Scalable and Distributed Processing of Scientific XML Data

Elif Dede [1], Zacharia Fadika [2], Chaitali Gupta [3], Madhusudhan Govindaraju [4]

Grid and Cloud Computing Research Laboratory,
Department of Computer Science, SUNY Binghamton, NY 13902
[1] edede1@binghamton.edu, [2] zfadika1@binghamton.edu, [3] cgupta1@binghamton.edu, [4] mgovinda@binghamton.edu

***Abstract*—A seamless and intuitive search capability for the vast amount of datasets generated by scientific experiments is critical to ensure effective use of such data by domain specific scientists. Currently, searches on enormous XML datasets is done manually via custom scripts or by using hard-to-customize queries developed by experts in complex and disparate XML query languages. Such approaches however do not provide acceptable performance for large-scale data since they are not based on a scalable distributed solution. Furthermore, it has been shown that databases are not optimized for queries on XML data generated by scientific experiments, as term kinship, range based queries, and constraints such as conjunction and negation need to be taken into account. There exists a critical need for an easy-to-use and scalable framework, specialized for scientific data, that provides natural-language-like syntax along with accurate results. As most existing search tools are designed for exact string matching, which is not adequate for scientific needs, we believe that such a framework will enhance the productivity and quality of scientific research by the data reduction capabilities it can provide. This paper presents how the MapReduce model should be used in XML metadata indexing for scientific datasets, specifically TeraGrid Information Services and the NeXus datasets generated by the Spallation Neutron Source (SNS) scientists. We present an indexing structure that scales well for large-scale MapReduce processing. We present performance results using two MapReduce implementations, Apache Hadoop and LEMO-MR, to emphasize the flexibility and adaptability of our framework in different MapReduce environments.**
[1]

I. Introduction

Data-centric programming paradigms and specialized query languages have steadily grown to very large sizes. In this progression, the technical knowledge required to manage such languages has slowly become more complex. Unfortunately, the powerful capabilities of these new technologies, currently, can only be leveraged by computer scientists who have mastered the complexity of specialized query languages and scripts, and who also have expertise in dealing with complex and disparate software tools. It is therefore essential to abstract away the fundamental complexity of scientific metadata and provide an elegant, intuitive, simple, but powerful free-form query based search framework to end users and scientists. The nature of scientific data in itself and the requirements of its search do not permit the direct use of existing search engine techniques. Unlike linked web documents, scientific datasets for their part, are stored in a flat or hierarchical file structure, where the index depth of their content is not a relevant criteria, but rather one that is required to capture the information in all the scientific datasets in the repository.

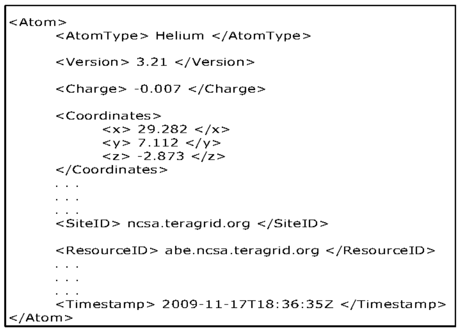
Scientific user queries contain ranges for data and attributes and cannot be processed with direct string matching alone, as is the case with text search engines. Such values need to be retrieved within specified ranges, and according to algebraic relationships. Additionally, the query terms given by the scientist may not directly match with the strings in the datasets, and various equivalences to the query may need to be matched to capture the intent of the scientist. For example, terms such as *degrees, temp, temperature, and weather* may need to be recognized as kins in the context of a scientific search. Similarly, terms such as *facility* and *site* will have to be reconciled in a scientific search, whereas they are divergent in a purely string based search. For such a search to be efficient and effective, a data indexing scheme needs to be employed. Without an indexing scheme, search requires scanning of each and every file in the repository. Indexing provides a mechanism to reduce the number of files relevant to the given search query, thus greatly limiting search time and enhancing accuracy.

With the amount of available scientific data growing from terabyte sizes to petabytes, the number of files being generated is also scaling into tens and hundreds of millions. Scientists conducting experiments using the Spallation Neutron Source (SNS) and High Flux Isotope Reactor (HFIR) at Oak Ridge National Laboratory (ORNL) generate hundreds of terabytes of data annually [1]. An effective scientific data search framework requires an efficient mechanism to generate indexes based on the metadata for the large number of scientific data files produced. Considering the scale of the datasets in question, it stands that performing the work of index creation on a single machine is simply not feasible.

The specific algorithms for semantic matching has been studied extensively by the Web information retrieval and database community including related work on just semantic matching of queries and search terms for grid by two of our co-authors [2], [3]. This paper is focused on the design of the indexing framework using MapReduce [4], which can scale, and can be used along with a semantic matching module.

