
**Longitudinal Analytics of Web Archive data**

*European CommissionSeventh Framework Programme*
*Call: FP7-ICT-2009-5, Activity: ICT-2009.1.6*
*Contract No: 258105*

# D4.1
# Exploratory Report on Distributed Web Analytics Technologies
# Version 1.0

| | |
|---|---|
| Editor: | MPG |
| Work Package: | WP4 |
| Status: | Final |
| Date: | 13.9.2011 |
| Dissemination Level: | Public |

## Project Overview

**Project Name:** LAWA – Longitudinal Analytics of Web Archive data

**Call Identifier:** FP7-ICT-2009-5

**Activity Code:** ICT-2009.1.6

**Contract No:** 258105

## Partners:

1. Coordinator: Max-Planck-Institut für Informatik (MPG), Germany
2. Hebrew University of Jerusalem (HUJI), Israel
3. Internet Memory Foundation[formerly European Archive] (IM), Netherlands
4. Hungarian Academy of Sciences (MTA-SZTAKI), Hungary
5. Hanzo Archives Limited (HANZO), United Kingdom
6. University of Patras (UP), Greece

## Document Control

**Title:** D4.1 – Exploratory Report on Distributed Web Analytics Technologies

**Author/Editor:** AvishekAnand (MPG), Klaus Berberich (MPG), Nikos Ntarmos (UP), Marc Spaniol (MPG), Spyros Zoupanos (MPG) and Gerhard Weikum (MPG)

## Document History

| Version | Date | Author/editor | Description/comments |
|---|---|---|---|
| V0.1 | 21.7.2011 | Marc Spaniol | Draft created |
| V0.2 | 26.8.2011 | All | First complete version |
| V0.3 | 6.9.2011 | Nikos Ntarmos | Feedback on initial version |
| V1.0 | 13.9.2011 | Marc Spaniol | Final Version |
| | | | |
| | | | |

# Table of content

# 1 Introduction

This document presents our research on distributed Web analytics technologies. We describe analytic tasks that are relevant for LAWA. In particular, we discuss typical tasks as building blocks towards added value within LAWA's Virtual Web Observatory. We present related research dealing with Hadoop-centered projects. Here, we introduce parallel dataflow systems inspired by the MapReduce [DeGh04, Hadoop] paradigm, solutions which mix MapReduce with DBMS and cloud computing platforms. Based on these considerations, we then present the LAWA architecture. We discuss the characteristics of LAWA's Web archive data, the storage model, and computation as well execution issues. Within our architecture indexing is highly relevant in order to dynamically derive ad-hoc subsets of a large document collection. To this end, we introduce several indexes that can be employed for filtering, ranging from simple meta-data filters up to keyword-supporting temporal filters. Finally, we discuss scalability issues. We explain how data partitioning and algorithmic patterns influence system performance and which steps should be taken to make LAWA's analytics technologies scale to Web (archive) size.

# 2 Analytic Tasks

Big-data analytics on the live Web has been a hot topic for some time. However, the Web of the Past is an equally important topic for both academics and real-life applications. Academically, longitudinal data analytics is even more challenging and has not received due attention. The sheer size and content of such Web archives lends itself to wide applicability for analysts in a great number of different domains.

These archives host a wealth of information, providing a gold mine for sociological, political, business, and media analysts. For example, one could track and analyze public statements made by representatives of companies such as Google, characterizing the evolution of patterns in their attitude towards energy efficiency. Another example could be tracking, over a long time horizon, a politician's public appearances: which cities she/he has visited, which other politicians or business leaders she/he has met, and so on. Analyses of this kind could also be carried out on large news archives, but this can be seen as variant of Web archive analytics; moreover, the Web (and especially the recent Web 2.0) has a wider variety of coverage, potentially leading to the discovery of more interesting patterns and trends.

## *2.1 Typical Analytic Tasks*

Web archives contain timestamped versions of Web sites over a long-term time horizon. This longitudinal dimension opens up great opportunities for analysts. For example, one

could compare the notions of “online friends” and “social networks" as of today versus five or ten years back. Similar examples relevant for a business analyst or technology journalist could be about “tablet PC” or “online music”. This requires finding all Web pages from certain eras that contain these and/or other related phrases, disambiguate the entities contained and, finally, group and/or filter them accordingly. Going beyond the term-level statistics that were traditionally used, statistics over n-grams (or phrases) help expose the contextual information in a better way. They can also capture the names of entities, quotes/slogans, sentiment expressions, etc., in a natural way. In the following we will introduce several tasks that are linked with the before mentioned analytics scenario.

### 2.1.1 Sequence mining, with time, entity, or phrase filtering

Techniques for analyzing, mining and automated extraction of knowledge contained in the raw text have become quite sophisticated and are widely used due to the promise of actionable insights that can be derived. In many text mining tasks, computing n-gram frequency statistics is used as a fundamental preprocessing step. Given a large collection D of text documents, a maximum phrase length $\lambda$ and a frequency threshold $\tau$, the goal of n-gram frequency statistics computation is to obtain all text phrases that occur more than $\tau$-times in D and have length less than or equal to $\lambda$.

Frequencies of variable-length word-level n-grams are an important building block for various applications including language models, employed for retrieval and translation purposes, as well as text analytics. For example, learning with variable or unrestricted length character n-grams could better capture spelling mistakes, spam characteristics (punctuation, etc.) or sub-words (implicit stemming) and phrasal features. Other benefits of using variable-length character n-grams could come from the more robust statistics captured by substrings of the text. However, computing n-gram frequencies is expensive for large-scale datasets.

In the most general setting, one is interested in determining these frequencies holistically, i.e., for all n-grams occurring in the document collection. One can restrict this and consider only n-grams that (i) occur at least $\tau$ times in the collection, (ii) occur at least in $\tau$ distinct documents, or (iii) are no longer than a fixed length. Note that, for ease of explanation, we use the term n-gram to refer to variable-length sequences of words (up to length n); whenever we refer to sequences of a specific length, this length is explicitly stated or a different identifier is used.

### 2.1.2 Disambiguate all mentions in a corpus

Detecting named entities in Web pages and thus lifting the entire analytics to a semantic rather than keywords level is a grand challenge already for standard text mining. The difficulties arise from name ambiguities, thus requiring a disambiguation mapping of

mentions (noun phrases in the text that can denote one or more entities) onto entities. For example, the mention “Bill Clinton” can be the former US president William Jefferson Clinton, but Wikipedia alone knows five other William Clintons. If the text says only “Clinton”, the number of choices increases, and phrases like “the US president” or “the president” have a wide variety of potential denotations. For established kinds of data cleaning and text mining, methods for entity resolution (aka. record linkage) have made reasonable progress (e.g. by using statistical learning for collective labelling), and could handle a good fraction of such cases. In the Web archive case, some additional aspects are assets while others pose major obstacles. The timestamp of an archived Web page can help to narrow down the disambiguation candidates for phrases like “the US president”. Similarly, the connection with previous and successive versions of the same page can help to identify changes at specific timepoints, which may in turn be cues for entity resolution. Cases where the temporal dimension introduces new complexity are when names of entities have changed over time. Examples are people's name changes after getting married or divorced (or simply out of some mood), or organizations that undergo restructuring in their identities.

