A Grid-Based Infrastructure for Distributed Retrieval

Fabio Simeoni[1], Leonardo Candela[2], George Kakaletris[3], Mads Sibeko[4],
Pasquale Pagano[2], Giorgos Papanikos[3], Paul Polydoras[3],
Yannis Ioannidis[3], Dagfinn Aarvaag[4], and Fabio Crestani[1]

[1] Department of Computer and Information Sciences, University of Strathclyde, Glasgow, UK
{fabio.simeoni, f.crestani}@cis.strath.ac.uk
[2] Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo" – CNR, Pisa - Italy
{candela, pagano}@isti.cnr.it
[3] Department of Informatics and Telecommunications, University of Athens – Athens, Greece
{g.kakaletris, g.papanikos, p.polydoras, yannis}@di.uoa.gr
[4] Fast Search & Transfer ASA, Oslo, Norway
{mads.sibeko, dagfinn.aarvaag}@fast.no

Abstract. In large-scale distributed retrieval, challenges of latency, heterogeneity, and dynamicity emphasise the importance of infrastructural support in reducing the development costs of state-of-the-art solutions. We present a service-based infrastructure for distributed retrieval which blends middleware facilities and a design framework to 'lift' the resource sharing approach and the computational services of a European Grid platform into the domain of e-Science applications. In this paper, we give an overview of the DILIGENT *Search Framework* and illustrate its exploitation in the field of Earth Science.

1 Introduction

The problem of retrieving information which is widely disseminated and autonomously managed has a wide range of possible solutions. Variability is in terms of:

– *models*: from those in which queries and data are structured or semi-structured (data retrieval), to those in which they are mostly or entirely unstructured (document or content-based retrieval) [1];
– *approaches*: from those in which queries are distributed along with the data (distributed retrieval) [2,3], to those in which the data is centralised around the queries (content crawling and metadata harvesting) [4];
– *architectures*: from those in which queries emanate from clients and data is held at servers (client-server architectures), to those in which both may originate from any of a number of peers (peer-to-peer architectures) [5].

Across the spectrum, the core challenges remain those associated with the latencies of the underlying network and the heterogeneity of data, tools, means, and purposes which may be observed across communicating nodes. It is increasingly recognised that the scale of these challenges requires infrastructural support to reduce the development costs normally associated with state-of-the-art solutions [6]. It is equally recognised that, in most areas, infrastructural support remains piecemeal and revolves around open-source implementations of standard protocols and formats.

Issues of large scale distribution and heterogeneity are particularly acute in e-Science communities, where infrastructures are called upon to enable secure, cost-effective, and on-demand *resource sharing* [7]. This is the Grid vision [8], and current-generation infrastructures increasingly realise it under a service-based paradigm and for low-level computational resources, such as networks, storage, and processing cycles [9,10]. Building on these platforms, next-generation infrastructures set out to extend the vision into application domains, where the scope for resource sharing broadens to include, among others, retrieval services [11]. The impact is potentially non-trivial, for co-ordinated sharing of application resources may invalidate cost analyses of retrieval solutions which assume more conventional deployment scenarios: solutions with high adoption costs may be *outsourced* to the Grid infrastructure.

The DILIGENT project[1] is one of the first attempts to systematically lift into the Digital Library (DL) domain the facilities of a European Grid platform[2] (see also [12,13]). The expected outcome is a rich infrastructure of internetworked machines, *middleware services*, *domain services*, and *application services* in which resource sharing is an implication of *virtual digital libraries*, i.e. DLs that are: (*i*) assembled *declaratively* from community-provided datasets and application services; and (*ii*) deployed and re-deployed *on-demand* across machines by middleware services, according to availability, performance, and functional constraints. This is genuinely ambitious, for it reflects a model of application-level sharing which encompasses not only data resources but also domain and application services: like computing cycles, storage, and data before, application logic becomes a commodity within an infrastructure which abstracts over its physical location at any given time. The dynamic deployment of services is the key challenge of DILIGENT and its primary contribution to the Grid vision for application domains.