The framework we have developed easily integrates into a MapReduce framework allowing the XML-based data index-

[1]This work was supported in part by NSF grant CNS 0958501.



```
<Atom>
      <AtomType> Helium </AtomType>

      <Version> 3.21 </Version>

      <Charge> -0.007 </Charge>

      <Coordinates>
            <x> 29.282 </x>
            <y> 7.112 </y>
            <z> -2.873 </z>
      </Coordinates>
      . . .
      . . .
      . . .
      <SiteID> ncsa.teragrid.org </SiteID>

      <ResourceID> abe.ncsa.teragrid.org </ResourceID>
      . . .
      . . .
      . . .
      <Timestamp> 2009-11-17T18:36:35Z </Timestamp>
</Atom>
```

Fig. 1: A sample XML file portraying a scientific experiment. Values for diverse fields expressed in the experiment are recorded in the document according to their relationship to adjacent fields. *Coordinates* in this context, being a field, is also the parent of the x, y and z fields. The data reduction queries by scientists cannot be fulfilled with direct string matching. A simple sample query may require checking for *versions* greater or lesser than a value, *dates* in a given range, and *site* and *resource* names that includes substrings such as ncsa

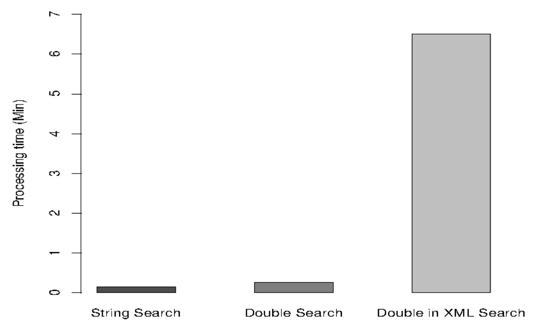


Fig. 2: Comparison of search times for a single string, a double and a double value within an XML document. Searching for numeric values in XML documents shows the highest processing time.

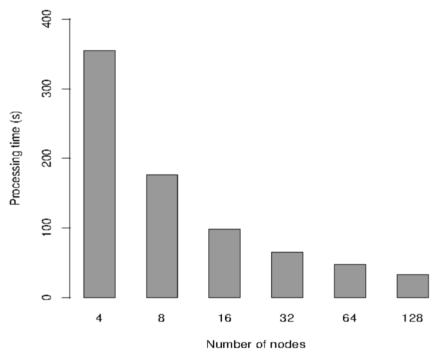


Fig. 3: Processing time for double search over 50 Million elements with changing cluster sizes in Hadoop. Processing time is in inverse ratio with the number of nodes. As the number of nodes increases the processing time scales down.

ing necessary for the search to occur. We focus on XML-based scientific data as an illustrative example. The scientific computing community is still dominated by scientific data formats, such as netCDF and HDF, which integrate metadata with array data into binary-portable, self-descriptive, and self-contained files. While the general approach proposed in this work applies to these data formats for indexing purposes, enabling semantic match for such formats remains a challenge. We plan to address these formats in future work. In this paper we use and refer to TeraGrid and SNS datasets, such as metadata, available in XML format, which is growing to large sizes at a rapid pace. The work represented here is designed to be integrated with a web-search like search engine that operates on XML/XSD formatted datasets [5]. We integrated this indexing and search system within two different MapReduce implementations, as to show its adaptability.

The contributions of this paper are the following:

- We provide a MapReduce based XML data indexing framework, which can be used along with a semantic matching framework, capable of indexing and efficiently searching large data sets.
- We present details on how well-known ideas on creating an inverted-index can be applied to XML scientific datasets to enable seamless integration into any framework that espouses the MapReduce model. To illustrate the design flexibility we present performance results with two different MapReduce implementations, including the widely used Apache Hadoop framework.

II. Metadata in XML Format

A sample metadata file in XML format is shown in **Fig.** 1. The sample XML file shows the various data types on which a scientist may query. The types in the sample include strings, date for time stamp, and doubles. A scientific search may require identifying datasets wherein the *AtomType* is *Helium* and *version* is greater than *3.1*, or datasets wherein the *date* is after *Oct 2009*. As a result, direct string matching does not suffice and constraints also need to be checked.

A user query asking for documents with the *versions between 3.6 and 5.8* requires comparison of two doubles to check if a value falls into the specified interval. In order to perform mathematical operations, strings need to be parsed to double first which in turn comes with a cost to be reflected on performance. **Fig.** 2 shows the cost of performing search on double values. String to double conversions are expensive operations and as the data sizes grow large, the cost associated with such operations similarly grows. It is to be noted that search on XML documents needs to take into account the parsing of both the XML tags and XML element values. Range search in XML datasets requires additional parsing of values stored as strings to numeric types.

As seen in **Fig.** 2, searching over 50 million XML elements with a single machine will take more than 6 minutes. An obvious alternative to reduce the processing time would be distributing the search over a cluster of nodes. **Fig.** 3 shows that distributing the same search with Hadoop [6] provides a remarkable improvement in performance as the number of nodes approaches 128. On the other hand, datasets with terabytes of sizes require extra measures, like indexing to achieve such improvement. On large volumed XML datasets, the cost of parsing is magnified as the number of elements to be parsed significantly increases. By indexing, we achieve a data reduction which lessens the volume of the data to be searched, and as a consequence the number of conversions to apply.

