2013 IEEE International Conference on Cloud Engineering

Scaling SeerSuite in the Cloud

Pradeep Teregowda
Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802, USA
pbt105@psu.edu

C. Lee Giles
Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802, USA
giles@ist.psu.edu

***Abstract*—The SeerSuite digital library search engine framework is used to build tools such as CiteSeer^x. It includes a complex metadata extraction system capable of extracting elements, such as author name, title, citations and citation contexts that are crucial bibliometric data and for building a citation graph. The workload faced by the exractor is dynamic in nature and this variability makes CiteSeer^x attractive for hosting in a cloud computing environment. Given its application binary dependencies and its reliance on a specialized infrastructure, the current extractor has several limitations. These limitations motivated the design and implementation of the metadata extraction system proposed in this study. A message oriented middleware architecture is used with a publish/subscribe pattern to build a scalable, flexible system that can be deployed across a range of cloud infrastructure. To demonstrate the broad applicability of the proposed system, we evaluate it in terms of its reference implementation across different scenarios of deployment and in regard to its scalability.**

***Keywords*-Information Retrieval, Cloud Computing, Message Oriented Middleware, Information Extraction.**

I. Introduction

A framework for building digital library search engines, SeerSuite [1] has several features in common with information retrieval systems such as search engines, digital libraries, and repositories in the form of index, crawlers and storage. SeerSuite systems index large quantities of information; for example, one such system CiteSeer^x [1] indexes over 2 million documents and 40 million bibliographic entities and has a repository that exceeds 4 TB in size. A study of SeerSuite in the context of cloud infrastructure and migration is, therefore, relevant to a wide cross section of information retrieval systems. Several aspects of existing information retrieval systems such as design, the lack of support for scaling, and the complexity of the code present challenges in regard to scaling SeerSuite and migrating it to the cloud. In this paper, we address these challenges through designing and implementing an information extraction system that can be effectively scaled and hosted using cloud infrastructure.

Information extraction and document processing systems are popular components hosted in the cloud [2], [3]. The availability in the cloud of distributed computing frameworks such as Hadoop [4] have enabled information extraction systems to be scaled in order to process larger collections and datasets than had previously been possible. In the present paper, we scale the information extraction system in SeerSuite by taking advantage of cloud computing services available across cloud infrastructure providers such as Windows Azure [2] and Amazon EC2 [3].

We identify the limitations of the existing metadata extraction system in SeerSuite and present a design for a new extraction framework. We use message oriented middleware to distribute the metadata extractor across local and cloud infrastructure. Through an examination of the performance of the system in different deployment scenarios and addressing the scalability of the system using message oriented middleware in the cloud, we demonstrate the feasibility and scalability of the proposed design.

This paper begins with a discussion of the background and related work in section II followed by a description of SeerSuite and its design in section III. In section IV, we present our proposal in the context of descriptions of our proposed metadata extraction system and of the present system in SeerSuite, including a discussion of the limitations of the latter in section IV. We then discuss a reference implementation of our proposed framework and evaluate our implementation in section V. A discussion of distributed computing and message oriented middleware is provided in section VI and a discussion of the extractor's workflow is presented in section VII. We discuss different scenarios relating to deploying the metadata extraction system in section VIII and then describe scaling the system across instances in section IX. A discussion on securing the system is provided in section X. Lessons learned are discussed in section XI, and directions for future work are discussed in section XII.

II. Background and Related Work

Exploiting cloud infrastructure and cloud models has been discussed in the context of digital libraries and information retrieval systems [5], [6]. In particular, researchers have explored the ways in which the economics and decision

[1] http://citeseerx.ist.psu.edu

[2] http://www.windowsazure.com/en-us

[3] http://aws.amazon.com/ec2

978-0-7695-4945-3/13 $26.00 © 2013 IEEE
DOI 10.1109/IC2E.2013.41

making [7], [8], [9] relating to cloud migration are connected to the availability of cloud infrastructure and they have also discussed migrating systems and particularly scientific computing and information retrieval systems as they function in the cloud. Specifically, [10] focuses on migrating an open source software framework with an emphasis on the technical aspects of such a migration and makes recommendations, accordingly. Whereas [11] provides a discussion of data hosting. These studies focus on existing applications and aspects of the application to be migrated to the cloud. In this regard, the present study breaks new ground by considering moving a specific component and scaling a framework in the cloud.

The task handled by the extractor at the level of the document is embarrassingly parallel. In this regard, a discussion of the use of distributed computing frameworks available in the cloud is particularly relevant. We considered the use of the MapReduce framework Hadoop [12] for distributing the computing across multiple instances. However, the input format of the extractor is a PDF document which would require serialization. In addition the extractor requires model files, dictionaries and gazetteers which must be accessible to each instance.

The use of high performance computing, grid or specialized architectures [13] and resources would restrict the ways in which the extractor could be used and require the user to build instances in order to gain access to similar resources.

Message oriented middleware [14], [15] offers several attractive features, for distributing the metadata extractor. It allows components to operate in isolation from each other: for example the crawler can operate in isolation from the extraction system, and the extraction system in isolation from the ingestion. Additionally, message-oriented middleware can be deployed across a number of architectures and heterogeneous systems.