The DILIGENT infrastructure retains the service-orientation of the underlying platform and organises its services across three layers: the *Collective Layer* (CL), where middleware services define, deploy, secure, and otherwise support the operation of DLs; the *Digital Library Layer* (DLL), where domain services manage the data and orchestrate processes against it; and the *Application Specific Layer* (ASL), where application services mediate between users and the services of the CL and DLL layers. Within the DLL layer, in particular, the infrastructure offers novel opportunities for supporting application development: not only may its domain services be *invoked* to offer sophisticated functionality, they may also be designed so as to be *specialised and extended* into application services which are tailored to the bespoke needs of adopting communities. In this case, a service-oriented infrastructure becomes a *service-oriented framework*.

In this paper, we focus on a core part of the DILIGENT DLL layer in which infrastructural support is largely in terms of a framework for application services. In particular, we abstract away from DLL services dedicated to content, metadata, annotation, and workflow management, and concentrate instead on the DILIGENT *Search Framework*, i.e. the set of DILIGENT services for the distributed retrieval of both data and

[1] `http://www.diligentproject.org/`

[2] Enabling Grids for E-sciencE, `http://public.eu-egee.org/`

documents. We discuss the framework in Section 2 and we report on its exploitation within the domain of Earth Science, a challenging e-Science, in Section 3. Finally, we conclude in Section 4, where we outline directions for further work.

2 The DILIGENT Search Framework

Within a service-oriented framework, the notion of 'service' acquires structure and granularity to accommodate functional composition and abstraction, respectively (cf. Figure 1). Precisely, our service model distinguishes between: (*i*) *service classes*, i.e. flat groupings of services within the same functional area (e.g. the *Index* class), (*ii*) *services*, i.e. abstract instances of service classes (e.g. the *LookupService* in the Index class), and (*iii*) *web services*, i.e. entry-points to concrete implementations of abstract services (e.g. the *LookupFactoryService*). Service implementations are dynamically deployable, and doing so on one or more *Host Nodes* of the DILIGENT infrastructure (DHNs) yields *running instances* of the service (RIs).

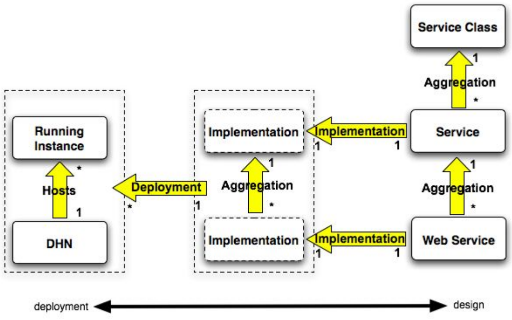

Fig. 1. Service Model

Against this model, the support offered by the framework is twofold. Firstly, it provides a set of *core services* which coordinate the functionality of application and domain services towards a wide range of distributed search processes; depending on the semantics of the orchestrated services, processes may fall at arbitrary points within the content-based vs. data-based spectrum and operate upon different forms of content and metadata (full-text or multimedia search, similarity search, structured and semi-structured search, etc.). Discussed in Section 2.1, the core services provide foundations to support the heterogeneity and dynamicity of data and processing requirements which can be observed within e-Science scenarios. Secondly, the Search framework offers design blueprints, partial implementations, and libraries for the development of application and domain services compliant with second-generation Web Service standards [14,15]. Two distinguished classes of such services, namely the index management services and the service for content description, selection and result fusion, are outlined in Sections 2.2 and 2.3, respectively.

2.1 The Core Services

The DILIGENT *Search Service* is a *Service Class* that groups all fundamental functional elements (i.e. Services) related to Information Retrieval (IR), but not directly bound to a more specific thematic area, such as Indexing, Distributed Information Retrieval (DIR), etc. The overall search management as well as several gluing elements fall within its scope. In contrast, we refer to the *Search Engine* as the full fledged set of elements that serve IR in DILIGENT. Finally, we refer to *Search Framework* when implying not only software but also protocols, rules and even guidelines for implementing and extending DILIGENT IR.

