An open source infrastructure for quality assurance and preservation of a large digital book collection

Sven Schlarb; Austrian National Library; Vienna, Austria

Abstract

This article presents an open source infrastructure for processing large collections of digital books available at the Austrian National Library with a special focus on quality assurance tasks in the context of the European project SCAPE (www.scape-project-eu). It describes the cluster hardware and the software components used for building the experimental IT infrastructure.

More concretely, a set of best practices for the data analysis of large document image collections on the basis of Apache Hadoop will be shown. Different types of Hadoop jobs (Hadoop-Streaming-API, Hadoop MapReduce, and Hive) are used as basic components, and the Taverna workflow description language and execution engine (www.taverna.org.uk) is used for orchestrating complex data processing tasks.

Introduction

Today's libraries are curating large digital collections, indexing millions of full-text documents, and preserving terabytes of data for future generations. This means that libraries must adopt new methods for processing large amounts of data. And this is exactly where the European project SCAPE (www.scape-project.eu) comes into play. The project has a focus on long-term preservation and is creating an open source infrastructure together with a variety of tools and services designed for the distributed processing of large data sets. The purpose of this infrastructure is to be able to perform quality assurance tasks, like analysing cropping errors with possible text loss, for example, as part of managing large digital book collections.

The article gives an overview on the experimental set-up at the Austrian National Library including hardware and open source software components. Furthermore, it describes experiences and provides best practices for the data analysis of large document image collections on the basis of Apache Hadoop (http://hadoop.apache.org). Different types of Hadoop jobs (Hadoop-Streaming-API, Hadoop MapReduce, and Hive - http://hive.apache.org) are used as basic components, and the Taverna workflow description language and execution engine (www.taverna.org.uk) is used for orchestrating complex data processing tasks.

Experimental environment

A dedicated test environment was created in physical proximity to the Austrian National Library's digital book repository in order to be able to run different types of large scale data processing scenarios on real institutional data sets. In the following, the hardware and software environment will be described.

Cluster software

The SCAPE Preservation Platform provides an extensible infrastructure based on Apache Hadoop together with a set of preservation components containing migration, characterisation and quality assurance components that can be used for creating composite preservation workflows.

Ubuntu server 10.04 is used as the operating system of all cluster nodes and Java (version 6 update 33) is required for most of the SCAPE platform, tools, and services.

Hadoop [2] is an essential component of the SCAPE platform and provides an implementation for MapReduce [3]. It is a programming model for the distributed processing of large datasets. The Cloudera CDH3 Update 5 (CDH3u5) distribution has been used to make Apache Hadoop available in version 0.20.2. Additionally, a MySQL database is used as Hive Metastore. The Taverna workflow description language [1] and execution engine (www.taverna.org.uk) is used for orchestrating complex data processing tasks.

Cluster hardware

The computing and distributed storage cluster consists of one master node controlling five worker nodes, as shown in figure 1. The master node is a server with two 2.4 GHz Quadcore CPUs (16 HyperThreading cores), 24 gigabyte RAM, and three one terabyte disks configured as RAID5. The worker nodes are servers with one 2.53 GHz Quadcore CPU (8 HyperThreading cores), 16 gigabyte RAM, and two one terabyte disks configured as RAID0 (no disk system level redundancy).

The master node runs only the Hadoop Job Tracker daemon, which distributes MapReduce tasks to the task trackers, and the Hadoop Name Node daemon, which manages the Hadoop Distributed Filesystem (HDFS).

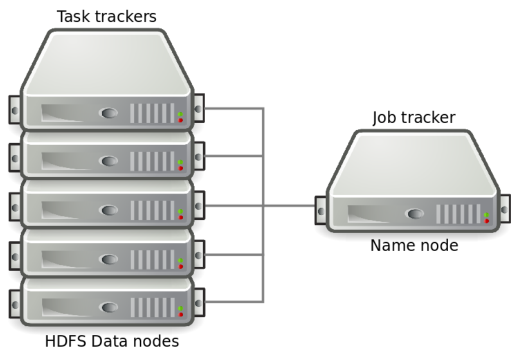

Figure 1: Experimental cluster: One controller and five worker nodes.