In particular, we were influenced by the use of grid based service oriented middleware in WSPeer [16] for message exchanges, Meghdoot [17] for content based publish/subscribe in Peer to Peer environments. In addition, we drew on the discussion on software engineering and middleware presented in [18]. Message oriented middleware is already part of SeerSuite and CiteSeer^x deployment in a limited way, i.e., in the form of a messaging framework between components of the crawler.

III. SeerSuite

The SeerSuite information retrieval framework was been built, using service oriented architecture and the framework's design features loose coupling among modules. The framework is available on the web under an open source license [4] in order to encourage users to adopt SeerSuite or its modules in their work. SeerSuite collections are built by crawling the web for academic and scientific documents. Extensive metadata are extracted automatically from documents and citations in documents, indexed, and made available to users of the collection. The citation metadata are used to build a citation graph that enables documents to be linked and to be ranked based on citations. This linking of documents and citations allows users to navigate the collection using citations. Features such as disambiguating author records and, indexing embedded objects such as tables are continually being developed and made available to users. Many of these features provided by SeerSuite draw on metadata extracted from documents.

The components of the framework are designed to be *portable* such that they can be deployed across different systems, *reliable* such that they can function with and recover from failures of resources and supporting modules, and, *flexible* such that they can be deployed as part of other systems or infrastructure such as the cloud.

The SeerSuite framework (architecture shown in Figure 1) consists of components for extracting and, ingesting documents drawn from crawling web and an interface for interacting with the user. The components are loosely coupled and interact with each other using service oriented methods and data access objects.

The acquisition system includes a crawler, which is responsible for discovering and fetching documents on the web. An information/metadata extraction system operates on the documents retrieved from the web. After the metadata have been extracted, the document and related metadata are ingested into the system, by adding information extracted to a database, adding files to a repository, and generating a citation graph. The metadata and the citation graph are made available to users through the web interface. Separate processes are responsible for updating the full text index and generating statistics such as a citation based ranking of authors, papers, and citations.

The role of the metadata extraction system in such an application is crucial. We, therefore, describe in detail the metadata extractor and extraction workflow in SeerSuite.

IV. Metadata Extraction

The metadata extraction system is part of the document acquisition process. The series of steps beginning with crawling the web to find a relevant document and ending with ingesting a document into the system is broadly labeled as metadata extraction. Each document is processed individually, and each step in the extraction process can be mapped to a distinct function or module.

The workflow is illustrated as part of the SeerSuite architecture in Figure 1. The web is crawled to identify relevant documents (step 1), documents are processed by a document converter, which covert PDF/PS document into plain text. At this point the document is split into two sections: the body and the references (step 2), the body and the references

[4] http://sourceforge.net/projects/citeseerx

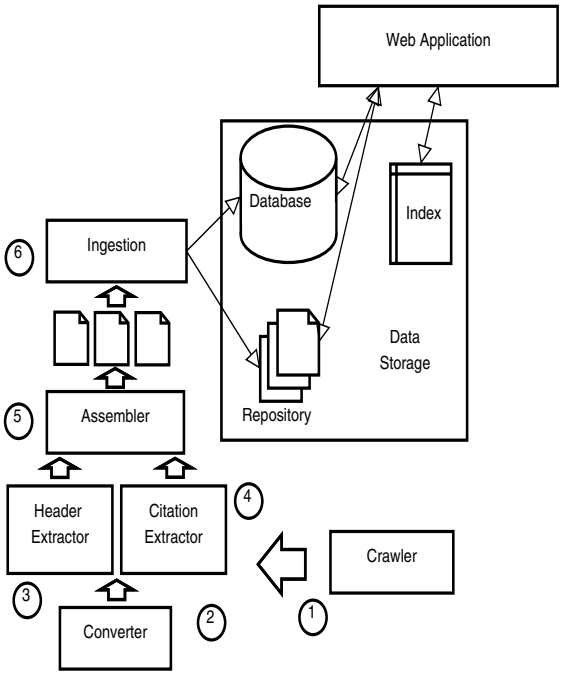

Figure 1. SeerSuite Architecture

are processed by the header extractor (step 3). The body and the references are processed by the citation extractor (step 4). The extracted metadata are assembled (step 5). The extracted metadata are ingested (step 6). The workflow enables us to identify the flow of data through the process and to determine the dependencies such as the dependency of the classifier on the conversion and the dependency of the assembler on the classifier.

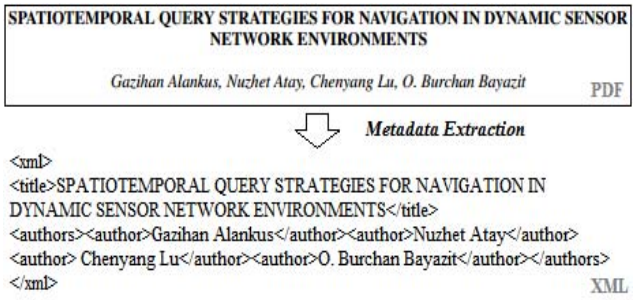

Figure 2. Metadata Extraction Sample

An excerpt of entities extracted as part of the information extraction procedure is shown in Figure 2.

A. Extractor Workload

We examine the source of documents serving as input to the extractor. A major source is the crawler. The workload of the metadata extractor is directly proportional to the number of the documents fetched by the crawler (Figure 3).