### 2.1.3 Group by time (range), entity (class), or phrase (pattern)

In order to support entity level analytics of Web archives we need to identify mentions (candidates for being a named entity) from free text, disambiguate them and extract interesting relations among them. The input might be an arbitrary (textual) Web document $d_j$ for which the following (meta) data needs to be extracted:

- Entities: Mapping of entity mentions (surface strings) onto canonical entities, based on the YAGO ontology [SKWe07,SKWe08]
- Time points / intervals: Determination of time points and intervals mentioned in temporal expressions
- Relations of interest: Identification of relations between the entities contained in the document and their validity time point or interval (if applicable)

Therefore the following (meta) data for tokens $t_n$ of a text document should be extracted

- $[t_i,t_{i+1},...|entity_k]$, where $entity_k$ is a unique YAGO identifier such as `yago:Albert_Einstein`
- $[t_{l+1},t_{l+1},...|(dd\text{-}mm\text{-}yyyy:dd\text{-}mm\text{-}yyyy)]$, where $(dd\text{-}mm\text{-}yyyy:dd\text{-}mm\text{-}yyyy)$ represents a canonical time point / interval for a temporal expression
- $[rel_x(entity_{k'}, entity_{k''})@(dd\text{-}mm\text{-}yyyy:dd\text{-}mm\text{-}yyyy)]$ where $rel_x$ specifies a relation between $entity_{k'}$ and $entity_{k''}$ valid at $(dd\text{-}mm\text{-}yyyy:dd\text{-}mm\text{-}yyyy)$

In addition to free text queries (see Section 2.5), the above (meta) data can then be exploited to support analytics of Web archives based on the semantically enriched documents. This will allow the following search and analysis functionality:

- Query for entity in text:
  Given an entity, return documents where the entity is mentioned
- Query for entity in text with time filter:
  Given an entity and a time point (or interval), return documents where the entity is mentioned and there is an overlap between the given time point (or interval) with creation date, archival date or a temporal expression contained in the document
- Query for related entities (with or without time filter):
  Given an entity $e$ (optionally with time filter), return all entities that are mentioned together with $e$ in some document
- Query by related entities:
  Given an entity $e$, and additional constraints such as type(s), relation(s) and/or geographic location(s), group all occurrences of $e$ with other entities by these related entities and return groups with their aggregated frequencies
- Query for entity timelines (with or without additional constraints):
  Given an entity $e$ and a time interval, return related entities (with or without additional constraints) along the temporal dimension (monthly or yearly granularity) based on their aggregated frequencies

### 2.1.4 Group by similar documents

Grouping by similar documents is an efficient method for near-duplicate elimination. The issue of near duplicate elimination in Web archives is an important issue for several reasons. First, it helps to sanitize term statistics by leveraging the impact of multiple copies from the same (or almost the same) Web content. Second, it helps to identify Web spam based on identifying replicas of high-quality contents within spam farms. Last but not least, it might also help to reduce the overall size of a Web archive in compacting it by dropping archived versions that contain insignificant changes only.

In order to compare the similarity of documents, usually a vector space model on the contained terms is being applied. Typically terms are single words, keywords, or even phrases. However, such an approach is computationally expensive given the fact that (ultimately) each and every document needs to be compared with each other. Hence, more efficient pruning strategies need to be investigated. In addition, the before mentioned approach is not capable of identifying semantically related documents. For instance, a document containing the terms “Barack Obama” and “Big Apple” might be (semantically) very similar with another one mentioning “The US President” and “NYC”.

To this end, we need to investigate how to incorporate entity-level analytics into the computation of document similarity.

## *2.2 Specific Analytic Tasks of High Relevance within LAWA's Virtual Web Observatory*

In the following, we describe specific analytic tasks that are of high relevance for longitudinal Web archive analytics. The first example describes large scale named entity disambiguation, which allows raising surface strings on the entity level. The next example introduces knowledge harvesting about fairly ephemeral (temporal) facts. The last one deals with variations of frequent n-gram mining. All of them are not to be considered in isolation, but as part of a comprehensive analytics machinery within LAWA's Virtual Web Observatory.

### 2.2.1 Massive-scale coherence-based disambiguation of named-entity mentions in large archives

Entity detection, disambiguation and tracking is currently under development as a system called AIDA (Accurate Online Disambiguation of Named Entities) [HYB*11,YHB*11]. AIDA harnesses context from knowledge bases and employs a new form of coherence graph. It unifies prior approaches into a comprehensive framework that combines three measures: the prior probability of an entity being mentioned, the similarity between the contexts of a mention and a candidate entity, as well as the coherence among candidate entities for all mentions together. The method builds a weighted graph of mentions and candidate entities, and uses a greedy algorithm for computing a "heavy" subgraph that approximates the best overall mention-entity mapping. Experiments show that the new method significantly outperforms prior methods in terms of accuracy, with robust behavior across a variety of inputs. However, in its current implementation, AIDA is centralized and we are now investigating how to parallelize (particularly hadoopify) the needed computations. Since the overall process is computationally expensive, there is a great potential for a Hadoop-based approach, especially, to scale entity tracking, resolution and disambiguation up to Web (archive) size.

### 2.2.2 Large-scale temporal fact harvesting based on PRAVDA

Large-scale temporal fact harvesting from Web (archive) data is currently being developed within a system called PRAVDA (label Propagated fAct extraction on Very large DAta) [WYQ*11]. Temporal facts distil the evolving knowledge over time, such as winners of sports competitions, and occasionally even CEOs and spouses. This kind of temporal knowledge is an indispensable asset to support many information needs by advanced users. For example, consider the following example questions: "Who were

the team mates of Diego Maradona during the 1990 FIFA World Cup?" "When did Madonna get married, when did she get divorced?" None of these questions are supported by existing knowledge bases. The problem that we therefore tackle is to automatically distil, from news articles and biography-style texts such as Wikipedia, temporal facts for a given set of relations. By this we mean instances of the relations with additional time annotations that denote the validity point or span of a relational fact. For example, for the *winsAward* relation between people and awards, we want to augment facts with the time points of the respective events; and for the *worksForClub* relation between athletes and sports clubs, we would add the timespan during which the fact holds. This can be seen as a specific task of extracting ternary relations, which is much harder than the usual information extraction issues considered in prior work.

PRAVDA gathers fact candidates and distils facts with their temporal extent based on a new form of label propagation (LP). This is a family of graph-based semi-supervised learning methods, applied to (in our setting) a similarity graph of fact candidates and textual patterns. LP algorithms start with a small number of manually labelled seeds, correct facts in our case, and spread labels to neighbors based on a graph regularized objective function which we aim to minimize. The outcome is an assignment of labels to nodes which can be interpreted as a per-node probability distribution over labels. In our scenario, the labels denote relations to which the fact in a correspondingly labelled node belongs.

In its current implementation we adopt the specific algorithm of [TaCr09], coined MAD (Modified Adsorption), with an objective function that combines the quadratic loss between initial labels (from seeds) and estimated labels of vertices with a data-induced graph regularizer and an L2 regularizer. The graph regularizer is also known as the un-normalized graph Laplacian, which penalizes changes of labels between vertices that are close. We develop substantial extensions, and show how to judiciously construct a suitable graph structure and objective function. Notably, we consider inclusion constraints between different relation labels for the same node in the graph. For example, we may exploit that a relation like *joinsClub* (with time points) is a sub-relation of *worksForClub* (with time spans).

Currently, PRAVDA is centralized and we are now investigating how to parallelize (particularly hadoopify) the needed computations. Since the overall approach incorporates large scale data analytics and a computationally expensive label propagation algorithm, there is a great potential for a Hadoop-based approach.

### 2.2.3 Frequent n-in-m-grams mining

Another analytic task of interest in our project is the efficient distributed computation of n-gram frequencies or relaxed notions thereof. Statistics about occurrence frequencies of n-grams (i.e., consecutive sequences of n words) have applications, e.g., in natural language processing and information retrieval. Higher-order language models [MRS09],

as a concrete application in information retrieval, make use of those statistics to determine the relevance of a document to a query, taking into account the order of words. In addition, those statistics can provide insights, e.g., into the evolution of language or into how the popularity of specific named entities changed over time, as demonstrated by recent work [LMJ07, MRS09] based on n-gram frequencies computed on millions of digitized books.

We demonstrate the usefulness of n-gram frequencies with the following concrete use case: For an article about recent political developments in Russia, a journalist wants to get an idea about how the perception of Vladimir Putin by western media has changed during the past decade. To this end, the journalist specifies that she is interested in western media (e.g., major newspaper such as The New York Times, Guardian and The Independent) and documents published since 2001 that talk about Russian politics. Our system then computes n-gram frequencies per year, i.e., grouping documents according to their year of publication. When inspecting the frequencies of n-grams that mention Putin, our journalist discovers that n-grams such as [the, democratic, leader, vladimir, putin] and [elected, president, vladimir, putin] have become less frequent during the past decade, whereas the frequency of n-grams such as [strong-armed, russian, leader] and [russian, ruler, putin] have increased.

