Grid Information Retrieval using OGSA – DAI

Muhammad S. Sarwar

October 10, 2008

MSc in High Performance Computing
The University of Edinburgh
Year of Presentation: 2008
Abstract

This dissertation is an attempt to present an IR system for grid computing environment. OGSA-DAI and Lucene have been used to come up with a solution for GridIR. Creation of a distributed test collection for grid network is described. Execution of search queries on distributed indexes has been presented. Merging approaches for integrating the search results have been shown. It has been shown that standard IR system evaluation techniques, precision and recall, are not particularly useful in evaluation of ranked retrieval systems and instead relevancy scores provide a good criteria for evaluating ranked retrieval systems.
## Contents

Chapter 1 Introduction ........................................................................................................1

1.1. Grid Information Retrieval .........................................................................................2

Chapter 2 Background Theory .........................................................................................4

Chapter 3 Requirements ...................................................................................................8
  3.1 Collection Manager Service ......................................................................................8
  3.2 Indexing Service .......................................................................................................9
  3.3 Query Processing and Searching Service .................................................................10
  3.4 Security Requirements ............................................................................................11
  3.5 User experience requirements ................................................................................11
  3.6 Technology requirements .......................................................................................11

Chapter 4 Design & Implementation ..............................................................................14
  4.1 Basic model of interaction .......................................................................................14
  4.2 Distributed model of interaction ............................................................................15
  4.3 System Architecture ...............................................................................................16
    4.3.1 Collection Manager Service (CMS) ...............................................................19
    4.3.2 Indexing service ..............................................................................................21
    4.3.3 Query Processing and Searching Service ......................................................22
  4.4 CollectionsDB .........................................................................................................24
  4.5 Document collections and Indexes ..........................................................................26
  4.6 Resources ................................................................................................................26
    4.6.1 Resource description .......................................................................................27
    4.6.2 Resource selection ..........................................................................................27
List of Tables

Table 1: Output of Indexer ................................................................. 4
Table 2: Users table ........................................................................... 25
Table 3: Collections table ................................................................. 25
Table 4: Indexes table ....................................................................... 26
Table 5: Precision-Recall table for concatenation merge .................. 46
Table 6: Precision-Recall table for round-robin merge ................... 46
Table 7: Precision-Recall table for ordered merge ........................... 46
Table 8: document relevancy scores for concatenation merge .......... 50
Table 9: document relevancy scores for round-robin merge ............. 50
Table 10: document relevancy scores for ordered merge .................. 50
Table 11: Precision-Recall and score table for concatenation merge as a function of no. of documents returned .................. 51
Table 12: Precision-Recall and score table for round-robin merge as a function of no. of documents returned .................. 52
Table 13: Precision-Recall and score table for ordered merge as a function of no. of documents returned .................. 52
Table 14: Precision-Recall and score table for concatenation merge as a function of no. of documents returned .................. 53
Table 15: Precision-Recall and score table for round-robin merge as a function of no. of documents returned .................. 54
Table 16: Precision-Recall and score table for ordered merge as a function of no. of documents returned .................. 54
Table 17: Precision-Recall and score table for concatenation-merge .... 56
Table 18: Precision-Recall and score table for round-robin merge ....... 56
Table 19: Precision-Recall and score table for ordered merge ...........................................57
Table 20: Precision-Recall and score table for concatenation-merge ....................................58
Table 21: Precision-Recall and score table for round-robin merge .......................................58
Table 22: Precision-Recall and score table for ordered-merge ............................................58
Table 23: Precision-Recall and score table for concatenation-merge ....................................59
Table 24: Precision-Recall and score table for round-robin merge .......................................60
Table 25: Precision-Recall and score table for ordered-merge ............................................60
List of Figures

Figure 1: Model of an IR system ................................................................. 1
Figure 2: Model of a GridIR system ............................................................ 2
Figure 3: use-case diagram for Collection Manager Service ....................... 9
Figure 4: use-case diagram for Indexing service ......................................... 10
Figure 5: use-case diagram for Query Processing and Searching Service .... 11
Figure 6: Use-case diagram of the system .................................................. 13
Figure 7: Basic model of interaction ............................................................ 15
Figure 8: Distributed model of interaction .................................................. 16
Figure 9: High level system architecture diagram ....................................... 16
Figure 11: Architecture diagram of Collection manager ............................ 21
Figure 12: Architecture diagram of Indexing service .................................. 22
Figure 13: Query processing and searching service .................................... 23
Figure 14: concatenation merge ................................................................. 28
Figure 15: Round robin merge ................................................................. 29
Figure 16: Ordered merge ........................................................................ 30
Figure 17: class diagram .......................................................................... 32
Figure 18: GridIR system implementation diagram .................................... 33
Figure 19: LuceneDataResource ................................................................. 34
Figure 20: LuceneResourceAccessor interface ......................................... 34
Figure 21: SimpleLuceneResourceAccessor ............................................. 35
Figure 22: LuceneDataResourceState interface ........................................ 35
Figure 23: SimpleLuceneDataResourceState ............................................ 36
Acknowledgements

I find myself extremely humble before ALLAH (SWT) who blessed me to complete this project. I offer a special tribute to Prophet MUHAMMAD (PBUH). His teachings have given us a bright vision of this world & life and persuaded us to seek knowledge from cradle to grave. I want to express my profound thanks, appreciation and special gratitude to my parents and my loving friends.

I must mention extended cooperation and guidance of my project Supervisor Mr. Alistair Grant that finally lead to completion of this Project. I am thankful to my friend Mr. Wahid Irshad, who was my soul source of encouragement & commitment throughout the project. I am thankful for the support of my sister, Tahira Sarwar, who was there to proof read my work.

Muhammad Sulaman Sarwar
Chapter 1

Introduction

Information Retrieval (IR) is the science of finding and retrieving information from huge collections of semi-structured and unstructured documents. Search engines (e.g. Google [7], Yahoo [8]) are an example of IR systems. A document is a set of words and a set of documents is called a collection or document collection.

There are three major components of any IR system: Collection Manager (CM), Indexer and Searcher. Figure 1 shows the model of an IR system. CM is responsible for collecting documents for the purpose of indexing. Indexer performs the indexing of the documents collected by CM and Searcher performs the search on the indexed document collections. In order to perform a search, user of the IR system enters the search query, a set of words typed, in the interface provided by the IR system. Searcher collects the input query and searches the indexed collection against the query, compiles the results and sends them back to the user.
Goal of this project is to develop an IR system that is able to retrieve the documents from a collection that are most relevant to the user’s information need. Information need of the user is provided to the IR system by using a query. Information need is the topic about which the user wishes to know more and is different from query which is actually used to transform the information need in a form, understandable to the IR system. A document is considered relevant if the user is satisfied by the information that it contains otherwise the document is considered irrelevant or non-relevant.

1.1. Grid Information Retrieval

Grid computing is the coordinated resource sharing and problem solving in dynamic virtual organization. GridIR is the application of IR in a grid computing environment. Modern search engines are good at retrieving information present on the World Wide Web (WWW) but a huge volume of data present on grids is inaccessible to these search engines. In order to implement the IR over grids, components of the IR system, explained in previous section and shown in Figure 1, are implemented as grid services. Figure 2 shows the model of a GridIR system. Data sources are indexed document collections, deployed anywhere on the grid.

Figure 2: Model of a GridIR system

The purpose of this project is to use OGSA-DAI [10] framework for the implementation of GridIR system. Lucene [11] is an open source text search library and
is used to implement the searching and indexing services. Issues in GridIR addressed in this project are: executing a query on distributed indexes using OGSA-DAI and merging the resulting ranked results. By developing a GridIR system it will be possible to exploit the benefits of a traditional IR system in a grid computing environment.

Test collections are standard data sets used to evaluate the IR system’s effectiveness in information retrieval. There are several test collections available for this purpose. Some of these collections freely available others need subscription. There are no particular test collections available for testing GridIR systems. We will make use of Cranfield test collection which is one of freely available test collections. Cranfield test collection is a small collection consisting 1400 textual abstracts. It also contains a set of 225 queries and relevance results against these queries. Cranfield collection is one of the oldest test collections created by hand-picked text files. Query set and relevance results are compiled by hand. As there are no test collections available for evaluating GridIR systems therefore an ad-hoc distributed test collection has been created by dividing the Cranfield test collection in two halves, each half containing 700 documents. Three more distributed ad-hoc test collections have been used (one used during the development of system, second for unit testing and third for testing the system behaviour in real time) during the lifecycle of the project.

Chapter 2 presents the background theory. Chapter 3 presents the requirements of the system. Chapter 4 describes the details of system design and implementation. Chapter 5 outlines the details of system testing and a discussion on evaluation test results has been drawn. Conclusions, drawn from the work done and future extensions of the project are presented in Chapter 6.
Chapter 2

Background Theory

Vannevar Bush [4] proposed the idea of computerized information retrieval, in 1945. The term information retrieval was coined by Calvin Moore [5] in 1948/50. IR has passed through many different phases since then. An IR system is analyzed in terms of four basic functions:

i- Analysis and indexing of document content

ii- Searching on document collections

iii- Evaluation and assessment of search results

iv- Access to physical content of documents.

Three components of an IR system (CM, Indexer and Searcher) perform these functions. CM (known as Crawler in search engines) is responsible for collecting documents and document collections for indexing. Analysis and indexing of document content is done by the Indexer. During the indexing process Indexer takes the input document and chops the text of the document into pieces called tokens. For example:

Input document: The quick brown fox jumps over a lazy dog;

Output:

| the | quick | brown | fox | brown | fox | Jumps | over | a | Lazy | dog |

Table 1: Output of Indexer

These tokens are called terms. Terms are often stored in lower case format and any punctuation is removed. Output in Table 1 shows that ‘The’ has been converted to ‘the’ and ‘;’ has been removed. Data in the documents is usually in natural language format (text) and there are certain words in the natural language format which are very common and can be excluded from the sentence without losing meaning of the statement. These words are called stop words. Removing stop words saves space and speeds up the searching process. For instance, if ‘the’ and ‘a’ are excluded from the output shown in Table 1, there will be no notable loss in meaning of the statement.
Appendix – B contains the lists of stop words used, for indexing document collections, for this project.

Searcher performs the searching operation on indexed document collections. Collection Manager and Indexer are most of the time automated components that work without involving manual input but Searcher is exposed by a graphical user interface (GUI) which is used by the user of the IR system to input the search query.

A query, for example: “Multidimensional online analytical processing”, is a piece of text composed of words. Each word in the query is called a term or query term. Queries formed in the way shown in example are called free-text queries where no query language is used to specify the search terms instead the query is a set of words. A query language can also be used to specify a query. For example, boolean queries are formed using combination of AND, OR and NOT boolean operators. The above mentioned query when converted to boolean expression will become “multidimensional AND online AND analytical AND processing”. Queries in free-text form are very easy to specify and are widely used in IR systems and this project also uses this same query format.

When a query is submitted to the IR system, Searcher performs the search on the index against the input query and finds the documents that are assumed to be relevant to the query, ranks them and displays to the user. Relevancy is the measure of how well a particular document matches the search query. Relevancy of a document is relative to the information need of the user, who submitted the query, and hence is the best person to judge whether a document is relevant to his information need or not. An IR system employs a ranking mechanism to order the search results. In order to determine the rank of a document a score is computed for every query-document pair. Documents that are assigned higher scores are assumed to be more relevant to the query and hence ranked higher in the search results. IR systems that employ ranking strategies for ordering the search results are known as ranked retrieval systems.

