Grid-Based Digital Libraries: Cheshire3 and Distributed Retrieval

Ray R. Larson
School of Information Management and Systems
University of California, Berkeley
Berkeley, California, USA, 94720-4600
ray@sherlock.berkeley.edu

Robert Sanderson
Department of Computer Science
University of Liverpool
Liverpool, L69 3BX United Kingdom
azaroth@liverpool.ac.uk

ABSTRACT

The University of California, Berkeley and the University of Liverpool are developing a Information Retrieval and Digital Library system (Cheshire3) that operates in both single-processor and "Grid" distributed computing environments. This paper discusses the architecture of the system and how it performs Digital Library tasks in a Grid computing environment.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval—*retrieval models, search process*; H.3.7 [**Information Systems**]: Digital Libraries—*systems issues* **General Terms:** Algorithms, Performance, Design

Keywords: Grid-based Information Retrieval

1. INTRODUCTION

In order for Information Retrieval (IR) in the evolving "Grid" parallel distributed computing environment[1] to work effectively, there must be a single flexible and extensible series of "Grid Services" with identifiable objects and a known API to handle the IR functions needed for Digital Libraries (DL) and other retrieval tasks. The Cheshire3 system builds upon the Cheshire project[2] to define and implement an set of objects with precisely defined roles that permit DL operations to be distributed over many nodes on a network, vastly increasing the throughput of data for compute and storage intensive processes with little overhead beyond single processor solutions.

A recent invitational workshop on "Grid-based Digital Libraries" held at UC Berkeley, defined the potential of using Grid technology to address current issues in the development of digital libraries. The conclusions reached there suggested that Grid technologies can contribute to increasing efficiency of the back-end functions of digital libraries such as data backup, automated replication, and archiving.

JCDL '05 June 7–11, 2005, Denver, Colorado, USA.

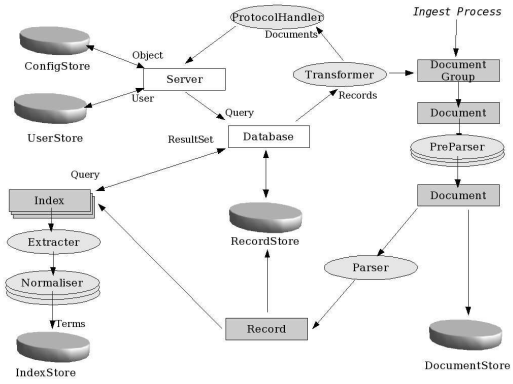

Figure 1: Cheshire3 Object Model

Data curation systems could float on top of localized storage. Use of Grid technology could facilitate federated digital libraries across institutions. It was concluded that much of the current research on digital library infrastructure could be translated into appropriate architectures for a grid-based systems.

In this paper describe our first steps in this direction. First we discuss the object structure of the Cheshire3 system and how it works in the Grid environment. We also present some efficiency tests using DL workloads. More on system performance and operations will be presented at the meeting.

2. CHESHIRE3 OBJECT MODEL

The Cheshire3 system uses an object-oriented design with an object hierarchy consisting of two main object types: those objects that represent data and storage, and those that represent processes. The main data objects in the system, most of which are shown graphically in Figure 1.

All objects which are not user-supplied data or the results of a process, are configured via XML records. These records include the type of object to instantiate, an identifier for it as well as the parameters or other information necessary to allow the component to perform its task. These configuration records can then be stored, accessed and manipulated in the same way as any other record within the system, including distribution via IR protocols such as OAI-PMH for bulk harvesting or SRW/U for search and retrieval. This allows resources to be shared seamlessly and configurations

distrubuted between instances of the Cheshire3 framework. However this does not, yet, prevent the necessity to share lines of user configured code in order to process localised requirements, which is a requirement for successful grid based information retrieval. The objects include:

DocumentGroup	A set of Documents.
Document	Unparsed data representing a single item.
Record	Parsed XML based data representing a single item.
Query	A CQL query parse tree.
ResultSet	An ordered set of symbolic pointers to Records.
Index	An ordered list of terms extracted from a Record
User	An authenticated user of the system.

