Deploying Lucene on the Grid

Edgar Meij and Maarten de Rijke
ISLA, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam
The Netherlands
emeij,mdr@science.uva.nl

ABSTRACT

We investigate if and how open source retrieval engines can be deployed in a grid environment. When comparing grids to conventional distributed IR, the lack of a-priori knowledge about available nodes is one of the most significant differences. On top of that, it is also unknown when a particular node has time and resources available and starts a submitted job. Therefore, conventional methods such as RMI are not directly usable and we propose a different approach, using middleware designed specifically for grids. We describe GridLucene, an extension of the open source engine Lucene with grid-specific classes, based on this middleware. We report on an initial comparison between GridLucene and Lucene, and find a minor penalty (in terms of execution time) for grid-based indexing and a more serious penalty for grid-based retrieval.

The used middleware can gather a set of physical resources to form a single logical resource with some abstract properties. The user-definable properties can be used during indexing and retrieval to let GridLucene know which files it needs to access. By using this kind of semantic information, grid nodes can "discover" which indices exist on the grid and which particular documents need to be indexed.

GridLucene is available for downloading under the same license as Lucene.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Measurement, Performance, Experimentation

Keywords

Grid, grid computing, metadata, parallel indexing

OSIR 2006 August 6-11, 2006, Seattle, USA

1. INTRODUCTION

Grids are an emerging infrastructural technology for resource sharing. They enable the collaborative use of computational resources (commonly referred to as *nodes*), towards one or more common goals. Distributed or parallel computing is not a novelty, but there are significant differences when compared to grids. Within *clustered* computing, the individual components are tightly coupled, usually located near to each other and very homogeneous. Within a *distributed* approach, the homogenity is much less guaranteed, individual elements are loosely coupled and they can be geographically apart. Grids are like distributed systems, but they can cross national, as well as organizational boundaries. Grids also have interesting new properties that on the one hand ask for new solutions but can, on the other hand, also have interesting new potential.

The use of grids is still gaining in popularity and the enabling technologies are maturing as well. Increasingly, grids are being deployed in multi-site research projects and/or in projects where large amounts of data need to be stored or processed [31].

In this paper, we address the exploratory question if and how (open source) search engines can be deployed in a grid environment. Within our research group we make heavy use of Lucene [29], the well-known open source search engine, for various retrieval experiments and in various ways.[1] The performance of Lucene decreases with the ever-increasing size of document collections. How would it perform in a grid environment? What, if anything, is the difference between deploying an engine such as Lucene in a more conventional setup and deploying it in a grid environment? Are there any special challenges, benefits, or disadvantages? A significant difference between a grid-based approach and conventional distributed information retrieval for example, is the lack of a-priori knowledge about available nodes in a grid. This makes load balancing at runtime difficult, something which is still an ongoing research topic [1, 35]. Indeed, on the grid, it is usually not known in advance when a particular node

[1]We have used Lucene in a local fashion for example, with either a single index or with multiple indices for our MoodViews project [27] and have also done experiments with a distributed setup. We have even augmented Lucene with an in-house developed Language Modeling extension [26] to replace the default similarity calculations. Faced with retrieval tasks that require substantial amounts of storage space, such as the TREC Terabyte retrieval track [10, 9], we turn to grids.

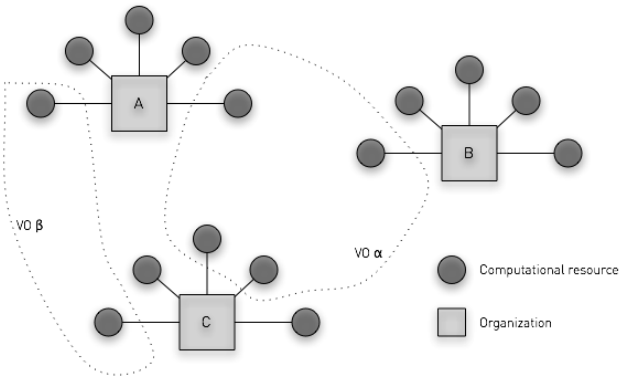

Figure 1: Organizations, sharing resources through Virtual Organizations.

starts a certain job, how many resources it has available, and where it is located—because of this, methods such as RMI (Remote Method Invocation) are not directly usable. To be able to take advantage of the distributed nature of grids, we propose a different approach, using grid-specific storage middleware. To evaluate this approach, we create a number of grid-specific implementations around Lucene.

