XML Processing in the Cloud: Large-Scale Digital Preservation in Small Institutions

Peter Wittek*, Thierry Jacquin, Hervé Déjean, Jean-Pierre Chanod†, and Sándor Darányi*
*Swedish School of Library and Information Science
University of Borås
Borås, Sweden
†Xerox Research Centre Europe
Meylan, France

***Abstract*—Digital preservation deals with the problem of retaining the meaning of digital information over time to ensure its accessibility. The process often involves a workflow which transforms the digital objects. The workflow defines document pipelines containing transformations and validation checkpoints, either to facilitate migration for persistent archival or to extract metadata. The transformations, nevertheless, are computationally expensive, and therefore digital preservation can be out of reach for an organization whose core operation is not in data conservation. The operations described the document workflow, however, do not frequently reoccur. This paper combines an implementation-independent workflow designer with cloud computing to support small institution in their ad-hoc peak computing needs that stem from their efforts in digital preservation.**

***Index Terms*—Digital Preservation, Cloud Computing**

I. Introduction

Digital preservation (DP) combines policies, strategies and actions to ensure that digital objects remain authentic and accessible to users and systems over a long period of time, regardless the challenges of component and management failures, natural disasters or attacks [1]. This includes the preservation of materials resulting from digital reformatting, but particularly of information that is born-digital and has no analog counterpart.

While DP has a vast literature exposing the wide array of associated issues, here we would like to focus on only three aspects: migration and transformation, scalability, and reusability.

The first aspect refers to the problem of keeping the content of legacy file formats accessible. This problem is most prominent with proprietary file formats for which documentation is not available. Once the vendor stops support for the associated software products, these files face digital obsolescence. A common solution to the problem is migration in which older formats are transformed to a more persistent format.

Secondly, dynamic collections and environments for DP require technical scalability to face technology evolution. Existing static collections, for instance, a digitized historical archive, where no new items will be added, will have a fixed data size. Although it will not be necessary to add new components to increase the storage capacity, it may be necessary to replace components or transform the objects in the collection. These requirements ask for scalability [2]. Achieving this of DP may require specific investments in an infrastructure for storing, maintaining, and managing data. Such costs can be prohibitive for organizations whose core business is not data conservation and that do not have a considerable budget for investments in information technology.

The third important aspect of DP that we would like focus on is to enable the reuse of digital content. Reuse of digital content covers its subsequent verification and its exploitation with a novel purpose, potentially by new consumers. Since consumers are unable to refer back to the creators, reuse of preserved digital objects depends on proper descriptions provided through the archive [3]. In this sense, reuse of digital content asks for metadata on both the content and how it was transformed to its most recent form. This is where document process preservation helps, which provides an architecture-independent description of the intent behind a document process [4].

The above three points are not unrelated. Migration and transformation can be regarded as part of a larger problem described in the third point: we need to preserve not just the documents, but also how they were processed. The second point is, of course, crucial: let it be migration or metadata extraction, these operations are computationally expensive.

SHAMAN (Sustaining Heritage Access through Multivalent ArchiviNg) is an integrated project, co-funded by the European Commission under the seventh RTD Framework Programme. The aim of this project is to investigate the long-term preservation of large volumes of digital objects in a distributed environment, by developing a preservation framework that is verifiable, open and extensible [5]. SHAMAN is focusing its research on: integrating data grid, digital library and persistent archive technologies; developing support for context representation and annotation, with deep linguistic analysis and corresponding semantics; and modeling of preservation processes [6]. In this paper, we describe a system developed in SHAMAN which combines a document process designer for digital preservation with scalable computational resources that are available for small organizations: cloud computing.

This paper is organized as follows. Section II outlines the importance of document processing pipelines in digital preservation and the need to preserve the process itself. Document processing, however, is both data and compute-intensive, a possible way out is using the cloud; the problem is detailed in Section III. Bringing together these two themes, document workflows and cloud computing, we propose a solution that works for organizations with limited computational resources or restricted IT budget in Section IV by moving the processes related to preservation to the cloud. Section V further discusses the implementation and reveals some experimental results. After the discussion, a section on related work highlights in what ways our approach is different from existing similar work (Section VI), Section VII discusses our planned future research efforts, and finally Section VIII concludes the paper.

II. Workflows and Digital Preservation

Preserving the intent behind the activities and projects that led to the production of digital data can be as meaningful and important in the long term as preserving the digital data themselves. Indeed, an important goal of digital preservation is to enable reuse of digital content by securing the long term understanding of the intent behind the preserved data.

Production and reuse of digital content from the archive do not coincide, as reuse may have a novel purpose or may operate in a totally different environment than was available at production time. Preservation must bridge gaps in time, space, semantics, knowledge, objectives and other dimensions. This is why the description of preserved digital objects must include means to understand the context in which the data was initially produced and used. The context is not only defined by the digital objects themselves, but also by the processes, in which they were created, ingested, accessed and re-used.

A. The Xeproc Domain-Specific Language

The Xeproc Domain-Specific Language (DSL), developed within SHAMAN, addresses precisely the need to capture the intent behind document processes, so that they can be preserved and reused in future unknown infrastructures. To that end, Xeproc preserves not only production processes, but also instrumented specifications of a document processing project.

Xeproc technology can be used to build a wide range of applications based on document processing, including transformation, extraction, indexing and navigation. It can be easily integrated with more global business processes and customized to match specific requirements and infrastructures. In the spirit of service-oriented architecture (SOA), Xeproc embeds references to services and documents and provides loose coupling not only to services but also to data resources, with respect to both their location and format.