The notion of n-grams used in existing work requires that the n words occur consecutively. In our work, we consider two relaxed notions of n-grams that allow for controllable gaps between words. The motivation behind allowing for small gaps between words is to also discover sequences of words that would normally not be discovered, since they occur in a large number of slight variations that individually are not frequent. As a concrete example, n-grams such as [vladimir, putin, met, with, french, president] and [vladimir, v., putin, met, the, german, president] may not be frequent individually. When forbidding gaps between words, the potentially insightful word sequence [vladimir, putin, met, president] is not discovered and reported. Our first relaxation, coined (n-in-m)-grams, demands that the n words occur in order within a window of width m. Our second relaxation, coined (n,g)-grams, requires the n words to occur in order but allows for gaps of at most g other words between them. In Figure 1, we illustrate our relaxed notions of n-grams by showing all (3,1)-grams and (3-in-5)-grams for the example sentence “vladimir v. putin was appointed prime minister”.

| **Example sentence:** “vladimir v. putin was appointed prime minister” | |
|---|---|
| **(3,1)-Grams:** | **(3-in-5)-Grams:** |
| [vladimir, v., putin] | [vladimir, v., putin] |
| [vladimir, v., was] | [vladimir, v., was] |
| [vladimir, putin, was] | [vladimir, v., appointed] |
| [vladimir, putin, appointed] | [vladimir, putin, was] |

| | |
|---|---|
| [v., putin, was] | [vladimir, putin, appointed] |
| [v., putin, appointed] | [vladimir, was, appointed] |
| [v., was, appointed] | [v., putin, was] |
| [v., was, prime] | [v., putin, appointed] |
| [putin, was, appointed] | [v., putin, prime] |
| [putin, was, prime] | [v., was, appointed] |
| [putin, appointed, prime] | [v., was, prime] |
| [putin, appointed, minister] | [v., appointed, prime] |
| [was, appointed, prime] | [putin, was, appointed] |
| [was, appointed, minister] | [putin, was, prime] |
| [was, prime, minister] | [putin, was, minister] |
| [appointed, prime, minister] | [putin, appointed, prime] |
| | [putin, appointed, minister] |
| | [putin, prime, minister] |
| | [was, appointed, prime] |
| | [was, appointed, minister] |
| | [was, prime, minister] |
| | [appointed, prime, minister] |

*Figure 1: Relaxed notions of n-grams illustrated*

Existing work typically computes n-gram frequencies on a document collection in its entirety, assumes the document collection to be static, and hence treats it as a one-time computation. However, as demonstrated by the above use case, one would often be interested in looking only at a well-defined subset of the document collection specified in an ad-hoc manner (e.g., documents published by western media since 2001).

Moreover, often only a small fraction of n-grams is of interest and has the potential to provide insights. We therefore allow the considered n-grams to be restricted by means of various constraints. For example, one may only be interested in relaxed n-grams that contain certain words (e.g., adjectives expressing sentiments) or those that include a reference to a specific entity (e.g., The European Parliament). An obvious but naive strategy to handle such constraints is to first compute frequencies for all relaxed n-grams and then filter based on the constraints. We plan to investigate how such constraints can be pushed into the computation, along the lines of work on constrained itemset mining [GRSh02], and thus be already considered early on.

# 3 Related Work

In this section we present Hadoop-centered projects and related projects like HDFS, HBase, Hive and Pig. Furthermore, we discuss current research on parallel dataflow systems inspired by the MapReduce paradigm, solutions which try to mix the MapReduce model with DBMS, and two of the most well known cloud computing platforms.

## *3.1 HDFS*

Hadoop comes with a file system called HDFS, which stands for Hadoop Distributed File System [SKR*10]. This is a network based file system, which manages storage across a network of machines. The main components of HDFS are analyzed in the sequel.

*NameNode*: The *NameNode* is responsible for the file and directory hierarchy of the file system represented as *inodes*. The file contents are split into large blocks (typically 128 megabytes) and each block is replicated at multiple *DataNodes*. When writing data, the NameNode nominates three DataNodes to host the block replicas. HDFS keeps the entire namespace in RAM. The inode data and the list of blocks belonging to each file comprise the image. The modification log of the image is called *journal*. On restart the NameNode uses the *image* and the *journal* to reach its final state. A secondary role of the NameNode is to work either as CheckpointNode or BackupNode. NameNode is also responsible for the replication management.

*CheckpointNode/BackupNode*: The *CheckpointNode* periodically combines the existing checkpoint and journal to create a new checkpoint and an empty journal. The *BackupNode* is also capable of creating periodic checkpoints but in addition it maintains an in-memory up-to-date image of the file system namespace that is always synchronized with the state of the NameNode.

*DataNode*: During start-up, each DataNode connects to the NameNode and performs a handshake to verify the namespace ID and the software version. In the sequel the DataNode registers with the NameNode, which makes it recognisable even with different IP address or port. The DataNode identifies block replicas in its possession by sending a block report to the NameNode. During normal operation the latter receives *heartbeats* from the DataNodes showing that the nodes are operating and that the replicas that they host are available. The NameNode replies to the heartbeats to give instructions to the NameNodes.

*HDFS Client* is used for user application access to the file system and collaborating with the NameNode and the DataNodes identifying the right blocks to be read or written.

**HDFS and other distributed systems** HDFS stores file system metadata and application data separately. The NameNode is responsible for the management of the

metadata. The use of a metadata-dedicated server is also found in other distributed file systems like PVFS [CLR*00, TPG08], Lustre [LFS], GFS [GGL03, KuQu09].

Unlike Lustre and PVFS, the DataNodes in HDFS do not use protection mechanisms as RAID to make the data durable. Instead, like GFS, the file content can be replicated to multiple DataNodes for reliability. This results, apart from durability, in bigger throughput and better data locality.

Distributed implementations of the namespace can be encountered in several distributed systems. Ceph[WBM*06] has a cluster of namespace servers (MDS) and uses a dynamic subtree-partitioning algorithm to map the namespace tree to MDSs evenly. GFS is also evolving into a distributed namespace implementation and Lustre is evolving towards a clustered namespace where the main idea is to stripe a directory over multiple metadata servers, using a hash function, each of them containing a disjoint portion of the namespace.

## *3.2 HBase*

HBase [HBase] is a distributed column-oriented database built on top of HDFS or Amazon's simple storage service (S3) [AS3] and based on the principles of Google's Bigtable [CDG*06]. HBase is in essence a Hadoop application, providing real-time read/write random-access to very large datasets.

Most of the distributed implementations of RDBMS have a difficulty to scale to many machines, to be installed and maintained or there is a severe compromise to the RDBMS feature set.

HBase attacks the scaling problem from the opposite direction. It is built from the ground-up to scale linearly by adding nodes. HBase is not relational and does not support SQL, but it is able to outperform relational DBMS, given a proper problem space, by hosting large, sparsely populated tables on clusters made from commodity hardware.

Similar systems to HBase, also based on BigTable, are Hypertable [Hypertable] and Cassandra [Cassandra]. The latter was open sourced by Facebook.

## *3.3 Hive*

Hive [Hive, TSJ*09, TSJ*10] is an open-source data warehousing solution built on top of Hadoop. Hive supports queries expressed in an SQL-like declarative language – HiveQL – which are compiled into MapReduce jobs that are executed using Hadoop. In addition, HiveQL enables users to plug in custom MapReduce scripts into queries. The language includes a type system with support for tables containing primitive types, collections like arrays and maps, and nested compositions of the above. The underlying IO libraries can be extended to query data in custom formats. Hive also includes a

system catalog - Metastore – that contains schemas and statistics, which are useful in data exploration, query optimization and query compilation.

**Other large data processing systems** SCOPE [CJL*08] is an SQL-like language on top of Microsoft's proprietary Cosmos MapReduce and distributed file system. Pig [Pig] allows users to write declarative scripts to process data. Hive is different from these systems since it provides a system catalog that persists metadata about tables within the system. This allows Hive to function as a traditional warehouse, which can interface with standard reporting tools. HadoopDB [APA*09] reuses most of Hive's system, except it uses traditional database instances in each of the nodes to store data instead of using a distributed file system.