The number of documents fetched by the crawler vary by day reaching peaks of several thousand documents per day to zero on some days. The average number of documents fetched per day is 164 documents with a high degree of variance in the number of documents fetched. The lack of predictability makes it challenging to adopt workload prediction techniques such as those proposed in [19]. It is also worth noting that changes (e.g., the addition of features, or objects) to the extractor may require entire sets of documents both those identified through crawling and those ingested to be reprocessed.

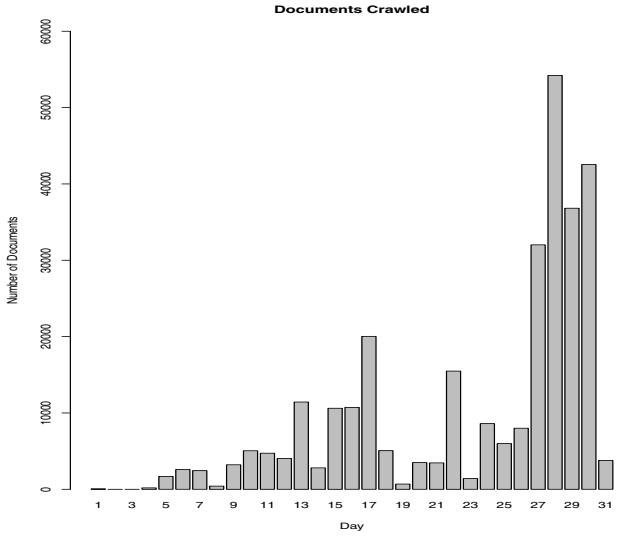

Figure 3. Documents Identified by Crawling the Web per Day

Other sources of documents added to the collection are user submissions and acquisitions from public repositories. User submissions vary daily, such that they do not generate a predictable number of documents. Public repositories might share parts of their collections or allow crawling of their entire collection. Repositories contribute to the depth of the collection, as many are focused on specific topics.

Constraining the crawler or user submissions by either restricting or scheduling crawls could undermine the freshness of the collection, and so draw adverse user feedback. Acquisitions from other repositories can be constrained, but this too could undermine the depth of the collection available to user.

These workload constraints and characteristics point to the fact that a deployment to the cloud and scaling would improve the freshness of a collection. Benefits of the pay as you go model would save both on cost and maintenance.

B. Present Implementation

The present metadata extraction system makes use of multiple open source applications and modules to process documents. The PDF document is converted into text using

a PDF to text converter such as PDFLib TET or PDFBox. The text thus converted, is filtered and identify documents of interest using regular expressions. This filtered document text is then divided into the body and reference sections. A header parser uses support vector machines [20], [21] to identify the title, author and abstract entities from the body. The body and the reference sections are then analyzed by a conditional random field based citation parser [22] in order to extract the citations and to establish the context of these citations. Finally the extracted entities are aggregated and assembled to generate a XML document, which includes the header metadata and the citation metadata.

The individual converters, filters and parsers are implemented as standalone modules, enabling the user to generate different workflows and extraction pipelines according to his needs. To process large collections, the system is scaled by being distributed across systems and data manually partitioned.

1) Limitations: The present metadata extraction system is subject to several limiting factors. Its complex code base is an obstacle to any refactoring; setting up the system, requires intimate knowledge of the machine learning methods and convertors, and adding new features and making improvements each require significant effort. The system's application dependencies prevent the extractor from being deployed across infrastructure such as across different file systems and operating systems.

In terms of the system's infrastructure requirements, the extractor requires shared storage in order to interact with other components of the SeerSuite framework. These interactions include accessing files fetched by the crawler and creating and modifying the files to be consumed by the ingestion system. And in order to scale the system, it is necessary to coordinate manually between instances. Due to the I/O bound nature of the extraction process, scaling up resources available to the system, does not result in a corresponding improvement in the system's throughput.

C. Proposed Framework

In order to improve the metadata extraction system, we designed and developed a framework for metadata extraction. Our design improves on the existing metadata extraction system. Specifically, the design uses a portable programming language and programming constructs to render the extractors implemented in this framework portable and extensible.

Figure 4 shows the workflow of an extractor implemented using the framework. This includes the PDF to text converter (label 1), the classifier(s) (label 2) with the associated model and gazetteer, and an assembler (label 4). A defined set of interfaces allows, modules to be developed independently of each other. The number of stages and the information available to each stage can be controlled by the class implementing the relevant interface. For example, the classifier could ignore features such as font and layout information provided by the converter and emulate the existing extractor.

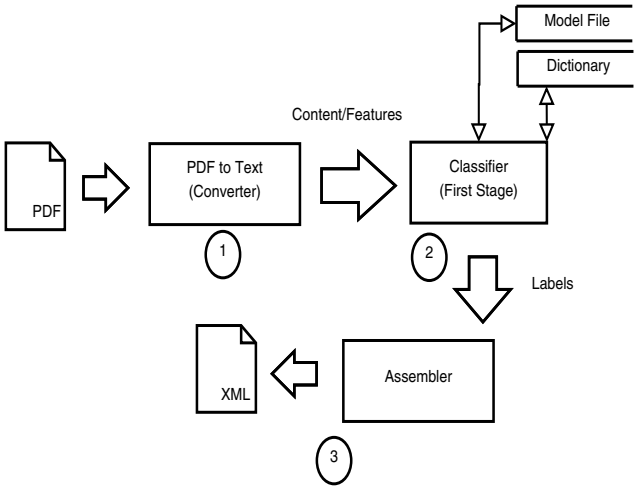

Figure 4. Proposed Metadata Extraction Framework

The proposed extractor include modules which provide services for document conversion, parsing and assembly of parsed metadata.