These capture the intent behind the workflow irrespective of the implementation at a given point in time (see Figure 1). These abstract representations are preserved, so that the Xeproc models can be seen as independent specifications to be instantiated and deployed over time and as technology evolves.

Available on Eclipse 3.5.1 under the Eclipse Public License[1], Xeproc combines a DSL, an associated graphic designer and extension application programming interfaces. A DSL is a programming language or specification language dedicated to a particular problem domain. The associated designer lets one define and design document processes while producing an abstract representation that is independent of the implementation.

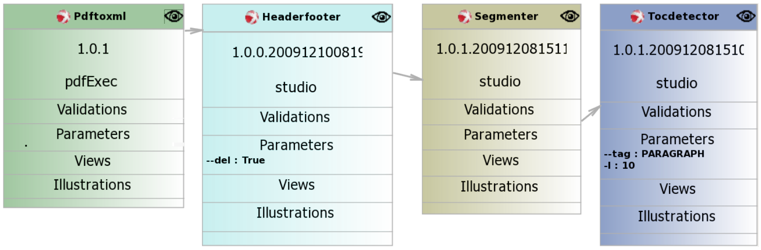
B. XML processing pipelines

XML is ideally suited to representing the logical structure of documents (e.g. titles, sections, chapters, paragraphs, reading flow) independently of their visual rendering. XML is also able to represent the semantics or meaning of documents, by explicitly encoding the various elements that make up the document, such as authors, dates, organization or product names, financial data, etc. It hence provides a natural bridge between databases and content. By explicitly encoding a document's structure and meaning, XML opens up the possibilities for document lifecycle, including content reuse and repurposing, quality assurance and security. The sheer volume and heterogeneity of document collections (numerous authoring systems and proprietary formats such as PDF, PS, Word, and TIFF) as well as the nature and characteristics of the information to be encoded (data-oriented documents such as purchase orders, complex and implicit structures such as maintenance manuals) makes document conversion to XML a complex and delicate task.

XML processing often involves transformations from one XML format to another. Extensible Stylesheet Language Transformations (XSLT) is a declarative, XML-based language used for describing these transformations [7]. For instance, XSLT is often used to convert XML data into HTML or XHTML documents for display as a web page. In the context of DP, the content may be stored in a richly annotated XML format, and when the content is presented to the user, an XSLT transformation can be used to render the page in a legible format. An XSLT processor is an algorithm takes an XML file and an XSLT file, and outputs the transformed XML file. An XSLT processor may be a stand-alone program, or a library to use from a wide range of programming languages.

From an automatic processing point of view, a great advantage of XML is that the format distinguishes the concepts of syntactic correctness and of validity with respect to a document type definition. Syntactic correctness is inherent in the language, while further restrictions on the elements and attributes can be defined, and a document can be validated against these. The validity is verified via XML schemas, which are descriptions of types of XML documents, typically expressed in terms of constraints on the structure and content of documents of those types, above and beyond the basic syntactical constraints imposed by XML itself. All XML documents must be syntactically correct (that is, they must be well-formed), but it is not required that a document be valid unless the XML parser is validating, in which case the document is also checked for conformance with its associated schema. Documents are only considered valid if they satisfy the requirements of the schema which they have been

[1] http://marketplace.eclipse.org/content/xeproc



--del : True

--tag : PARAGRAPH
-l : 10

Fig. 1. Document process designer showing a process to extract table of contents

associated with. These requirements typically include such constraints as:

- Elements and attributes that must/may be included, and their permitted structure;
- The structure as specified by a regular expression syntax;
- How character data is to be interpreted, e.g. as a number, a date, a URL, a Boolean, etc.

Within the context of SHAMAN, Xeproc has been specifically used to model XML processing pipelines and XML validation checkpoints. Validation checkpoints may be defined in any of several schema languages such as Document Type Definition or Relax NG. Those XML processing pipelines focus on identifying structural metadata describing the document organization. The pipelines have been designed and customized for each of the three following collections:

- DNB collection of electronic PhD theses: The Deutsche Nationalbibliothek (DNB) holds a continually growing collection of currently more than 95,000 PhD theses in electronic form originally provided by German universities. These theses are fully processed for library and preservation purposes and constitute an important subcollection of the overall digital collection at DNB. All objects in this collection are text documents, more than 95% of them are PDF files.
- Digitised Collections from the Göttinger Digitalisierungszentrum: The Göttinger Digitalisierungszentrum (GDZ) is one of the two German competence centres for digitization and has digitized over 5 million pages across a variety of projects.
- INEX collections: INEX (Initiative for the Evaluation of XML) promotes the evaluation of focused retrieval by providing large test collections of structured documents, uniform evaluation measures, and a forum for organizations to compare their results. The 2009 Book structure, track builds on a collection of digitized books, provided by Microsoft Live Book Search and the Internet Archive (for non-commercial purposes only). The corpus consists of over 50,000 digitized out-of-copyright books.

The pipelines eventually produce metadata in Metadata Encoding and Transmission Standard (METS, [8]) addressing various aspects of the document collections, namely:

1) Extraction of logical organization: logical elements reflecting the document organization are marked-up (table of contents, chapters);
2) Extraction of the page numbering (for book navigation purpose);
3) Extraction of illustrations and associated captions;
4) Generation of an XML file for indexing purpose and reflecting the logical organization of the books.