## *3.4 Pig*

Pig [Pig] is a platform to analyze large data sets and is consisted of a language used to express data flows, named PigLatin, coupled with an execution environment to run Pig programs. PigLatin is a textual language permitting the user to easily program, understand and maintain complex tasks comprised of multiple interrelated data transformations. Moreover, because of the way tasks are encoded, the system can automatically optimize them, letting the user to focus more on the semantics than efficiency. The users can also create their own functions for special-purpose processing. These properties make the produced programs highly parallelizable, enabling them to handle large datasets.

## *3.5 Parallel data flow systems*

In this Section we present three parallel data flow systems inspired by the MapReduce model.

### 3.5.1 Microsoft's Dryad

Dryad [Dryad, IBY*07] is a massively parallel execution engine for data flows, which is designed to scale from multi core single computers to large clusters. A Dryad application combines computational "vertices" with communication "channels" to form a directed acyclic data flow graph (DAG). Vertices run arbitrary user code and edges represent communication channels between vertices. The vertices are simple usually sequential programs provided by the application developer. Dryad runs the application by executing the vertices of this graph on a set of available computers, communicating as appropriate through files, TCP pipes, and shared-memory FIFOs.

Dryad is responsible for the vertex scheduling on computers and CPUs and where multiple vertices may run simultaneously on multiple CPU cores of one computer. The application can discover the size and placement of data at run time, and modify the graph as the computation progresses to make efficient use of the available resources.

The application can also handle and recover from computer failures. On top of Dryad high-level systems like Dryad/LINQ [YIF*08] and SCOPE [CJL*08] have been built.

**Dryad and MapReduce** The primary goal of Dryad was to support large-scale data mining over clusters of thousands of computers and it shares many similarities with Google's MapReduce, which addresses a similar domain. The biggest difference of these two systems is that in Dryad the user writes arbitrary communication DAGs, while in MapReduce he/she is restricted by the sequence of map/distribute/sort/reduce operations. On the other hand, writing well performing data flow graphs is significantly harder than writing MapReduce jobs.

### 3.5.2 ASTERIX

ASTERIX is a joint project of UC Irvine, UC Riverside and UC San Diego and its goal is to store, manage, index, query and process semi-structured data [Asterix, BBC*11]. The scalable information management system, which they build, operates on large, shared-nothing commodity computing clusters. One of the key features of the system is the support of short and long running queries over evolving data sets. Some of the basic components of ASTERIX is the ASTERIX data model (ADM), the ASTERIX Query Language (AQL) and Hyracks execution layer [BCG*11]. The latter allows users to express a computation as a DAG of data operators and connectors. Operators operate on partitions of input data and produce partitions of output data, while connectors repartition operators' outputs to make the newly produced partitions available at the consuming operators. As stated in [BCG*11] Dryad is a low level platform and Hyracks could have been built on top of it if it had been available under an open source license.

### 3.5.3 Stratosphere

Stratosphere System [Stratosphere] is an outcome of a common research project by TU Berlin, HU Berlin and HPI Potsdam. At the current state, the system consists of the PACT Programming Model [BEH*10], the Nephele Execution Engine [BEH*10, WaKe09] and a compiler/optimizer, which converts PACT programs to parallel data-flows for the Nephele Engine [BEH*10]. The programming model is centered around key/value pairs and Parallelization Contracts (PACTs). The latter are second-order functions that define properties on the input and output data of their associated first-order functions (or user functions). The system utilizes these properties to parallelize the execution of the user functions and to apply optimization rules. The resulting compiled program is forwarded to Nephele, the physical execution engine of the Stratosphere System. Nephele executes DAG-based data flow programs on dynamic compute clouds and is responsible for task scheduling. Moreover, it is responsible for the set-up of the required communication channels, the allocation of computing resource and it also provides fault-tolerance mechanisms.

The PACT programming model is very similar to MapReduce. However, there are some crucial differences. First, in PACT they have added additional functions that fit to problems, which are not naturally expressible as a map or a reduce function. Second, they separate the programming and execution model and they use a compiler to generate the parallel data flow. Third the PACT model preserves more semantic information than MapReduce that only keeps that a function is either a map or a reduce task.

## *3.6 Hybrid solutions – HadoopDB*

MapReduce is widely known and used for its scalability, fault tolerance and flexibility. On the other hand, parallel databases demonstrate good performance and efficiency. HadoopDB [APA*09] is a hybrid system, which tries to combine the advantages of these two different architectures. MapReduce is used as a communication layer above multiple nodes running single-node DBMS instances. The queries, expressed in SQL, are translated into MapReduce with a goal to maximize the work pushed to the DBMS instances. The authors have selected Hadoop [Hadoop] as their communication layer, PostgreSQL [PostgreSQL] as their database layer and Hive [Hive] as their translation layer. The code is released as open source.

## *3.7 Cloud services*

Amazon Web Services [AWS], Windows Azure Platform [Azure] and Rackspace [Rackspace] are some of the available cloud computing platforms. In addition, there are also software platforms for the implementation of private cloud computing on computer clusters like Eucalyptus [Eucalyptus]. In this Section we focus on the two most prominent cloud computing platforms, provided by Amazon and Microsoft.

### 3.7.1 Amazon Web Services

Amazon provides a cloud-computing platform comprised of a collection of remote computing services (web services). Two of the most known services, Amazon Simple Storage Service and Amazon EC2 are analyzed in the sequel.

**Amazon Simple Storage Service (S3)** [AS3] is cloud-based persistent storage, which permits the user to store arbitrary objects in buckets. The size of the objects can be at maximum 5 GB and the buckets exist in a flat namespace shared among all Amazon S3 users. The objects are identified within each bucket by a unique, user-assigned key. Buckets and objects can be created, listed, and retrieved using either a REST-style HTTP interface or a SOAP interface. Additionally, objects can be downloaded using the HTTP GET interface and the BitTorrent protocol [BitTorrent].

**Amazon EC2** [AEC2] is a virtual computing environment, allowing to use web service interfaces to launch instances with a variety of operating systems, load them with a custom application environment, manage the network's access permissions, and run an image using as many or few systems is needed. Amazon EC2 is called elastic because it allows the user to increase or decrease the number of reserved instances within minutes. The instances are completely controlled by the user, through the provided root shell access and web service API. Furthermore Amazon EC2 is flexible because the user can choose from a variety of instance types, operating systems and software packages. It is designed to work in conjunction with other Amazon services to provide a complete solution for computing, query processing and storage across a wide range of applications.

### 3.7.2 Azure Services Platform

Windows Azure Platform [Azure] is Microsoft's proposal to cloud computing allowing Microsoft's data centres to host and run applications. The cloud operating system used (Windows Azure) serves as a runtime for the applications and provides a set of services that allows development, management and hosting of applications. All Azure Services and applications built using them run on top of Windows Azure. Some of the main services of the platform are the following:

- *Windows Azure Compute* provides developers with an internet-scale hosting environment. It is built from one of the roles which are the Web role, the Worker role or the Virtual Machine role.
- *Windows Azure Storage* provides persistent and durable storage in the cloud through four storage services: Binary Large Object Service, Table Service, Queue Service and Windows Azure Drive.
- *Windows Azure Virtual Network* provides a simple and easy-to-manage mechanism to setup IP-based network connectivity between on-premises and Windows Azure resources. Moreover it allows customers to load-balance traffic to multiple hosted services.
- *Microsoft SQL Azure Database* is a highly available and scalable cloud database service built on SQL Server technologies.
- The *Service Bus* provides secure messaging and connectivity capabilities that enable building distributed and disconnected applications in the cloud, as well as hybrid applications across both on-premise and cloud resources.

The Windows Azure Platform provides an API built on REST, HTTP and XML that allows the developer to interact with the available services.

# 4 LAWA Analytics Architecture

This section outlines a suitable architecture for performing analyses like the ones described above. First, we describe the input and output data of these analyses and its characteristics. Following that, we detail on different alternatives to physically store and access our data, and conduct our analyses (i.e., perform computations on the data). Finally, orthogonal to storage and computation, we look into different ways to deploy the proposed architecture (e.g., on a local cluster vs. on the cluster of a cloud service-provider).