1) Motivation for XML based Metadata Search: XML based scientific metadata, programming paradigms, and specialized query languages such as SPARQL [7] have steadily grown in complexity and the technical knowledge to manage them has also become difficult to master for domain scientists. Just as most computer users today do not have to write programs, domain scientists should be shielded from the low-level details of XML syntax and structure. The integration of Natural Language Processing (NLP) and Information Retrieval (IR) technologies in web search engines have made it possible for end-users to easily and effectively obtain information that is stored in billions of web pages. Users do not need professional programming expertise or technical knowledge of the structure and format in which web pages are stored by search engine servers. However, as there is little context to the information that is indexed and searched via web search engines, they typically return multiple links to the end user. A key difference for search in XML based scientific metadata information is that unlike web search engines that return multiple web pages, domain scientists require the exact information in response to their queries. For many scientific instruments, such as the SNS facility, the number of metadata files is also very large, as each data file, or small groups of files can have an XML based metadata file associated with it. A fast and scalable metadata based data reduction system is critical for making such information easily accessible.

2) Metadata Sources from TeraGrid: The TeraGrid information, that can be searched using free-form queries in our framework, is based on the data that can be extracted by crawling their portal pages [8]. The collected data provides information about TeraGrid resources, capabilities, software, services, science gateways, and other infrastructure elements.

3) Scope of Free-form Queries: The scope of free-form queries in our framework is based on expressing search queries in plain English language, and scientists do not need to learn any formal expression syntax, just as in web search. Scientists can express search constraints using natural-language-like specifications. Our work adapts several techniques from Information Retrieval and Semantic Web, to enable context-rich free-form queries. The problem of processing and acting upon arbitrary English is an extremely challenging research topic being actively addressed in the AI community. To serve a scientific query, however, it suffices for our system to understand a limited form of English, wherein the vocabulary is based on scientific terminology. There is no formal query specification language that scientists need to learn, just as in web search. In this paper, we do not discuss the application of ontologies for the semantic analysis of a user query. This ontological module takes place in a single node and the research involved in the semantic network has been discussed in other venues, [3] and [5].

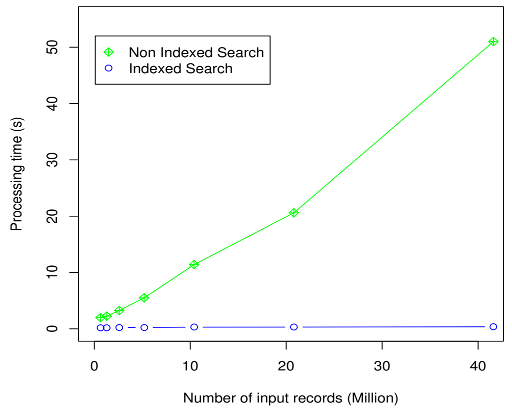


Fig. 4: Non-indexed search, and indexed search times of TeraGrid data. Indexing in this case is paramount to good search performance. While the advantages of indexing is well known, this graph quantifies the exact cost difference for searching XML datasets.

III. Indexing XML-based Scientific Metadata using MapReduce

In providing a scientist with the ability to easily, intuitively and effectively search through massive data sets, it is paramount to develop an efficacious indexing methodology. The primary objective of any XML indexing technique is to reduce the time and space complexity for search queries on XML-based scientific datasets. When a search query is received, the indexing metadata is reviewed and compared with keywords present in the query. According to search terms and conjunctions present in the search query, the corresponding index file is selected, and the inverted index object is loaded into memory to serve the query. The indexes are stored in separate files as storing the index for petabytes of data in one very large file is a highly inefficient option. As the indexes are stored in separate index files, the search space for the keywords present in the query gets substantially reduced. An inverted index is created for each element value present in all the XML metadata files, which results in significantly faster query processing. Without an indexing scheme, data reduction will require scanning every file in the repository, which requires extensive computing for each query and significantly increases the response time of every request, as shown in **Fig.** 4. In **Fig.** 4, the search time for a non-indexed search through 40 million records takes nearly 1 minute, whereas, that of an indexed search take a 1/3 of a second. If this occurs for several searches per-day, it can result in a massive loss of productivity. The performance bottlenecks presented by a single-node search system approach make it impractical. Furthermore, the large number of index files, and as a consequence the disk-space occupied by the indices can be overwhelming for a single system. The MapReduce paradigm is a perfect candidate for the generation of indexes and subsequent searching, as the model is designed to be scalable and fault-tolerant in data-intensive applications [4].