Idea behind ranked retrieval is that if a query term appears more number of times in a document relative to number of times the term appears in all documents in the collection; the more relevant that document is to the query [12].

Precision and recall are the two measures used to evaluate the effectiveness of any IR system.

\[
(1) \quad Precision = \frac{|\text{no. of relevant items retrieved}|}{|\text{no. of retrieved documents}|}
\]

\[
(2) \quad Recall = \frac{|\text{no. of relevant items retrieved}|}{\text{total no. of relevant documents}}
\]
Or in other words,

Precision: How many relevant items have been retrieved out of the total number of retrieved items?

Recall: How many relevant items have been retrieved out of the total number of relevant items?

If there is no relevant document is retrieved, precision of the system in zero (0). Relevancy scores computed and assigned to documents can also be used to evaluate the effectiveness of the IR system, in terms of document relevancy. A higher relevancy score assigned to a document means that the document is potentially a highly relevant document.

An IR system provides access to physical content of documents by displaying the text of the document on screen. Some IR systems provide the facility of downloading the document or printing the contents of document.

Early IR systems used Boolean retrieval model. A boolean retrieval model views the document as a set of terms. Any query in the form of a boolean expression, a combination of boolean AND, OR and NOT operators, can be posed to the IR system. A document contains many fields such as title, author, abstract or summary, keywords, body of the document and date of publication. Boolean retrieval systems only indexed author, title and keywords. After tokenizing the document text, tokens were sorted alphabetically and then stored in the index table. Boolean retrieval systems perform exact matching of the query terms to the document terms in the index therefore if a document matches the query terms it is considered relevant otherwise not.

The problem with Boolean search is that using AND operator yields high precision but low recall values while using OR operator gives high recall and low precision values. It is very hard to find a balance between precision and recall values. In a Boolean search, if the Boolean query expression is found in a document that document is considered relevant otherwise not. Problem with this approach is that if in one document the query terms have appeared only once and on the other hand in another document the query terms appear several times there is no way to determine which document is more relevant. Solution to this problem is to record term frequency, number of times a term appears in a document. Boolean retrieval systems do not incorporate any result ordering mechanism. Solution to this is to perform ordering or ranking of the search results based on the term frequency scores. If a document has a higher term frequency score it will be listed higher in the output list and so on.

Boolean retrieval models have been heavily in use till early 1990s when World Wide Web (WWW) came into being. With the arrival of Internet, WWW and search engines, Boolean retrieval models were replaced with ranked retrieval models such as Vector Space Model (VSM). Ranked retrieval systems used free-text queries, set of one or more words, instead of boolean queries that used a precise language for specifying the query. Ranked retrieval systems also incorporated the full text search (where body of the document is also searched whereas in Boolean retrieval systems only title, author and keyword matching was performed) capability to the IR systems, for the first time.
Most of the IR systems now (including search engines) use VSM model for information retrieval.

A VSM views a document in a collection as a vector of terms. Every term in the document is assigned a weight according to the importance of that term. Weight assignment was done manually through editorial reviews but since it is not possible to review every document manually therefore automated term weighting schemes are used. Simplest of these approaches is tf-idf (term frequency – inverse document frequency) weighting. Term frequency (tf) is simply the number of times a term appears in a particular document. Document frequency (df) represents the importance of a term in that document. Inverse document frequency (idf) is obtained by taking logarithm of df. A combined tf-idf value is obtained by multiplying tf and idf values. Documents that have higher tf-idf scores are considered more relevant to the query and hence ranked higher in the search results.

IR over grids is a new concept. Grid computing presents a different perspective to the emerging need of computational power and data storage problems. There are many different types of grid networks but basic purpose of any grid network is sharing of computing resources to fulfil the demand of any user. Grid computing has a concept of Virtual Organizations (VO). A VO can span over a vast geographical area physically but is tightly coupled logically for the reasons of resource sharing and utilization. This projects aims to develop a GridIR system that is able to retrieve information from distributed document collections present on grid networks.

There are no particular live GridIR systems available to date therefore relative comparison of the effectiveness in information retrieval of this system is not possible but to determine the effectiveness of information retrieval, standard IR evaluation techniques have been used. Creation of distributed test collections for evaluation of GridIR systems is also very important as currently there are no distributed test collections available.
Chapter 3

Requirements

This chapter describes the requirements of the system. Requirements of the system have been identified by exploring different search engines and by examining the architectural and requirements specification documents which have been developed by Grid IR Working Group [13]. System’s requirements have been described for each of the Grid IR components, shown in figure1.

3.1 Collection Manager Service

Collection Manager Service (CMS) is exposed by a Graphical User Interface (GUI). An administrator uses the GUI to administer the document collections. CMS keeps record of the document collections that have been indexed or need to be indexed.

CMS is required to provide functionality for adding document collections to system for indexing. A document collection is a set of documents that are contained inside a folder. A collection owner shall forward the request of adding his collection to the system either by providing the address/location of his collection or by physically hosting his collection on the central location, where system is deployed.

CMS shall provide functionality for removing document collections from system. If a collection owner wishes to withdraw his collection from the system for whatever reason, he shall be able do so by forwarding a request to system administrator.

Collection owners and system administrators shall be able to view the indexes hosted by system. System should incorporate some functionality to display a list of indexes which are being searched by searching component of the system.

If a collection owner wishes to submit his collection that is already indexed then he shall be able to do so. This will enable plug and play behaviour to the system. Anytime an already indexed collection can be hooked to the system and the system shall be able to do the searching on newly added collection without using any complex procedures for discovering newly added collections. A collection owner shall be able to host his indexed collection either on the central location, where the system has been deployed, or by providing the location of the indexed collection.
CMS shall provide functionality to remove the hosted indexes. If a collection owner wishes to withdraw his collection, he shall be able to do so by putting a request to the system administrator. Figure 3 shows the use-case diagram for the Collection Manager Service.

![Use-case diagram for Collection Manager Service](image)

**Figure 3: use-case diagram for Collection Manager Service**

### 3.2 Indexing Service

Indexing service is responsible for creating indexes of document collections. An administrator uses the GUI to create indexes. Document collections are data sources, input to indexing service. Indexing service processes each document in the input collection to create an index.

Indexing service is required to take the documents, from document collections registered by CMS, as input. Indexing service should then perform the analysis of the input documents and index them. It is required to chop off the very common words, stop words, to minimize the size of index. If a collection owner wishes to submit the stop words particular to his collection, system should be able to utilize provided stop words list during indexing. Strategy for storing the index shall also be devised. Indexer should be able to do indexing operations on document collections available on central location as well as on remote locations.

Figure 4 shows the use-case diagram for indexing service.
3.3 Query Processing and Searching Service

Query processing service (QPS) takes input query submitted by the user and submits it to searching service. Searching service then performs search against the input query and returns the search results to QPS. QPS transforms the search results, applies formatting and displays them to user.

When a user submits a query for search, QPS shall accept the query and submit it to Searching service. Searching service is required to perform searching operation against the query on indexes available on central location as well as indexes on remote locations. Before searching the query terms, searching service should clean the input query by removing the stop words from the query. After the search results have been acquired they should be sent to QPS. QPS should then merge the search results returned by searching service, apply necessary formatting to merged results and display them to the user. Result merging approach should be developed to efficiently merge the search results.

Figure 5 shows the use case diagram for QPS and Searching service.
3.4 Security Requirements

No user shall be allowed to access document collections or indexed collections without identifying himself. A login mechanism is required for the administration of CMS, Indexing, QPS and Searching services.

3.5 User experience requirements

A web based application is required, to enable the end users interact with the system, for the purpose of submitting their queries. Web application shall be able to display the search results in a readable and easily browse-able format. A web application is required for administrator user as well to help him administer the CMS, Indexing, QPS and Searching services.

3.6 Technology requirements

System shall be implemented as a set of grid enabled services and for this purpose OGSA-DAI framework shall be used to access local and remote data sources (indexed collections). Lucene is an open source text searching library and should be used for the purpose of implementing searching and indexing services. System is required to run on any operating system and therefore Java language should be used. For the purpose of implementing web applications, for end users and administrators, JSP should be used. OGSA-DAI framework has two versions, one for Axis [14] based web applications and other for Globus Toolkit [15] based web applications. This project shall use OGSA-DAI Axis framework for the deployment of web applications.
Figure 6 shows the use-case diagram of system.
Figure 6: Use-case diagram of the system
Chapter 4

Design & Implementation

This chapter presents system design and implementation. To start with system design, firstly basic and distributed model of interaction is presented. After presenting communication model, system architecture is explained. The system has been designed as a series of grid services deployed on a central location over a grid network. System architecture is further explained in detail in terms of individual system components. Design class diagram has been drawn to further explain the system in terms of objects.

Core architecture for server side and client side activities is explained in the system implementation. Structure and implementation of GridIR system specific data resources is described. Unit test suite design and implementation has been presented in the last section of the chapter.

4.1 Basic model of interaction

The system is a three-tier distributed client/server application. Figure 7 shows the basic model of interaction of the system. Top layer is the user interface layer provided by the GridIR Application. The middle-tier consists of process management services such as, process creation, process execution, process monitoring and process termination. OGSA-DAI framework is also present on middle-tier. Business logic components in middle tier are GridIR Application specific activities and resources. Data sources are present on third-tier. Tomcat is used as an application server on middle-tier.

A user accesses the GridIR Application by using a web browser. GridIR Application is deployed inside the application server. GridIR Application uses the services provided by OGSA-DAI framework to access the data source i.e. the indexed document collection.
A user initiates the communication cycle by accessing the user interface of the GridIR system and submits the request for information. GridIR system receives the request and by using activities and resources of OGSA-DAI framework and GridIR system specific activities and resources accesses the indexes. A search is performed by GridIR system and search results are sent back to the user. The communication cycle is completed once user has received search results.

4.2 Distributed model of interaction

Distributed model of interaction, shown in Figure 8, suggests how the GridIR system is used in a distributed environment. In a distributed model of communication, GridIR System needs to be hosted at only one central location and system specific activities and resources are deployed on each of the remote application servers. Remote application servers can exist anywhere on the internet and host OGSA-DAI framework and GridIR application specific activities and resource, in order to successfully interact with the GridIR application. Distributed calls are handled by OGSA-DAI framework. Distributed model of GridIR application is hidden to the user of the system as user of the system always accesses the user interface layer, which is deployed on central location. Also user of the system is unaware of the location of indexed collections.
A user initiates the communication cycle by accessing the user interface of the GridIR system and submits the request for information. GridIR system receives the request and by using activities and resources, defined in OGSA-DAI framework, and business logic components of GridIR system, to access the local and remote indexes. A search is performed on local and remote indexes and search results returned by remote servers are then merged to produce one integrated list of search results. Merged search results are sent back to the user. The communication cycle is completed once the user has received search results.

4.3 System Architecture

System architecture describes the conceptual representation of the GridIR system. System architecture diagram in Figure 9 shows the structure of GridIR system, major components of the system and their working. GridIR system has tree major components, Collection Manager, Indexing service and Query processing and searching service. Besides the GridIR system components, OGSA-DAI framework hosts the activities and data resources required for the working of GridIR system. Activities are units of work, described in the form of classes, and a group of activities constitutes a workflow. Data resources are files that are abstractions of the indexed document collections.
Figure 9: High level system architecture diagram
Data resources contain information about: activities that are required to run the workflow, metadata information of the index (such as fields of the index to search upon, fields of the index to output, location of the index), and login and access parameters. Section 4.6 discusses the data resources in detail.