The rest of this paper is organized as follows. Before we describe our approach in more detail, we first provide some background on grids (Section 2) and discuss several key components, common to most (institutional) grids, in Section 3. In Section 4 we describe the changes we made to Lucene and some additional implementations. In Section 5 we describe the setup used to evaluate indexing and searching performance of our grid-based solution. In particular, we compare indexing and retrieval speeds of our grid-enabled Lucene with a "traditional," locally running Lucene installation. The results can be found in Section 6, and concluding remarks are in Section 7.

2. BACKGROUND

A grid enables the integrated use of resources, which are typically owned by multiple organizations and/or individuals and is in fact a system consisting of distributed, but connected resources [12]. It also encompasses software and/or hardware that provides and manages logically seamless access to those resources [13, 24]. Grids can be roughly classified in two categories: institutional grids (IG's) and global computing or P2P (GCP) systems [3, 23, 11].

GCP systems typically harvest the computing power provided by individual computers, using otherwise unused bandwidth and computing cycles in order to run very large and distributed applications [22, 15]. Some examples include SETI@home [38], LookSmart's Grub [28] (a voluntary initiave to crawl the Internet in a distributed fashion), and ZetaGrid [40]. ZetaGrid is an attempt to verify Riemann's Hypothesis using grid technology, with a reported peak performance rate of around 7000 GFLOPS. There are also (open source) packages such as XtremWeb [11], and Q^2ADPZ [30] which allow to setup, deploy and run GCP projects. BOINC (the Berkeley Open Infrastructure for Network Computing) is another open source platform for public-resource distributed computing [3] and currently the enabling system for SETI@home, LHC@home, Einstein@home, Climateprediction.net, and many more.

Our work however, takes place in an IG environment. IGs cross organizational boundaries and are based on the virtual organization (VO) paradigm [20, 17, 21]. A VO is a dynamic collection of resources and users unified by a common goal and potentially spanning multiple administrative or organizational domains [18]. A figure depicting the notion of VOs is given in Figure 1 (adapted from a figure in [32]). IG's involve intensive computations and very large amounts of data, that also require secure resource sharing for which the current Internet infrastructure is unadequate. While GCP systems can be categorized as using weak to no authentication, IGs use strong authentication systems, based on the VO paradigm [19, 39]. They target broad scientific collaborations where various individuals, groups and institutions perform diverse calculations. There is considerable variation in the range of scientific grid applications, depending on the interest and scale of the community in question. A practical example of the use of an IG is the numerical solution of the long-open "nug30" quadratic optimization problem [33]. As opposed to mostly single machines within GCP systems, an IG resource might also be a cluster of machines, a storage system, database, or any other scientific instrument of considerable value that is administered in an organized fashion.

3. GRID COMPONENTS

In this section we describe the institutional grid environment in which we work and in which our experiments take place. We start out by describing the general environment and then zoom in on a particular kind of storage middleware and job submission details.

3.1 General

The construction of an institutional grid requires the establishment of standards for infrastructure, communication protocols and application programming interfaces [20, 37]. The Open Grid Services Architecture (OGSA) defines such a standard service-oriented IG framework [21]. The goal of OGSA is to standardize practically all the services one finds in an IG, such as VO/user management, security, resource management, job management, and data services. The implementation of these services provides a virtualization of a grid's resources and form an IG's middleware. Today, the Globus Toolkit [16, 25] is the de facto standard for any IG's middleware, with the Globus Toolkit 4 (GT4) being the most recent version [14]. GT4 is an open source toolkit and a realization of the OGSA requirements. It consists of services, programming libraries, and development tools which can be used to create IG-based applications [32].

Since there is no persistent storage on any grid node, specific middleware has been developed to accomodate storage requirements across an IG. Within GT4 there are several data management components such as GridFTP [20, 2] and OGSA-DAI (a service-based architecture for database access over the grid) [4]. Another implementation of a storage middleware solution will be presented in the next section.

3.2 Storage Resource Broker

The Storage Resource Broker (SRB) is a grid storage implementation, which includes data abstraction as well as metadata handling [5]. It provides a uniform API to heterogenous storage resources in a distributed fashion and has been used succesfully in various applications and fields, such as biomedicine, astronomy, digital libraries, and more [31].