C. XML processing pipeline deployment

The resulting Xeproc logical and persistent descriptions, when associated with the accurate components, are interpreted or translated into any SOA orchestration language to produce logically structured documents. For instance, one may use Eclipse Modeling Framework (EMF) based generation technologies such as openArchitectureWare (OAW) to package production-ready pipelines. EMF is an Eclipse-based modeling framework and code generation facility for building tools and other applications based on a structured data model. Models can be specified using XML documents, modeling tools, and a few other methods, then imported into EMF. Eventually, the packaged pipelines can be deployed towards a production platform.

This paper proposes a novel deployment strategy targeting clouds. From a digital preservation perspective, this highlights how Xeproc enables straightforward migration from one operational infrastructure to another.

III. The computational need of digital preservation

The steps of a workflow tend to be computationally expensive. For instance, when migrating from one XML format to the other, one may insert an XSLT transformation in the pipeline. XSLT processors are increasingly optimized, they use the kind of optimization techniques found in functional programming languages and database query languages. For example, static rewriting of the expression tree to move calculations out of loops, and lazy pipelined evaluation to reduce the use of memory for intermediate results and allow "early exit" when the processor can evaluate an expression without a complete evaluation of all subexpressions. There are also processors which use tree representations that are much more efficient than a general purpose Document Object Model (DOM) implementation, which is a non-optimized cross-platform and language-independent convention for representing and interacting with objects in an XML document. However, even with these optimization, a processing a single large document tree can take hours, provided that the computer has the necessary resources, particularly a satisfactory amount of physical memory.

XML transformations, while costly, are not the most expensive operations. Automatic metadata extraction may involve high-complexity natural language processing tasks such as deep parsing and named entity recognition. The use of these tasks can be prohibitive even for smaller collections due their computational requirements.

It is important to recognize the ad-hoc nature of these computations. Metadata extraction and migration are not frequently performed. Small organizations, or organizations that do not have a considerable budget for investments in information technology do not have to maintain the resources permanently to deal with these operations. As many text mining applications prove, cloud computing is ideally situated for such situations.

The concept of MapReduce is often associated with cloud computing. When talking about cloud computing we refer to what is more precisely known as utility computing [9]. As the name implies, the idea behind utility computing is to treat computing resources as a metered service, like electricity or natural gas. Under this model, a user can dynamically provision any amount of computing resources from a (cloud) provider on demand and only pay for what is consumed [9]. Technically, this means that the user is paying for access to virtual machine instances that run a standard operating system such as Linux. The virtualization technology enables the cloud provider to allocate available physical resources and enforce isolation between multiple users that may be sharing the same hardware. Once one or more virtual machine instances have been allocated, the user has full control over the resources and can use them for arbitrary computation. When the virtual instances are no longer needed, they are destroyed, thereby freeing up physical resources that can be redirected to other users. Resource consumption is measured in machine-hours, breaking down to CPU-hours, bandwidth usage, etc. MapReduce provides the appropriate level of abstraction to this utility model by hiding the complexity of scaling to an arbitrary number of nodes which may fail.

MapReduce was originally developed by Google in the early 2000s, and the framework was first published in 2004 [10]. The publication spawned several open source efforts to implement the framework. Hadoop, now a top-level Apache project, was first released in 2006, and it has become the most popular open source implementation [11].

MapReduce is not a novel framework in distributed computing. It draws on well-known principles in parallel and distributed computing and assembles them in a way to scale to collections of sizes unseen before. The fundamental principles of MapReduce are summarized below [9]:

- Scale out: use a large number of low-cost commodity servers instead of a small number of expensive HPCs.
- Failures are common: a failure is not the exception, but the norm. Redundancy is required.
- Move processing to the data: instead of moving large volumes of data, it is worth moving the code around. This saves network bandwidth.
- Avoid random access: hard disks seek times did not improve at the same pace as capacities. to reduce latency coming from seek times, data is stored in sequential files.
- Hide system-level details from the developer: programming parallel, and especially distributed algorithms is a complicated process, MapReduce hides most details to simplify it.
- Seamless scalability: running the algorithm on a cluster of ten or a thousand computers should not burden the developer with additional programming.

Exploiting the resources in the cloud can be problematic for digital preservation, as it requires persistence and high-reliability [12], and, as outlined in the points above, the MapReduce framework helps. The framework is designed to be fault tolerant in an unpredictable, massively distributed environment in which individual nodes may fail frequently. Redundancy is built-in, with a computation launched at least three nodes simultanously, and if one result is different than the ones on the other two nodes, the subtask is executed again. Debugging, however, can be a major issue. Given the complexity of the task, a gradual scaling out can help identify errors. A task could be launched locally or in a local pseudo-cluster, than in a single-node cloud instance, then a full-scale launch may follow. However, if the input collection is inconsistent, finding out how the problem persists in the output collection is an unresolved challenge.

The framework is inspired by map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms [10].

In the map step, the so-called master node (a coordinating computer in the cluster) takes the input, chops it up into smaller sub-problems, and distributes those to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes that smaller problem, and passes the answer back to its master node. A mapper commonly performs input format parsing, projection (selecting the relevant fields), and filtering (removing records that are not of interest). The basic data structure of the framework is key-value pairs. Keys and values can be arbitrarily complex data structures. For instance, for a collection of web pages, the keys can be the URLs and the values are the HTML content. For a graph, keys can be the node identifiers, while values are the adjacency lists of those nodes. The output of the mapper is a sorted list of key-value pairs.