Each worker node server has 4 physical cores. With hyper-threading technology 8 logical cores are available. In the selected configuration, 5 logical cores are assigned to Map tasks, 2 cores to Reduce tasks, and 1 core is reserved for the operating system. Basically, this means that a total of 25 Map tasks and 10 Reduce tasks can run in parallel.

Regarding the overall storage capacity, in this configuration only worker node hard disks are configured to be part of the Hadoop Distributed Filesystem (HDFS). 1.74 terabytes of each worker node are assigned to HDFS, and the rest is reserved to the operating system. This leads to a total physical storage space of around 8.7 terabytes. The cluster is configured with redundancy factor 2 which means that each data block is located at least on two different machines. Therefore, half of the physical storage space, around 4.35 terabytes, is available in the HDFS.

Book page layout analysis

One scenario that will be discussed in detail here is about parsing a large collection of HTML book page layout and text content files (>25 million pages) in order to extract structured information that can be used to identify possible quality issues.

The HTML files – more precisely, hOCR [4] files – are part of a book collection where each HTML page represents layout and text of a corresponding book page image. These HTML files have block level elements described by properties of a 'div' HTML tag. Each 'div' element has a "position", "width" and "height" property representing the surrounding rectangle of a text or image block.

Assuming a typical book page layout, the identified text block width (multiple columns summing up to one text block) plus the white margins should approximately match the book page image width. Based on this assumption, the central hypothesis is that if there is a significant mismatch for several pages of the same book, it can be interpreted as an indicator for possible cropping errors with text loss.

However, the main concern of this article is not to verify the validity of this hypothesis but to show the principle of approaching this kind of data processing scenario by means of the MapReduce programming model.

In this sense, the scenario presented here presents the parallel execution of HTML parsing and the subsequent average text block width per page calculation using MapReduce.

Before this MapReduce job can be applied, it is necessary to prepare the data which will be described in the following section.

Data preparation

First of all, dealing with lots of HTML files means that we are facing Hadoop's "Small Files Problem" [5]. In brief, this is to say that the files we want to process are too small for choosing them directly as input for a MapReduce job.

As a matter of fact, loading 1000 separated HTML files into HDFS would already take a considerable amount of time. What is worse, however, if these 1000 HTML files were used as input the Hadoop JobTracker would create 1000 Map tasks because at least one Map task is created per input file. Given the task creation overhead this would result in a bad processing performance. In short, Hadoop does not like small files but, on the contrary, the larger the better.

One approach to overcome this shortcoming is to create one large file, a so called *SequenceFile*, that holds key-value pairs. Related to the scenario described above the key is a book page identifier and the value is the content of the "small file", here the HTML plain text content. The *SequenceFile* is used for sequential processing of the key-value pairs, since random access is not intended. The sequence file is created by reading the HTML files in parallel from network attached storage. Just to give a rough impression, depending on file server and network load, aggregating 25 million pages and loading it as a single *SequenceFile* into HDFS took up to 24 hours in the experimental environment described in this article (see section "Experimental environment").

Further down in this article it will be made clear how this relates to the processing time of MapRecuce jobs. At this point it's just worth noting that the purpose of this data preparation is a "write-once, read-many-times pattern" [6] where data is kept in HDFS in order to perform different types of analysis and information extraction tasks.

Given the limited storage capacity of the experimental cluster, it is not possible to make the complete digital books data set available in the cluster, especially the image data remains on network attached storage.

Data flow modelling

Depending on expected output and requirements, complex data processing patterns are usually not performed by a single MapReduce job, but they are combined in a processing pipeline executing a sequence of jobs. For orchestrating complex data processing tasks, the Taverna workflow description language and execution engine (www.taverna.org.uk) is used in the SCAPE project.

The diagram in Figure 2 shows a graphical representation of a Taverna workflow. On the top of the diagram are the so called input ports which are variables that get string values (e.g. by user input) when starting a workflow run. Arrows indicate the data flow from one element to the subsequent one. The element on the bottom of the diagram is the output port which is the final outcome of the workflow run. The components in the middle of the diagram are "tool service" (http://dev.mygrid.org.uk/wiki/display/taverna/Tool+service) components which start Hadoop jobs in bash scripts and return the standard output of these processes to the subsequent component.