SRB provides an abstraction layer over the actual storage devices, which can include various type of filesystems, as well as databases and archival storage systems. Because of this transparancy, SRB can accommodate virtually limitless amounts of data and various types of data scattered in multiple geographic locations can be accessed in a single representation from any grid node. Files and directories within SRB are organized in so called *collections*. With this abstraction, data items which are stored in a single collection can in fact be stored on heterogenous and geographically distant storage systems, but be referenced to by a single URI [5]. Individual files or directories can be accessed in the same manner. The available interfaces include a UNIX-like file I/O interface, a Java API called JARGON [8], and an interface that supports *get* and *put* operations of individual files, directories, or entire collections.

SRB also provides a metadata catalog service (MCAT), by which collections can be annotated, queried and retrieved, providing an abstract definition for resources. These annotations take the form of user-definable *attribute-value* pairs. They can be arbitrarily modified and queried, for example through JARGON [8]. This makes it possible to query SRB as one would a database management system, but instead of records you receive URI's of files or collections. Using these features, one can gather a set of distinct physical resources to form a single logical resource with some abstract properties.

3.3 Job Submission System

Unfortunately, GT4 was not yet functioning properly on the grid we are using for these experiments. We therefore used the tools made available through the European Data Grid project [6] to perform the current grid experiments with Lucene. These tools include a number of job planning, submission, and tracking tools (`edg-job-list-match`, `edg-job-submit`, and `edg-job-get-status` respectively), written in Python. Job descriptions are written in specialized configuration files, marked up using the Job Description Language.

4. A GRID-ENABLED LUCENE

To make Lucene run in our grid environment, we implemented a number of additions on top of the standard Lucene implementation. The main ones concern Lucene's interaction with SRB and the usage of MCAT, the metadata catalog service. The resulting implementation is called GridLucene; below, we briefly describe the additions that distinguish GridLucene from Lucene.

4.1 Storage

As described in subsection 3.2, each node on the grid has uniform access to files through storage middleware, such as SRB. Although it is possible, to some extent, to define on which particular machines or nodes our jobs will run, we decided not to make use of this functionality since it is not possible to define *when* our jobs will run. What we do know is that it is possible to describe which data from the storage middleware must be accessed and that this storage is available to all nodes through the network.

So, in order for Lucene to communicate with SRB, we have subclassed Lucene's (abstract) `Directory` class. This makes communications from Lucene with SRB transparent, because we can regard an SRB collection as any other local or RAM directory. Because of the implicit networking overhead, we wanted to evaluate the indexing and searching performance and compare it with a local, disk-based Lucene installation. The results of this evaluation can be found in Section 6.

4.2 Metadata

Using MCAT, the metadata catalog service within SRB introduced in Section 3.2, one can select certain files or collections from SRB, based on definable criteria. These criteria can be either more "regular" properties, such as filenames, sizes, and creation dates, but also manual annotations. For example, one can label a SRB collection and all the files therein `{collection:INEX, year:2005, indexed:no}`. A different set of files might have the same annotations, but a different value for one key: `year:2006`. When queried for `collection:INEX`, the SRB server will return the URI's of all the files in both collections. It would also be possible to just return the URI's of the collections themselves. Single files, directories, or collections can thus be located in a more semantic way than is normally possible using regular filesystems.

To make use of this new possibility, we have written Java classes which can partition a given document collection (consisting of multiple files) into subparts, put these on SRB, and annotate them. In a typical setting, one would split the collection into subsets, based on filesizes or the number of files, depending on the collection or task at hand.

This approach makes it possible for nodes to "discover" on their own which documents remain to be indexed. This means that, during indexing, each node that starts its job can select one of the remaining subparts of the document collection on the fly. It then indexes the documents in this subcollection and labels it as *done*. The indexing itself is preceded by transferring the actual documents from SRB and could be optimized by using either the local disk or RAM as a buffer. The index is written directly onto SRB or transferred afterwards from the buffer, so that it will be available in a later stage for retrieval. There are various possible approaches to transferring files from and to SRB, on which we elaborate in section 5.2.

The entire indexing process is shown graphically in Figure 2. Once the index has been stored on SRB, it is again possible to annotate it with specific metadata. It can, for example, be labeled with a description of its contents, the time taken to create the index, or the used `Analyzer`. Since the metadata is not restricted to pre-defined fields, the only limit is one's imagination or needs.