Search Overview. DILIGENT Search Engine, modular even in its internal structure, captures its requirements through a complex, yet straightforward, set of concepts and mechanisms, which are depicted in Figure 2.

The main idea behind the presented architecture is that the actual work of retrieving and processing the information and data contained in DILIGENT or other sources, is not an integral part of the Search Service, but can rather be off-loaded to entities that focus on different (D)IR and data processing aspects. Yet, the Search Service comes bundled with a set of such entities to enable the out-of-the-box use of the system.

The search task is captured by a set of steps which manage:

– Interaction with the ultimate consumer of the service (e.g. the User Interface) through a query language and various alternative interaction staging facilities;
– Consolidation of information regarding the status and availability of resources in need / reach;
– Preparation of the IR process through advanced facilities that potentially impose changes over the initial query;
– Operation planning, producing a near-optimal workflow of low-level search operations, in terms of resource utilisation;
– Execution of the workflow, i.e. invocation of the low-level search operations, (potentially off-loaded to external engines) and progress monitoring;

In this operation, the *Search Orchestrator*, which falls is the class of *Services* in the service model, is the entry point of the Search Engine, and acts as the manager of the IR procedure. Under its co-ordination, collaborating, yet independent, sets of service classes, such as the DIR, the Index and the Metadata Management ones, form the backbone of DILIGENT Search Engine.

The major hooks for extending functionality that renders the overall engine capable of capturing the requirements for custom processing, raised by its addressed communities, have been sketched in the above. These entail both the IR preparation steps, masked under the *Query Preprocessor* concept, as well as the run time processing performed by single components of the *Search Operators* set. Within the adopted architecture,

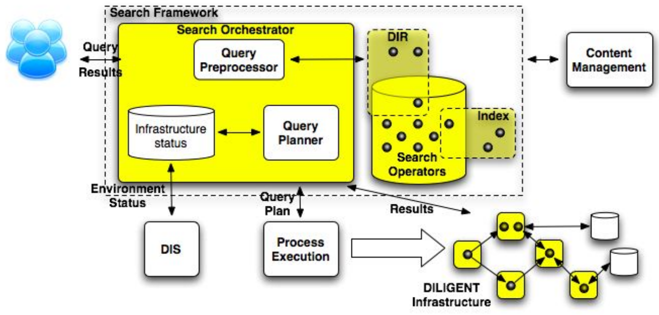

Fig. 2. The DILIGENT Search Framework

the quality and performance of the IR operations are, to a large degree, subject to the performance of particular elements plugged-in at these predefined placeholders.

In the following paragraphs we deepen on selected aspects of the DILIGENT Search Engine and its operation.

Serving a Request. The first step, even prior to searching, is the construction of a user query, i.e. an XML based, tree-like, abstract representation of the IR processing tasks involved in a search operation, that the engine must perform on behalf of the user. This query, formulated in a strict structure, is passed to the *Search Orchestrator* and has its validity thoroughly verified.

This procedure, i.e. validation, is an informed decision based on the availability of infrastructure resources. This information is gathered by the DIS, the DILIGENT CL service playing the role of global registry commonly used in both SOA architectures and Grid infrastructures to monitor and discover resources. The DIS has a two-fold role in the context of Search. The first is the discovery of the hosting environment the engine resides and the resources it can allocate to a task, e.g. Web Services, RIs, DHNs. This environment does not only drive the construction of query execution plans, but in conjunction with the dynamic deployment feature and the modular nature of service oriented approaches, it can imply the need to create new resources to utilise in future evaluations. The second key role of the DIS is to actually enable the modular architecture the Search framework provides. Available service instances define their semantics within the search context by publishing them to the DIS in a common way. These semantics are used by the *Query Planner* to produce execution plans.

The *Search Orchestrator* is constantly aware of the environment, through repetitive interactions with the DIS. Given a user query, the service supervises the individual steps that produce the query results. These steps include validation, preprocessing, such as injecting personalised information and pre-selection over the available sources, and linguistic processing. The enriched query and the environment information are passed to the *Query Planner*, in order for it to construct the execution plan.