The *HadoopSequenceFileCreator* Taverna component in Figure 2 is basically a Map function reading HTML files directly from the file server, and storing a book page identifier as 'key' and the content as *BytesWritable* 'value' (key-value-pair).

Figure 3 illustrates the process of creating a plain text input file containing all file paths and then assembling the content into one *SequenceFile*.

The network attached storage is mounted on each worker node at the same mount point, i.e. each processing node of the cluster has access to the files on the file server. Given that each worker node executes several tasks simultaneously using the available CPU cores, the *SequenceFile* is created in a parallel manner, limited basically by the bandwidth of the internal network (*SequenceFile* creation is mostly I/O bound).

The *SequenceFile* is then split into 64 megabytes splits (default configuration) so that the TaskTracker parses a bundle of HTML files in each task. The runtime of a task in a Hadoop job is an important factor for the overall performance. On the one hand, choosing a smaller split size can have the effect of creating many tasks with short runtime so that the task creation overhead has a negative impact on the overall performance. On the other hand, a bigger split size with fewer tasks can have the effect of creating tasks with long runtime where, at the end of the Hadoop job, a few tasks are still running while the remaining processing cores lie idle. Cluster monitoring services, like Ganglia (http://ganglia.sourceforge.net), can help to optimize the use of cluster resources [7].

Once data is loaded into HDFS, the *SequenceFileInputFormat* can be used as input in the subsequent MapReduce job which parses the HTML files using the Java HTML parser Jsoup (http://jsoup.org) in the Map function and calculates the average block width in the Reduce function. This is done by the *HadoopHocrAvBlockWidthMapReduce* Taverna component illustrated in Figure 4.

The interoperability between Hadoop jobs is in this example simply ensured by the first job writing the output HDFS path to standard out which the second job then takes as the HDFS input path. The second job only starts after the first job has completely finished.

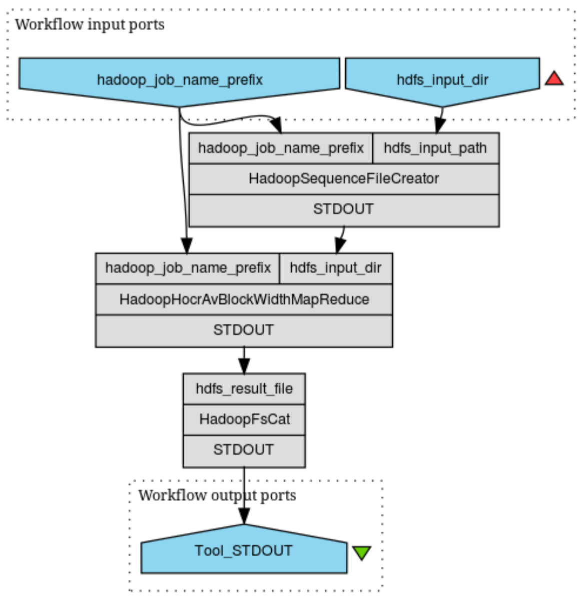

Figure 2: Taverna workflow modelling a sequence of Hadoop jobs.

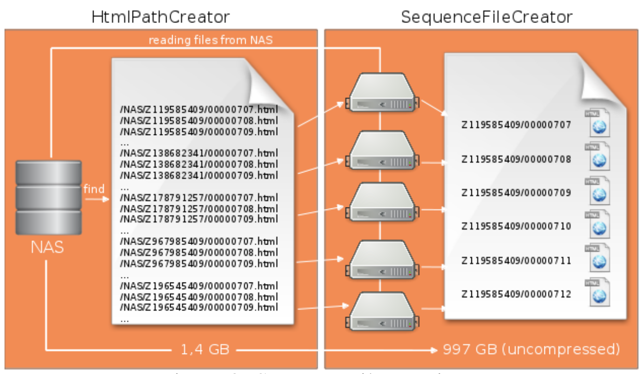
HtmlPathCreator
SequenceFileCreator
reading files from NAS
/NAS/Z119585409/00000707.html
/NAS/Z119585409/00000708.html
/NAS/Z119585409/00000709.html
...
/NAS/Z138682341/00000707.html
/NAS/Z138682341/00000708.html
/NAS/Z138682341/00000709.html
...
/NAS/Z178791257/00000707.html
/NAS/Z178791257/00000708.html
/NAS/Z178791257/00000709.html
...
/NAS/Z967985409/00000707.html
/NAS/Z967985409/00000708.html
/NAS/Z967985409/00000709.html
...
/NAS/Z196545409/00000707.html
/NAS/Z196545409/00000708.html
/NAS/Z196545409/00000709.html
...
find
NAS
Z119585409/00000707
Z119585409/00000708
Z119585409/00000709
Z119585409/00000710
Z119585409/00000711
Z119585409/00000712
1,4 GB
997 GB (uncompressed)

Figure 3: SequenceFile creation.