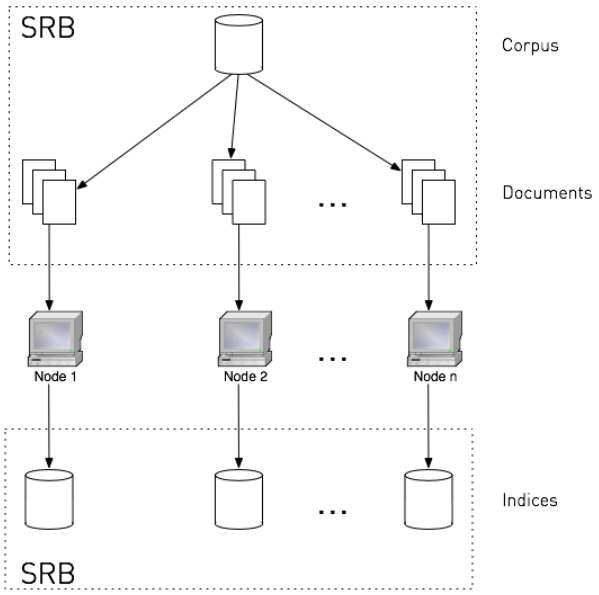

Figure 2: Indexing on a grid, using GridLucene. The document collection is split up into subsets. Each indexing node "discovers" which part(s) remain to be indexed based on SRB metadata. It selects a part, downloads and indexes the documents it contains into a separate index and transfers the index back to SRB. The index can then again be labeled with user definable metadata.

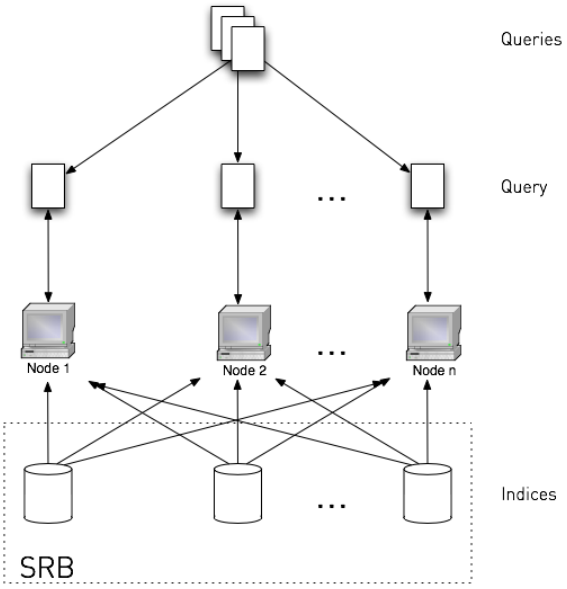

Figure 3: Searching on a grid. The separately created indices can be "discovered" by a searcher node, using SRB metadata. Each retrieval node then searches the appropriate index or indices.

At retrieval time, the searcher node(s) can query SRB to discover which directories have certain annotations and thus contain relevant indices for a certain query. They can then either load them into a Lucene `MultiSearcher` (if there is more than one relevant index), or a single `IndexSearcher` (If only a single index has been created, or if multiple indices have been merged into one). Both approaches can again make use of our SRB `Directory` class. In experimental information retrieval settings where very large batches of queries are more common, such as the TREC Terabyte track, it may also be beneficial to divide the queries into subsets and distribute these among the searcher nodes as depicted in Figure 3.

5. EXPERIMENTAL ENVIRONMENT

Now that we have described GridLucene, we turn to an evaluation of the efficiency of the additions implemented in GridLucene, and, more generally, of GridLucene.

We investigated the performance of the SRB-related classes, in order to assess response times and the scalability of the approach. Our goal is to see how Lucene performs in a grid setting, with varying document numbers to index and search, using the classes described in the previous section. Below, we present preliminary outcomes of "evaluations in progress." That is, at this stage our experiments do not yet provide an exhaustive benchmark analysis, but they do provide a sanity check in that they indicate how the performance of using SRB in combination with GridLucene relates to more conventional approaches.