## *4.1 Data*

Our analyses are performed on archived web contents, predominantly textual contents represented in formats such as plain text, HTML or XML. From an abstract point of view, our input data thus consists of time-stamped document versions, each of which is composed of its actual (textual) content and meta-data. The timestamp associated with a document version reflects either the time when it was published (if this can be determined precisely, which is not always the case on the Web) or the time when our crawler harvested it. Document contents are regarded as character sequences at this point. Note that, in practice, these character sequences may have embedded structure (e.g., the document structure as reflect by HTML mark-up) and also already contain rich annotations. Examples of such annotations include part-of-speech tags (as the result of a linguistic analysis) or embedded RDFa triples [RDFa] (e.g., linking to the URIs of named entities mentioned in the document). The meta-data associated with time-stamped document versions include both information about how the document version was obtained (e.g., from which IP address it was fetched by a crawler) and information about its actual contents (e.g., information about authorship, language or format of the document).

The output of our analyses is less homogeneous and thus more difficult to characterize concisely. When performing named-entity disambiguation, for example, for every document version considered, the output consists of resolved entity mentions (e.g., represented as a region of the document content together with a URI identifying the resolved entity). Frequent relaxed n-gram mining, as our second example task, on the other hand, produces as output n-grams together with their respective frequency in the ad-hoc subset of the document collection.

## *4.2 Storage*

Having outlined the input and output data of our analyses, the next question to address is how it can be physically stored. The options here, as described in detail in Section 3, include distributed file systems (e.g., HDFS and S3) and distributed key-value stores

(e.g., HBase, Cassandra and Hypertable). To recall the benefits and drawbacks: the former provide efficient sequential access to the data and do not impose a schema on it; the latter support efficient random accesses based on a key (e.g., a URL) and allow structuring the data in a columnar format.

Given our requirement, motivated in Section 2, that analyses can be performed on subsets of the document collection specified in an ad-hoc manner, it is clear that we need efficient random access to individual documents in our collection. Therefore, we employ a distributed key-value store, specifically HBase, to store the archived web contents that serve as an input to our analyses. While HBase provides the necessary functionality to store the time-stamped document versions described above and separate their components (i.e., contents and meta-data), it does not provide advanced indexing functionality (e.g., to support keyword queries on the document contents). Later, in Section 5, we discuss such advanced indexes and the challenges in implementing them on top of HBase. Figure 2 sketches how archived web contents can be stored in a single HBase table (further referred to as the "web table" similar to [CDG*06]).

| | **CONTENT** | **CONTENT_LENGTH** | **LANGUAGE** | **…** |
|---|---|---|---|---|
| org.x.www/contents/summary.html | <html>…@*t1* | 4,096@*t1* | EN@*t1* | |
| org.x.www/contents/index.html | <html>…@*t4* | 10,223@*t4* | EN@*t4* | |

*Figure 2: Example of Web content storage for LAWA in HBase*

The row key, as can be seen from the figure, is derived from the URL of the archived web contents in such a way that contents from the same top-level domain or host are clustered together (e.g., the first row in the table corresponds to the URL http://www.x.org/contents/summary.html). The three example columns in Figure 2 capture the document content, content length and language. Note that, in practice, the table contains many more columns and can be extended when additional columns are needed (e.g., to store analytic results such as annotations of the document content). Moreover, every cell in the table is time-stamped (as indicated by the appended timestamps in Figure 2).

For the output of our analyses (e.g., entity annotations of relaxed n-gram frequencies), the appropriate way to store it depends on its follow-up use. Entity annotations, on the one hand, are likely to be used in follow-up analyses (e.g., when analysing the popularity of entities over time as reflected by their number of mentions). Thus, it makes sense to store them alongside the raw data, i.e., as an extension of the time-stamped document version. In practice, this can be achieved by adding a column to the web table in HBase that stores the annotations (i.e., pairs of document region and URI of referenced entity) relative to the original document content or a version of the original document content with embedded annotations. Frequencies of relaxed n-grams, on the

other hand, are likely to be scouted manually for insights or used by other applications (e.g., a text-search engine) that are not immediately part of our architecture. Thus, it seems more appropriate to store them directly in the distributed file system (e.g., HDFS) using a flat or indexed (e.g., the MapFile provided as part of the Hadoop API) file format.

## *4.3 Computation*

How do we implement our analyses, so that they can run in a distributed environment and make efficient use of a cluster of machines? Again, the suitable tool depends on the analysis at hand. While we expect Hadoop/MapReduce to be suitable for most tasks, in some cases other tools may be more appropriate. For example, when the data is already structured and the task at hand is simple (e.g., performing a grouping followed by an aggregation), we may implement our analysis using the high-level data-flow language PigLatin (and let Pig compile and execute the analysis on a Hadoop cluster). Similarly, when performing many simple analyses that can be expressed using simple relational operators (i.e., selection, projection and join) on structure data, it can become beneficial to insert the data into the distributed data-warehouse Hive and implement analyses using its SQL-inspired query language HQL. At the other end of the spectrum, when performing extremely sophisticated analyses (e.g. entity disambiguation with subsequent clustering) that depend on algorithms requiring more elaborate coordination of nodes than provided by Hadoop/MapReduce, it can become rewarding to use other tools (e.g., the MPISQUARE message-passing interface developed by MPG) for their implementation.

## *4.4 Execution*

Choosing the right tools to store/access our data and implement our analysis is important. An equally important yet orthogonal question to address is that of execution, i.e., where the tools are physically deployed. Here, possibilities range from a local deployment (e.g., on a single machine or cluster housed by one of the partners), wide-area deployment (connecting machines across partners' sites) to cloud deployment on the machines of a service provider (e.g., Amazon or Rackspace). In what follows, we discuss these possibilities in more detail and outline when we deem them appropriate.

Local execution of analyses (i.e., on locally deployed tools such as an in-house Hadoop cluster) is sensible, e.g., when only local data is needed or while the implementation of the analysis is still being tested and finalized.

Wide-area execution on geographically distributed machines (e.g., hosted by the project partners) on a wide-area deployment of tools (e.g., Hadoop Kelvin [Kirk11]) is suitable, for instance, when required computing resources exceed a single partners' capacities. Hadoop Kelvin is a network monitoring system designed for the Hadoop Map-Reduce

framework. It monitors data (not control) traffic between Hadoop nodes and provides multiple ways to store, visualize and access the stored monitoring data. It is designed to be easily extensible, flexible and to operate with a minimal effect on the running time of Hadoop jobs.

Hadoop Kelvin consists of two main parts, a statistics server and a statistics client [Kirk11]. The statistics server is a program which runs on a single machine in the cluster (typically one of the master machines in the cluster if a single statistics server is present; alternatively a set of slave machines can be used if several such servers are required) and serves as a sink for all the traffic reports arriving from the cluster nodes. The server operates a set of user-configurable (via XML) data storers (which are write-only), data retrievers (which are read-only) and data manipulators (which provide read-and-write access) to which measurement data is stored and from which queries about past measurement data are completed.The statistics client allows a 3rd party program to access the data stored inside the data storers of the statistics Server and also to submit reports of its own. The protocol all Hadoop Kelvin traffic uses is HTTP. The monitoring data collected by Kelvin about the Hadoop cluster can then be subsequently visualized via a Web browser in order to better understand the cluster's execution performance.

Execution of analyses and deployment of tools in the cloud (e.g., on the machines provided by a service provider such as Amazon) is appropriate only in rare cases, for instance, when required computing resources are vast or when systematically evaluating the scalability of the developed methods through so-called scale-out experiments.

# 5 Indexing issues

When performing analyses like the ones describe above, one is often interested in considering only an ad-hoc subset of a large-scale document collection (e.g., a web archive in our setting), rather than considering the entire document collection. Such ad-hoc subsets of the document collection may be defined based on document meta-data (e.g., their URL, host, language or publication time) but also based on their content (e.g., requiring that they are relevant to a specific keyword query or mention a specific named entity). In our use case from Section 2, as an example, the ad-hoc document set was defined based on host (i.e., the host has to be one of www.guardian.co.uk, www.nyt.com or www.independent.co.uk), publication time (i.e., between 2001 and 2011) and content (i.e., relevant to Russian politics).