The individual steps of this plan are performed by elements which are labelled as *Search Operators*. These operators are actually services that enable both information retrieval and data processing. Processing can refer to standard operations, e.g. sort, join, merge, but also to more complex ones, such as mathematical and logical expression evaluation, aggregation and even branching evaluation, all within the same workflow. Following the service-oriented approach, search operators build up a pool of dynamically selectable services, allowing unconstrained extensibility of the Search Service through the addition of new, custom, processing blocks. These very concepts equip the Search Engine with the capability to exploit elements such as DataFusion, Indices and Metadata handling services, as Search Operators themselves, even if they are distinct components. Search Operators follow the aforementioned Service Model, thus it is possible to have the domain-specific Services embedding multiple operation steps in a single Operator Service, leaving room for trade-off evaluation of distributed computational power versus highly optimised local transactions.

Planning and Optimising Execution. In order to produce the execution plan, the *Query Planner* matches abstract queries to concrete service instance invocations. This is accomplished using templates that specify the query sub-trees a service can satisfy.

The outcome of the procedure is a graph of service invocations, adopting the orchestration model of the Business Process Execution Language (BPEL). According to this, there is a global view of the participant service instances, under the central control of the execution engine (the DILIGENT Process Execution Service). This paradigm accommodates, among others, better performance, sophisticated control and fine grained error handling. The final plan is subsequently passed to the available execution engine which executes and monitors it so that the desired results are produced and passed back to the user.

Due to the dynamic nature of the underlying infrastructure, the query planner does not have inherent knowledge of available service instances, but instead receives this information externally from the *Search Orchestrator*. This favours both extensibility, by adding/removing service instances, and flexibility, by changing existing instances.

It is a fact that, due to the plethora of resources, a single user query can be served by more than one execution plan, with semantic equivalence yet significant diversion in terms of resource requirements. Under this observation, the Query Planner, and more specifically its optimisation component, is responsible not only to construct a semantically and operationally valid execution plan, but also to achieve minimal resource consumption while maintaining a certain quality of service level. Yet, this *optimisation* has to accomplished within a reasonable time limit, since searching for the best execution plan is generally a computationally intractable problem and involves enumerating and checking a potentially huge number of alternative options.

Query optimisation is managed with optimisation techniques borrowed from distributed databases [16,17]. It traverses through the solution space of alternative execution plans, estimates their costs and chooses the best candidate. This involves not only selecting the best set of services that can compute a user query, but also choosing the best hosting nodes of these service instances. Optimisation takes place in two-steps [18]. First, an abstract execution plan is produced and then site selection is performed.

A mathematical cost formula, tailored to the DILIGENT service oriented architecture, is employed. It takes into consideration factors like data source statistics, service operational complexity, intermediate results size, cost of messages among service invocations, communication initialization/termination overheads, intermediate result fragmentation, etc. Cost estimation is constantly enhanced by an active component that monitors the plan execution and refines both the formula and the stored statistics [19].

Moving Large DataSets. Throughout the execution of a search workflow, the need to transfer potentially large amounts of data between services is evident. Given the particular hosting environment of the Grid, enriched by the possibilities provided by the dynamic deployment features of DILIGENT, it is essential to utilise a common framework for data transferring.

The separation of concerns achieved by extracting relevant concepts out of the main functional logic of the services, allows them to be easily and uniformly reused for all search elements (i.e. Services). This lead to the DILIGENT ResultSet framework, i.e. the leveraging system data transfer mechanism be comprised of the *ResultSet* service and client elements. Its utilisation aims at hiding the complexities of underlying protocols and data locations. Yet, benefits rising from their exploitation are offered to consumer / producer services, which are able to take advantage of them with the minimum cost and complexity.

The encapsulated logic allows on-the-spot processing of data, eliminating unnecessary data movements, usually avoiding protocol stack and network engagement.

Furthermore, pipelining of execution is enabled through paged data transfer facilities, integrated in the framework. Based on this feature, the ResultSet can also act as a flow control mechanism, freeing component services of unnecessary resource consumption in a uniform manner.