OGSA-DAI framework is a set of activities and resources that are available for use and can be extended to suit the needs of any particular application and in this case GridIR system. GridIR system specific activities are set of activities that are required to execute the workflows of GridIR system. CollectionsDB is a database and is implemented in MySQL. CollectionsDB stores information about document collections and indexes. Collection Manager Service of the GridIR system interacts with CollectionsDB to either update or select the information regarding document collections and indexes.

A user uses the web browser to place a request for information need. When a user inputs his query, the Query Processing Service (QPS) accepts the query and forwards it to the searching service. The QPS has four major functions. It forwards the query, submitted by the user, to searching service. The QPS selects the indexes that need to be searched upon by the Searching service. After the searching service has produced search results, these results are sent back to QPS. The QPS integrates the search results received from searching service and produces single merged list of results. Merged results are formatted in an appropriate format and sent back to the user. QPS interacts with the user interface components, index.jsp and results.jsp in Figure 9, to collect the input query and display the search results. QPS also interacts with searching service. Interaction is in the form of function calls on objects of searching service. The QPS has been discussed in detail in section 4.3.3.

Searching service is responsible for performing searches for the query forwarded by the QPS. The query submitted by the QPS is input to the searching service. Searching service interacts with OGSA-DAI framework activities and GridIR system specific activities and resources to search the indexes deployed on local and remote application servers. Search results returned by local and remote application servers are sent back to QPS. Searching service and been discussed in detail in section 4.3.3.

Collection manager is responsible for managing the document collections. Collection manager interacts with collections data-store (CollectionsDB), which is implemented in MySQL and stores information about document collections and indexes. Indexing service is fed with document collections whose record is stored in CollectionsDB. After the indexing service has finished creating the index, system administrator updates the CollectionsDB and inserts a new record for newly create index. QPS selects the indexes that need to be searched, by Searching service, from OGSA-DAI resources repository. After selecting the indexes, QPS inputs them to the Searching service. Searching service then performs the searching operation on the selected indexes. Section 4.3.1 discusses the Collection Manager Service in detail while Section 4.4 discusses the structure of the database in detail.

Indexing service is responsible for creating indexes on the document collections provided by the collection owners. Document collections are the input to Indexing service and an index is the output of the Indexing service. When an index is created for
a document collection. CollectionsDB is updated with the information of newly created index. When the information of newly created index is added to the CollectionDB, the index becomes available for searching by Searching service. Section 4.3.2 discusses the architecture and operations performed by Indexing service in detail.

4.3.1 Collection Manager Service (CMS)

Figure 10 shows the architecture of CMS. CollectionsDB is the database which stores the information about the document collections and indexes. Collection manager service is exposed by a graphical user interface. CMS performs five major functions:

i- Adding new document collections to the collections database (CollectionDB)

Process of adding new document collections begins when a request is received from a collection owner to register his documents collection for use with the GridIR system. Once the request has been approved by the GridIR system administrator and the document collection has been made available (either by providing the address of the document collection where it has been hosted or by placing the document collection on the central location, where the GridIR system has been deployed), a system administrator uses the user interface of the Collection Manager Service to add the information of the collection into the CollectionsDB. A new record is inserted in the Collections table (Table 3) of the CollectionsDB.

ii- Removing document collection from database on the request of collection owner

If a document collection owner wishes to withdraw his collection from the GridIR system, he sends a request to the system administrator. The system administrator then, by using the user interface of the Collection Manager Service, deletes the record of the concerned document collection from the CollectionsDB. Two possible scenarios can happen in this situation:

1- The document collection has not been indexed by the Indexing service

If the document collection has not been indexed then only one table (Collections table, Table 3) in the CollectionsDB is updated.

2- The document collection has been indexed by the indexing service and is being utilized by the Searching service

If the document collection has been indexed then two tables in the CollectionsDB (Collections and Indexes, Table 3 and Table 4) are updated.

iii- Adding new indexes to the database (which are available for searching once they have been added to the CollectionsDB)

It is possible to add already indexed collections to the GridIR system. If a collection owner wishes to provide an indexed collection, he sends a request
to the system administrator of GridIR system. After the request has been approved by the system administrator, the collection owner provides his indexed collection either by providing the address of the remote location where collection has been hosted or by placing the collection on the central location, where the GridIR system has been deployed. The system administrator uses the user interface provided by the Collection Manager Service to insert the record of the indexed collection into CollectionsDB.

When an already indexed collection is added to the GridIR system, only one table in the CollectionsDB (Indexes, Table 4) is updated and a new record is added in the table. As soon as the record has been inserted into the database the collection becomes available for search.

iv- Removing indexed collections from the CollectionDB on the request of collection owner

If a document collection owner wishes to withdraw his index from the GridIR system, he sends a request to the system administrator. The system administrator then, by using the user interface of the Collection Manager Service, deletes the record of the concerned indexed collection from the CollectionsDB.

When removing an indexed collection from the GridIR system, only one table (Indexes, Table 4) in the CollectionsDB is updated. A record is deleted from the table and the collection is no more available to the Searching service.

v- Providing the facility to view the document collections and indexes being hosted by the GridIR system.

If a system administrator wishes to view the document collections and indexes being used by the GridIR system, he uses the user interface provided by the Collection Manager Service to view the list of collections being used by the GridIR system.
It is imperative to implement a security mechanism in order to prevent the misuse of the system and for this purpose a login mechanism is used. The user accesses the user interface of the CMS and provides his username and password to the CMS. The CMS performs the authentication process and if the credentials provided are correct, access to perform the management operations is granted.

4.3.2 Indexing service

Figure 11 shows the architecture of Indexing service. Indexing service is responsible for creating indexes on the document collections that are provided as input to the service. Doc.Coll-1 and Doc.Coll-R in Figure 11 are the document collections input to the indexing processes. CollectionsDB is the database used to select the document collections that are to be indexed. Index-1 and Index-R in Figure 11 are the output of indexing processes.
An authenticated user is granted access to invoke the indexing process. Indexing service is provided with document collections that are to be indexed. Indexing service can perform indexing in following two scenarios:

1- Document collections are available on central location, where GridIR is deployed

   If document collections are provided on central location, the index is also created and stored on central location.

2- Document collections are placed on remote locations

   If document collection exists on remote location and address of the remote location is provided, then index is created and stored on the location advised by the collection owner.

   Indexing service performs indexing on .txt files only.

4.3.3 Query Processing and Searching Service

Figure 12 shows the architecture of Query processing and searching service. This service consists of two sub-services, Query processing service and Searching service. Query processing service accepts the query submitted by the user. Query processing service then selects the data resources from OGSA-DAI resources repository. Each data resource file in the OGSA-DAI resources repository represents a data source. QPS only selects those data resources which are related to the GridIR system. One resources have been selected they are submitted to the searching service along with the input query. Searching service makes use of indexes and data resources, selected by Query
processing service, to search local and remote indexes. Data resources that provide mapping of the indexes hosted on the same server are called local resources while those that provide information of the remotely hosted indexes are called remote resources.

Figure 12: Query processing and searching service

Searching the indexed collections involves following two scenarios:

1- Searching the collections that are hosted on the central location, where GridIR system is deployed

If the index has been hosted on the same server where GridIR system is deployed then searching service is provided with the data resource file, by the Query processing service, of the index which is to be searched. Searching service reads the contents of the resource file and performs a search on the index. Data resource files provide direct mapping to the underlying index which is to be searched. In case where the index is hosted on the central server the data resource file provides the exact location (address of the index on the file system), metadata fields (the fields of the index that needs to be searched) and output fields (the fields of the index from where the output text should be selected) of the index. A detailed discussion on data resources has been laid out in section 4.6. Search results are sent back to Query processing service.

2- Searching the collections that are hosted on remote locations
If the index has been hosted on the remote server then searching service is provided with the data resource file, by the Query processing service, of the index which is to be searched. Searching service reads the contents of the data resource file and executes the search operation on the remote server where the index has been hosted. In this scenario remote resources are not direct mappings of the remote index. Instead they provide the location of the remote server, where the index has been hosted. Remote resources contain the id of the data resource file, which is hosted on the remote server and provides direct mapping to the index hosted on the remote server.

Remote search operation starts with reading the contents of the remote resource. Searching service reads the location of the remote server and id of the data resource on the remote server, from the remote resource file. Searching service then uses this information to execute the remote search operation. GridIR system specific activities on the remote server are contacted and id of the data resource is provided to the activities on the remote server. GridIR system specific activities on the remote server then use the resource id to access the resource file in the OGSA-DAI resource repository and after reading the contents of the resource file access the indexed collection. A search is performed on the index and search results are sent back to the searching service. Searching service sends results back to the Query processing service.

Upon receiving the search results from the searching service, Query processing service performs merging of the results. There are three merging approaches used to merge the search results. These merging approaches have been discussed in detail in section 4.7. Once the merging operation is complete, the Query processing service applies formatting on the search results and sends them back to the user.

### 4.4 CollectionsDB

CollectionsDB is the database used by Collection manager and Indexing service to store and retrieve the information regarding document collections and indexes. This database contains three tables. First table, users, store the user names and passwords for system administrators, the persons that can access Collection Manager and Indexing service. Table 2 shows the structure of users table. Second table, collections, stores the information about collections and Table 3 shows the structure of this table. Third table, Indexes, contains the information about the indexes. Table 4 shows the structure of Indexes table.

<table>
<thead>
<tr>
<th>Item no.</th>
<th>Fields</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>userId</td>
<td>auto-number</td>
<td>This is an auto-number field that keeps count of the number of users and works as primary key of the table as well.</td>
</tr>
<tr>
<td>2.</td>
<td>userName</td>
<td>varchar(16)</td>
<td>This field stores the user name. Usernames can be upto 16 characters long.</td>
</tr>
</tbody>
</table>
userPassword | Varchar(16) | This field stores passwords.

**Table 2: Users table**

Table 2 contains three fields: userId, userName, userPassword. GridIR system has two types of users. One type of users are general who wish to perform searches and these users do not need to register with the system. Any person can use the user interface layer to perform search. Second group of users are system administrators who are involved with the administration and management of the system and they need to be registered with the system. Passwords are stored in clear text format.

<table>
<thead>
<tr>
<th>Item no.</th>
<th>Fields</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>collectionId</td>
<td>auto-number</td>
<td>This is an auto-number field that keeps count of the number of records and works as primary key of the table as well.</td>
</tr>
<tr>
<td>2.</td>
<td>collectionOwner</td>
<td>varchar(16)</td>
<td>This field contains the name of collection owner.</td>
</tr>
<tr>
<td>3.</td>
<td>collectionName</td>
<td>Varchar(16)</td>
<td>This field stores the name of the collection.</td>
</tr>
<tr>
<td>4.</td>
<td>collectionType</td>
<td>Varchar(6)</td>
<td>This field describes whether the collection is hosted on central location or remotely. Two possible values are: local or remote</td>
</tr>
</tbody>
</table>

**Table 3: Collections table**

Collections table, Table 3, contains four fields. This table is populated by the CMS.

<table>
<thead>
<tr>
<th>Item no.</th>
<th>Fields</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>indexId</td>
<td>auto-number</td>
<td>This is an auto-number field that keeps count of the number of records and works as primary key of the table as well.</td>
</tr>
<tr>
<td>2.</td>
<td>indexOwner</td>
<td>varchar(16)</td>
<td>This field contains the name of index owner.</td>
</tr>
<tr>
<td>3.</td>
<td>indexResourceId</td>
<td>Varchar(64)</td>
<td>This field stores the resource name which contains the descriptions about index.</td>
</tr>
</tbody>
</table>
4. **indexType**  Varchar(6)  This field describes whether the index is hosted on central location or remotely. Two possible values are: local or remote.