In the reduce step, the master node then takes the answers to all the sub-problems and combines them in a way to get the output - the answer to the problem it was originally trying to solve. As the processing gets more complex, this complexity is generally manifested by having more MapReduce jobs, rather than having more complex map and reduce functions. In other words, a developer thinks about adding more jobs, rather than increasing the complexity of the jobs. The two steps, map and reduce, are juxtaposed by an intermediate step, sort and shuffle. The sort phase orders the key-value pairs by a similarity function on the keys (note that keys can be arbitrarily complex structures), and the shuffle phase transfers the map outputs to the reducers as inputs.

In many ways, MapReduce is a major shift from the ruling Neumann model, which has a bottleneck between the CPU and the memory. Since CPU speed and memory size have increased much faster than the throughput between them, the bottleneck has become more of a problem, a problem whose severity increases with every newer generation of CPU. The abstraction provided by MapReduce circumvents this problem allowing programmers to organize computations not over individual computers, but over entire clusters. Pushing the data word by word between the CPU and the memory is no longer a concern in the framework.

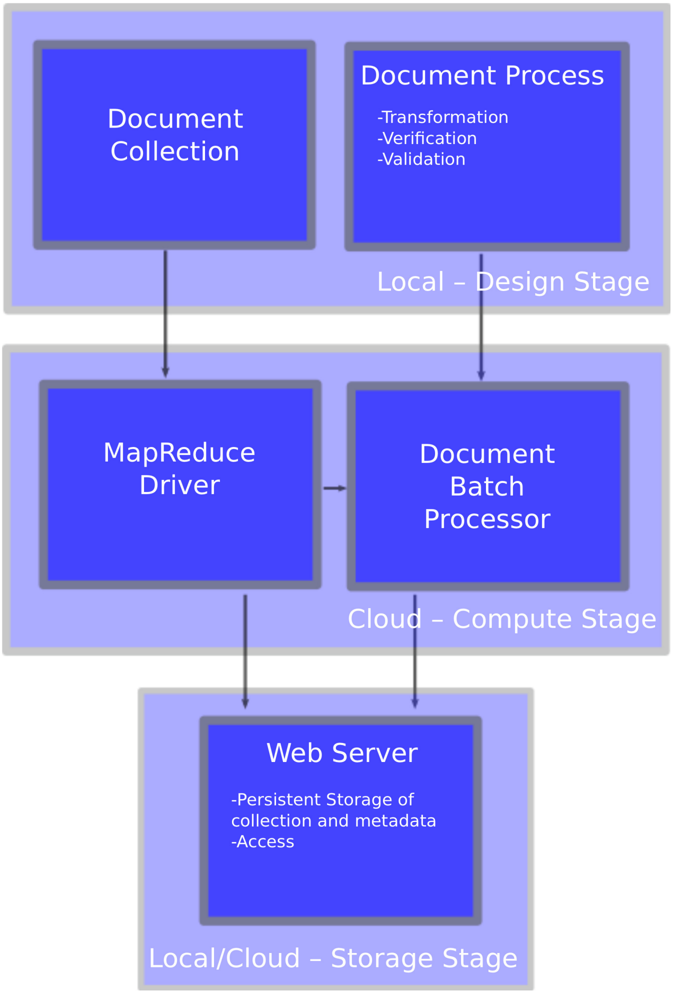


Fig. 2. Architectural overview of XML processing in the cloud to support digital preservation

IV. Proposed Architecture: XML workflow in the cloud

We believe that enterprises whose core business is not data conservation are going to have an increased demand for knowledge and expertise in logical preservation solutions to keep their data accessible. These organizations are often hardly aware of changes in their technological environment. This can have serious effects on their long-term ability to access their highly valuable digital assets. A flexible document processing pipeline with an associated designer, and the ability to perform the transformations in the pipeline in the cloud are essential for local digital preservation.

The MapReduce framework, while hiding the complexity of the underlying architecture, forces the user to think in terms of map and reduce operations. When it comes to document processing, it is difficult to divide a single document to smaller pieces. The unit for the map operation is therefore a single XML document.

This approach integrates well with document pipelines and document validation checkpoints modeled by Xeproc which allows one to define and design document processes while producing an abstract representation that is independent of the implementation. A workflow designed for a document is invoked from the map routine of a MapReduce job, with the identity operator acting as the reduce function. The output is the transformed document which then may undergo further processing in subsequent MapReduce jobs.

To implement the above approach, we choose Apache Hadoop, which is a software framework that supports data-intensive distributed applications. Hadoop is essentially a MapReduce implementation (the MapReduce engine) combined with a distributed file system (HDFS). We are not measuring the scalability of the software for distributed computing, hence we opted for Hadoop as stable and widely used tool, and did not consider other options.

The output of an XML processing pipeline is a collection of XML documents in METS format, as mentioned in Section II-B. Going beyond digital preservation, further automatic content extraction may be performed at document or collection level. Such further processing may be used to support digital curation (see also Section VII). Indexing and machine learning algorithms are likely to require a different XML input format. Interfacing between the document processing pipeline and further steps can be done with simple XSLT transformations.

V. Discussion

A. Implementation

The implementation relies on the decoupling of the MapReduce framework from the XML processing pipeline, and since there are no internal dependencies for the processes, the workload is naturally parallel. A process designed in Xeproc is exported via the EMF interface to Python, and the process is executed on individual documents that are mapped out to the nodes in the cloud by a relatively simple MapReduce driver.