IV. Scientific data search

An indexing program needs to be flexible enough to run at pre-determined times and also on-demand when a large

```
<xs:element maxOccurs="unbounded" name="Atom" form="unqualified">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="AtomType" . . . />

                  <xs:element name="Version"  . . . />

                  <xs:element name="Charge"  . . . />
                  <xs:sequence name = "Coordinates" . . . >
                        <xs:element name="x" . . . />
                        <xs:element name="y" . . . />
                        <xs:element name="z" . . . />
                  </xs:sequence>
                  . . .
                  . . .
                  . . .
                  <xs:element name="SiteID" . . . />

                  <xs:element name="ResourceID" . . . />
                  . . .
                  . . .
                  . . .
                  <xs:element name="Timestamp" . . . />
            </xs:sequence>
      </xs:complexType>
</xs:element>
```

Fig. 5: XSD example used to specify the structure of scientific metadata. This XSD file uniquely identifies each element in the scientific experiment it portrays, as well as the child-parent and siblings relationships between such elements.

Element Name	Code Value
Atom	[0-0]
AtomType	[0-0-0]
Version	[0-0-1]
Charge	[0-0-2]
Coordinates	[0-0-3]
. . .	
. . .	
. . .	
SiteID	[0-0-356]
ResourceID	[0-0-357]
. . .	
. . .	
. . .	
TimeStamp	[0-0-562]

Fig. 6: XSD numbering code table for metadata fields. The numbering scheme starts from 0 for each element at each level of the tree. Codes for each subsequent element are incremented by 1, but also generated based on the element's parent code and its assigned number under its parent element.

experiment has completed and data reduction needs to be performed immediately. Another similarity with web search is the requirement of an inverted index to process user queries and map the terms to relevant documents and rank the results.

The design of the inverted index is challenging for data reduction, compared to web search, in two fundamental ways: (1) the user query contains ranges for attributes and datasets that need to be retrieved for values within the specified range; (2) the query terms may not match directly with the strings in the datasets. For example, the query may have the terms *site* and *degrees/temp*, while the metadata schema may refer to the same concepts as *facility* and *temperature*. Due to the tree-based semi-structured format of XML metadata, the indexing is substantially different from database indexing techniques. The aim of our indexing mechanism, which uses the Dewey labeling scheme, is to uniquely identify each element in a scientific metadata document, whose structure is specified by an XSD document, as **Fig.** 5 shows.

In every level of the tree representation of the XSD file, which constitutes our metadata file, each element is given a numbering code, as shown in **Fig.** 6. The numbering scheme starts from 0 for each element at each level of the tree. As the root element is always the only element in its level and has no ancestors, it is always denoted with the code 0. Codes for each subsequent element are generated based on its parent code and its assigned number under its parent element.

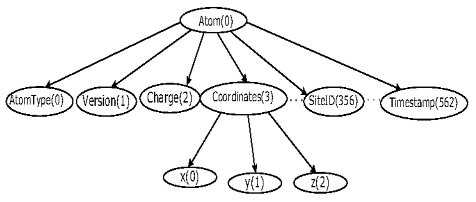


Fig. 7: Numbering scheme based on the XSD document used for application metadata files in SNS.

Fig. 5 shows the content of a sample XSD file similar to the ones generated by SNS scientists, while **Fig.** 7 shows the numbering model for the same sample metadata XSD file. The element *Atom* is the root element and is assigned the code 0. The element *AtomType*, a child of *Atom*, is assigned the code 0_0, while *Version* is assigned 0_1, *Charge*, 0_2 and *Coordinates*, 0_3. Similarly the element *x*, a child of *Coordinates*, is assigned the code 0_3_0. The advantage of this coding scheme is that each node also contains the id of its ancestor. For example, by examining the code of element *y*, which is 0_3_1, we can tell who is the parent, which in this case is *Coordinates*. The uniqueness of element codes also ensures that elements with similar names, but with different ancestry structures, are distinguishable. This feature is required as scientific metadata often have identical element names in different sub-trees. For example, the element *Version* with code 0_1 can be a child of *AtomType*, in which case with the code 0_0_1, and be present in different paths of the XSD tree with its parents as *Atom* and *AtomType*. These two codes for the element *Version* also add semantic value to the context of any query. Thus, a user query with keywords *Atom* and *Version* will get processed only in the *Version* subtree, whereas a query with *Version* and *AtomType* will get processed in the *AtomType* subtree. Similarly, for a user query such as *find Atom of samples that have Version 4093*, our system can determine which particular name to search for. The addition of new elements (increase in depth or breadth), also does not require re-numbering and re-assigning codes of existent nodes in XML data.

Fig. 8 shows the index generation steps. The XSD file for the given scientific metadata is parsed using the XPP XML parser [9] and each element is assigned a unique code. The codes are then stored in a code repository for the unique identification of each element. For each leaf-element in the XML data files (conforming to the XSD), a key-value pair is created, which we denote as the inverted index. The key is the data-value of the leaf element in the XML file. The value is a set containing all the file ids, because there is also the possibility of the same leaf element value being present in multiple XML files.