5. **indexName**  Varchar(16)  This field stores the name of the index.

<table>
<thead>
<tr>
<th>Table 4: Indexes table</th>
</tr>
</thead>
</table>

Indexes table, Table 4, has four fields. Whenever an index is created on a document collection, a new row is added to the Indexes table.

### 4.5 Document collections and Indexes

Document collections are input to the Indexing service of the GridIR system and are required to create indexes. An index has been created on an ad-hoc document collection that has been created to assist the implementation of the GridIR system. The ad-hoc collection contains text files in various different formats including .txt, .java and .html. As the GridIR system is a distributed application therefore the document collection has been divided into two halves and an index has been created on both halves, by using the Indexing service of the GridIR system. One of the indexes is hosted on a central server while other is hosted on the remote server.

For the purpose of evaluating the GridIR system Cranfield test collection has been used. Cranfield collection is one of the oldest and trusted test collections. It is a small collection that contains 1400 hand picked abstracts. Collection also contains a set of 225 queries and relevance judgements of all (query, document) pairs. Test collection was divided into two halves. Each half contained 700 documents. First half was indexed and deployed on central server while second half was indexed and deployed on remote application server.

### 4.6 Resources

OGSA-DAI resources are structured files that contain the details of underlying data sources. These resource files describe the type of data source; the activities that are required during the running of application, login details and any other important parameters that application may require during the course of its run. OGSA-DAI provides implementation for data resources that map the relational databases, XML data sources and file system based data sources. Data resources that are provided by the OGSA-DAI can be classified in two categories. One is called local resource and other type is called remote resource. Local resources map the underlying data source present on the central server and remote resources provide the details of remote server that hosts the remote collection.

New data resources can be created for GridIR system specific data sources to meet the requirements of a particular application. The GridIR system uses indexed document
collections as its data source therefore new data resource types, LuceneResource and RemoteLuceneResource, have been defined. LuceneResource type represents the data resources that describe the indexed document collections present on the central OGSA-DAI server while RemoteLuceneResource type represents the data resources that are present on remote OGSA-DAI servers. Resource implementation details are provided in section 4.9.1.

Distributed model of information retrieval means that indexed collections can be hosted on remote locations and a resource file is created against each indexed collection.

### 4.6.1 Resource description

Resource descriptions provide information on the contents of indexes. GridIR system is hosted on one central location and detail about all the indexes whether they are local to the server or exist on remote servers is made available on this central location. These details are contained in the data resource files that are hosted in the OGSA-DAI resources repository. Every time an index owner wishes to make his collection available for searching either he can submit the document collection to the CMS and then indexing service can perform the indexing or he can do the indexing himself and provide a resource file that contains the metadata of his index. This metadata in the resource file is used by the searching service to perform the search on the collection.

There are two scenarios in providing data resource and data resource description.

1. Providing data resource and description for the indexed collections that are hosted on the central server

   If an index is hosted on the central server then only one resource files, local resource file, is needed to access the index.

2. Providing the data resource and description for remotely hosted index

   If the index is hosted on a remote location then we need two resource files, local resource and remote resource, to access this remote index. Both of these resource files are provided by the collection owner or are created by the system administrator at the request of collection owner. Remote resource is hosted on the central location while the local resource is hosted on the server where index is hosted.

### 4.6.2 Resource selection

Resource selection is the process of finding out which indexed collections need to be searched against the input query. Resource selection is done by the Query processing service. Query processing service selects the indexes that need to be searched and for the sake of simplicity we search all document collections against the query therefore we select all the data resources provided on the central location.
4.7 Merging ranked search results

Query processing service performs the merging of search results. Three methodologies have been used to merge the search results. These are concatenation merge, round-robin merge and ordered merge. Each of these approaches is outlined below.

4.7.1 Concatenation merge

To perform a concatenation merge we take the lists of search results as input to the ConcatMerge activity and create a new list of results. Figure 13 shows the flow chart of the concatenation merge.

![Flow chart of concatenation merge](image)

Figure 13: concatenation merge

Concatenation merge algorithm takes the search result lists as input. The algorithm processes the input lists in first come first serve basis. A new merged list is created and all the search result lists are appended to the new list.

4.7.2 Round-robin merge

Round-robin merge, as the name suggests, merges the search results by taking one result from each of the input result lists. RoundRobinMerge activity takes the search result lists as input and creates a new list to hold the results of round-robin merge. Input search result lists are then traversed such that one result is selected from each list at a time and added to the newly created results.
Figure 14: Round robin merge

Explanation of the steps mentioned in algorithm is as follows:

1- Start of algorithm

2- Total number of lists to be merged are stored in N

3- If all the records, in each lists, have been traversed then break the loop otherwise continue to step 4

4- Start the for-loop, initialize loop variable (i) to zero. Iterate over the input lists, accessing one record from each list.

5- Check to see if value of (i) is equal to N (total lists to merge). If the value of (i) is equal to N, then one record from each of the input lists has been traversed and stored in merged list (step-7) therefore reset the loop variable (i) to 0 (step-4) but before resetting the i=0 perform step-3

6- Store the next record of the current list in the merged list. Getting the record from the current list (loop variable (i) points to the current list) and storing it into the merged list is a single step and once a record has been traversed from the current list, list pointer points to next record automatically. So next time
when the list is accessed, new record is accessed because the list is never reset only the loop variable (i) is reset.

7- Merged list

8- Go to next list

4.7.3 Ordered merge

Ordered merge performs the merging of search results in order of ranking scores of the search results. A merge sort algorithm is applied to merge the input search result lists and a new sorted list of search results is produced as a result of merge sort. The results of the sorted list are then sent in inverse order. Figure 15 shows the flow chart of the ordered merge activity.

Figure 15: Ordered merge

Ordered merge activity does its work in two stages. In first stage, a simple concatenation merge is performed to create a merged list of search results. In the second stage a merge sort is performed on the merged list of search results. Search results are sorted on the basis of relevancy score.
4.8 Design class diagram

Figure 16 shows the class diagram of GridIR system.
Figure 16: class diagram
4.9 Implementation

Two OGSA-DAI Axis servers have been deployed in order to implement the GridIR system. One of the servers has been identified as the central server and the other as remote server. GridIR system has been deployed on the central server while GridIR specific activities and resources have been deployed on both servers. Indexed collections have also been hosted on both of the servers.

Figure 17 shows the implementation diagram of GridIR system.

![GridIR system implementation diagram](image)

A user accesses the GridIR system services by using a web browser. GridIR system services communicate with DataRequestExecutionService to utilize the OGSA-DAI services. When a user submits the query, GridIR system builds the workflow of activities and submits the workflow to the DataRequestExecutionService. DataRequestExecutionService uses DataRequestExecutionResource and other GridIR system specific resources to perform the search on the indexed collections.

4.9.1 Resource implementation

LuceneResource and RemoteLuceneResource contain the details of the underlying indexes. In order to access the details of the indexes these resources need to be accessed so that the information in the data resource files can be read. LuceneResource and RemoteLuceneResource are GridIR system specific resources and therefore contain fields that are only relevant to GridIR system. Data resource implementations provided
by the OGSA-DAI framework are generic in nature and therefore cannot be used with GridIR system specific data resources. In order to successfully process the GridIR system specific data resources, implementations have been provided for LuceneResource and RemoteLuceneResource.

Following classes and interfaces have been developed for the implementation of LuceneResource.

**LuceneDataResource**

LuceneDataResource class provides implementation for LuceneResource. Figure 18 shows the diagram of LuceneDataResource class.

```
LuceneDataResource
+createResourceAccessor()
+getResourceIndexMetadata()
+getResourceURL()
+getResourceFields()
+getResourceID()
+getState()
```

**Figure 18: LuceneDataResource**

**LuceneResourceAccessor**

LuceneResourceAccessor is an interface that provides access to LuceneResource. Figure 19 shows the diagram of the interface.

```
interface LuceneResourceAccessor
+getResourceIndexMetadata()
+getResourceURL()
+getResourceFields()
+getResourceID()
+getState()
```

**Figure 19: LuceneResourceAccessor interface**

**SimpleLuceneResourceAccessor**

SimpleLuceneResourceAccessor class implements the LuceneResourceAccessor interface. Figure 20 shows the diagram of the SimpleLuceneResourceAccessor.
LuceneDataResourceState

LuceneDataResourceState is a wrapper for generic data resource state allowing for a more usable access of GridIR system-specific resource information. Figure 21 shows the diagram of LuceneDataResourceState interface.

SimpleLuceneDataResourceState

SimpleLuceneDataResourceState class implements the LuceneDataResourceState interface. Figure 22 shows the diagram of SimpleLuceneDataResourceState.
Following classes and interfaces have been developed in order to provide implementation for RemoteLuceneResource.

**LuceneRemoteResource**

LuceneRemoteResource class provides implementation for RemoteLuceneResource. Figure 23 shows the diagram of LuceneRemoteResource class.

**LuceneRemoteResourceAccessor**

LuceneRemoteResourceAccessor is an interface providing access to the remote resource. Figure 24 shows the diagram of the LuceneRemoteResourceAccessor.
SimpleLuceneRemoteResourceAccessor

SimpleLuceneRemoteResourceAccessor class implements the LuceneRemoteResourceAccessor interface. Figure 25 shows the diagram of the SimpleLuceneRemoteResourceAccessor.

LuceneRemoteResourceState

LuceneRemoteResourceState is a wrapper for generic data resource state allowing for a more usable access of GridIR system-specific resource information. Figure 26 shows the diagram of LuceneRemoteDataResourceState interface.
4.9.2 Activities

This section briefly describes the server side and presentation layer activities that have been developed in order to implement the GridIR system.

**QueryProcessor**

QueryProcessor activity is part of the Query processing service. QueryProcessor activity accepts the input query from users, selects the indexes and data resources from OGSA-DAI resources repository and forwards them to the searching service. List of data resources is used by the LuceneQueryActivity and RemoteLuceneQueryActivity to search the collections. Figure 28 shows the structure of QueryProcessor activity.

**LuceneQueryActivity**

LuceneQueryActivity extends from MatchedIterativeActivity. LuceneQueryActivity is responsible for performing search operations on indexes that are hosted on the central server. This activity takes a query and a list of local resources as the inputs and performs the search. Figure 29 and Figure 30 show the structure of LuceneQuery and LuceneQueryActivity classes. LuceneQuery class is the presentation layer and is used to send inputs to and receive outputs from LuceneQueryActivity. Both these classes are part of the searching service.
RemoteLuceneQueryActivity

RemoteLuceneQueryActivity extends from MatchedIterativeActivity. RemoteLuceneQueryActivity is responsible for performing search operations on remote indexes. This activity takes a query and a list of remote resources as an input and performs the search on remote indexes. Figure 31 and Figure 32 show the structure of RemoteLuceneQuery and RemoteLuceneQueryActivity classes. RemoteLuceneQuery class is the presentation layer and is used to send inputs to and receive outputs from RemoteLuceneQueryActivity. Both these classes are part of the searching service.
**ConcatMergeActivity**

ConcatMergeActivity extends from MatchedIterativeMultipleInputActivity. Search results obtained from local and remote resources are sent to this activity and a list of merged results is produced as the output. Input search results are appended in order of occurrence. Figure 33 and Figure 34 show the structure of ConcatMerge and ConcatMergeActivity. ConcatMerge class is the presentation layer and is used to provide inputs to and outputs from ConcatMergeActivity. setNumberOfDataInputs function sets the total number of search result lists to be merged and connectDataInput function of ConcatMerge class takes the search result list as input. Both these classes are part of the Query processing service.