Let k_1 be the identifier of the HTML file (data type: org.apache.hadoop.io.Text), and v_1 be the value holding the content of the HTML file (data type: org.apache.hadoop.io.BytesWritable). The key-value pair $(k_1\ \ v_1)$ is the input of the Map function shown in equation 1.

$$map(k_1\ \ v_1) \rightarrow list(k_1\ \ v_2) \tag{1}$$

As a simple example, let us assume one key-value pair input with the page identifier 00001 *html* and the HTML file content as value as shown in example 2.

$$(\text{"00001.html"}\ \ \text{"}<html>[content]<\ \ html>\text{"}) \tag{2}$$

The mapper will produce a list of key-value pairs, one key-value pair for each text block the parser finds, as output. Assuming the parser finds two text blocks with 1200 pixel and 1400 pixel width, the output list would be as shown in example 3.

$$((\text{"00001.html"}\ \ 1200)\ \ (\text{"00001.html"}\ \ 1400)) \tag{3}$$

The Mapper outputs a list of key-value pairs $list(k_1\ \ v_2)$ with v_2 being the text block width value and k_1 the HTML page key simply repeated for each value. The value is a string with coordinates, representing width and height of the block element.

For each HTML page k_1, the Reduce function aggregates all text block width values $list(v_2)$:

$$reduce(k_1\ \ list(v_2)) \rightarrow k_1\ \ v_3 \tag{4}$$

During this step it sums up the $list(v_2)$ text block width values and calculates the average width v_3 (data type: org.apache.hadoop.io.LongWritable).

Related to example 3, this means that the Reduce function would calculate the average width like shown in example 5.

$$(\text{"00001.html"}\ \ (1200\ \ 1400)) \rightarrow (\text{"00001.html"}\ \ 1300) \tag{5}$$

Finally, the *HadoopFsCat* Taverna component returns part of the result for demonstrating a successful workflow run.

Combining different types of Hadoop jobs

The example in the previous section introduced the principle of modelling a data flow, including different Hadoop processing steps, using the Taverna workbench while data processing and exchange is done exclusively in HDFS.

In this section, an extension of this workflow that reuses the components *HadoopSequenceFileCreator* and

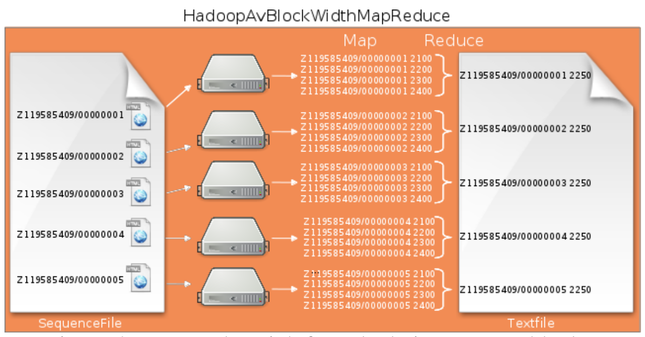

Figure 4: Map Reduce job for calculating average block.

HadoopHocrAvBlockWidthMapRecude will be shown. The workflow is illustrated in Figure 5 and includes different types of Hadoop components in a data flow with two branches, it has:

a Hadoop Streaming API component (HadoopStreamingExiftoolRead) reading image metadata using Exiftool.

a Map/Reduce component (HadoopHocrAvBlockWidthMapReduce) calculating the average text block width per page.

Hive components for creating data tables (HiveLoad*Data) and performing a test query on the results (HiveSelect).