For local experiments, we used a workstation with 24 GB of main memory, one quadro-core Intel Xeon E5620 CPU with two logical units in each core and 2 TB of storage, running in a 64-bit environment. For cloud computations, we used Amazon Web Services (AWS). It is possible to run Hadoop on Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage

Service (S3). While Amazon Web Services is undoubtedly the market leader in cloud computing, other providers, including at least one open source solution, exist. We decided in favor of AWS due to the maturity of its products.

The Python module requires bootstrapping across all instances in the cluster. This involves copying the Xeproc player from S3 to the local drive, then installing modules needed to run it.

We use two types of Amazon EC2 instances: small standard instances (m1.small) and large standard instances (m1.large). The former consists of 1.7 GB of main memory, one EC2 Compute Unit (one virtual core with one EC2 Compute Unit), 160 GB instance storage (150 GB plus 10 GB root partition), and the software architecture is 32-bit. A large instance includes 7.5 GB of main memory, four EC2 Compute Units (two virtual cores with two EC2 Compute Units each), 850 GB instance storage, and the platform is 64-bit.

B. Document processing pipeline

Demonstrating the capabilities of Xeproc, we use a process which recognizes and extracts the table of contents of a document [13], [14]. This pipeline is composed of the following steps (see also Figure 1):

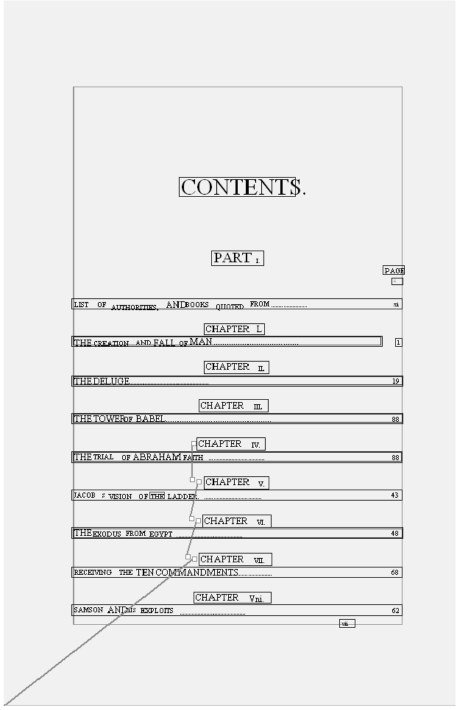
- Entry-level PDF to XML converter[2] for PDF files;
- Ad-hoc XSLT transformations for XML files;
- Page header and footer recognition;
- Text reading order reconstruction and paragraph segmentation;
- Caption detection;
- Table of contents analysis.

The first steps consist in recognizing and deleting document elements which can introduce noise during the table of contents analysis: page headers and footers are recognized and deleted in order to reduce noise (running titles); captions are deleted to eliminate tables of figures as potential table of contents. The text reading order and paragraph segmentation step generates a proper content flow which is used by the table of contents component. Finally the table of contents is extracted, and titles in the document body are marked up as heading elements (Figure 3).

Validation checkpoints are associated with each step. One type of validation aims at validating the step output against an XML Schema. Others perform more specific validations triggering errors or warnings and using XSL Transformations (especially XSLT 2.0) or ad hoc engines. For instance, an XSLT 2.0 validation detects overlapping paragraphs after the paragraph segmentation step and triggers a warning which points out some difficult or unexpected content layout. The user interface allows for visual inspection of the list of errors and warnings.

C. The Dataset

The collection being studied is a collection of doctoral theses hosted by the German National Library[3]. At the point of downloading the collection, it contained 94,437 theses. The total volume of the PDF files is above 500GB.



CONTENT$.

PART I.

PAGE

vii.

Fig. 3. An example of an identified table of contents in a PDF file [14]

The accompanying metadata is in Dublin Core format, which is a set of elements providing a small and fundamental group of text elements using only fifteen fields through which most resources can be described and cataloged.

The collection is multilingual. Approximately 75 % of the collection is in German, and over 20 % are in English. The presence of other languages is almost negligible.

D. The overhead of using a MapReduce framework

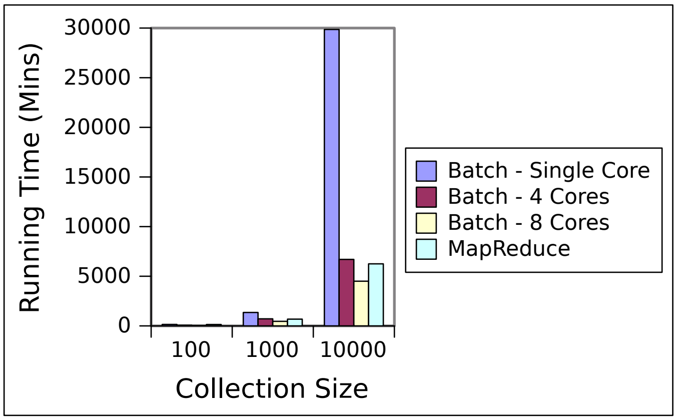


Fig. 4. Comparison of running times on a single local node

On a small subset of a hundred documents, MapReduce performance is close to single-core batch processing (Figure

[2] http://sourceforge.net/projects/pdf2xml/

[3] http://deposit.d-nb.de/index_e.htm

4). This is probably due to the overhead of setting up the framework. For larger collection sizes, MapReduce traces the quadro-core performance. This is expected, since we see three to four map tasks running simultaneously.