V. Reference Implementation

We developed a reference implementation as a guideline for developing extractors with the proposed framework. The reference implementation uses the PDFBox document conversion library, and extends the extraction capabilities to include features such as layout and font information along with the text of the PDF document.

We used a supervised machine learning method, a Random Forest [23] classifier trained on 4000 labeled lines across categories. The training set was generated and labeled manually. The set was labeled across categories such as authors, titles and citations. Open source libraries such as Weka [24] were used to build the classifier. Generated from training with labeled samples stored in a serialized format, the model was used by the classifier to label unlabeled lines.

The assembler organized the labeled lines into objects such as complete titles, and author names and order and linked related objects such as citations with their context. This process result in the generation of an an XML file containing metadata.

A. Evaluation

The performance of the reference implementation was evaluated by comparing it to the performance of the existing extractor.

1) Quality of Extraction: The quality of the extraction in terms of identifying various entities such as the title, author, and citations in the document was examined. To evaluate the quality, we manually examined metadata from

Extractor	Titles	Authors (Recovered)	Citations (Recovered)
Reference	84%	88% (86.05%)	98% (87.90%)
Existing	80%	90% (93.70%)	98% (79.99%)

Table I
QUALITY OF EXTRACTION

Infrastructure	Average Time (sec/doc)
Local	1.55
Amazon EC2	2.25
Azure	3.6

Table II
PERFORMANCE IN THE CLOUD

50 documents. From the metadata extracted by the reference and the existing extractor. we accurately identified the number of documents for which a particular entity was extracted and we did the same for entities such as authors and citations (where multiple authors and citations existed within a single document), On this basis, we found the percentage of entities extracted accurately (recovered). Results are shown in Table I

From the results, it is clear that the proposed reference metadata extractor improves on the existing method in regard to extraction of titles and citations; however, in regard to authors, the performance of the proposed extractor marginally worse than the existing extractor. An examination of the errors of the reference implementation shows that there were issues with identifying author names containing hyphens and Unicode characters, In addition certain entities such as university name and department were labeled as author names.

2) Throughput: We examine the throughput of the existing extractor and the reference extractor implementations with a set of 2498 documents drawn from CiteSeer^x. The average file size of the documents being processed was 700 KB of which 97% were below 2 MB.

In this case the existing extractor took an average of 3.32 seconds(s) to process a single document compared to the average 3.2(s) taken by the reference extractor. The experiments were performed on a dual core, 3 GB RAM system. Therefore, in regard to processing a single document, the proposed reference extractor performed better than the existing extractor.

The performance of the reference extractor is similar to that of the existing extractor in regard to throughput, where as there is room for improvement in extracting authors.

3) Performance in the Cloud: To prepare for deploying the metadata extraction system in the cloud, we evaluated the performance of the extractor over a small set of documents in both IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) cloud environments.

The extractor processed 120 documents all of which were located in the local storage/file system for this evaluation. The output of the extractor (XML) was also stored in the local file system. Other than changes made for packaging for specific cloud infrastructure no changes were made to the application code. In the case of Amazon EC2, the extractor was packaged as a jar file and a copy of the virtual machine, the model files and gazetteers were shipped to the instance. In the case of Azure, existing tools for packaging applications along with environmental parameter settings were used for packaging. Data in the form of PDF files were shipped along with the packages.

We used the Large Instance with a worker role in Azure and Extra Large Instance in Amazon EC2 and a similar system locally [5]. Results are shown in Table II.

The Amazon EC2 instance provides the closest performance to the local deployment, where as the performance on Azure indicates that there is scope to improve the performance of the system. The performance has improved by a factor over that of the existing extraction system; however, we can take advantage of, several aspects of the process to further improve on the throughput performance of the extractor.

Based on the discussion for the need to distribute the processing of documents and of the constraints faced, we decided to consider message oriented middleware. In the next section, we discuss message oriented middleware, its features in the context of distributing metadata extraction.

VI. MESSAGE ORIENTED MIDDLEWARE

We choose message oriented middleware as a mechanism for distributing the metadata extraction system. The use of the publish/subscribe pattern provides flexibility, loose coupling and scalability in the acquisition pipeline. Further new metadata extractors such as those for extracting algorithms [25] as subscribers can be added without refactoring. It is possible also to group extractors with topic exchanges without affecting existing components in the pipeline.

A. Wrapper/Adapter

The metadata extraction system does not by itself provide interfaces to interact with other components. For example, the system does not provide the ability to interact with storage systems such as Amazon S3 [6], WebHDFS, or a simple REST based system [7], as well as queue operations such as poll and queue methods.

Building methods to interact with various infrastructure in the metadata extraction itself would result in multiple multiple methods and greater complexity for a researcher interested in extracting metadata. Any modification to the infrastructure would entail refactoring the entire metadata extraction system. This is particularly limiting as we planned

[5] 1 CPU, 4 Cores, 16 GB RAM
[6] https://aws.amazon.com/s3

ID
URL (input)
Destination
Destination End Point (output)
Destination Type

Table III
MESSAGE TEMPLATE AT THE EXTRACTOR QUEUE

to deploy the extractor not just on the local physical infrastructure, but also across different infrastructure, abstractions. We used a wrapper or adapter pattern to provide methods to interact with storage systems, the queue and other SeerSuite components. By discarding the wrapper, the metadata extractor can function in the same way as the existing extractor.