V. Indexing with MapReduce

We applied the MapReduce model to the indexing method described above in processing of TeraGrid data by first integrating the XML Pull Parser [9] into the Hadoop [6] distributed environment; and subsequently, LEMO-MR [10], our

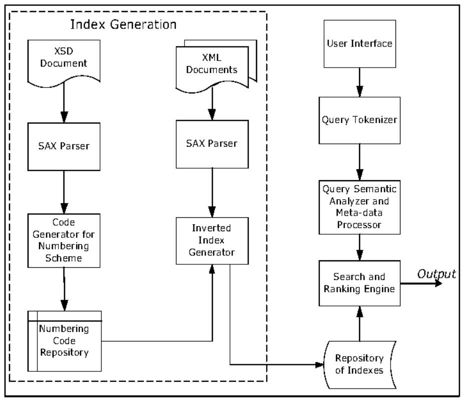


Fig. 8: System architecture for indexing of XML datasets. The numbering code is based on the XSD file. New XML datasets are indexed and the relevant information is stored in the repository. The dotted boxed outline represents the focus of this paper.

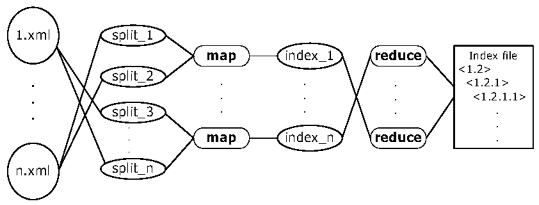


Fig. 9: Shows the MapReduce model applied to our indexing framework. The XML files are the metadata files that need to be searched for data reduction.

MapReduce implementation, also hosting the Pull Parser. The Pull parser is the implementation of our indexing mechanism described in Section III.

Map and Reduce: Both MapReduce frameworks implement our indexing module as a *map* task solely focused on the shard of input a particular node is responsible for processing. All nodes apply the same *map* function to the input chunk they receive. The indexes are then filtered through the *reduce* function by designated *reducers* and written to individual index files. In the case of searching, the input is a particular index file relevant to the user search query. That index file is distributed among nodes and searched as a part of the *map* function. Reducing here only consists of presenting the result of the search.

Fig. 9 shows the structure of our indexing framework integrated into both MapReduce environments. XML and XSD files are taken as input, the XML files are split, and then communicated to the various *mappers*, represented by each node's

Key	Values
0-0-0	(Helium, exp1543.xml), (Ammonium, exp2001.xml), ...
0-0-1	(2.10, exp2001.xml), (3.21, exp1543.xml), (3.21, exp2021.xml), ...
....	
....	
....	
0-0-562	(2009-11-12T08:32:04Z, exp2001.xml),(2009-11-17T18:36:35Z, exp1543.xml), ...

Fig. 10: Data-structure representation of the "sub-indexes", held by each mapper in the MapReduce cluster. "sub-indexes" held by each mapper are each a piece of the entire index structure. As memory limitations proper to a single machine do not allow storage of the entire structure on one node, MapReduce offers an advantage as bigger indexes, and as such bigger input data can be processed and stored in parallel.

0_0_0.index

Value	Filename
Ammonium	exp2001.xml, exp8471.xml
Helium	exp1543.xml, exp6492.xml
...	
...	
...	

0_0_1.index

Value	Filename
2.10	exp2001.xml
3.21	exp1543.xml, exp2021.xml
...	
...	
...	

0_0_562.index

Value	Filename
2009-11-12T08:32:04Z	exp2001.xml
2009-11-17T18:36:35Z	exp1543.xml
...	
...	
...	

Fig. 11: Sample index files produced by the indexing scheme described in **Section** IV. Index filenames are driven by the codes of the corresponding values shown in the code table **Fig.** 6. Input filenames in this sample are linked to the values they host.

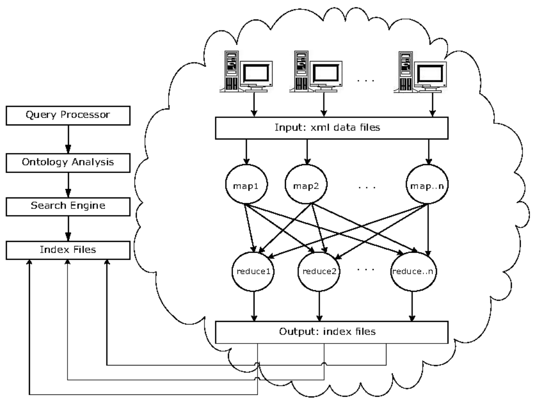


Fig. 12: MapReduce based search framework. The MapReduce system (clouded) performs indexing from input XML data files and metadata XSD files, as described in **Section** V. Resulting index files are then fed into the scientific data search engine, which processes user queries and performs searching over index files.

processing cores. The worker nodes apply each XSD file to their XML chunks and progressively build "sub-indexes" by parsing both XSD files, XML chunks, and matching their tag elements. "Sub-indexes" are represented in memory as **Fig.** 10 portrays. Each node holds a piece of the general data structure as shown by **Fig.** 10. "Sub-indexes" are then communicated to all the *reducers*, whose role is to organize all the intermediary indexes by key into a final index file corresponding to a particular search tag. The resulting index files shown in **Fig.** 11 allow for the search to directly select the relevant index file, and quickly match the searched element within.