5.1 Test Collection

For our experiments, we used the INEX 2005 test collection, which contains about 12,000 XML documents with an average size of 50 KB each [34]. We split it up randomly, based on filenumbers rather than sizes. In each experiment, we calculated the overall time taken using GridLucene and compared this with the results of Lucene running on a single machine (using the same Lucene version and settings, Java version, and document collection, but with everything stored locally on disk). This machine has dual Pentium 4 processors, each running at 3.0 GHz, 2GB of RAM, and Linux with kernel version 2.6.12 as operating system. We used Lucene version 1.9.1 and SRB version 3.4.1. Every indexing and searching run has been executed multiple (40), consecutive times, to compensate for any caching that may be involved; the times reported below are averaged over these 40 runs.

For indexing, we created document subcollections with an increasing numbers of files, starting with 1,000 documents and adding 1,000 at each increment, with an upper limit of 10,000 documents. The presented indexing times for GridLucene include transferring the documents from SRB and putting the generated index back on SRB, as described in Section 4.2. For searching, we used the indices generated during the indexing stage. GridLucene searches the indices stored on SRB, whereas the locally running Lucene uses locally stored indices. As queries we have taken 200 actual INEX topics[2] for this corpus and sequentially searched the index for each of them.

[2] `http://inex.is.informatik.uni-duisburg.de/2005/`

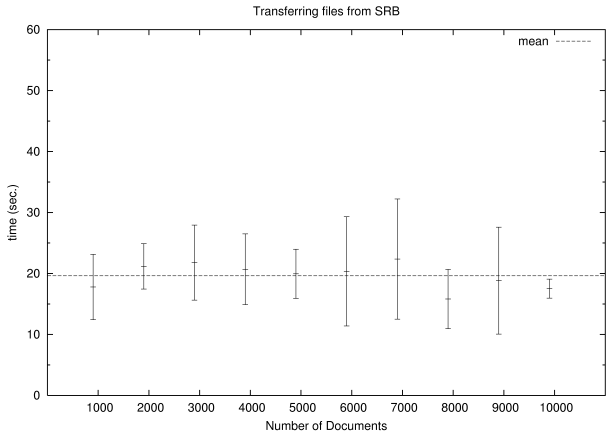

Figure 4: Time it takes to transfer an increasing number of files from SRB to a grid node. Due to the multi-threaded, parallel implementation, there is no noticable slowdown.

5.2 Transferring Files

We experimented with different ways of transferring the documents from and to a grid node. There are basically two distinct ways to transfer files from and to SRB [8]. The first is using methods from a Java `FileStream` like class, written to interact with SRB, which has roughly the same functionality as its Java counterpart. With its methods it is, for example, possible to seek a position in a file and read in an array of bytes. The other method is a higher-level class, which is able to transfer files using multiple threads. In turn, each thread can send multiple files in parallel, thus greatly improving performance.

For indexing we started out by using the `FileStream` approach to read in the source documents. This proved to be error-prone due to connection issues; a slight lapse in network connectvity caused the sequential stream to break unrecoverably. On top of that, this kind of transfer is also relatively slow. We therefore switch to a caching approach, in which we use the higher-level method to first download all the necessary documents to the indexing node. This does require a receiving node to have at least enough diskspace available to hold the files it needs to index. Figure 4 shows the average time it takes to transfer varying numbers of documents from SRB to a grid node, using the higher-level `copyTo` method. Further research is neccessary to determine how this method scales up beyond 10,000 documents and with varying document sizes. It seems that the 1 Gb/s ethernet connection is definitely not a bottleneck. When the index has been created it is transferred back to SRB using the same method. Before doing so, we first merge the resulting index segments into a single compound file.

We keep the original `FileStream` method on the retrieval side, since the searcher classes only need to access certain parts of the indexfile(s), namely those which contain the relevant parts of the inverted index and document mappings. We use the approach as described in section 4.1. For this evaluation we further use a single index, growing in size, and a single retrieval node. However, the implementation used can easily be extended to use multiple SRB indices in a Lucene `MultiSearcher`. We use the default retrieval model within Lucene.

6. EVALUATION

We wanted to know how long it takes GridLucene to index a certain set of documents from SRB to an SRB index and compare this with Lucene installed on a standalone machine. We also wanted to do a similar comparison for retrieval, as described in the previous section. The numbers in the figures presented in this section are the means of each run, with accompanying standard deviations.