The interface to the storage system makes it possible to make GET/POST requests to storage, and to use, queue and poll methods to interact with the middleware. Modifications to the wrapper or adapter can be made without any specialized knowledge of the metadata extraction system. In the same way, modifications to the metadata extractor can be made without specialized knowledge of the infrastructure being used.

The workflow of the extractor now begins with the wrapper enabled metadata extractor consuming messages from the exchange. The extractor can now fetch PDF documents stored in the virtual store and post XML files to the virtual storage produced during extraction through wrapper methods. Finally a completion message, a message for the ingestion system is placed in the relevant exchange through the wrapper.

B. Message Format

Generated by the crawler and consumed by the metadata extractor, the message is the primary means of communication between the various components. To communicate between the extraction and the ingestion processes, the metadata extractor generates messages for the ingestion system. The message format for communications from the crawler to the extractor includes a description of the resource to be processed in the form of a URL or a location in the physical storage. The destination, could represent either a local file location or a virtual/cloud storage location. A combination of the destination and the destination type (type of service such as REST/WS) allows us to decide where the output of the metadata extraction system should be located. As the end point is embedded in the message, the placement of the input and output can be changed without requiring a change in the deployment. The format for the message is shown in Table III.

VII. INGESTION WORKFLOW WITH WRAPPER

The use of the wrapper enables the metadata extraction system to interact with message oriented middleware A typical deployment is shown in Figure 5.

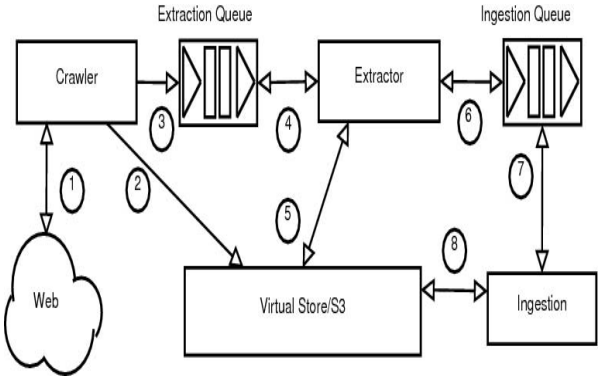

Figure 5. Deployment Architecture

The workflow of the process proceeds as follows. The crawler on a successful identification and fetching of a document from the web (steps 1 and 2), posts a message to the extractor queue (step 3). This message either points to the URL of the document or the URL in the virtual storage. The extractor then consumes messages (step 4) from the queue and processes the document (step 5), generating an ingestible document. A message of completion (step 6) is posted to the ingestion queue, which is then handled by the ingestion system (steps 7 and 8).

The use of middleware allows us to isolate individual modules from each other. A failure of the crawler or of the extractor will not affect the ingestion system. Though this is possible with the present implementation, the middleware provides coordination among many similar and or special purpose extractors. Asynchronous operation of the extraction module is now possible and allows greater flexibility in terms of scheduling the extraction process. These factors improve both the reliability and the efficiency of the system.

A. Input Output Costs

One hundred and twenty documents were hosted in a virtual storage system and processed both in physical systems and cloud instances, with the generated metadata stored in the virtual storage. A breakdown of the costs associated with accessing the queue, fetching documents from the storage, processing storing the output in the storage are provided in Table IV.

Component	Queue Access (s)	Fetch (s)& Processing (s)	Storage (s)
Physical	0.02	4.49	4.56
Amazon EC2	5.08	57.43	19.07

Table IV
TIME CONSUMED FOR I/O OPERATIONS

The costs of data access and storage of a document dominate the cost breakdown. These results indicate that the I/O

costs are particularly high and methods that use threading can improve the throughput of the system enormously. We therefore, adopted threaded workers for all further evaluation and processing metadata. To reduce startup times for each thread, We load the classifier before any threads are initialized and sharing the pre loaded classifier across threads.

VIII. Deployment Scenarios

With the ability to function in both cloud and local infrastructure, the extraction service allows a great deal of flexibility in deployment. We demonstrate some of these deployment possibilities and to examine the performance of the metadata extraction system in these scenarios. We deployed the extractor in a completely local environment, completely in the cloud and finally a hybrid combination of instances in cloud and local infrastructure.

A. Local Deployment

When a suitable number of extractors are available in the local infrastructure, the coordination provided by the middleware and storage abstraction by the virtual storage address concerns associated with distributing the extractor workload and the need to use specialized shared storage for operation (Figure 6). This approach, allows the user to automate the distribution of extraction workload among instances, to avoid manual partitioning, and scheduling of the extractor. Other scenarios in which such an approach would be relevant are in testing and where there are policy restrictions related to data exist.

B. Cloud Deployment

The use of cloud computing resources can greatly reduce the need to invest in physical resources, as dynamic provisioning allows us to scale to order, to meet workload demands. A cloud infrastructure based deployment can make use of the local messaging middleware and virtual storage. It can also be scaled by including multiple instances based on need. This deployment can also be used when the cost of operating in the cloud makes the use of cloud resources more attractive than operating on physical infrastructure.