<table>
<thead>
<tr>
<th>ConcatMerge</th>
</tr>
</thead>
<tbody>
<tr>
<td>-DEFAULT_ACTIVITY_NAME</td>
</tr>
<tr>
<td>+setNumberOfDataInputs(in numberOfInputs : int)</td>
</tr>
<tr>
<td>+connectDataInput(in SingleActivityOutput, in index : int)</td>
</tr>
<tr>
<td>+getResultOutput()</td>
</tr>
<tr>
<td>+getInputs()</td>
</tr>
<tr>
<td>+getOutputs()</td>
</tr>
<tr>
<td>+validateIOState()</td>
</tr>
</tbody>
</table>

**Figure 33: ConcatMerge**

**ConcatMergeActivity**

<table>
<thead>
<tr>
<th>ConcatMergeActivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>-INPUT_TUPLES</td>
</tr>
<tr>
<td>+process()</td>
</tr>
</tbody>
</table>

**Figure 34: ConcatMergeActivity**

**RoundRobinMergeActivity**

RoundRobinMergeActivity extends from MatchedIterativeMultipleInputActivity. Search results obtained from local and remote resources are sent to this activity and a list of merged results is produced as the output. Output of this activity is search results appended in order of occurrence. Figure 35 and Figure 38 show the structure of RoundRobinMerge and RoundRobinMergeActivity. RoundRobinMerge class is the presentation layer and is used to provide inputs to and outputs from RoundRobinMergeActivity. setNumberOfDataInputs function sets the total number of search result lists to be merged and connectDataInput function of RoundRobinMerge class takes the search result list as input. Both these classes are part of the Query processing service.

<table>
<thead>
<tr>
<th>RoundRobinMerge</th>
</tr>
</thead>
<tbody>
<tr>
<td>-DEFAULT_ACTIVITY_NAME</td>
</tr>
<tr>
<td>+getResultOutput()</td>
</tr>
<tr>
<td>+connectDataInput()</td>
</tr>
<tr>
<td>+setNumberOfDataInputs()</td>
</tr>
<tr>
<td>+RoundRobinMerge()</td>
</tr>
</tbody>
</table>

**Figure 35: RoundRobinMerge**
OrderedMergeActivity

OrderedMergeActivity extends from MatchedIterativeMultipleInputActivity. Search results obtained from local and remote resources are sent to this activity and a list of merged results is produced as the output. Output of this activity is search results appended in order of occurrence. Figure 35 and Figure 38 show the structure of OrderedMerge and OrderedMergeActivity. OrderedMerge class is the presentation layer and is used to provide inputs to and outputs from OrderedMergeActivity. setNumberOfDataInputs function sets the total number of search result lists to be merged and connectDataInput function of OrderedMerge class takes the search result list as input. Both these classes are part of the Query processing service.

4.10 Unit test design and implementation

Figure 39 shows the test suite class diagram. The classes in test suite have been implemented for the purpose of unit testing. Junit framework has been used to implement the unit testing functionality. Two data resource files have been created and deployed on the central server and remote server. First data resource file maps the index, which has been created specifically for the purpose of unit testing, hosted on central server and second resource file maps the remotely hosted index.
4.10.1 BaseTest

BaseTest is the base class for all the testing classes and extends from the TestCase class of junit framework. This class contains the functionality related to setting up the environment prior to executing a test. A new index is created every time a test is run. Arrays of string type have been used to store the text which is indexed by using the Indexing service of the GridIR system. Index is stored in the file system.

4.10.2 LuceneQueryTest

LuceneQueryTest class extends from BaseTest and contains the functionality to test the LuceneQuery activity. Input is added statically to the LuceneQuery and the searching process is invoked. If the search completes successfully search results are returned and the search process is considered as working correctly. The search results are verified by comparing with the static answers.

4.10.3 RemoteLuceneQueryTest

RemoteLuceneQueryTest class extends from BaseTest and contains the functionality to test the RemoteLuceneQuery activity. Input is added statically to the RemoteLuceneQuery and the searching process is invoked. If the search completes successfully search results are returned and the search process is considered as working correctly. The search results are verified by comparing with the static answers.

4.10.4 ConcatMergeTest

ConcatMergeTest class extends from BaseTest and contains the functionality to test the ConcatMerge activity. Once the search results have been obtained they are sent to the ConcatMerge activity. Merged results are then compared with the static answers to verify that ConcatMerge activity works correctly.
4.10.5 RoundRobinMergeTest

RoundRobinMergeTest class extends from BaseTest and contains the functionality to test the RoundRobinMerge activity. Once the search results have been obtained they are sent to the RoundRobinMerge activity. Merged results are then compared with the static answers to verify that RoundRobinMerge activity works correctly.

4.10.6 OrderedMergeTest

OrderedMergeTest class extends from BaseTest and contains the functionality to test the OrderedMerge activity. Once the search results have been obtained they are sent to the OrderedMerge activity. Merged results are then compared with the static answers to verify that OrderedMerge activity works correctly.
Chapter 5

Results and Analysis

This chapter presents the results of testing and evaluation of GridIR system. First of all unit testing results have been presented and after that system evaluation results have been presented.

5.1 Unit testing

JUnit has been used to implement the unit testing. Details of test implementation have been presented in previous chapter (Section 4.10). Figure 40 shows the results of unit testing.

<table>
<thead>
<tr>
<th>Class</th>
<th>Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseTest</td>
<td>testBaseTest</td>
<td>Success</td>
</tr>
<tr>
<td>ConcatMergeTest</td>
<td>testConcatMerge</td>
<td>Success</td>
</tr>
<tr>
<td>ConcatMergeTest</td>
<td>testBaseTest</td>
<td>Success</td>
</tr>
<tr>
<td>LuceneQueryTest</td>
<td>testLuceneQuery</td>
<td>Success</td>
</tr>
<tr>
<td>LuceneQueryTest</td>
<td>testBaseTest</td>
<td>Success</td>
</tr>
<tr>
<td>OrderedMergeTest</td>
<td>testOrderedMerge</td>
<td>Success</td>
</tr>
<tr>
<td>OrderedMergeTest</td>
<td>testBaseTest</td>
<td>Success</td>
</tr>
<tr>
<td>RemoteLuceneQueryTest</td>
<td>testRemoteLuceneQuery</td>
<td>Success</td>
</tr>
<tr>
<td>RemoteLuceneQueryTest</td>
<td>testBaseTest</td>
<td>Success</td>
</tr>
<tr>
<td>RoundRobinMergeTest</td>
<td>testRoundRobinMerge</td>
<td>Success</td>
</tr>
<tr>
<td>RoundRobinMergeTest</td>
<td>testBaseTest</td>
<td>Success</td>
</tr>
</tbody>
</table>

Figure 40: Unit test results
5.2 Evaluation

Evaluation tests have been performed to evaluate the correctness of the system. Evaluation test runs have been performed on pre-formulated queries which are provided in the Cranfield collection. Cranfield collection also provides the relevance results of the provided query set. System was run against the queries provided and results were compared to the pre-compiled results.

It is very hard to get highly accurate results with Cranfield collection because the queries are long and in natural language. Top ten search results are used to determine the effectiveness of the system. These top ten results will be compared to relevance results provided by the Cranfield test collection.

Following is the first query in the collection:

*What similarity laws must be obeyed when constructing aero elastic models of heated high speed aircraft?*

This query is known to have 29 relevant documents as listed below:

\{d_{184}, d_{29}, d_{51}, d_{12}, d_{51}, d_{102}, d_{13}, d_{14}, d_{15}, d_{57}, d_{378}, d_{859}, d_{185}, d_{30}, d_{37}, d_{52}, d_{142}, d_{195}, d_{875}, d_{56}, d_{66}, d_{95}, d_{162}, d_{197}, d_{858}, d_{876}, d_{879}, d_{880}, d_{486}\}

When this query is run, records are returned in order of relevance with most relevant result being the first in the list and least relevant being the last. We only consider top ten results returned by the system to evaluate the effectiveness of the retrieval. We have used three different approaches to obtain the final results. By using these approaches we get following documents:

**Concatenation-merge:** \{d_{184}, d_{486}, d_{12}, d_{13}, d_{51}, d_{14}, d_{311}, d_{195}, d_{172}, d_{141}\}

**Round-Robin merge:** \{d_{184}, d_{1268}, d_{486}, d_{858}, d_{12}, d_{1362}, d_{13}, d_{792}, d_{51}, d_{746}\}

**Ordered-merge:** \{d_{184}, d_{1268}, d_{486}, d_{12}, d_{13}, d_{51}, d_{878}, d_{14}, d_{1362}, d_{792}\}

Two measures for evaluating the results are precision and recall. These measures have been explained in chapter 2.

For all these approaches we can calculate the precision and recall values according to the returned values. Table 5, Table 6 and Table 7 below show the results for the above mentioned approaches. For each value of N, number of documents returned, we determine whether the retrieved document is relevant or not. If the document is relevant, we add 1 to the value of n otherwise the value stays same. For each value of N and n we calculate precision and recall values.
<table>
<thead>
<tr>
<th>No. of returned (N)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of relevant in returned (n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Precision (n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.85</td>
<td>0.87</td>
<td>0.77</td>
<td>0.70</td>
</tr>
<tr>
<td>Recall (n/29)</td>
<td>0.03</td>
<td>0.06</td>
<td>0.10</td>
<td>0.13</td>
<td>0.17</td>
<td>0.20</td>
<td>0.20</td>
<td>0.24</td>
<td>0.24</td>
<td>0.24</td>
</tr>
</tbody>
</table>

**Table 5: Precision-Recall table for concatenation merge**

<table>
<thead>
<tr>
<th># of retrieved(N)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td># of relevant in returned(n)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Precision (n/N)</td>
<td>1.00</td>
<td>0.50</td>
<td>0.67</td>
<td>0.50</td>
<td>0.60</td>
<td>0.50</td>
<td>0.57</td>
<td>0.50</td>
<td>0.55</td>
<td>0.50</td>
</tr>
<tr>
<td>Recall (n/29)</td>
<td>0.03</td>
<td>0.03</td>
<td>0.06</td>
<td>0.06</td>
<td>0.10</td>
<td>0.10</td>
<td>0.13</td>
<td>0.13</td>
<td>0.17</td>
<td>0.17</td>
</tr>
</tbody>
</table>

**Table 6: Precision-Recall table for round-robin merge**

<table>
<thead>
<tr>
<th># of retrieved(N)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td># of relevant in returned(n)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Precision (n/N)</td>
<td>1.00</td>
<td>0.50</td>
<td>0.67</td>
<td>0.75</td>
<td>0.80</td>
<td>0.83</td>
<td>0.71</td>
<td>0.75</td>
<td>0.67</td>
<td>0.60</td>
</tr>
<tr>
<td>Recall (n/29)</td>
<td>0.03</td>
<td>0.03</td>
<td>0.06</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.17</td>
<td>0.20</td>
<td>0.20</td>
<td>0.20</td>
</tr>
</tbody>
</table>

**Table 7: Precision-Recall table for ordered merge**

Figure 41 shows the precision of the search results as a function of the number of results returned.
Figure 41: Precision vs. no. of returned documents for concat, round-robin and ordered merge

Figure 42 shows the recall of the search results as the function of documents returned.