6.1 Indexing

Figure 5a compares the total indexing time against an increasing number of documents. This also includes, in the case of GridLucene, the time to transfer documents from and indices to SRB. There is a slight variation in the transfer times of files from and to SRB, as can be seen from Figure 4. This variation also affects the results presented in figure 5a. It was clear from the beginning that GridLucene indexing times would be higher for any number of documents, due to the implicit networking overhead.

Although indexing using GridLucene is slower than a local setup, the extra time needed is limited and does not exceed 30 seconds. It also seems constant over every run, which may indicate that a trade-off point exists between conventional and grid-based indexing, at which the extra time needed for file I/O can be compensated by making subsets of the task and distributing these to multiple nodes. The additional indexing time involved when using GridLucene averages around 20 seconds per indexing task for these experiments, which may be an important predictor for this constant. Indexing 6,000 documents for example, would take around 60 seconds using a locally installed Lucene. Roughly the same amount of time is needed for GridLucene to index 2,000 documents. So, when dividing these 6,000 documents into three subsets and dispatching each to a grid node, the overall time taken is the same for both approaches. We have collected too little data however, to accurately determine if and where exactly the trade-off point lies and whether this would still hold for more and/or larger documents.

The fact that grid nodes running GridLucene can discover the annotated subcollections on their own is novel and can be seen as a step forward towards the acceptance of using grids within information retrieval.

6.2 Searching

Figure 5b shows a less optimistic picture. The searching times for GridLucene seem to increase more with a growing index size, than with a locally installed Lucene: file access method we currently use obviously fails for the task. We did not use the `copyTo` method from Section 5.2 for retrieval, thinking that it would not be necessary to copy the entire index to a searcher node. Instead, we held on to the FileStream approach, which clearly has its adverse effects on response times. Additionally, we did not perform any kind of local caching, so each time the index gets queried for a term, the request travels through the network. We had no means of quantitatively measuring the network overhead

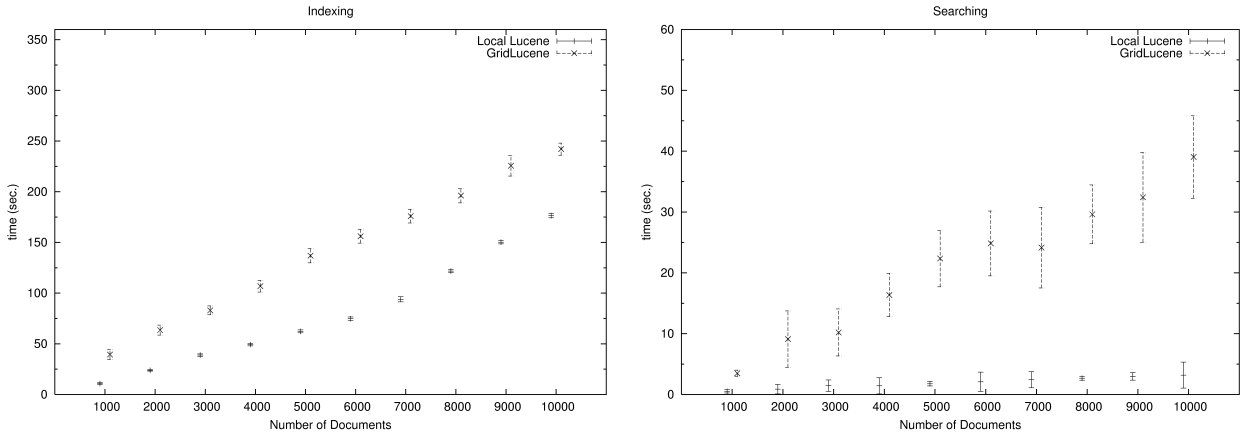

Figure 5: Indexing times of GridLucene (using SRB) compared with a local installation of Lucene and the time taken for 200 queries to be retrieved, with varying document numbers.

using the tools available on our grid, so we can only guess that caching the most-accessed parts of an index locally on a retrieval node will be beneficial.

Had we used the same method to transfer the index files as we did during indexing, the picture would probably have been similar to Figure 6a, with the response times of GridLucene being consistently higher (adding around 20 seconds) than the local Lucene. Whether this is in fact the case and to which extent this statement holds, remains a topic for further research.