C. Hybrid Deployment

In cases in which the local infrastructure handles a fixed set of documents, cloud infrastructure can be used to meet large workloads. This allows the application to be scaled without the need for additional physical infrastructure to handle peak load conditions, or to take advantage of features such as spot instances. The compute instances are hosted in the cloud with the input or output stored in a virtual storage or S3. The deployment is shown in Figure 7.

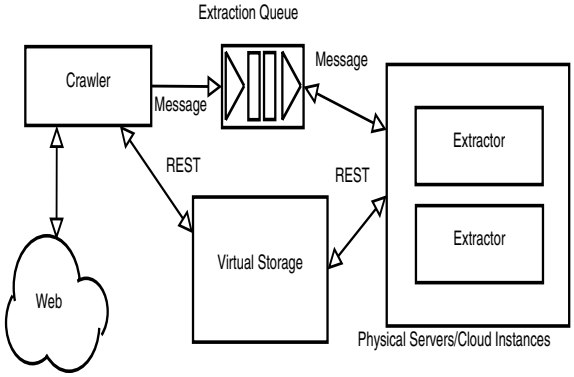

Figure 6. Physical or Cloud Deployment

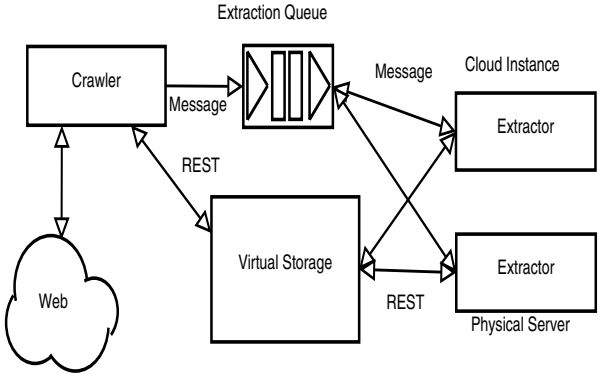

Figure 7. Hybrid Deployment

D. Evaluation

In order to study the performance of the metadata extractor in the deployment scenarios such as local physical infrastructure, cloud infrastructure and hybrid operations, we evaluated the performance of the extractor in these scenarios with Amazon EC2 and local infrastructure instances. We based our evaluation of the system on 2498 documents (discussed in section V) with two instances. In this case, the documents to be processed were stored in a virtual storage system and processed with local metadata extractor instances and or cloud based instances.

We used rabbitmq [7] which supports AMQP [8] with a publish subscribe topic exchange for local operations, For cloud based systems, the Amazon SQS system was used. The queue servers were hosted independently of the extractor. In case of AMQP, messages were published to the queue by the crawlers and consumed through the wrapper through the extractors.

Figure 8 shows the performance and range for each scenario. The relative performance of the cloud instance is worse than those of other deployments. This could be ex-

[7]http://www.rabbitmq.com
[8]http://www.amqp.org

plained by network latency issues between the local storage and the cloud instance. Both instance and local systems were sized to be similar [9]. The physical instances performed best, but there was significant variation in the results.

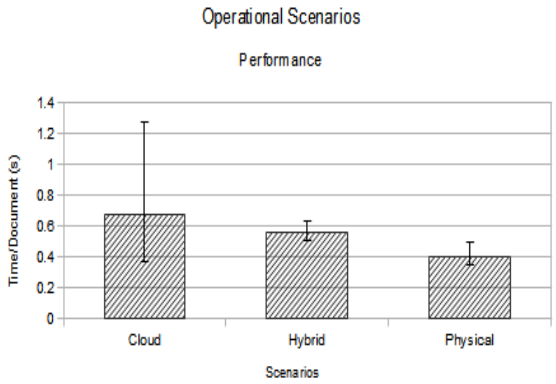

Figure 8. Performance in different scenarios

IX. Scalability

We studied the performance of the system across cloud instances to identify whether such a system could be scaled beyond a few instances and to determine whether there were any improvements to the throughput of the system. For this study 2498 documents (discussed in section V) were stored in a virtual storage system and processed by metadata extractor instances in the cloud. The output of these extractors were assigned to be stored to a virtual storage system. The extractor was packaged and deployed into the instances [10], and a pool of 10 threads was available at each instance for extraction.

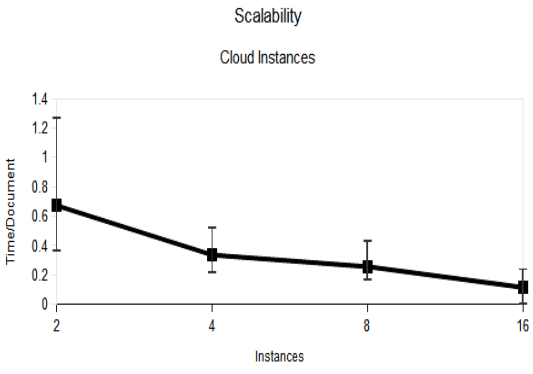

Figure 9. Scalability

The graph shown in Figure 9 illustrates the performance of the metadata extraction system using message oriented middleware and virtual storage across an increasing number of instances. From the graph it is clear that the metadata extraction system is scalable and the throughput of the extraction system has increased significantly.

