hierarchical layers of associations (the first layer corresponds to the list in the lower left corner). “Color” and “battery” are among the top associations, reflecting two product features with different polarity i.e. positive vs. negative sentiment,

[3] `https://api.weblyzard.com/doc/ui/#/Document_API`

[4] Tuu, H.H. and Olsen, S.O. (2016). The Satisfaction-Loyalty Relationship in Marketing: A Critical Review and Future Research, Journal of Economics and Development, 18(1): 92-116

[5] Woodham, O.P., Williams, J.A. and McNeil, K.R. (2016). Toward Understanding the Impact of Attributes on Satisfaction in Different Price Tiers, Journal of Consumer Satisfaction, Dissatisfaction and Complaining Behavior, 29: 91-117.

Figure 19: Screenshot of the ASAP dashboard with an analysis of POFFER descriptions (09-12/2016) based on a query for the Samsung Galaxy product series including keyword associations, a list of major sources, brand comparison over time, and regional distribution of geographic references contained in the POFFER descriptions

respectively. They are also represented in the line chart and the bar chart, together with the other product lines and the search query itself (dotted line). The main content area shows the dominant POFFER sources for the Samsung Galaxy series, together with source-specific term associations. In the upper right corner, the geographic map depicts the regional distribution of geographic references identified in the product offers. Most references were identified for France and the United Kingdom.

Fig. 20 demonstrates the use of knowledge extraction techniques to compare product associations for the Samsung Galaxy series between POFFERs (left), news media coverage (center), and social media coverage (right). It is interesting to note that there are significant differences across these sources. Product offers center around the brand itself and key product features, predominantly employing positive language. News media mostly report issues of national or global

Figure 20: Product associations for the Samsung Galaxy series comparison based on POFFER descriptions (left), news media articles (center), and social media postings (right)

relevance, such as the September 2016 recall of the Galaxy Note 7 after a manufacturing defect in the batteries resulted in fires and a reported loss of USD $5 billion[6]. The recall impacted social media coverage as well, although to a lesser extent since social media authors tend to focus more on their individual user experience, or technical discussions such as how to upgrade a smartphones firmware or gain root access.

The shown examples demonstrate the potential of semantic technologies in conjunction with visual tools to automatically transform noisy and unstructured Web content into valuable repositories of actionable knowledge[7]. For a business intelligence tool in conjunction with price comparisons, supplemental metadata attributes are particularly important. Addressing this requirement, the drill-down sidebar developed in WP6 and shown in Fig. 21 extends the capabilities of the platform to show the temporal distribution of various metadata attributes, and to compare results across content sources.

The line chart compares the average sentiment for the Samsung Galaxy series by source (product offers, social media, news media) over time, the bar chart presents the same data in aggregated form, and the scatterplot the frequency vs. sentiment distribution of the major content sources. Two very significant drops in the average sentiment of news media articles towards the Galaxy series were triggered by the product recall (15 Sep 2016) of the Galaxy Note7, and the global sales stop (10 October 2016). While the number of Note7 product offers declined rapidly, the sentiment of other Samsung Galaxy offers remained unaffected by the recall (since offers typically do not discuss current events or other models). The strong rebound of sentiment in November correlates with an increase of Samsung's fourth-quarter profit by 50% to $7.9 billion (on flat revenue), the

[6] http://www.fortune.com/2017/02/21/samsung-galaxy-7-reputation

[7] Scharl, A., Weichselbraun, A., Gbel, M., Rafelsberger, W. and Kamolov, R. (2016). "Scalable Knowledge Extraction and Visualization for Web Intelligence", 49th Hawaii International Conference on System Sciences (HICSS-2016). Kauai, USA: IEEE Press. 3749-3757.

Figure 21: Screenshot of the ASAP dashboard showing the drill down sidebar to compare average product sentiment for the Samsung Galaxy series by source, including a scatterplot for cross-media analysis, a projection of referenced locations, and a tooltip for on-the-fly query refinement

company's best numbers in three years[8].

On the right, the geographic map reflects the major increase in scalability achieved in Y3 compared to the Y2 version of the dashboard, now projecting the entire result set instead of just the 50 top-ranked documents. The adaptive tooltip in the lower right corner enables on-the-fly query refinements, either to replace the search query with a new term, or to apply the Boolean operators AND (Restrict), OR (Extend), AND NOT (Exclude). Using the dashboards view synchronization mechanism, the tooltip is tightly coupled with the tag cloud which highlights the terms "battery" and "color" as the strongest associations with the hovered keyword "Samsung".

[8] http://www.usatoday.com/story/tech/talkingtech/2017/01/27/samsung-makes-strong-case-rebound-2017/97135248

6 Discussion and concluding remarks

We now summarize the conclusions that can be drawn from the study that precedes. The outcome of using the ASAP platform can be split in two parts: benefits brought by the high-level approach supported by ASAP, and new insights on some aspects of data processing workflow management that are implicitly part of the platform design, and suggest improvements and extensions.