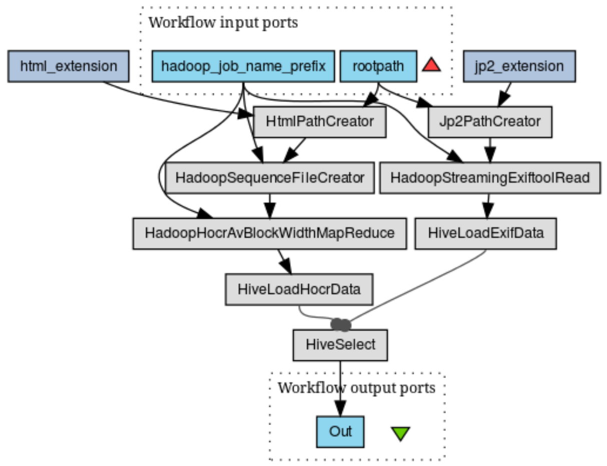

Figure 5: Taverna workflow modelling a sequence of Hadoop jobs.

The workflow has two input ports, the first input port *hadoop_job_name_prefix* is an identifier for the Hadoop jobs started by the workflow. This allows tracking Hadoop job progress using the HadoopJobTracker. The second input port *rootpath* is a directory root path from where files will be retrieved recursively. In this demonstration, the Unix tool 'find' is used to create a text file containing all file paths with a specific extension in the corresponding directory. This is done by the component *HtmlPathCreator* for retrieving all file paths with HTML extension and by the component *Jp2PathCreator* for all file paths with JP2 extension (JPEG2000).

The components *HadoopSequenceFileCreator* for sequence file creation and *HadoopHocrAvBlockWidthMapRecude* for reading the average text block width per page of the workflow shown in Figure 2 are reused in the left branch of this workflow.

In the right branch, image metadata from the JPEG2000 image files are read by executing Exiftool using Hadoop's streaming API. In this case a bash script is defined as the Mapper and no Java implementation for the Hadoop job is required.

Both branches of this workflow return a two columns tab delimited plain text files where the first column is the page identifier, and the second column is the average text block width for *HadoopHocrAvBlockWidthMapRecude* and the image width for *HadoopStreamingExiftoolRead* respectively.

These results can then be loaded into a Hive database, the *HiveLoadHocrData* component creates a Hive table *hocrdata* with an identifier column *hocrdata id* and average text block width column *hocrdata width*. And the *HiveLoadExifData* component creates a *exifdata* table width an identifier column *exifdata id* and an image width column *exifdata width*.

The workflow is intended to be a data preparation component which makes data available for doing analytic queries using Hive's MySQL-like query language. It is now possible to execute analytic queries in Hive's SQL-like syntax, the *HiveSelect* component executes a simple test query in order to verify if the result data has been created successfully. It executes a SELECT query with a JOIN on the two tables created by Hive like shown in listing 1.

```
select
  hocrdata.id, hocrdata.width, exifdata.width
  from hocrdata
  inner join exifdata
  on hocrdata.identifier=exifdata.identifier;
```

Listing 1: SQL Join Query using the two result tables

An example result of this query is shown in table 1 where the first column is a combined book/page identifier, the second column the corresponding average text block width (Avg. width) and the third column the image width (Exif width).

Table 1: Hive select result table

Identifier	Avg. width	Exif width
00010.html	1041	2210
00011.html	826	2245
00012.html	1122	2266
00013.html	1092	2281
00014.html	1026	2102
00015.html	1046	2217
00016.html	864	2263

Outlook and further work

The institutional scenario outlined in this article focused on I/O intensive tasks, like reading metadata and parsing plain text HTML files.

A different challenge is posed by computing intensive quality assurance components developed in the SCAPE project, like the one described in [8]. Based on image feature extraction, this software enables creating modularised workflows for a variety of digital library data processing scenarios, like image duplicate detection in books or digital book version comparison, for example. Various processing steps of this software are computing intensive and data needs to be shared in between. Using of the software in the context of the distributed data processing platform used in the SCAPE project will open new requirements to the way how the software components interact and how data can be shared in a distributed computing environment.

At the Austrian National Library, the SCAPE platform and software components will be used in the context of data management and quality assurance of the digital book collection. It is planned to integrate more software tools for data format migration, file format validation, data analysis, and quality assurance into this framework to enable the analysis of large digital book collections combining the library book metadata with the technical metadata created by workflows described in this article.