An advantage of distributing the system across instances along with threading is that it makes the system more reliable. The failure of one instance does not result in the failure of the entire extraction system.

Compared to the existing extractor, the distributed, metadata extraction system can process documents several factors faster. The scalability of the extractor allows us to process peak loads and large collections in less time, ensuring the freshness of the collection.

X. Security

Locating resources on physical infrastructure or in the cloud presents several issues related to information security. The proposed deployment does not include any features explicitly designed to secure the extraction system. However aspects of the system have been designed to allow for secure communications (with SSL between the storage, queue and clients) by taking advantage of the features already available features in the system and modules. To maintain the integrity of a message or transfer, a checksum would be included in the message. Confidentiality has a low priority given the public nature of the data being transferred, i.e., (documents crawled from public web sites).

Authentication mechanisms at the virtual storage, and the mechanisms that interact with them can be implemented as part of the wrapper used by the metadata extractor. This allows considerable flexibility in regard to how requests are handled.

A. Availability

Metadata extraction services can be substantially improved through the use of distributed systems, a publish/subscribe pattern and hardware redundancy across sites together with the use of hybrid deployment, persistent queues replication at the queue server and storage availability.

XI. Lessons Learned

In this section, we identify and explore some of the lessons learned from our experience building and scaling the metadata extraction system.

A. Scaling

In this study we used both threading and distribution across instances to scale the metadata extraction system. Though the I/O operations performed by the extractor suggest threading, the approach to threading needs to be studied carefully, as discussed earlier the size of the model files, dictionaries required us to share these across threads. A different choice of classifier(s) could very well remove this

[9] 4 Cores (8 ECU), 15 GB and 8 GB
[10] 4 Cores (8 ECU), 15 GB and 8 GB

constraint. In our case, in addition to the application profile, the limits placed by legacy infrastructure limited the size of thread pools available, the use of messaging middleware, allowed us to distribute the extractor across many systems and scale up to meet demands. The combination of threads and instances is an aspect of the system which can be tuned to meet infrastructure constraints and required throughput.

B. Packaging

The use of wrappers and portable programming language and methods greatly simplifies the deployment of the extractor in various cloud infrastructures. However, considerable amount of effort is still required to package the system for these deployments. The use of configuration files resolves many of these issues. In the case of PaaS systems, the placement of files, the availability of space in the application and data storage spaces might not be explicit and may require the application to explore the instance and the environment in which it is deployed to enable effective deployment.

C. Message Oriented Middleware

Substantial differences exist between different kinds of middleware such as those based on AMQP and those based on Amazon SQS. The need for a publish/subscribe pattern in our context, can be emulated by using a zero visibility timeout SQS queue. The issues with the order of messages, delays in message availability while not limiting, can be relevant to user experience. In our experiments we did not encounter these issues, however, monitoring the queue and the metadata extraction process allow tracking of message ordering and delays to be tracked. The immediate availability of client and support libraries reduced the time required to develop the wrapper and its methods for each of these middleware.

XII. Future Work

Avenues for further work include exploring the reference implementation as a guideline for improving the performance of the extraction system. This includes both the quality of the metadata extraction and the throughput of the system. With improvements to SeerSuite and the addition of new features such as a function to index objects embedded in documents e.g., figures algorithms and tables, the range and amount of metadata extracted could undergo significant changes.

Workload prediction would allow extraction services to be scheduled, however the variety of documents fetched by the crawler at any time, presents challenges. Simplifying and developing of the metadata extraction framework presents a definite step toward the goal of building an automatic ingestion system for SeerSuite, in which the document acquisition processes of crawling, extraction and ingestion are automated. However a more in depth study is required in order to establish how best to scale and schedule services. Beyond the framework and its workload, the ingestion system could be further improved in its ability to handle larger volume of documents.

Message oriented middleware is used in our study to provide scalability and coordination, but several features of such middleware need be explored further. Monitoring the queue and identifying documents that have not been successfully processed are accomplished by use of timeouts and manual examination of the output. Several approaches are available to resolve these issues, including the use of an approach similar to RPC by including acknowledgments and by introducing a monitoring agent system for the queue.

Though the security of the system has been discussed briefly, a more elaborate authentication mechanism along with access control would extend the capabilities of SeerSuite far beyond the existing set of features in regard to hosting and ensuring access to documents in the collection. Several issues, including those of user access, administration, and maintenance need to be addressed in this regard.

There are several cloud infrastructure, and resources that we have not discussed as part of this work. These include compute engines [11], storage and context aware cloud infrastructure [26]. A more exhaustive study would be helpful in identifying and exploiting a varied set of cloud infrastructure. Though we have considered a hybrid implementation with a combination of cloud and physical based instances, combination of instances in different cloud infrastructure also needs to be studied. We would also like to examine in more depth, distributed computing frameworks such as MapReduce. We hope to fully explore resources in the context of SeerSuite and information retrieval in more depth in the future.

XIII. Conclusion

We discussed scaling the SeerSuite framework and proposed a metadata extraction framework. We presented the workload of the metadata extraction system to, which support cloud hosting of the metadata extraction system. We implemented an extraction system for reference using the framework and examined its performance for quality and throughput. To further improve the performance of the extraction system, we discussed approaches to parallelizing the extraction process and the constraints of using each approach. We decided to use message oriented middleware to distribute the extraction process. We examined the performance of the extractor using middleware in different scenarios. and we also examined the scalability of the system with cloud infrastructure. Lessons learned from our experience were discussed and directions for future work were identified.