High-level workflow management

ASAP leads to a modeling of workflows as independent, system-agnostic DAG of operators connected by data flows. This brings to the design of big data management the standard and well-known advantages of declarative data management systems: (i) ability to rely on a stable, well-defined and expressive model, (ii) independence between the general specification and a specific materialization, and (iii) opportunity for last minute optimization and tuning. These advantages are often blurred in distributed engines that are more focused on system aspects (load balancing, fault tolerance) than on user friendliness concerns. In our particular case, modeling the abstract workflow showed how stable is our data acquisition method since 3 years, something that was not obvious as we often change the implementation, running engine, and data storage platform. Initally, for instance, our datasets were stored in HBase and we decided relatively recently to move to Elastic Search which is both more efficient and more flexible (at least for our needs). This involved a refactoring of our platform that would have been clarified and probably simplified if our workflow had been modeled at an abstract level in the first place.

Specifying explicitly our operators, their tasks, the characteristics of the data that they consume and produce, encourages the management of changes and handling of particular situations at a conceptual level. As an example, in a near future, we plan to adopt Flink as a distributed engine at least for a part of the workflow. The multi-engine nature of ASAP workflow makes it easy to implement a node with such a new engine and to experiment its features without changing the rest of the workflow. Of course, a disciplined design adopted from the beginning would have permitted this flexibility as well, but using a framework a definitely an asset when it comes to coordinate a team of engineers who face many technical challenges. Moreover, ASAP brings this idea one step further than existing declarative workflow systems (e.g., Pig, Hive, Spark of Flink workflow specification languages) by allowing the full replacement of a data engine by another, and the comparison of both solutions with minimal integration effort.

New insights on the management of big data workflows

ASAP main goal is to profile a large set of (materialized) operators, some of them implementing the same algorithms, and to decide at run-time which materialization must be used in a specific settings and with specific optimization targets. This can be seen as inspired from relational systems which propose several (internal) equivalent implementations for the same operator (e.g., join

algorithms). In the case of data processing workflows, there are strong differences, though. First, the workflow nodes are not limited to a small algebra, but can implement any functional operation, and the number of possibilities is virtually unbounded. Second, the burden of implementing the alternative materializations of an operator is on the application developer, who has to ensure that the materializations are and remain iso-functional throughout maintenance and extensions.

The main target for experimenting this feature in our case is the classification algorithm. During the project we experimented three distinct methods, and variants that can either run in centralized or distributed mode. A problem there is that obtaining similar classification results with different implementations is hard, and the classification quality, measured essentially by precision in our case, in likely to always differ from one implementation to the other. For our use case, we will definitely favor quality over pure perfomance. This calls for a larger definition of the "optimization" concept, as a decision process based on any application-dependent factor that can hopefully be evaluated automatically. Currently, we add explicitly the measured precision (obtained by our own evaluation) in the operator description to drive the choice of the IReS optimizer. A future extension of the platform could include a generic mechanism to incorporate the optimization factor as the result of an external function.

If we depart from the idea of keeping strictly equivalent (in terms of functionality) materializations of an abstract operator, we obtain a broader perspective where a node achieves a high-level functionality, possibly by quite different means. Our data acquisition operator is an illustration, where the whole workflow can be fed by a "black box" that supplies flows of web pages, no matter the data source it relies on. This "specialization" (to speak in object-oriented terms) of a workflow function is quite convenient to further enlarge the applicability scope of a workflow and make it more generic. This brings a factorization effect that is likely to save time and development efforts to the development team, and make the workflow useful to a wider group of people, for instance as a testing environment where a new implementation can be checked, or as a materialization focused on a sub-part of the workflow to closely examine the properties of its data consumption and production, a feature well appreciated by our content management team.

In summary, our collaboration to ASAP has led to a significant improvement of the workflow management procedure at IMR. Initially, a workflow was an opaque, monolithic piece of software, implemented by our engineers, and beyond control of our content managers regarding its detailed definition. Moreover, the functional specification, the internal performance of its components, and their dependencies were mostly ignored. Entering into the process of defining the workflow with a high-level approach unveiled several quite interesting aspects related to the modeling (clarification of the various steps involved, and high-level, implementation-agnostic perspective on each operator), modularity (replacement of a materialization but an alternative, study of dependencies between operators, optimal associations), and performance (individual profiling, and perspective of achieving an optimal allocation of resources based on the in/out throughput and internal performance of each materialization).

References

[1] ASAP. Continuous Query Prototype. ASAP Deliverable 8.3, 2016.

[2] K. Crammer, O. Dekel, J. Keshat, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, pages 551–585, 2006.

FP7 Project ASAP

Adaptable Scalable Analytics Platform

End of ASAP D8.4 Results of the performance and scalability study

WP 8 – Applications: Web Content Analytics

Nature: Report

Dissemination: Public