Having defined an ad-hoc subset of the document collection by means of meta-data filtering and/or content filtering (e.g., a keyword query), it is essential to efficiently retrieve documents therein to make them available as an input to the following stages of the analysis. To this end, we need to equip our architecture from Section 4 with indexes that support meta-data and/or content filters efficiently. In the following, we describe the

different types of required indexes and characterize the challenges that an implementation on top of a distributed key-value store (e.g., HBase in our architecture described in Section 4) poses.

## 5.1 Indexes for simple meta-data filters

One shortcoming of distributed key-value stores such as HBase and Cassandra is that they only provide efficient access based on the key (e.g., a combination of URL and timestamp in our setting), but do not support efficient look-ups based on other attribute values. Whereas such functionality is integral to relational database systems, it often has to be implemented from scratch when using a distributed key-value store (e.g., HBase in our case). Thus, if we want to support a simple meta-data constraint such as identifying and retrieving all documents that are written in French (i.e., identifying all rows in our document table that have the value “FR” in their LANGUAGE column) or contain less than 1,000 words (i.e., having a value between 0 and 1,000 in their “CONTENT_LENGTH” column), without indexes, we have to scan the entire table for relevant rows.

To support such simple meta-data constraints and circumvent the above shortcoming, a common solution is to build so-called secondary indexes. In essence, such a secondary index for a single or multiple columns (e.g., LANGUAGE) consists of an auxiliary HBase table that uses the observed values (e.g., “FR”) as its keys. The columns of a row for a specific value (e.g., “FR”) then keep track of the keys in the original table, where that value can be found in the indexed columns (e.g., LANGUAGE).

## 5.2 Indexes for temporal filters

Secondary indexes as described above are useful when dealing with simple meta-data constraints. However, for certain temporal constraints they are insufficient, as we explain in the following. Say, we are interested in an analysis that is based on a snapshot of the Web, to the extent covered by our web archive, as of a specific point in time (e.g., September 11$^{th}$, 2011) and thus need to retrieve all document versions that existed on the specified date. This corresponds to identifying all document versions whose associated valid-time interval (spanning from their time of publication until they were replaced by a newer version) includes the given point in time. This type of query is known as a stabbing query or time-point query and various main-memory index structures (e.g., interval trees and segment trees [BBO*00, Same06]) and disk-resident index structures (e.g., the TSB-Tree and Multi-Version B-Tree [SaTs99, Same06]) have been proposed to support it efficiently in centralized settings. Supporting this type of query efficiently in a distributed setting, for instance, by implementing one of the existing index structures on top of a distributed key-value store (e.g., HBase) is a challenging problem that we are addressing in the project.

Of these structures, segment trees are the ones that seem the best candidate thus far for our setting. For any given set of intervals, a corresponding segment tree is a tree data structure so that:

- Its leaf level consists of an ordered set of “elementary intervals”; that is, all intervals resulting from layout out all interval ends and taking all combinations of two subsequent ends.
- Internal nodes correspond to unions of (elementary) intervals below them.
- Each leaf/internal node stores (or is responsible for) all original intervals that span through its interval but not through the interval of its parent node.

In a centralized setting, such a tree can be built in O(N * *log*N) time and space and can answer to stabbing queries in O(logN + k) time, where N is the number of original intervals and k is the number of retrieved intervals. Several issues arise, though, when one tries to port this data structure to a Hadoop/HBase setting. First, one needs to decide on where and how to store the index data; all of the choices outlined in Sections 3 and 4.2 are viable solutions, offering different tradeoffs between performance, scalability, portability, and ease of development and deployment. Next, designing an efficient algorithm to scan through the vast amount of data in the distributed data store to build this index, is also in itself a formidable task. In this field, our focus is on Hadoop/MapReduce-based methods, so as to take advantage of the rest of the infrastructure; again, the same tradeoffs as earlier apply regarding single- vs multi-pass MapReduce approaches and the ability to use combiners or other Hadoop/MapReduce optimization primitives. Last, a very interesting aspect of this work from both the research and applied points of view, is the way this index will be used during query processing. An implementation that would plug the index data (and logic) directly in the query processing engine (e.g., tapping in the MapReduce execution engine), would surely result in higher throughput and lower latency; on the other hand, managing and accessing the index data like any other in the data store – in which case, making use of the index would require a nested query of sorts – would provide much better portability and (probably) scalability, but for lower performance gains (note, though, that even in this case preliminary results show a more than 10-fold decrease in query processing time, compared to the current state-of-art).

## *5.3 Indexes for content-based filters*

As motivated above, it is also important to allow ad-hoc subsets of the document collection to be defined based on content constraints, for instance, using keyword queries (e.g., to identify document versions relevant to Russian politics) or phrase queries (e.g., to identify document versions that mention a specific entity). The standard index structure to support such queries efficiently is the inverted index that has been studied intensively [ZoMo06] and is at the core of virtually all large-scale search

systems. The challenge that we have to address to support content-based querying is to build such an inverted index on the contents of our web archive. One possible solution is to index the contents of our web archive using a mature existing text-search engine such as Lucene and thus to add an additional component to the architecture described in Section 4. An alternative approach that we consider, similar to the other types of indexes described above, is to implement an inverted index on top of HBase as our distributed key-value store.

### *5.4 Indexes for keyword querying with temporal filters*

Thus far, we have assumed that meta-data constraints and content constraints are dealt with in isolation, i.e., using separate indexes and combining the results obtained from them. However, it is conceivable that certain combinations of constraints are so common that building a dedicated index to support these combinations efficiently makes sense. As a concrete case, one may often be interested in document versions that existed at a given point in time (e.g., September 11$^{th}$, 2001) and are relevant to a keyword query or mention a specific entity (e.g., Barack Obama), i.e., a combination of the content-based constraints and the temporal constraints described above. Such a combination of temporal constraints and content constraints, coined time-travel text search, has been looked at [BBT*07, ABB*11]. Existing work builds on the inverted index to implement this functionality, extending the inverted index by temporal information and applying compression as well as partitioning techniques to achieve quick response times. None of the existing approaches, however, targets a distributed setting like ours or makes use of Hadoop for efficient indexing. Therefore, an interesting open challenge is to adapt the existing approaches, so that they can be implemented on top of Hadoop and HBase and support combinations of temporal constraints and content constraints efficiently.

# 6 Scalability issues

Depending on the kind of algorithm we run in MapReduce, a proper data partitioning may help to increase performance. Data can be partitioned before the beginning of the computation but also during the MapReduce execution.

### *6.1 Data partitioning before starting the MR computations*

The input data may be correlated and this may affect the execution that will follow. For example, a naïve computation of the ten most probable words in every chapter of a book may require all the pages of one chapter to be processed by one mapper or reducer. Having not organized the pages in the disk in such a way will result in major reshuffling which should be avoided.

The partitioning can be done manually, using some external tools (e.g. using some specialized tools to partition a graph into neighbourhoods, in a graph related problem) or by defining a partitioning function and performing a MapReduce task only for this partitioning.

## *6.2 Data partitioning and load balancing during the MR execution*

By default, data is shuffled during the shuffle phase which takes place after the end of the map phase and before the beginning of the reduce phase. A hash function is responsible for the right partitioning but it can also be replaced by a user-defined partitioning function, which should respect the data properties (e.g. data locality) and should create partitions of the same size.

In the case of iterative algorithms with many map/reduce rounds, knowing the partitioning scheme permits the separation of the static data from the non-static data avoiding shuffling non-necessary information. Moreover, choosing an appropriate partitioning function may minimize the shuffled data across different computers and may permit the use of combiners. These cases will be analyzed in the sequel.

### 6.2.1 Separating static and non-static data

Reducing the data that the mappers output but also minimizing inter-peer shuffling can minimize the exchanged data in the shuffling phase. To reduce the outputted data from the mapper, a careful design of the algorithm is needed to output only the data that changes. Data remaining static across rounds can be stored outside the MapReduce flow. When following such an approach, it is very important to know the partitioning scheme in order to partition the static data in the same way. Then data from the same partition can be merged with the non-static data in the mapper or the reducer. Having decided to make such a separation, the next question that rises is where to store the static-data. We present some of the available options.