Figure 42: Recall vs. no. of returned documents for concat, round-robin and ordered merge
Figure 43: Precision-recall curve for concatenation merge, round-robin merge and ordered merge

Figure 43 shows the precision-recall curve for the results mentioned in Table 5, Table 6 and Table 7.

Firstly a comparison of the three merging strategies has been presented and after that discussion on search results has been presented.

Concatenation merge strategy seems to work best among the three strategies. By comparing the results of three merging strategies directly from Figure 43 it is clear that concatenation merge has higher recall and higher precision than other two strategies.

On the other hand concatenation merge is also susceptible to the ordering of document collections. This phenomenon can be explained by a simple example. In concatenation merge, search results are merged and a new merged list is created, by adding the search results of every document collection from every OGSA-DAI server at the end of the list. Now for example, there are two document collections and a search is performed against a particular query - "breast cancer" - without ordering the document collections. First document collection contains documents of generic nature while second collection contains medical articles. The search returns 150 results from first collection and 50 results from second collection. As the document collections are not properly ordered therefore results from medical collection are added at the end of the results from the generic collection. Since second document collection is a specialized collection and the query posed by the user also falls into the same category as the collection is therefore search results from second document collection are, may be few but, potentially highly relevant. Since first document collection contains documents of no particular category therefore finding more precise and relevant documents is hard. As a result, search results displayed to the user may not be relevant to his information need. On the other
hand if document collections are sorted prior to performing the search then search results from second document collection will be placed and displayed up in the order hence increasing the chances of finding relevant documents.

Overall the results in Figure 43 show that whenever there is a drop in precision there is a drop in recall as well. GridIR system retrieves at least five and at most seven relevant results in top ten which is good.

There is another interesting fact which is related to ordered-merge and round-robin merge only and that is, there is a sudden drop in precision for second document but after that the graph seems to progress smoothly until for few last documents for ordered merge while there are frequent ups and downs for round-robin merge approach.

Let’s just consider the case of ordered merge for explaining this phenomenon and explanation will actually suffice the round-robin merge results as well. The sudden drop means that the document retrieved at second number is actually not relevant to the input query. So it is implied that a non-relevant document has been retrieved so high in the list. Remember that ordered merge returns the results according to the relevance score and round-robin merge returns the results by picking one result each from search results lists returned by application servers (in this case OGSA-DAI servers). Explanation of the reasons behind this situation is presented in a moment but first another method for evaluating the effectiveness of the GridIR system is presented.

There is another quantitative measure to determine the effectiveness of system. This measure is called F measure and is defined as follows:

\[
F = \frac{2PR}{P+R}
\]

F measure takes into account both precision and recall values. Precision and recall values of 10th document returned from Table 5, Table 6 and Table 7 have been used. Therefore:

Concatenation merge: \(P=0.70, R=0.24\) and \(F=0.36\)

Round-robin merge: \(P=0.50, R=0.17\) and \(F=0.25\)

Ordered merge: \(P=0.60, R=0.20\) and \(F=0.30\)

It is clear that concatenation merge performs better than other two merging strategies but it should be noted that only top 10 returned results are being considered. As limitation of concatenation merge has earlier been explained with an example therefore it is very hard to say that concatenation merge is the best merging approach. Furthermore, only top 10 search results are being considered and if more results are considered, situation might be different and some other approach may prove more effective.
Both these measures quantitatively measure the effectiveness of the three approaches. Precision-recall curve and F measure is a two state model of evaluation i.e. either the retrieved result is relevant or it is not relevant. There is though no record of how much relevant a relevant-retrieved result is. All that is known is that the search is producing handful number of relevant results in top ten.

Since GridIR system is performing ranked retrieval therefore every result also has a relevancy score assigned to it and that score tells the magnitude of relevance of the retrieved result. GridIR system uses tf-idf scoring formula. Table 8, Table 9 and Table 10 show the document-relevancy score for each of approach.

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d_{184}</th>
<th>d_{486}</th>
<th>d_{12}</th>
<th>d_{13}</th>
<th>d_{51}</th>
<th>d_{14}</th>
<th>d_{311}</th>
<th>d_{195}</th>
<th>d_{172}</th>
<th>d_{141}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>28.34%</td>
<td>19.93%</td>
<td>16.24%</td>
<td>16.10%</td>
<td>15.14%</td>
<td>11.44%</td>
<td>7.96%</td>
<td>7.88%</td>
<td>7.28%</td>
<td>6.80%</td>
</tr>
</tbody>
</table>

Table 8: document relevancy scores for concatenation merge

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d_{184}</th>
<th>d_{1268}</th>
<th>d_{486}</th>
<th>d_{478}</th>
<th>d_{12}</th>
<th>d_{1362}</th>
<th>d_{13}</th>
<th>d_{792}</th>
<th>d_{51}</th>
<th>d_{766}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>28.34%</td>
<td>24.85%</td>
<td>19.93%</td>
<td>12.89%</td>
<td>16.24%</td>
<td>9.88%</td>
<td>16.09%</td>
<td>9.78%</td>
<td>15.14%</td>
<td>9.61%</td>
</tr>
</tbody>
</table>

Table 9: document relevancy scores for round-robin merge

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d_{184}</th>
<th>d_{1268}</th>
<th>d_{486}</th>
<th>d_{12}</th>
<th>d_{13}</th>
<th>d_{878}</th>
<th>d_{14}</th>
<th>d_{1362}</th>
<th>d_{392}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>28.34%</td>
<td>24.85%</td>
<td>19.93%</td>
<td>16.24%</td>
<td>16.09%</td>
<td>15.14%</td>
<td>12.89%</td>
<td>11.44%</td>
<td>9.88%</td>
</tr>
</tbody>
</table>

Table 10: document relevancy scores for ordered merge

By contrast to previous quantitative measurements, values in Table 8, Table 9 and Table 10 do not reflect the same results. Even though there are a handful number of relevant results in the top ten but they don’t seem to be highly relevant in terms of the relevancy score. Top most result in the list is actually only 28.34% relevant to the input query and the magnitude of relevance drops as low as 6.80% in just top ten results.

Now the discussion of the behaviour earlier identified and depicted in Figure 43 is resumed and explanation of the possible causes of this phenomenon has been outlined. Discussion of this trend is also relevant in here because there is a co-relation between the precision-recall values and document-relevancy pairs.

It was said that second result in the list, d_{1268}, is not relevant to the input query. By looking at the relevancy scores it is clear that this non-relevant result has a higher relevancy score than other relevant results appearing further down in the list and for this reason it managed to appear in the second spot. Now the question arises, how come a non-relevant document gets a higher relevancy score? And the answer is that this is because of the input query. The input query is in natural language and GridIR system is...
not capable of handling natural language queries very well. This effect can be named as natural language pollution. Following query is being used to perform the search:

*What similarity laws must be obeyed when constructing aero elastic models of heated high speed aircraft?*

By looking at the query it is clear that there are lot of terms in the query. When system performs a search it actually cuts off the unnecessary words from the query. These words are called stop words. The results we presented above are computed using a stop-word list of 31 words. Stop word list can be found in Appendix – B.

There are many query cleansing approaches we can use to enhance the precision, recall and relevancy scores. Some of these approaches are:

- Using a stop-word list that has higher number of stop words
- Using free text queries instead of natural language queries
- Stemming and lemmatization (for both index and query)

Some of these approaches increase precision others increase recall and yet some result in increased relevance score.

### 5.2.1 Using stop words

GridIR system has been experimented by using two additional stop-word lists containing 154 and 524 stop words each and by using free-text queries. Stop word lists that we used can be found in Appendix-B.

Table 11, Table 12 and Table 13 show the results when using 154 stop words list.

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d_{486}</th>
<th>d_{184}</th>
<th>d_{13}</th>
<th>d_{51}</th>
<th>d_{14}</th>
<th>d_{141}</th>
<th>d_{195}</th>
<th>d_{38}</th>
<th>d_{252}</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Precision (n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.86</td>
<td>0.88</td>
<td>0.78</td>
</tr>
<tr>
<td>Recall (n/29)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.21</td>
<td>0.24</td>
<td>0.24</td>
</tr>
<tr>
<td>Score</td>
<td>21.15%</td>
<td>18.16%</td>
<td>13.79%</td>
<td>13.67%</td>
<td>8.96%</td>
<td>7.85%</td>
<td>6.74%</td>
<td>6.69%</td>
<td>5.76%</td>
</tr>
</tbody>
</table>

*Table 11: Precision-Recall and score table for concatenation merge as a function of no. of documents returned*
<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d486</th>
<th>d878</th>
<th>d184</th>
<th>d1268</th>
<th>d12</th>
<th>d746</th>
<th>d13</th>
<th>d875</th>
<th>d51</th>
<th>d247</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>0.50</td>
<td>0.67</td>
<td>0.50</td>
<td>0.60</td>
<td>0.50</td>
<td>0.57</td>
<td>0.63</td>
<td>0.67</td>
<td>0.60</td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.03</td>
<td>0.07</td>
<td>0.07</td>
<td>0.10</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.21</td>
</tr>
<tr>
<td>Score</td>
<td>21.15%</td>
<td>11.01%</td>
<td>18.16%</td>
<td>9.92%</td>
<td>13.79%</td>
<td>8.21%</td>
<td>13.67%</td>
<td>7.47%</td>
<td>8.96%</td>
<td>5.77%</td>
</tr>
</tbody>
</table>

Table 12: Precision-Recall and score table for round-robin merge as a function of no. of documents returned

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d486</th>
<th>d184</th>
<th>d12</th>
<th>d13</th>
<th>d878</th>
<th>d1268</th>
<th>d51</th>
<th>d746</th>
<th>d14</th>
<th>d875</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>0.67</td>
<td>0.71</td>
<td>0.63</td>
<td>0.67</td>
<td>0.70</td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.10</td>
<td>0.14</td>
<td>0.14</td>
<td>0.14</td>
<td>0.17</td>
<td>0.17</td>
<td>0.21</td>
<td>0.24</td>
</tr>
<tr>
<td>Score</td>
<td>21.15%</td>
<td>18.16%</td>
<td>13.79%</td>
<td>13.67%</td>
<td>11.01%</td>
<td>9.92%</td>
<td>8.96%</td>
<td>8.21%</td>
<td>7.85%</td>
<td>7.47%</td>
</tr>
</tbody>
</table>

Table 13: Precision-Recall and score table for ordered merge as a function of no. of documents returned

Figure 44 shows the precision-recall curve for retrieved results when using 154 stop words. No overall change has been observed in the precision and recall values. Also note that no real change has occurred in the relevancy scores due the usage of more number of stop words.
Table 14, Table 15 and Table 16 show the search results when using 524 stop words. It can be seen clearly that there is no notable change in the relevancy scores.