7. CONCLUSIONS

We investigated if and how open source retrieval engines can be deployed in a grid environment. When comparing grids to conventional distributed information retrieval, the lack of a-priori knowledge about available nodes is one of the most significant differences. On top of that, it is also unknown when a particular node has time and resources available and starts a submitted job. Conventional methods such as RMI are therefore not directly usable and we propose a different approach, using middleware designed specifically for grids. As an example we have taken Lucene and implemented some grid-specific classes, based on this middleware. The resulting implementation, GridLucene, is available for downloading.

Some properties of the used grid storage middleware open up new and interesting possibilities. With relatively minor additions to Lucene we were able to transparently incorporate metadata about document (sub)collections and indices. Using user definable annotations, GridLucene can automatically find documents remaining to be indexed. The current approach thus makes it worthwhile to semi-automatically "parallelize" the indexing of large document collections.

Indexing new, additional documents in a later stage is also straightforward using a similar approach. They can be stored in a new index, or added to an existing one based on provided annotations. SRB also provides a theoretically infinite storage capacity, because additional storage space can be added without having to deal with changing file names and locations.

On the retrieval side, GridLucene can use the annotations to discover which indices exist on the grid and, most importantly, might be relevant. Especially in a grid setting, where sharing and collaboration are some of the main driving forces, these new possibilities give way to a whole new range of applications. In a sense, one can simply "plug in" additional indices during a search or discover what others might already have indexed. However, the way in which results from different information providers should be combined is still an ongoing issue within the distributed IR research field.

With GridLucene, a framework has been created in which research from the distributed information retrieval community can be tested in a grid setting. For example, the resource *description*, *selection* and *merging* algorithms for distributed/federated text databases [7] can be more or less directly applied in a multi-organizational grid environment.

We carried out some initial benchmarkings on the proposed implementations and found that GridLucene incurs a minor and consistent penalty when indexing on SRB, while the online performance during document retrieval incurs a far more dramatic penalty. This is most likely due to the chosen file-access methods—while appropriate in a grid-based environment, it proves to be a relatively slow and error-prone file access method in the current setting. It seems there is a trade-off factor between the performance of distributing the indexing task among grid nodes (using GridLucene) and local indexing. We did not collect enough data to test whether this is in fact the case and where it plateaus. This is therefore something we intend to study further, for example using larger and more heterogeneous corpora

We have not yet investigated whether a lower level alteration of Lucene might also be fruitful in a grid environment. When GT4 becomes available on our grid, we intend to answer that question as well. Furthermore, we plan to use and evaluate

divide-and-conquer grid API's such as IBIS [36, 35] in an information retrieval setting.

8. AVAILABILITY

GridLucene is available for download at `http://ilps.science.uva.nl/Resources/`, under the same license as Lucene is distributed with.

9. ACKNOWLEDGMENTS

Thanks to Gilad Mishne for his insights into the inner workings of Lucene, to Machiel Jansen for his last minute comments, and to the reviewers for their constructive comments.

This work was carried out in the context of the Virtual Laboratory for e-Science project (`http://www.vl-e.nl`). This project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ).

Maarten de Rijke was supported by the Netherlands Organisation for Scientific Research (NWO) under project numbers 017.001.190, 220-80-001, 264-70-050, 354-20-005, 612-13-001, 612.000.106, 612.000.207, 612.066.302, 612.069.006, and 640.001.501.

10. REFERENCES

[1] M. Aldinucci, M. Coppola, S. Campa, M. Danelutto, M. Vanneschi, and C. Zoccolo. Structured implementation of component based grid programming environments. In *Dagstuhl Seminar Future Generation Grid 2004*, CoreGRID series. Springer Verlag, 2005.

[2] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, and I. Foster. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *MSS '01: Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, page 13, Washington, DC, USA, 2001. IEEE Computer Society.

[3] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[4] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. P. C. Hong, B. Collins, N. Hardman, A. C. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N. W. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead. The design and implementation of grid database services in ogsa-dai. *Concurrency and Computation: Practice and Experience*, 17(2–4):357–376, 2005.

[5] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Procs. of CASCON '98*, Toronto, Canada, 1998.

[6] R. Berlich, M. Kunze, and K. Schwarz. Grid computing in europe: from research to deployment. In *CRPIT '44: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 21–27, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[7] J. Callan. Distributed information retrieval. In W. B. Croft, editor, *Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval*, The Kluwer International Series in Information Retrieval, pages 127–150. Kluwer Academic Publishers, 2000.