Storage facilities exist for each of these object classes. In addition there are several main processing objects including:

PreParser	Convert a Document into another type of Document
Parser	Convert a Document into a Record
Transformer	Convert a Record into a Document
Extracter	Extract data of a given format or type for indexing
Normaliser	Convert data from one format or type to another
ProtocolHandler	Take a request in a known protocol and converts it to an internal representation.

Also, there are three abstract objects:

Server	A logical collection of databases
Database	A logical collection of records and indexes
Workflow	An object that can take input, go through a user-defined sequence of processing steps and produce output.

The last abstract object above is the component that allows the Cheshire3 system to work effectively in a distributed environment. Workflow objects are configured like any other object within the system, and hence share their ease of portability. Each workflow contains a series of instructions in its configuration, and when the object is requested, these are compiled dynamically into executable code. Thus, the requirements for the system operations are specified by describing the logical flow of data through the various types of processing objects, and no additional programming is needed. Specification of the workflow is done via object identifiers, and workflows can call other workflows as required, allowing users to set up a workflow for specific tasks but still have that subsystem integrated within a larger processing environment.

Each object in the system has a (usually unique) identifier as its reference. It is possible, however, to re-use identifiers at a lower levels of the system, allowing a database to override a particular server object without requiring changes to code or configurations refering to it. The utility of this becomes apparent in the context of distributed workflows, as not only can one workflow to call another, one very high level workflow might interact in the same way with many different database-specific workflows, each of which performs customised operations to achieve the same end result.

Once the distributed infrastructure has been defined, one or more 'master' workflows divide the processing requirements among 'slave' processes. Our implementation uses PVM as a very fast communication layer over a gigabit switched network, but we are implementing an MPI based version as well. When a slave process has completed its assigned workflow, it returns the results to the master to be merged. Object interactions within the architecture are well defined, the result from any workflow is also well defined, based on the last processing object. This is very important for trees or graphs of distributed workflows.

Because each database is a logical collection of records, it can be easily split across many nodes or combined at a single location. This means that each node on the cluster can either look after a slice of the database or do the processing required and then return the record for central storage. The same applies to indexing to a lesser extent; if the index files at each node store all of the terms for the records at the node, then interacting with the sub database is very fast, but the intermediate resultsets will need to be merged before a global query can be answered. If each node maintains a portion of the terms for all of the records, then it can answer the query authoritatively, however this requires some central authority to manage the division of the index terms.

3. PERFORMANCE AND CONCLUSIONS

Early performance tests used 1 'master' and 8 slave indexing processes (in a cluster of 16 a dual 1.6Ghz Athlon machines with 2Gb of RAM, with gigabit ethernet). Using this configuration on a workflow that parses MARC records, converts them to XML, parses the XML, stores it in an RecordStore and indexes it, we are able to achieve a sustained indexing rate of about 10,000 records per second (even when competing with other tasks on the cluster). For larger, more complex, records we tested with 640 Mb of TEI encoded documents, and were able to parse, store and index the entire collection in 3 minutes and 20 seconds.

We are in the process of integrating Cheshire3 with the Storage Resource Broker (SRB)[3] DataGrid storage system. Because each object in the SRB is a "document" that can be processed and indexed and identified by its unique SRB ID, we can retrieve the original from storage on demand without needing to store an XML representation of it. Since SRB is also being integrated into the DSpace Digital Library Framework, we will soon have an advanced indexing and search systems for DSpace and SRB installations.

We believe that current digital library architectures and standards can be added to the existing "Grid services" to provide enhanced performance, collaboration and functionality across the developing computational Grid.

4. ACKNOWLEDGMENTS

Design of the Cheshire3 system was supported by the NSF and JISC (U.K) under the *International Digital Libraries Program* award #IIS-9975164.

5. REFERENCES

[1] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Elsevier, Amsterdam, Second edition, 2004.

[2] R. R. Larson, J. McDonough, P. O'Leary, L. Kuntz, and R. Moon. Cheshire II: Designing a next-generation online catalog. *Journal of the American Society for Information Science*, 47(7):555–567, July 1996.

[3] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.-Y. Chen, and R. Olschanowsky. Storage resource broker - managing distributed data in a grid. *Computer Society of India Journal*, 33(4):42–54, 2003.