VI. TeraGrid data Processing with MapReduce

We applied our indexing framework, along with Hadoop and LEMO-MR, to search and extract information on TeraGrid. The architecture of the framework is shown in **Fig.** 12. The clouded area represents the MapReduce system as part of the overall framework. We subsequently developed an automated tool that crawls the web pages of the TeraGrid Information Services portal, converts to appropriate RDF/OWL formats, and stores them in our search framework.

TeraGrid REST "web-apps" interfaces currently do not provide access to all the information, instead, access is available through the HTTP GET based WEbMDS. Our crawler also

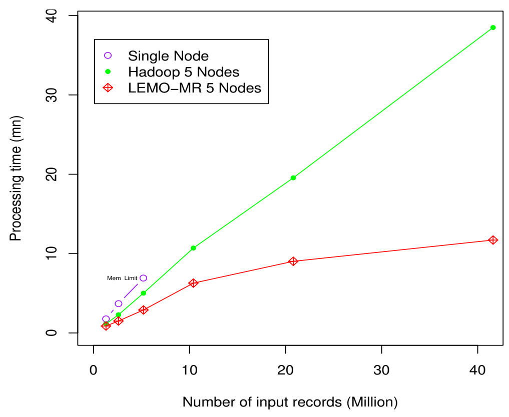


Fig. 13: Shows the execution of our indexing framework in two a distributed settings. Both Hadoop and LEMO-MR clusters are setup with five hosts. Both frameworks are running identical Map and Reduce code. Input here varies from .1 to 40 million records. The single computer system shows slower, and finds itself out of memory around 5.2 Million records.

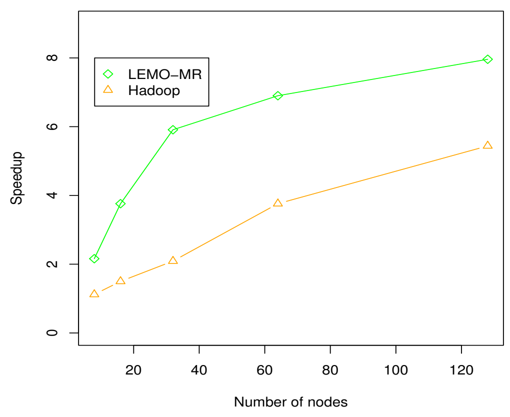


Fig. 14: Speed-up computed from LEMO-MR, and Hadoop clusters, against a single node search system. In this experiment, both MapReduce frameworks search over 40 million records with various cluster sizes, ranging from 8 to 128 nodes. Speed-up is computed as $\frac{T_1}{T_p}$ and represents how fast each cluster performs relative to the single-node search system. Both clusters roughly scale in a similar fashion, even as LEMO-MR runs faster in this context. This graph shows the applicability and performance of the MapReduce model, even with single-node-search system capable data sizes. In this particular example, indexing time is cut by a factor of 8, and as such can allow for more frequent indexing, and as a consequence, more accurate and up-to-date search results.

accesses hidden links that have data in XML format for listing of the kits, services, software, and resource descriptions. We used these hidden links to generate XSD files for the information available on the TeraGrid web pages and subsequently applied our indexing scheme. As there are various kinds of information available via TeraGrid Information Services, we combined the different XSDs under a single parent node to apply the node numbering scheme described in Section IV.

We run our tests on a selection of two machine classes:

- Dual core – One desktop-class machine, which has a single 2.4Ghz Intel Core 6600 with 2 GB of ECC RAM, and quad cores running Linux 2.6.24. The file system in use here is NFS v.4.
- Quad core – 1U nodes in a cluster, each of which has two 3.2Ghz Intel Xeon CPUs, 4 gigabytes of RAM 8 cores, and run a 64 bit version of Linux 2.6.15. Results on this class of machines are taken by averaging the timings produced on these nodes. The file system in use in the test directory is NFS v.4

In **Fig.** 13, the TeraGrid datasets presented here features a progression of input sizes going from 0.1 million records and scaling all the way up to 40 million records. As more data is gathered from TeraGrid, a distributed solution will be essential to address the limitations of a single node's memory and computational capacity. The single node search system featured in this experiment reaches the upper limits of its memory capacity with 5.2 million elements. Processing larger data then becomes incumbent upon both MapReduce frameworks, which scale up to over 40 Millions elements with 5 nodes. An advantage of the MapReduce model is the ability for its distributed filesystem to off-load application memory content into distributed files to be picked up by any node in the cluster. With enough nodes, this affords the model the ability to process all its input in memory at once. This experiment is not only aimed at showing the memory advantage and performance of a MapReduce cluster in scientific data reduction, but also quantify the threshold up to which a single-node's memory is capable of handling input sets. Even below the memory threshold of a single node solution, a MapReduce solution, although out of place with small input sizes and little processing, [11], is more efficient. Finally, **Fig.** 13 is meant to show that scientists processing smaller scale data, but requiring multiple indexing processes as data arrives frequently, can harvest greater productivity from a MapReduce framework with a limited number of nodes.