[8] S. D. S. Center. JARGON, a Java API for the DataGrid, 2006. `http://www.sdsc.edu/srb/jargon`.

[9] C. Clarke, N. Craswell, and I. Soboroff. The TREC Terabyte retrieval track. *SIGIR Forum*, 39(1):25–25, 2005.

[10] C. Clarke, F. Scholer, and I. Soboroff. The TREC 2005 Terabyte track. In *The Fourteenth Text REtrieval Conference (TREC 2005) Proceedings*, 2005.

[11] G. Fedak, C. Germain-Renaud, V. Neri, and F. Cappello. XtremWeb: A generic global computing system. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 582, Washington, DC, USA, 2001. IEEE Computer Society.

[12] I. Foster. Internet Computing and the Emerging Grid. *Nature*, 7 Dec. 2000.

[13] I. Foster. What is the grid? A three point checklist. *GRIDtoday*, 1(6), 2002. `http://www.gridtoday.com/02/0722/100136.html`.

[14] I. Foster. Globus Toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer Verlag, 2005.

[15] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, Feb. 2003.

[16] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[17] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[18] I. Foster, C. Kesselman, L. Pearlman, S. Tuecke, and V. Welch. The community authorization service: Status and future. In *Proceedings of Computing in High Energy Physics 03 (CHEP '03)*, 2003.

[19] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[20] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.

[21] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The Open Grid Services Architecture, Version 1.0. Technical report, Global Grid Forum, 2005. `http://www.ggf.org/documents/GFD.30.pdf`.

[22] C. Germain-Renaud, G. Fedak, V. Néri, and F. Cappello. Global computing systems. In *LSSC '01: Proceedings of the Third International Conference on Large-Scale Scientific Computing-Revised Papers*, pages 218–227, London, UK, 2001. Springer-Verlag.

[23] C. Germain-Renaud and D. Monnier-Ragaigne. Grid result checking. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 87–96, New York, NY, USA, 2005. ACM Press.

[24] GGF. Global Grid Forum, 2006. `http://www.ggf.org`.

[25] Globus. The Globus toolkit, 2006. `http://www.globus.org/toolkit`.

[26] ILPS. The ILPS extension of the Lucene search engine. `http://ilps.science.uva.nl/Resources`.

[27] ILPS. Moodviews. `http://www.moodviews.com`.

[28] LookSmart. Grub's distributed web crawling project, 2006. `http://grub.looksmart.com`.

[29] Lucene. The Lucene search engine. `http://lucene.apache.org/`.

[30] Q^2ADPZ. `http://qadpz.idi.ntnu.no`.

[31] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.-Y. Chen, and R. Olschanowsky. Storage resource broker - managing distributed data in a grid. In *Computer Society of India Journal, Special Issue on SAN*, volume 33, pages 42–54, October 2003.

[32] B. Sotomayo and L. Childers. *Globus Toolkit 4: Programming Java Services*. The Morgan Kaufmann Series in Networking. Morgan Kaufmann, 2006.

[33] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. In *Concurrency and Computation: Practice and Experience*, 2004.

[34] A. Trotman and M. Lalmas. Introduction to the inex 2005 workshop on element retrieval methodology. In *Proceedings of the INEX 2005 Workshop on Element Retrieval Methodology, Second Edition*, 2005.

[35] R. V. van Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, and H. E. Bal. Ibis: an efficient Java-based grid programming environment. In *Joint ACM Java Grande - ISCOPE 2002 Conference*, pages 18–27, Seattle, Washington, USA, November 2002.

[36] K. van Reeuwijk, R. V. van Niewpoort, and H. E. Bal. Developing Java grid applications with Ibis. In *Proc. of the 11th International Euro-Par Conference*, pages 411–420, Lisbon, Portugal, September 2005.

[37] G. von Laszewski, P. Z. andTan Trieu, and D. Angulo. The Java CoG kit experiment manager. Technical report, Argonne National Laboratories, 2006.

[38] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela. SETI@HOME – Massively distributed computing for seti. *Comput. Sci. Eng.*, 3(1):78–83, 2001.

[39] L. J. Winton. A simple virtual organisation model and practical implementation. In *CRPIT '44: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 57–65, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[40] ZetaGrid. `http://www.zetagrid.net`.