- *Memcached* [Memcached] is an in-memory key-value store for small chunks of arbitrary data. It can run on one computer or it can combine the memory from several computers. In the simple scenario, where there is one memcached instance per computer, it can be used as persistent memory across iterations where static-data or computer related data can be stored.
- A lightweight database can also be used to store such data. For example *Berkeley DB* [BerkeleyDB] is a key/value store that can be easily integrated to the available implementation since there are versions for Java and C with their corresponding API.
- Static data can be also stored (sorted) in the local disk of every computer. Then the right partition can be retrieved and combined with the sorted non-static data

that arrive from the MapReduce flow by performing a merge join. This technique has been followed in [LiSc10].

All the aforementioned techniques assume that the correct partition of static and non-static data is located at the same computer. Unfortunately, this cannot be guaranteed because the assignment of reducers to computers is decided by the Hadoop framework and is not controlled by the end-user. Even in the case where a partition of static data has to be fetched from a remote computer, following such techniques is better than incorporating static data in the MapReduce data flow [LiSc10].

In the case where static-data cannot be partitioned, solutions like Hadoop's DistributedCache can also be followed. This technique allows the user to define some files, which will be available through HDFS but will be stored in all available computers of the Hadoop cluster. In this case there is data redundancy but there will be a local copy on every computer accessible via HDFS.

### 6.2.2 Choosing the right partitioning

Choosing a suitable partitioning function for every problem is very important. For example in an iterative graph algorithm where vertex locality makes a big difference, it is crucial to implement a partitioning function, which will respect this locality and will cut the graph into sub graphs respecting vertex neighbourhoods. Imagine the case of an iterative graph algorithm where, on every round, the new value of every vertex is calculated and for this calculation the vertex' neighbours are needed. Figure 3: Sample of a simple MapReduce program shows a sketch of such an algorithm.

| map(id i, vertex v) | reduce (id i, vertex list lv) |
|---|---|
| **for every** n **in** v.neighbours | **find** v **from** lv |
| **emit**(n.id, v) | **calculate new value of** v **using** lv |
| **emit**(i, v) | **emit**(i, v) |

*Figure 3: Sample of a simple MapReduce program*

The map method will send a copy of vertex v to all its neighbours. Using the default hash function will result in major data shuffling across computers, which may be prohibitive in cases where large amounts of data are processed. Moreover, this major data shuffling will be repeated in every round of the algorithm.

Having a partitioning function, which respects the graph's neighbourhoods, will lead vertices of the same neighbourhood to the same reducer $R_1$. Let's say that for this round $R_1$ is scheduled to run at computer $C_1$. $R_1$ will initially try to store the results of its

computation to its local HDFS node, hosted by $C_1$, and if there is a problem (lack of space etc) other HDFS nodes will be tried. In the next round, the mapper of $C_1$ will first try to read the input files stored in the local HDFS node. At the end of the map phase, the data will be routed by the partitioning function again to $R_1$. It is not certain if the Hadoop framework, will assign $R_1$ again to $C_1$ but in an isolated environment with computers of the same characteristics, normally, there will not be many changes from one round to another. Achieving to run $R_1$ in $C_1$ most of the times means that minimum data exchange across peers is achieved, which leads to less network usage and a processing speed-up.

### 6.2.3 Using combiners

Partitioning the data properly can facilitate the use of combiners. A combiner can be roughly seen as a reduce function which runs on the map's output and at the mapper side. The purpose of the combiner is to perform as many reduce operations as possible at the mapper side and therefore to reduce the amount of data sent to the reducers. Since combiners are an optimization, the framework does not guaranty their usage. To overcome this obstacle, in-mapper combiners can be constructed as described in [LiDy10].

The main idea behind an in-mapper combiner is to construct a combiner inside the map function and therefore impose its execution. This will result to reduced network usage. Going back to the example of Section 6.2.2, the choice of a well selected partitioning method and also the use of in-mapper combiners will shift most of the computation at the mapper side. The calculations for the remaining few vertices, which would not have all their neighbours available at the mapper side, will be performed in the reducers or half in the in-mapper combiners and half in the reducers.

## *6.3 Algorithmic patterns*

We can divide the algorithms that we have studied so far into three main categories.

- The simplest case of MapReduce algorithms – at least seemingly – is the one, which contains algorithms needing one map/reduce round. The naïve computation of the appearances of each different word in a set of documents is an example of such algorithm.
- Some algorithms may need more than one rounds but the number of rounds is fixed or can be easily estimated. Consider, as a concrete example, the computation of PageRank scores [BrPa98, LaMe04] on a large-scale graph. The power-iteration method, as the standard approach, performs a series of matrix-vector multiplications, each of which can be implemented as a separate MapReduce job. The number of iterations needed until convergence can be

bounded solely based on PageRank's random-jump probability [LaMe04] and is typically at the order of a few dozens.

- There are also algorithms where we cannot estimate before the actual execution the number of rounds that will be needed. As a concrete example, consider an implementation of the a-priori algorithm [AgSr94] for frequent itemset mining in MapReduce. The algorithm determines frequent itemsets of increasing cardinality iteratively, using output from the previous iteration for filtering in the current iteration. Again, a natural way to implement the algorithm using MapReduce is to run one MapReduce job per iteration of the algorithm. The number of iterations needed depends on the cardinality of the largest frequent itemset and can thus not be determined or non-trivially bounded upfront.

# 7 Conclusion

In this report we have discussed the analytic tasks relevant to LAWA. We have presented relevant technologies and outlined our architectural design. In addition, we have identified issues we intend to investigate related to indexing and scalability. These issues have great relevance for the overall LAWA setup with regard to ad-hoc document selection and scaling analytics up to Web (archive) size.

# References

[ABB*11] A. Anand, S. Bedathur, K. Berberich, R. Schenkel: *Temporal Index Sharding for Space-Time Efficiency in Archive Search*, SIGIR 2011, July 2011.

[AEC2] Amazon Elastic Compute Cloud (Amazon EC2): http://aws.amazon.com/ec2/. [last visited: 24.08.2011]

[AgSr94] R. Agrawal and R. Srikanta: Fast Algorithms for Mining Association Rules, VLDB 1994

[APA*09] A. Abouzeid, K. B. Pawlikowski, D. Abadi, A. Silberschatz, A. Rasin: HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. In: Proc. VLDB Endow., Vol. 2, August 2009, pp. 922-933.

[AS3] Amazon Simple Storage Service (Amazon S3): http://aws.amazon.com/s3/. [last visited: 24.08.2011]

[Asterix] Asterix: http://asterix.ics.uci.edu/. [last visited: 24.08.2011]

[AWS] Amazon Web Services: http://aws.amazon.com/. [last visited: 24.08.2011]

[Azure] Windows Azure: http://www.microsoft.com/windowsazure/. [last visited: 24.08.2011]

[BBC*11] A. Behm, V. R. Borkar, M. J. Carey, R. Grover, C. Li, N. Onose, R. Vernica, A. Deutsch, Y. Papakonstantinou, V. J. Tsotras: ASTERIX: towards a scalable, semistructured data platform for evolving-world models, Distributed and Parallel Databases, Volume 29, Number 3, 185-216.

[BBD*10] S. Bedathur, K. Berberich, J. Dittrich, N. Mamoulis, G. Weikum: Interesting-Phrase Mining for Ad-hoc Text Analytics, PVLDB 3(1), pp. 1348-1357, 2010.

[BBO*00] M. de Berg, M. van Kreveld, M. Overmars and O, Schwarzkopf: Computational Geometry: Algorithms and Applications. Publisher: Springer-Verlag 2000.

[BBT*07] K. Berberich, S. Bedathur, T. Neumann, G. Weikum: *A Time Machine for Text Search*, SIGIR 2007, July 2007.

[BCG*11] V. R. Borkar, M. J. Carey, R. Grover, N. Onose, R. Vernica: Hyracks: A flexible and extensible foundation for data-intensive computing, ICDE 2011.