As shown in [10], LEMO-MR has less operational overhead, as it does not support all of Hadoop's features such as data replication and data shuffling, and thus performs slightly better than Hadoop as the data set scales from 0.1 million to tens of millions of elements.

In **Fig.** 14 speed-up is computed as a measure of how both clusters scale relative to a single-node search system. This graph shows that even with a relatively moderate cluster in size, performance up to 8 times faster than that of a single-node search system can be attained, while still operating with single-node memory capable datasets.

Fig. 15 shows the scalability of our indexing framework in a MapReduce environment, with 400 million data elements to process. As TeraGrid data is gathered, a single node system quickly becomes overwhelmed with the amount of data capable of fitting in its memory, as shown in **Fig.** 13. We address this shortcoming by adapting the indexing scheme to work with commodity machines in a MapReduce cluster. As more computing power is gradually added for each performance run, both clusters index their input faster. This enables the processing of over 0.4 billion elements, from 10 minutes with 8 nodes, to 1.5 minutes with a 128 node cluster. The Map construct here consists for each node in applying the index on the data they temporarily own and returning the produced index into the intermediate data pool, awaiting reducing. The reduce operation consolidates, organizes and

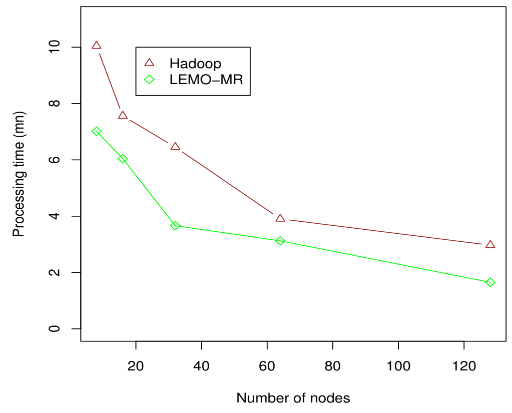


Fig. 15: Processing performance shown by both MapReduce clusters as nodes are gradually added to each of them in indexing a set containing slightly over 400 million elements.

eliminates duplicate indexes produced by similar parts of the input. LEMO-MR performs better as it is designed as a lightweight implementation of MapReduce, and is devoid of significant operational responsibility encountered by Hadoop MapReduce, such as maintenance of input chunk duplication factors, HDFS maps, and block mapping.

VII. Related Work

While we applied our indexing framework to Hadoop [4] and LEMO-MR [10], the indexing scheme we presented in this paper can be applied in an iterative manner to Twister [12] by allowing each step of the iteration to produce a sub-index as explained in section III. [13] details four different approaches for applying document indexing in MapReduce. They show that the MapReduce indexing strategy suggested in [4] generates too much intermediate data, which in turn causes too much overhead. [14] indicts scan-centric MapReduce approaches for processing queries as a cause for extra I/O overhead when a query addresses only a subset of the data. [15] uses MapReduce to address two important problems with spatial databases on a Google IBM cluster. The paper shows the application of MapReduce to R-Trees building and aerial image rendering. The work we presented in this paper focuses on scientific data indexing, and as such differs from the ones outlined above, as they present general data indexing approaches using MapReduce.

Structure-based XML-indexing is primarily based on the bi-similarity concept where the nodes in XML documents are grouped into structurally similar concepts. These structural summaries are the reflection of the underlying database structures. Milo et. al. [16] use template indexes or T-indexes for processing queries consisting of path expressions. T-indexing can be quite an expensive structure, but this can be considerably reduced in specific conditions. Chen et. al. [17] introduce the D(k)-index that is based on the concepts of bi-similarity. Their indexing technique derives adaptive structural summaries from the XML data and serves as indexes for evaluating path expressions.

XML data is modeled as labeled graphs, where the edges correspond to element-sub-element relationships and IDREF pointers. Spyglass [18] exploits spatial locality and skewed distribution of namespaces in file system metadata to index namespace hierarchy in a multi-dimensional kd-tree. The design of the system is influenced by real-time file system metadata usage characteristics that include search on multiple metadata attributes, multiple versions of same metadata attribute, and localized lookup. Although scientific metadata search exhibits similar characteristics, there is an additional requirement to support complex free-form queries, which cannot be satisfied by processing namespaces alone. Therefore our system focuses on semantics by including ontology-based techniques.

Our indexing and search system is complementary to these approaches as it focuses on matching free-form queries to the correct data files. Depending on the structure of the metadata, the appropriate scheme can be incorporated to augment framework we have developed. The work we present in this paper focuses on scientific data indexing, and as such differs from the ones outlined above, as they present general data indexing approaches using MapReduce.