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d_{486}</th>
<th>d_{134}</th>
<th>d_{12}</th>
<th>d_{13}</th>
<th>d_{51}</th>
<th>d_{135}</th>
<th>d_{14}</th>
<th>d_{141}</th>
<th>d_{78}</th>
<th>d_{311}</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Precision (n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>0.78</td>
<td>0.70</td>
</tr>
<tr>
<td>Recall (n/29)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.24</td>
<td>0.24</td>
<td>0.24</td>
<td>0.24</td>
</tr>
<tr>
<td>Score</td>
<td>21.15%</td>
<td>18.16%</td>
<td>16.09%</td>
<td>13.67%</td>
<td>8.96%</td>
<td>8.03%</td>
<td>7.85%</td>
<td>6.74%</td>
<td>5.76%</td>
<td>5.27%</td>
</tr>
</tbody>
</table>

Table 14: Precision-Recall and score table for concatenation merge as a function of no. of documents returned
Table 15: Precision-Recall and score table for round-robin merge as a function of no. of documents returned

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d_{486}</th>
<th>d_{478}</th>
<th>d_{184}</th>
<th>d_{1268}</th>
<th>d_{12}</th>
<th>d_{346}</th>
<th>d_{13}</th>
<th>d_{875}</th>
<th>d_{51}</th>
<th>d_{247}</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>0.50</td>
<td>0.67</td>
<td>0.50</td>
<td>0.60</td>
<td>0.50</td>
<td>0.57</td>
<td>0.63</td>
<td>0.67</td>
<td>0.60</td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.03</td>
<td>0.07</td>
<td>0.07</td>
<td>0.10</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.21</td>
</tr>
<tr>
<td>Score</td>
<td>21.15%</td>
<td>11.01%</td>
<td>18.16%</td>
<td>9.92%</td>
<td>16.09%</td>
<td>9.57%</td>
<td>13.67%</td>
<td>7.47%</td>
<td>8.96%</td>
<td>6.73%</td>
</tr>
</tbody>
</table>

Table 16: Precision-Recall and score table for ordered merge as a function of no. of documents returned

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d_{486}</th>
<th>d_{184}</th>
<th>d_{12}</th>
<th>d_{13}</th>
<th>d_{878}</th>
<th>d_{1268}</th>
<th>d_{346}</th>
<th>d_{13}</th>
<th>d_{875}</th>
<th>d_{51}</th>
<th>d_{195}</th>
<th>d_{14}</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>0.67</td>
<td>0.57</td>
<td>0.63</td>
<td>0.67</td>
<td>0.70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.10</td>
<td>0.14</td>
<td>0.14</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.21</td>
<td>0.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Score</td>
<td>21.15%</td>
<td>18.16%</td>
<td>16.09%</td>
<td>13.67%</td>
<td>11.01%</td>
<td>9.92%</td>
<td>8.96%</td>
<td>8.03%</td>
<td>7.85%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 45 shows the precision-recall curve for the data presented in Table 14, Table 15 and Table 16.

Figure 45: Precision-recall curve when using 524 stop words
By considering the patterns shown Figure 43, Figure 44 and Figure 45 it can be concluded that there has been no particular change in overall precision and recall values and also no big change in relevancy scores is observed.

When the results in Table 5, Table 6 and Table 7 were observed it was said that some of the results in top ten are not relevant to the input query and that they appeared so higher in the list because they have a higher relevancy score. It was also said that higher relevancy score has been given to these non-relevant results because the input query is in natural language and that natural language pollution is the cause of the higher relevancy scores for non-relevant documents. The reasoning for this statement is that a query is made up of terms. Each individual word in the query is treated as a term and index is searched against every term so many non-relevant documents may seem to be relevant to the system because certain query terms appear in a particular document more frequently than others. These frequently occurring terms may not represent the concept that the original query is trying to realize but because appear more frequently than other important terms and therefore cause a higher score computed for a non-relevant document.

More number of stop words have also been used to clean the pollution in the query caused by the natural language. Using more stop words did work and some of the higher ranked non-relevant documents went down the list but on the other side there has been no overall effect on precision and recall values. Since precision and recall values represent the effectiveness of the system in terms of how many returned results are actually relevant and therefore it is really hard to tell that either less stop words are producing better results or higher number of stop words are producing better results because overall precision and recall values do not change. However, since GridIR system performs ranked retrieval and as was earlier mentioned that relevancy score is a good criterion to judge the effectiveness of the system, particularly in this case where search results are sorted on the basis of relevancy score thereby looking at the relevancy scores in tables it is concluded that usage of less or more stop words has no dramatic effect on the scores except that when 154 and 524 stop words are used, relevancy scores for the returned results slightly dropped.

There is another interest aspect of this discussion and that is: Why to care about the relevancy scores at all when the GridIR system is returning 7 correct documents (in most cases) in the top ten and that should be an enough proof of effectiveness of the GridIR system. Answer to this question is that: The user who is performing the search is the only judge of the output and hence he alone can tell that a document declared relevant by the system meets his information need or not. If relevancy scores are considered as a criterion for measuring the effectiveness of results only then it can be predicted that the chances of acceptability of the results by the user are higher because if a document gets a score of, for example, 80% then the document can be assumed as a possible match to the user’s information need.

Another rationale for using different number of stop word lists is that number of terms in the query effect the searching process. If there are more terms in the query then the query becomes too long and if a comprehensive stop words list is not used then chances of getting a higher relevancy score will be low since it may not be possible for a
document to actually contain all the terms specified in the query. However we see that using higher number of stop words did help more relevant documents, to come at higher in the order but did not help to increase the relevancy scores. Reason for this behaviour is that although the potential spam words have been removed from the query but still the query is too long. Solution to this problem is using less but effective and high frequency words in the query. This solution actually produces better relevancy scores than previously observed and is discussed next.

5.2.2 Free-text queries

To increase the relevance free-text query can be used instead of natural language query. Following query has been derived from the query presented earlier.

*Similarity laws in aero elastic models of heated high speed aircraft*

**a- Using 31 word stop-word list**

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>$d_{184}$</th>
<th>$d_{486}$</th>
<th>$d_{12}$</th>
<th>$d_{51}$</th>
<th>$d_{14}$</th>
<th>$d_{195}$</th>
<th>$d_{141}$</th>
<th>$d_{78}$</th>
<th>$d_{435}$</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.87</td>
<td>0.76</td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.24</td>
<td>0.24</td>
<td>0.24</td>
</tr>
<tr>
<td>Score</td>
<td>49.82%</td>
<td>46.43%</td>
<td>37.84%</td>
<td>37.51%</td>
<td>24.57%</td>
<td>18.84%</td>
<td>18.36%</td>
<td>15.85%</td>
<td>15.8%</td>
</tr>
</tbody>
</table>

**Table 17: Precision-Recall and score table for concatenation-merge**

Table 17 shows the precision and recall values as the function of number of documents retrieved for concatenation-merge.

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>$d_{184}$</th>
<th>$d_{486}$</th>
<th>$d_{12}$</th>
<th>$d_{51}$</th>
<th>$d_{14}$</th>
<th>$d_{1268}$</th>
<th>$d_{13}$</th>
<th>$d_{146}$</th>
<th>$d_{745}$</th>
<th>$d_{51}$</th>
<th>$d_{747}$</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>0.50</td>
<td>0.67</td>
<td>0.50</td>
<td>0.60</td>
<td>0.50</td>
<td>0.57</td>
<td>0.63</td>
<td>0.67</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.03</td>
<td>0.07</td>
<td>0.07</td>
<td>0.10</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>Score</td>
<td>49.82%</td>
<td>29.14%</td>
<td>46.43%</td>
<td>26.28%</td>
<td>37.84%</td>
<td>21.73%</td>
<td>37.51%</td>
<td>19.77%</td>
<td>24.57%</td>
<td>15.29%</td>
<td></td>
</tr>
</tbody>
</table>

**Table 18: Precision-Recall and score table for round-robin merge**

Table 18 shows the precision and recall values as the function of number of documents retrieved for round-robin merge.
Table 19: Precision-Recall and score table for ordered merge

Table 19 shows the precision and recall values as the function of number of documents retrieved for ordered merge.

<table>
<thead>
<tr>
<th>in returned (n)</th>
<th>1.00</th>
<th>1.00</th>
<th>1.00</th>
<th>0.80</th>
<th>0.71</th>
<th>0.63</th>
<th>0.67</th>
<th>0.70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>0.71</td>
<td>0.63</td>
<td>0.67</td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.10</td>
<td>0.14</td>
<td>0.14</td>
<td>0.17</td>
<td>0.17</td>
<td>0.21</td>
</tr>
<tr>
<td>Score</td>
<td>49.82%</td>
<td>46.43%</td>
<td>37.84%</td>
<td>37.51%</td>
<td>29.14%</td>
<td>26.28%</td>
<td>24.57%</td>
<td>21.73%</td>
</tr>
</tbody>
</table>

Figure 46: Precision-recall curve when using free-text query with default (31) stop words

Graphs shown in Figure 45 and Figure 46 are almost identical to each other, with the exception of ordered merge, which means there is no overall change in precision and recall values while values in Table 17, Table 18 and Table 19 show an increase in the relevancy score. This relevancy score is more than double in magnitude in comparison to the relevancy scores in Table 14, Table 15 and Table 16. Change in the search results of ordered-merge is also due to this increased relevancy score.

From these results it can be concluded that free-text queries are bringing more relevant documents. While in real the document is relevant to the user’s information need or not is dependant on user’s perception of the document but at least the relevancy scores show increased chances of satisfying the user’s information need.

The effect of using free-text queries is immediately visible and relevancy scores for the top result are near to 50%. Remember that default stop word list has been used that contains 31 words. Next 154 stop words have been used to experiment with the GridIR system.
b- Using 154 word stop-word list

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d_{486}</th>
<th>d_{184}</th>
<th>d_{12}</th>
<th>d_{13}</th>
<th>d_{51}</th>
<th>d_{14}</th>
<th>d_{141}</th>
<th>d_{195}</th>
<th>d_{78}</th>
<th>d_{252}</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.86</td>
<td>0.88</td>
<td>0.78</td>
<td>0.70</td>
<td></td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.21</td>
<td>0.24</td>
<td>0.24</td>
<td></td>
</tr>
<tr>
<td>Score</td>
<td>58.04%</td>
<td>49.82%</td>
<td>37.84%</td>
<td>37.51%</td>
<td>24.57%</td>
<td>21.54%</td>
<td>18.49%</td>
<td>18.36%</td>
<td>15.8%</td>
<td>13.63%</td>
</tr>
</tbody>
</table>

Table 20: Precision-Recall and score table for concatenation-merge

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d_{486}</th>
<th>d_{878}</th>
<th>d_{184}</th>
<th>d_{12}</th>
<th>d_{13}</th>
<th>d_{14}</th>
<th>d_{51}</th>
<th>d_{875}</th>
<th>d_{51}</th>
<th>d_{747}</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>0.50</td>
<td>0.67</td>
<td>0.50</td>
<td>0.60</td>
<td>0.50</td>
<td>0.57</td>
<td>0.63</td>
<td>0.67</td>
<td>0.60</td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.03</td>
<td>0.07</td>
<td>0.07</td>
<td>0.10</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.21</td>
</tr>
<tr>
<td>Score</td>
<td>58.04%</td>
<td>29.14%</td>
<td>49.82%</td>
<td>26.28%</td>
<td>37.84%</td>
<td>21.73%</td>
<td>37.51%</td>
<td>19.77%</td>
<td>24.57%</td>
<td>15.29%</td>
</tr>
</tbody>
</table>

Table 21: Precision-Recall and score table for round-robin merge

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d_{486}</th>
<th>d_{184}</th>
<th>d_{12}</th>
<th>d_{13}</th>
<th>d_{878}</th>
<th>d_{12}</th>
<th>d_{13}</th>
<th>d_{875}</th>
<th>d_{14}</th>
<th>d_{875}</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>0.67</td>
<td>0.71</td>
<td>0.63</td>
<td>0.67</td>
<td>0.70</td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.10</td>
<td>0.14</td>
<td>0.14</td>
<td>0.17</td>
<td>0.17</td>
<td>0.21</td>
<td>0.24</td>
<td></td>
</tr>
<tr>
<td>Score</td>
<td>58.04%</td>
<td>49.82%</td>
<td>37.84%</td>
<td>37.51%</td>
<td>29.14%</td>
<td>26.28%</td>
<td>24.57%</td>
<td>21.73%</td>
<td>21.54%</td>
<td>19.77%</td>
</tr>
</tbody>
</table>

Table 22: Precision-Recall and score table for ordered-merge

Table 20, Table 21 and Table 22 show the precision, recall and relevancy score values against the search results.
It can be clearly seen that, relevancy score values have got better when using free-text queries and increased number of stop words.