Load balancing is not optimized for the batch processor, as it simply divides up the collection to tasks of approximately equal size. The size, however, does not correspond well with the actual time spent on computation. This gives a slight advantage to MapReduce, since it has a sophisticated mechanism for load balancing. This is not critical when comparing with the quadro-core performance, as the differences between the finishing times of individual threads are marginal for four concurrent threads. The differences in running eight processes are more significant, the slowest thread finishes 1.6 times slower than the fastest. Hence the eight-thread performance is not directly comparable with that of MapReduce.

E. Running time in the cloud

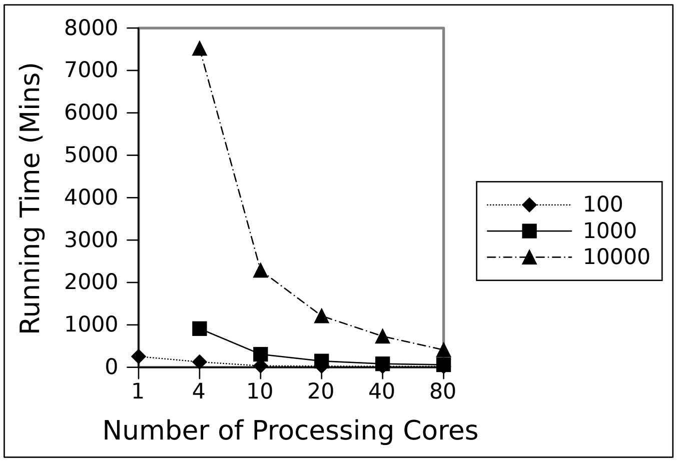


Fig. 5. Comparison of running times with different collection sizes

Before we compare the running times of executing a pipeline locally versus in the cloud, we must note some changes that are required for scaling out. The default configuration for the EC2 m1.small and m1.large instances has 1.7 GB main memory with 900 MB swap, and 7.5 GB main memory with no swap, respectively. As the memory requirements on the single local node hinted, this is not sufficient for running several processes simultaneously. An m1.small instance runs two child processes for MapReduce tasks, and and m1.large instance launches four. To accommodate the occasional peak memory usage, 4 GB of swap memory was configured for both type of instances. We observed a maximum of 2.1 GB swap memory used, which is a surprising result, since local single-node runs saw up to 8 GB of memory usage for a single Python process.

The document processing pipeline is not only memory-intensive, but is also demanding on the CPU. Since the Python interface does not comply with the stream interface of Hadoop, a map task which spawns the Python process may not be able to send "heartbeat" signals for long periods of time. These heartbeats signals are required by the framework, and if they are not received periodically, the task is considered failed. To avoid this, we removed the need for these signals. As the Python module always terminates, a permanent lock-up will not occur. It may happen that several pipelines require long computations simultaneously – a larger number of nodes will balance the load better.

The times shown in Figure 5 do not include the time needed to launch an instance or a cluster. This depends on the demand, the type of instances launched, and the number of instances requested. For a cluster of m1.large instances the starting time alone can be ten minutes or more. We focus our attention on the actual execution time of the document processes. The time and cost of moving data in and out of the cloud is negligible compared to the exectution time.

The closest equivalent to our local node is a single m1.large instance. The running times are marginally better on the local node for collection size above a hundred documents (75 % and 86 % of the running time compared to the single quadro-core cloud instance). This is in line with our intuition, as the local node has four times more main memory and does not have to rely on swap memory at all. However, Hadoop configuration is more fine-tuned for the virtual instances, and that may cause the better performance.

On small collections, there is little to gain by scaling to a high number of nodes or processing cores. As the collection size increases, it makes more and more sense to use a larger cluster. With a twenty-instance claster of m1.large nodes, the running time drops from two days to just seven hours. Cost, however, becomes an important aspect in choosing the correct cluster size.

F. Cost analysis of computing and storage

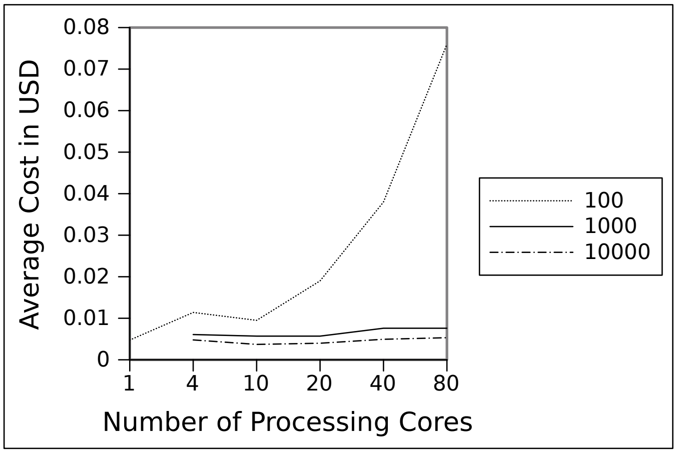


Fig. 6. Comparison of average cost of computations with different collection sizes

The price analysis is based on instance pricing in the EU West region (Ireland) as of September 2009. The prices indicated do not include VAT. The pricing for compute instance does not include a small fixed charge for using the MapReduce framework, it indicates the cost of using EC2 instances only.

Striking an optimal average price depends largely on the collection size. Since full hours have to be paid for partial hours as well, small data sets are the cheapest to process on a single or a few m1.small instances (Figure 6).

The scenario is different for larger collections. They tend to have a minimum average cost in the middle range. For a thousand documents, the lowest cost is US$0.0057 per document in clusters of ten or twenty instances. For ten thousand documents, the lowest average price is US$0.0037 with a cluster of ten m1.small instances. This also shows a trend the larger the collection the lower the average cost per processed object.