VIII. Conclusion

With current advances in modern scientific endeavors, coupled with the demands of an ever more connected scientific world, there exists a need for fast and efficient scientific data search framework. Domains like Cancer Research, Biomedicine, High Energy Physics, to cite a few, are all daily producing massive amounts of data, that need to be searched. According to CERN, [19], more than 40 million data elements are generated by the Large Hadron Collider, every second during the particle accelerator's operation. The utilization of a semantic framework along with an indexing scheme capable of efficiently indexing scientific metadata has the power of providing a simple yet powerful interface to domain scientists for querying and obtaining specific subsets of files and data they are interested in.

Scientific search differs from web text based search, as data ranges, term kinship, conjunction, disjunction, natural language and result precision need to be accounted for. We provide the ease-of-use of popular Web search engines, along with the ability to retrieve information related to user queries in the scientific domain. Our distributed indexing and search is based on the MapReduce model, as TeraGrid data sizes require the indexing and search to scale to very large datasets. The integration of our framework into a high performance distributed environment, allows domain scientists to harness more indexing and searching power. The specific contributions of our framework are the following:

- Our design provides an indexing framework capable of indexing and efficiently searching large-scale scientific XML data sets.
- To meet scalability and variety requirements, our framework is tailored for integration with any framework that espouses the MapReduce model.

IX. Future Work

In future work, we will modify the indexing data structures to optimize the storage of keys by combining frequently occurring or well-known ranges into a single key value. We also plan to incorporate the effects of disk I/O and memory size for testing terabyte size data for indexing. We will conduct benchmark studies to determine the best XML parser suited to parse a large number of XML files for faster indexing of the data.

X. Acknowledgements

We would like to thank Stephen D. Miller and the team at the Oak Ridge National Laboratory's Neutron Sciences group for the technical discussions and access to XML metadata that contributed to this work.

References

[1] Spallation Neutron Source (SNS). [Online]. Available: http://neutrons.ornl.gov/aboutsns/aboutsns.shtml

[2] C. Gupta and M. Govindaraju, "Semantic framework for free-form search of grid resources," in *ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 454–455.

[3] C. Gupta, R. Bhowmik, M. R. Head, M. Govindaraju, and W. Meng, "Improving performance of web services query matchmaking with automated knowledge acquisition," in *WI '07: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 559–563.

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] C. Gupta, R. Bhowmik, and M. Govindaraju, "Semantic framework for free-form search of grid resources and services," in *Grid Computing, 2009 10th IEEE/ACM International Conference on*, 2009, pp. 81 –88.

[6] Apache Hadoop. [Online]. Available: http://hadoop.apache.org

[7] SPARQL. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[8] TeraGrid Information Services. [Online]. Available: http://info.teragrid.org/

[9] XML Pull Parser. [Online]. Available: http://www.xmlpull.org/

[10] Z. Fadika and M. Govindaraju, "Lemo-mr: Low overhead and elastic mapreduce implementation optimized for memory and cpu-intensive applications," *Cloud Computing Technology and Science, IEEE International Conference on*, vol. 0, pp. 1–8, 2010.

[11] Z. Fadika, M. R. Head, and M. Govindaraju, "Parallel and Distributed Approach for Processing Large-Scale XML Datasets," in *10th IEEE/ACM International Conference on Grid Computing*, October 2009, pp. 5–7.

[12] Pervasive Technology Institute. [Online]. Available: http://www.iterativemapreduce.org/

[13] R. M. C. Mccreadie, C. Macdonald, and I. Ounis, "Comparing distributed indexing: To mapreduce or not?"

[14] M. An, Y. Wang, and W. Wang, "Using index in the mapreduce framework," in *Proceedings of the 2010 12th International Asia-Pacific Web Conference*, ser. APWEB '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 52–58. [Online]. Available: http://dx.doi.org/10.1109/APWeb.2010.12

[15] A. Cary, Z. Sun, V. Hristidis, and N. Rishe, "Experiences on processing spatial data with mapreduce," in *Proceedings of the 21st International Conference on Scientific and Statistical Database Management*, ser. SSDBM 2009. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 302–319. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02279-1_24

[16] T. Milo and D. Suciu, "Index structures for path expressions," in *ICDT '99: Proceedings of the 7th International Conference on Database Theory*. London, UK: Springer-Verlag, 1999, pp. 277–295.

[17] Q. Chen, A. Lim, and K. W. Ong, "D(k)-index: an adaptive structural summary for graph-structured data," in *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2003, pp. 134–144.

[18] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller, "Spyglass: fast, scalable metadata search for large-scale storage systems," in *FAST '09: Proccedings of the 7th conference on File and storage technologies*. Berkeley, CA, USA: USENIX Association, 2009, pp. 153–166.

[19] European organization for nuclear research. [Online]. Available: http://public.web.cern.ch/public/en/lhc/lhc-en.html