Various approaches to improving the metadata extractor, use of message oriented middleware, and the security of the

[11] http://www.openstack.org/software/openstack-compute

system were been discussed and should serve as motivation for research. The source code for this framework and the reference implementation will be made available as part of SeerSuite. With the metadata extraction framework and evaluation of the reference implementation we have demonstrated that the metadata extraction system can be built and scaled under the constraints imposed by the overall goals of the SeerSuite framework. There is also a need to improve the performance of each component of the SeerSuite framework.

ACKNOWLEDGMENT

We gratefully acknowledge funding received from from Amazon and NSF, which helped support this study. We also thank Bhuvan Urgaonkar for the insights and suggestions he provided.

REFERENCES

[1] P. B. Teregowda, I. G. Councill, R. Fernández, M. Khabsa, S. Zheng, and C. L. Giles, "Seersuite: Developing a scalable and reliable application framework for building digital libraries by crawling the web," in *Proceedings of USENIX WebApps*, 2010.

[2] R. Baumgartner, G. Gottlob, and M. Herzog, "Scalable web data extraction for online market intelligence," *VLDB Endowment*, vol. 2, no. 2, pp. 1512–1523, 2009.

[3] S. Blohm, *Large-Scale Pattern-Based Information Extraction from the World Wide Web*. KIT Scientific Publishing, 2010.

[4] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in *Proceedings of USENIX HotCloud*, 2010.

[5] R. Fox, "Library in the clouds," *OCLC Systems and Services*, vol. 25, no. 3, p. 156, 2009.

[6] F. Xiaona and B. Lingyun, "Application of cloud computing in university library user service model," in *Proceedings of ICACTE*, vol. 3, 2010, pp. V3–144.

[7] P. Teregowda, B. Urgaonkar, and C. Giles, "Cloud computing: A digital libraries perspective," in *Proceedings of International Conference on Cloud Computing*, 2010, pp. 115–122.

[8] A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, and P. Teregowda, "Decision support tools for cloud migration in the enterprise," in *Proceedings of International IEEE Cloud Computing*, 2011, pp. 541 –548.

[9] P. Teregowda, B. Urgaonkar, and C. Giles, "Citeseerx: a cloud perspective," in *USENIX HotCloud*, 2010.

[10] M. Chauhan and M. Babar, "Migrating service-oriented system to cloud computing: An experience report," in *Proceedings of IEEE International Conference on Cloud Computing*, july 2011, pp. 404 –411.

[11] A. Thakar and A. Szalay, "Migrating a (large) science database to the cloud," in *Proceedings of International Symposium on High Performance Distributed Computing*, 2010, pp. 430–434.

[12] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[13] M. Atkinson, J. van Hemert, L. Han, A. Hume, and C. Liew, "A distributed architecture for data mining and integration," in *International workshop on Data-aware distributed computing*. ACM, 2009, pp. 11–20.

[14] G. Banavar, T. Chandra, R. Strom, and D. Sturman, "A case for message oriented middleware," *Distributed Computing*, pp. 846–846, 1999.

[15] E. Curry, "Message-oriented middleware," *Middleware for communications*, pp. 1–28, 2004.

[16] A. Harrison and I. Taylor, "Service-oriented middleware for hybrid environments," in *Proceedings of ADPUC*, 2006, pp. 2–.

[17] A. Gupta, O. Sahin, D. Agrawal, and A. Abbadi, "Meghdoot: content-based publish/subscribe over p2p networks," in *Proceedings of ACM/IFIP/USENIX International conference on Middleware*, 2004, pp. 254–273.

[18] W. Emmerich, "Software engineering and middleware: a roadmap," in *Proceedings of ACM Conference on The future of Software engineering*, 2000, pp. 117–129.

[19] H. Li, W. Lee, A. Sivasubramaniam, and C. Giles, "Workload analysis for scientific literature digital libraries," *International Journal on Digital Libraries*, vol. 9, no. 2, pp. 139–149, 2008.

[20] H. Han, E. Manavoglu, H. Zha, K. Tsioutsiouliklis, C. L. Giles, and X. Zhang, "Rule-based word clustering for document metadata extraction," in *Proceedings of ACM Symposium on Applied Computing*, 2005, pp. 1049–1053.

[21] H. Han, C. Giles, E. Manavoglu, H. Zha, Z. Zhang, and E. Fox, "Automatic document metadata extraction using support vector machines," in *Proceedings of JCDL*, 2003, pp. 37–48.

[22] I. Councill, C. Giles, and M. Kan, "Parscit: An open-source crf reference string parsing package," in *LREC*, vol. 2008, 2008.

[23] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[24] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "The weka data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[25] S. Bhatia, S. Tuarob, P. Mitra, and C. Giles, "An algorithm search engine for software developers," in *Proceedings of International workshop on Search-driven development: users, infrastructure, tools, and evaluation*. ACM, 2011, pp. 13–16.

[26] J. Weissman, P. Sundarrajan, A. Gupta, M. Ryden, R. Nair, and A. Chandra, "Early experience with the distributed nebula cloud," in *International workshop on Data-intensive distributed computing*. ACM, 2011, pp. 17–26.