**c- Using 524 word stop-word list**

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d486</th>
<th>d184</th>
<th>d12</th>
<th>d13</th>
<th>d51</th>
<th>d195</th>
<th>d14</th>
<th>d141</th>
<th>d78</th>
<th>d311</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.88</td>
<td>0.78</td>
<td>0.70</td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.24</td>
<td>0.24</td>
<td>0.24</td>
<td>0.24</td>
</tr>
<tr>
<td>Score</td>
<td>58.04%</td>
<td>49.82%</td>
<td>44.14%</td>
<td>37.51%</td>
<td>24.57%</td>
<td>22.03%</td>
<td>21.54%</td>
<td>18.36%</td>
<td>15.8%</td>
<td>14.45%</td>
</tr>
</tbody>
</table>

**Table 23: Precision-Recall and score table for concatenation-merge**

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d486</th>
<th>d478</th>
<th>d184</th>
<th>d12</th>
<th>d1268</th>
<th>d12</th>
<th>d156</th>
<th>d13</th>
<th>d475</th>
<th>d51</th>
<th>d477</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>0.50</td>
<td>0.67</td>
<td>0.50</td>
<td>0.60</td>
<td>0.50</td>
<td>0.57</td>
<td>0.63</td>
<td>0.67</td>
<td>0.60</td>
<td></td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.03</td>
<td>0.07</td>
<td>0.07</td>
<td>0.10</td>
<td>0.10</td>
<td>0.14</td>
<td>0.17</td>
<td>0.21</td>
<td>0.21</td>
<td></td>
</tr>
</tbody>
</table>
Table 24: Precision-Recall and score table for round-robin merge

<table>
<thead>
<tr>
<th>Doc no.</th>
<th>d486</th>
<th>d184</th>
<th>d12</th>
<th>d13</th>
<th>d878</th>
<th>d1268</th>
<th>d746</th>
<th>d31</th>
<th>d195</th>
<th>d14</th>
</tr>
</thead>
<tbody>
<tr>
<td># of returned (N)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td># of relevant in returned (n)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Precision(n/N)</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.80</td>
<td>0.67</td>
<td>0.71</td>
<td>0.63</td>
<td>0.67</td>
<td>0.70</td>
</tr>
<tr>
<td>Recall(n/29)</td>
<td>0.03</td>
<td>0.07</td>
<td>0.10</td>
<td>0.14</td>
<td>0.14</td>
<td>0.14</td>
<td>0.17</td>
<td>0.17</td>
<td>0.21</td>
<td>0.24</td>
</tr>
<tr>
<td>Score</td>
<td>58.04%</td>
<td>49.82%</td>
<td>44.14%</td>
<td>37.51%</td>
<td>29.14%</td>
<td>26.28%</td>
<td>25.35%</td>
<td>24.57%</td>
<td>22.03%</td>
<td>21.54%</td>
</tr>
</tbody>
</table>

Table 25: Precision-Recall and score table for ordered-merge

Table 23, Table 24 and Table 25 show the precision, recall and relevancy score values when using 524 stop words.

Figure 48 shows the precision-recall curve when using 524 stop words. Graph looks very similar to graphs shown earlier.

![Precision-recall curve when using free-text query with 524 stop words](image)

**Figure 48: Precision-recall curve when using free-text query with 524 stop words**

By observing precision-recall curves in the figures above we conclude that there is no substantial overall difference in precision and recall values whether we use natural language queries or free-text queries, less stop words or more stop words.
While there is no change in precision and recall values there is a substantial difference in the relevancy scores that have been shown previously. Usage of more stop words has indeed proved effective and scores have just got better and consistent.
Chapter 6

Conclusions and Future directions

6.1 Conclusions

GridIR provides new directions to the existing IR world. Established IR methods and approaches are useful in developing and evaluating the effectiveness of a GridIR system but GridIR poses some new and unique challenges that are specific to the distributed nature of grid, such as creation of distributed test collections, execution of queries on distributed indexes and merging of rank retrieved results.

This project provides a platform for future developments in the field of distributed IR in general and GridIR particularly. This project is a demonstration of how open source components can be used to build an efficient and effective system. Although this system does not use many of techniques that are pretty much standard with modern search engines, like application of stemming and lemmatization techniques at the time of index generation, query and index term boosting, query expansion via thesaurus and relevance feedback mechanism, but provides the opportunity of system extensibility in this direction.

GridIR system uses ranked retrieval approach for retrieving the search results. Three different strategies have been used to merge the search results collected from remote document collections. These strategies are Concatenation merge, Round-robin merge and Ordered merge. Concatenation merge depends upon the ordering of document collections. Round-robin merge applies a fair policy for merging the search results. Ordered merge performs merging and sorting of search results based on the relevancy scores computed during the search operation.

Evaluation tests performed on Cranfield field test collection show that standard IR system evaluation methods, precision and recall, are not very useful when applied to the ranked retrieval model of information retrieval because the precision and recall values are dependant on the relevancy score computed against and assigned to each retrieved result. Therefore relevancy scoring scheme alone can also be used effectively to determine the effectiveness of the system.
Many different techniques and approaches have been used to improve the relevancy scores of the retrieved results but there are still many aspects that are open to future development. Outline of these potential developments is presented in the next section.

6.2 Future directions

There is a great potential of adding new features to the GridIR system. GridIR community profoundly needs a standard distributed test collection because without the availability of a distributed test bed it is very difficult to evaluate the effectiveness of the system. New and better approaches for merging search results need to be researched. Usage of stemming and lemmatization techniques is highly recommended as it will increase the relevancy scores. Query expansion via the usage of a thesaurus can bring better results as well. Intelligent approaches towards filtering the search results for possible spam are required. Moreover, creation of policies for index generation, access and ownership (specific to the grid environment) is also required. Another interesting development can be the extension of GridIR system to enable the retrieval of other types of unstructured information like images and videos.
Appendix A

Lucene resource file format

Following is an example of Lucene resource file.

```plaintext
id=LocalResource
type=uk.org.ogsadai.DATARESOURCE
creationTime=null
terminationTime=null
PROPERTIES
  uk.org.ogsadai.resource.dataresource.product=OGSA-DAI
  uk.org.ogsadai.resource.dataresource.vendor=EPCC/OGSA-DAI
  uk.org.ogsadai.resource.dataresource.version=3.0
END
CONFIG
dai.data.search.fields=contents
dai.index.uri=D:\javaprog\ogsadai\index8080\index
dai.index.metadata=title,summary,filename,score
END
ACTIVITIES
END
dataResourceClass=uk.org.ogsadai.tutorials.activity.LuceneDataResource
```
Appendix B

Stop words

B.1 List 1: Default stop words

a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, such, that, the, their, then, there, these, they, this, to, was, will, with

B.2 List 2: 154 stop words

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 000, $, about, after, all, also, an, and, another, any, are, as, at, be, because, been, before, being, between, both, but, by, came, can, come, could, did, do, does, each, else, for, from, get, got, has, had, he, have, her, here, him, himself, his, how, if, in, into, is, it, its, just, like, make, many, me, might, more, most, much, must, my, never, now, of, on, only, or, other, our, out, over, re, said, same, see, should, since, so, some, still, such, take, than, that, the, their, them, then, there, these, they, this, those, through, to, too, under, up, use, very, want, was, way, we, well, were, what, when, where, which, while, who, will, with, would, you, your, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

B.3 List 3: 524 stop words

a, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, around, as, aside, ask, asking, associated, at, available, away, awfully, b, be, became, because, become, becomes, becoming, been, before, beforehand, behind, being, believe, below, beside, besides, best, better, between, beyond, both, brief, but, by, c, came, can, cannot, cant, cause, causes, certain, certainly, changes, clearly, co, com, come, comes, concerning, consequently, consider, considering, contain, containing, contains, corresponding, could, course, currently, d, definitely, described, despite, did, different, do, does, doing, done, down, downwards, during, e, each, edu, eg, eight, either, else, elsewhere, enough, entirely, especially, et, etc, even, ever, every, everybody, everyone, everything, everywhere, ex, exactly, example, except, f, far, few, fifth, first, five, followed, following, follows, for, former, formerly, forth, four, from, further, furthermore, g, get, gets, getting, given, gives, go, goes, going, gone, got, gotten, greetings, h, had, happens, hardly, has, have, having, he, hello, help, hence, her, here, hereafter, hereby, herein, hereupon, hers, herself, hi, him, himself, his, hither, hopefully, how, howbeit, however, i, ie, if, ignored, immediate, in, inasmuch, inc, indeed, indicate, indicated, indicates,
inner, insofar, instead, into, inward, is, it, its, itself, j, just, k, keep, keeps, kept, know, knows, known, l, last, lately, later, latter, latterly, least, less, lest, let, like, liked, likely, little, look, looking, looks, ltd, m, mainly, many, may, maybe, me, mean, meanwhile, merely, might, more, moreover, most, mostly, much, must, my, myself, n, name, namely, nd, near, nearly, necessary, need, needs, neither, never, nevertheless, new, next, nine, no, nobody, non, none, noone, nor, normally, not, nothing, novel, now, nowhere, o, obviously, of, off, often, oh, ok, okay, old, on, once, one, ones, only, onto, or, other, others, otherwise, ought, our, ours, ourselves, out, outside, over, overall, own, p, particular, particularly, per, perhaps, placed, please, plus, possible, presumably, probably, provides, q, que, quite, qv, r, rather, rd, re, really, reasonably, regarding, regardless, regards, relatively, respectively, right, s, said, same, saw, say, saying, says, second, secondly, see, seeing, seem, seemed, seeming, seems, seen, self, selves, sensible, sent, serious, seriously, seven, several, shall, she, should, since, six, so, some, somebody, somehow, something, sometime, somewhat, somewhere, soon, sorry, specified, specify, specifying, still, sub, such, sup, sure, t, take, taken, tell, tends, th, than, thank, thanks, thanx, that, thats, the, their, theirs, them, themselves, then, thence, there, thereafter, thereby, therefore, therein, theres, thereupon, these, they, think, third, this, thorough, thoroughly, those, though, three, through, throughout, thru, thus, to, together, too, took, toward, towards, tried, tries, truly, try, trying, twice, two, u, un, under, unfortunately, unless, unlikely, until, unto, up, upon, us, use, used, useful, uses, using, usually, uucp, v, value, various, very, via, viz, vs, w, want, wants, was, way, we, welcome, well, went, were, what, whatever, when, whence, whenever, where, whereafter, whereas, whereby, wherein, whereupon, wherever, whether, which, while, whither, who, whoever, whole, whom, whose, why, will, willing, wish, with, within, without, wonder, would, would, x, y, yes, yet, you, your, yours, yourself, yourselves, z, zero
References