Beyond the compute instances, we also look at persistent storage in the cloud as an alternative to local hosting. Amazon S3 charges US$0.150 per gigabyte per month for the first fifty terabyte of data. To store the full collection, it would amount to approximately US$80.00 per month. This, however, is not the total cost of storage, as the access requests also cost US$0.01 per thousand. Depending on the purpose of the collection, this may prove to be a substantial sum. Public access to the collection stored in S3 would also need third-party solutions, further increasing the cost.

VI. Related work

Data grids provide several functions required by digital preservation systems, particularly when massive amounts of data must be preserved [1], offering a distributed infrastructure and services that support applications that deal with massive data collections stored in heterogeneous distributed resources. Focusing on the technical aspects first, grids are built using middleware software making fundamental aspects such as file management, user management and networking protocols, completely transparent. DP in a grid environment, however, suffers from a number of threats. These threats include component failures such as media faults, hardware faults, software faults, communication faults, network services failures. These faults are assumed as part of the normal operation in a MapReduce framework, and are addressed at lower level, application developers do not have to deal with them directly. Other threats in a grid environment include organizational issues, such as management failures and economic failures. These can be considered at a higher organizational level through preservation policies that include duplication of resources and integrity checks among others. With a cloud provider involved, these policies can be more difficult to enforce.

As part of our efforts in SHAMAN, we packaged Xeproc pipelines and components into standalone Python programs that can be deployed on any node of a grid. In the project, we actually targeted iRODS[4] for storage with Cheshire[5] used for full-text indexing. IRODS stands for Integrated Rule-Oriented Data System, and it is a data grid software system with applications in digital libraries, persistent archives, and real-time data systems. IRODS management policies (sets of assertions these communities make about their digital collections) are characterized in iRODS Rules and state information. At the iRODS core, a Rule Engine interprets the Rules to decide how the system is to respond to various requests and conditions. This ability to execute rules conditionally, and to define multiple rules implementing alternative means of achieving the same goal, provide a degree of flexibility that hold great promise for implementing automated digital curation and preservation applications [15]. The other component, Cheshire, is an XML search engine including support for XML namespaces, unicode, and a distributable object oriented mode. Cheshire proved to be a fast engine for digital libraries in a distributed grid environment [16], and integrates well with iRODS [**?**]. The present work is a viable alternative to the iRODS approach to digital preservation in the cloud. Further processing, such as indexing, may also be performed in the cloud.

Digital preservation requires persistence and, according to [17], the highest degree of persistence means fully persistent materials that enable high confidence for ongoing preservation and access. To see how cloud computing supports the cause, it is important to make a clear distinction between the cloud provider and its services and the MapReduce framework that enable an efficient use of the services. The technical issues were discussed in Section III. Here we would like to address the issues pertaining to the cloud provider. The underlying benefit of cloud computing is shared resources, which is supported by the underlying nature of a shared infrastructure environment. A service-level agreement (SLA) records a common understanding about services, priorities, responsibilities, guarantees, and warranties. Service level agreements span across the cloud and are offered by service providers as a service based agreement rather than a customer based agreement. Measuring, monitoring and reporting on cloud performance is based upon an end user experience or the end users ability to consume resources. The SLA may also specify the levels of availability, serviceability, performance, operation, or other attributes of the service. The “level of service” can also be specified as “target” and “minimum”, which allows users to be informed what to expect. In the context of DP, if the collection is preserved in the cloud, SLA has to explicitly define the persistency of storage. Cloud providers may have fundamentally different SLAs, which makes comparison difficult. An ongoing European Union funded Framework Programme 7 project, SLA@SOI, is researching aspects of multi-level, multi-provider SLAs within service-oriented infrastructure and cloud computing [18]. The project will eventually deliver predictability, dependability, and transparency in SLA management [19], which will make it easier to choose a cloud provider suitable for DP purposes.

VII. Future Work

Another aspect of the DP problem area is digital curation which is being increasingly used for the actions needed to add value to and maintain these digital assets over time for current and future generations of users [20]. Digital curation naturally

[4] https://www.irods.org/

[5] http://www.cheshire3.org/

involves the preservation of collections, and also entails the semantic and ontological continuity and comparability of the collection content. Document and collection-level metadata and metadata are crucial to support the goals of digital curation.

We anticipate that in the tested framework, machine learning algorithms can be useful to support further metadata extraction. This would involve indexing the outcome of a document processing pipeline, either the full-text documents or the extracted features. Once indexing is performed, various clustering, classification, and similar algorithms can be deployed to add metadata to individual documents or to the entire collection.

When choosing the machine learning component, one has to make a trade-off. Tools such as RapidMiner[6] enable designing a document indexing and machine learning in a similar fashion to Xeproc XML processing pipeline designer: they capture the intent behind the learning process and enable the preservation of the processing pipeline. However, this approach will not integrate readily with the MapReduce framework. On the other hand, machine learning libraries developed for Hadoop, such as Mahout [21], make designing the process difficult. Selecting the right tool that supports DP is an open research issue.

VIII. Conclusion

With a paradigm shift in the making toward a service-oriented architecture, digital preservation is one of the areas to benefit from this change. Considerations suggest that especially small organizations should welcome the turn as an attractive upcoming solution to their related problems. To test the feasibility of this assumption, we tested the Xeproc XML workflow designer in a cloud processing environment to show that the process is smoothly running.