Related work

A great variety of workflow engines have emerged around Apache Hadoop in the recent years, like Apache Oozie (http://oozie.apache.org) or Hamake (http://code.google.com/p/hamake), just to name two other Hadoop related workflow engines using an XML-based workflow description language.

A prominent approach in this field is the work presented by [9] about Meandre, a component-based framework for data-intensive computing. While the work presented in this article was focusing on using Hadoop as an open source solution to the large scale data processing in the context of a specific use case, this work describes a generic approach of using an SOA (Service Oriented Architecture) with a semantic component model used to model computing and I/O intensive data flows.

Conclusional remarks

The SCAPE project will continue developing scalable services for planning and execution of institutional preservation strategies on an open source platform that orchestrates semi-automated workflows for large-scale, heterogeneous collections of complex digital objects. More information is available at:

www.scape-project.eu

The principal use of the Taverna Workbench desktop application is to design component-based data flows and demonstrate a workflow run using sample data. Workflows can be shared using the myExperiment workflow publishing platform [10]. Hadoop jobs are usually long-running processes, therefore Taverna server offers a REST API which allows remote execution of workflows.

The Taverna workflows presented in this article are available on myExperiment:

www.myexperiment.org/workflows/3069
www.myexperiment.org/workflows/3105

The code for the Hadoop jobs employed in the workflows is available via the artefacts tb-lsdr-seqfilecreator and tb-lsdr-hocrparser in the openplanets repository on Github:

www.github.com/openplanets/scape

Visit the Github account of the Open Planets Foundation to get a preview on many of the software components created by the SCAPE project at:

www.github.com/openplanets

And via the myExperiment SCAPE group you can find Taverna workflows published by the SCAPE project:

www.myexperiment.org/groups/490

Acknowledgments

This work was partially supported by the SCAPE project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).

Special thanks go to my former colleague at the Austrian National Library, Martin Reisacher, who provided the primary use case for this article which is the detection of cropping errors causing text loss in the Austrian National Library's digital books collection.

References

[1] D. Hull, K. Wolstencroft, R. Stevens, C.A. Goble, M.R. Pocock, P. Li, and T. Oinn, Taverna: a tool for building and running workflows of services., Nucleic Acids Research, vol. 34, 2006, pp. 729–732.

[2] Andrzej Bialecki, Mike Cafarella, Doug Cutting, Owen O'Malley, "Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware", 2005, http://hadoop.apache.org/.

[3] Jeffrey Dean, Sanjay Ghemawat, MapReduce: simplified data processing on large clusters, 2008, pp. 107–113.

[4] Thomas M. Breuel, "The hOCR Microformat for OCR Workflow and Results.", Proc. ICDAR, pg. 1063-1067. (2007).

[5] Tom White, "The Small Files Problem.", Cloudera blog, 02/02/2009, Retrieved from http://blog.cloudera.com/blog/2009/02/the-small-files-problem/.

[6] Tom White, Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale (3. ed., revised and updated), O'Reilly, 2012, pg. 44.

[7] Jason Venner, Pro Hadoop, 2009, pg. 196.

[8] Reinhold Huber-Mrk, Alexander Schindler, Sven Schlarb, Duplicate Detection for Quality Assurance of Document Image Collections, Proc. iPres2012. (2012).

[9] Bernie Ács, Xavier Llorà, Loretta Auvil, Boris Capitanu, David Tcheng, Mike Haberman, Limin Dong, Tim Wentling, Michael Welge, A general approach to data-intensive computing using the Meandre component-based framework. Proc. 1st International Workshop on Workflow Approaches to New Data-centric Science. (2010).

[10] C. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, and D. De Roure, myExperiment: a repository and social network for the sharing of bioinformatics workflows, Nucleic Acids Research, 2010.

Author Biography

Sven Schlarb , Ph.D., studied Humanities Computer Science at the University of Cologne. He joined the Austrian National Library in 2008, participated in the EU funded projects PLANETS and IMPACT, and is now leading the Testbeds sub-project in SCAPE. Before, he worked as a web software developer in Cologne, and as software engineer (C++/Java) and SAP support consultant at SAP in Madrid.