[BerkeleyDB] Oracle Berkeley DB:http://www.oracle.com/technetwork/database/berkeleydb. [last visited: 24.08.2011]

[BitTorrent] BitTorrent: http://www.bittorrent.com/. [last visited: 24.08.2011]

[BEH*10] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, D. Warneke: Nephele/PACTs: A Programming Model and Execution Framework for Web-Scale Analytical Processing, SoCC 2010 pp. 119–130.

[BrPa98] S. Brin and L. Page: The anatomy of a large-scale hypertextual Web search engine, Computer Networks 30(1), pp. 107-117, 1998

[Cassandra] Cassandra: http://cassandra.apache.org/. [last visited: 24.08.2011]

[CDG*06] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber: Bigtable: A Distributed Storage System for Structured Data. In: OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.

[CJL*08] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, J. Zhou: SCOPE: easy and efficient parallel processing of massive data sets, Proc. VLDB Endow. 1, 2 (August 2008), 1265-1276.

[CLR*00] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur.“PVFS: A parallel file system for Linux clusters,” in Proc. of 4th Annual Linux Showcase and Conference, 2000, pp. 317–327.

[DeGh04] J. Dean and S. Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004, pages 137–150.

[Dryad] Microsoft's Dryad: http://research.microsoft.com/en-us/projects/dryad/. [last visited: 24.08.2011]

[Eucalyptus] Eucalyptus: http://www.eucalyptus.com/. [last visited: 24.08.2011]

[GGL03] S. Ghemawat, H. Gobioff, S. Leung. "The Google file system," In Proc. of ACM Symposium on Operating Systems Principles, Lake George, NY, Oct 2003, pp 29–43.

[GRSh02] M. Garofalakis, R. Rastogi, K. Shim: Mining Sequential Patterns with Regular Expression Constraints, TKDE 14(3), pp. 530-552, 2002.

[Hadoop] Hadoop: http://hadoop.apache.org/. [last visited: 24.08.2011]

[HBase] HBase: http://hbase.apache.org/. [last visited: 24.08.2011]

[Hive] Hive: http://hive.apache.org/. [last visited: 24.08.2011]

[HYB*11] J. Hoffart, M.A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, S. Thater and G. Weikum: "Robust Disambiguation of Named Entities in Text". In Proc. of EMNLP 2011: Conference on Empirical Methods in Natural Language Processing, Edinburgh, Scotland, UK, July 27–31, 2011.

[Hypertable] Hypertable: http://hypertable.org/documentation.html. [last visited: 24.08.2011]

[IBY*07] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly : Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. EuroSys 2007.

[Kirk11] S. Kirkpatrick: "LAWA Deliverable D3.1: Introduction to Hadoop-Kelvin". 4.5.2011, http://www.lawa-project.eu/uploads/D3.1.pdf. [last visited: 24.08.2011]

[KuQu09] M. K. McKusick, S. Quinlan. "GFS: Evolution on Fast-forward," ACM Queue, vol. 7, no. 7, New York, NY. August 2009.

[LaMe04] A. N. Langville and C. D. Meyer: Deeper Inside PageRank, Internet Mathematics 1(3), pp. 335-380, 2004

[LFS] Lustre File System. http://www.lustre.org. [last visited: 24.08.2011]

[LiDy10] J. Lin and C. Dyer: Data-Intensive Text Processing with MapReduce, Morgan & Claypool, 2010.

[LiSc10] J. Lin, M. Schatz: Design patterns for efficient graph algorithms in MapReduce, MLG 2010

[LMJ*07] E. Lieberman, J. Michel*, J. Jackson, T. Tang, and M. Nowak: Quantifying the Evolutionary Dynamics of Language, Nature 449, pp. 713-716, 2007.

[MCB*11] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, A. H. Byers: Big data: The next frontier for innovation, competition, and productivity. http://www.mckinsey.com/mgi/publications/big_data/pdfs/MGI_big_data_full_report.pdf, May 2011. [last visited: 24.08.2011]

[Memcached] Memcached: http://memcached.org/. [last visited: 24.08.2011]

[MRS09] C. D. Manning, P. Raghavan, H. Schütze: Introduction to Information Retrieval, Cambridge University Press, 2009.

[MSA*10] J. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, and E. Lieberman: Quantitative Analysis of Culture Using Millions of Digitized Books, Science 331(6014), pp. 176-182, 2011.

[Pig] Pig: http://pig.apache.org/. [last visited: 24.08.2011]

[PostgreSQL] PostgreSQL: http://www.postgresql.org/. [last visited: 24.08.2011]

[Rackspace] Rackspace: http://www.rackspace.com/cloud/. [last visited: 24.08.2011]

[RDFa] RDFa Primer: http://www.w3.org/TR/xhtml-rdfa-primer/. [last visited: 24.08.2011]

[SKR*10] K. Shvachko, H. Kuang, S. Radia, R. Chansler: The Hadoop Distributed File System. In: MSST2010, May 2010.

[SKWe07] Fabian M. Suchanek, GjergjiKasneci, and Gerhard Weikum.Yago: a core of semantic knowledge. In: Proc. of the 16th International World Wide Web Conference, 2007.

[SKWe08] Fabian M. Suchanek, GjergjiKasneci, and Gerhard Weikum. YAGO: A Large Ontology from Wikipedia and WordNet. Journal of Web Semantics 6(3), 2008.

[SaTs99] B. Salzberg and V. J. Tsotras: Comparison of Access Methods of Time-Evolving Data, ACM Computing Surveys 31(2), pp. 158-221, 1999.

[Stratosphere] Stratosphere: http://www.stratosphere.eu/. [last visited: 24.08.2011]

[Same06] H. Samet: Foundations of Multidimensional and Metric Data Structures, Morgan Kaufman, 2006.

[TaCr09] P. P. Talukdar and K. Crammer: “New regularized algorithms for transductive learning”. In: ECML/PKDD (2), pp. 442–457, 2009.

[TPG08] W. Tantisiriroj, S. Patil, G. Gibson. “Data-intensive file systems for Internet services: A rose by any other name ...” Technical Report CMU- PDL-08-114, Parallel Data Laboratory, Carnegie Mellon University, Pittsburgh, PA, October 2008.

[TSJ*09] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, R. Murthy: Hive - A Warehousing Solution Over a Map-Reduce Framework. PVLDB 2(2): 1626-1629 (2009).

[TSJ*10] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, R. Murthy: Hive - a petabyte scale data warehouse using Hadoop. ICDE 2010: 996-1005.

[WaKe09] D. Warneke, O. Kao: Nephele: Efficient Parallel Data Processing in the Cloud, MTAGS 2009.

[WBM*06] S. Weil, S. Brandt, E. Miller, D. Long, C. Maltzahn, “Ceph: A Scalable, High-Performance Distributed File System,” In Proc. of the 7th Symposium on Operating Systems Design and Implementation, Seattle, WA, November 2006.

[WNS*11] G. Weikum, N. Ntarmos, M. Spaniol, P. Triantafillou, A. Benczúr, S. Kirkpatrick, P. Rigaux and M. Williamson: “Longitudinal Analytics on Web Archive Data: It's About Time!” In: Proceedings of the 5th biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA, January 9 - 12, pp. 199-202, 2011.

[WYQ*11] Y. Wang, B. Yang, L. Qu, M. Spaniol and G. Weikum: “Harvesting Facts from Textual Web Sources by Constrained Label Propagation”. In: Proceedings of the 20th ACM Conference on Information and Knowledge Management (CIKM 2011), Glasgow, Scotland, UK, October 24-28, 2011. (to appear)

[YHB*11] M.A. Yosef, J. Hoffart, I. Bordino, M. Spaniol and G. Weikum: “AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables”. In Proc. of the 37th Intl. Conference on Very Large Databases (VLDB 2011), August 29 – September 3, Seattle, WA, USA (to appear).

[YIF*08] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, J. Currey: DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language, OSDI 2008.

[ZoMo06] J. Zobel and A. Moffat: Inverted Files for Text Search Engines, ACM Computing Surveys 38(2), pp. 1-56, 2006.