The competitive aspects of the new technology, including its cost assessments, are too early to address but as shown by parallel experiments on a high performance workstation, memory and CPU requirements of DP are quite considerable and may be beyond reach for several memory institutions. As first results by cloud computing suggest, even a single node m1.large instance has a performance close to that of a decent workstation which makes running Xeproc possible for everyone. Moreover, a single workstation may be more expensive to rent for a longer term than running the same calculations on a cluster, as shown in the average cost diagrams. By picking the cloud configuration with the lowest estimated average cost, bulky DP jobs can be done at an affordable price and with flexibility beyond that of fixed resources.

Therefore we believe that the architecture outlined in this paper advances the state-of-the-art in digital preservation for the following reasons:

- The procurement of an expensive server or a grid can be replaced by service-level agreements with the cloud provider;
- The flexibility is unprecedented in terms of scale and in terms document process design;
- Ad-hoc peak computations that are typical in document format migration are easily addressed;
- Persistent storage in the cloud is a viable alternative to local servers.

IX. Acknowledgment

We would like thank our collaborators at the University of Liverpool for their help in developing the original grid-based iRODS implementation.

The SHAMAN large-scale integrating project is co-funded by the European Union (Grant Agreement No. ICT-216736).

References

[1] J. Barateiro, G. Antunes, M. Cabral, J. Borbinha, and R. Rodrigues, "Using a grid for digital preservation," in *Proceedings of ICADL-08, 11th International Conference on Asian Digital Libraries: Universal and Ubiquitous Access to Information*, Kuta, Indonesia, December 2008, pp. 225–235.

[2] J. Barateiro, G. Antunes, J. Borbinha, and P. Lisboa, "Addressing digital preservation: Proposals for new perspectives," in *Proceedings of InDP-09, 1st International Workshop on Innovation in Digital Preservation*, Austin, TX, USA, June 2009.

[3] F. Engel, C. Klas, H. Brocks, A. Kranstedt, G. Jäschke, and M. Hemmje, "Towards supporting context-oriented information retrieval in a scientific-archive based information lifecycle," in *Proceedings of Cultural Heritage online. Empowering users: an active role for user communities*, Florence, Italy, December 2009, pp. 135–140.

[4] T. Jacquin, H. Déjean, and J.-P. Chanod, "Xeproc©: A model-based approach towards document process preservation," in *Research and Advanced Technology for Digital Libraries*, ser. Lecture Notes in Computer Science, M. Lalmas, J. Jose, A. Rauber, F. Sebastiani, and I. Frommholz, Eds., 2010, vol. 6273, pp. 538–541.

[5] P. Innocenti, S. Ross, E. Maceciuvite, T. Wilson, J. Ludwig, and W. Pempe, "Assessing digital preservation frameworks: the approach of the SHAMAN project," in *Proceedings of MEDES-09, 1st International Conference on Management of Emergent Digital EcoSystems*, Lyon, France, October 2009, pp. 412–416.

[6] J. Borbinha, "SHAMAN: Sustaining heritage access through multivalent archiving," *ERCIM News*, vol. 80, 2010.

[7] D. Tidwell, *XSLT: Mastering XML Transformations*. O'Reilly Media, Inc., 2007.

[8] M. Cundiff, "An introduction to the Metadata Encoding and Transmission Standard (METS)," *Library Hi Tech*, vol. 22, no. 1, pp. 52–64, 2004.

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

[10] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proceedings of OSDI-04, 6th International Symposium on Operating Systems Design & Implementation*, San Francisco, CA, USA, December 2004.

[11] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.

[12] K. Skinner and M. Schultz, *A Guide to Distributed Digital Preservation*. Educopia Institute, 2010.

[13] H. Déjean and J.-L. Meunier, "On tables of contents and how to recognize them," *International journal on document analysis and recognition*, vol. 12, no. 1, pp. 1–20, 2009.

[14] H. Déjean, "Numbered sequence detection in documents," *Document Recognition and Retrieval XVII*, vol. 7534, no. 1, pp. 753 405–12, 2010.

[15] M. Hedges, A. Hasan, and T. Blanke, "Management and preservation of research data with iRODS," in *Proceedings of CIKM-07, 1st Workshop on CyberInfrastructure: Information Management in eScience, in conjuction with 16th Conference on Information and Knowledge Management*, P. Mitra, C. Giles, and L. Carr, Eds., Lisbon, Portugal, November 2007, pp. 17–22.

[16] R. Larson and R. Sanderson, "Grid-based digital libraries: Cheshire3 and distributed retrieval," in *Proceedings of JCDL-05, 5th Joint Conference on Digital Libraries*, Denver, CO, USA, June 2005, pp. 112–113.

[6] http://rapid-i.com/

[17] W. LeFurgy, “Levels of service for digital repositories,” *D-Lib Magazine*, vol. 8, no. 5, 2002.

[18] M. Comuzzi, C. Kotsokalis, G. Spanoudakis, and R. Yahyapour, “Establishing and monitoring SLAs in complex service based systems,” in *Proceedings of ICWS-09, 7th International Conference on Web Services*, Los Angeles, CA, USA, July 2009, pp. 783–790.

[19] T. Metsch, A. Edmonds, and V. Bayon, “Using cloud standards for interoperability of cloud frameworks,” SLA@SOI, Tech. Rep., 2010.

[20] N. Beagrie, “Digital curation for science, digital libraries, and individuals,” *International Journal of Digital Curation*, vol. 1, no. 1, pp. 3–16, 2006.

[21] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in Action*. Manning Publications Co, 2010.