2.2 Index Generation and Management

The role of the Index service in the Search framework is to facilitate fast and scalable information retrieval from a number heterogeneous information sources over the distributed and unstable Grid environment. This is achieved by generating and maintaining a number of different indices, deeply integrated in the DILIGENT infrastructure. In collaboration with the Planner and the other Search services, both low response time services and complex search operators are made available.

The main obstacle to be overcome by the design of the Index service, was to be able to ensure both stability and low response times on a highly distributed and heterogeneous system. Clearly replication, both concerning files and services, was needed. As an indices might grow very big, replicating the full index is potentially a slow and bandwidth heavy process. With this in mind, it was decided to represent indices by way of *delta files*[3], which can both easily be replicated through DILIGENT services and be used to keep service replications up to date. In order to manage the delta files and replication process the Index service has been designed as service oriented framework characterised by services playing three distinct roles: (*i*) *Manager service*, i.e. a service managing and representing one specific index in terms of delta files used to build it; (*ii*) *Updater service*, i.e. a service responsible for consuming content from a content source, transforming this into delta files, and updating the Manager service of a specific index. Any number of Updater instances may be connected to a single Manager instance, allowing for highly distributed feeding of an index; and (*iii*) *Lookup service*, i.e. a service that will use the delta files maintained by a Manager in order to build a replication of the index locally on a node. Any number of Lookup instances can be connected to one logical index representations (Manager instances), thereby providing replication of the Lookup Service and the actual physical index, adding both stability, fault tolerance and query performance.

In order to ensure that the Manager-Updater-Lookup framework is easily and uniformly implemented current and future Index service implementations, a library handling delta files storage, management and retrieval in addition to providing needed Web Service operations for the different roles was implemented. Using the library, the three roles can be implemented in a single Service, or as distinctly separate Services. In this manner, four different index distribution configurations can be implemented: (*i*) *All-in-one*, all roles are implemented through the same Service thus ensuring low latency between document feeding and searchability; (*ii*) *Lookup separated*, promoting any number of lookup instances thus supporting query intensive environments even if it might not be applicable for large indices or indices with a large number of information

[3] Delta files contain information on how to transform each version of the index to the next a.k.a. the difference between the versions.

sources; (*iii*) *Updater separated*, keeps the Updater separated from the rest thus being able to handle a more intense feeding process for large indices or indices with a number of information sources where a low query frequency is expected; (*iv*) *One service per role*, promotes one service per role on diverse nodes thus offering replication both at the updater and lookup level.

Three standard index implementations have been created in the current DILIGENT implementation supporting three different indexing scenarios with distinct data types. A *Full Text Index* providing the functionality of retrieving entries based on text queries against the full text contents of the entries. By manipulating the index profile it is possible to customize this index, e.g. specify the index structure, the query processing supported and the result sets including advanced linguistic processing. The full text index is implemented using the one-service-per-role distribution. A *Forward Index* supporting simple and fast key-value lookups. It is implemented using the one-service-per-role distribution. A *Geographical Index*, supporting geospatial and spatio-temporal search over a very large set of objects described by their location in a geographical system. In expectation of highly distributed content sources, intense feeding process, and high query frequencies, it was decided to use the one-service-per-role index distribution. This lead to the Geographical index being implemented by way of the following three Web Services: GeoIndexManagementService, GeoIndexUpdaterService and GeoIndexLookupService. The functionality of the latter Web Service in respect to Earth Science will be further described in Section 3.

2.3 DIR Services

Complementing search and indexing services, three service classes provide further support for content-based retrieval within the framework: namely, *Content Source Selection* (CSS), *Content Source Description* (CSD), and *Data Fusion* (DF). Collectively, CSD, CSS, and DF services perform the tasks which characterise content-based *Distributed Information Retrieval* [3], an active field of research which has had limited uptake in the practice of information services so far [20]. In more detail: (*i*) CSD services generate and maintain summary descriptions of content sources, such as partial indices, collection-level term statistics, or result traces from training or past queries; (*ii*) CSS services limit the routing of queries to the sources which appear to be the best targets for their execution, where ‘goodness’ criteria include the relevance of content, the sophistication of retrieval engines, and the monetary costs associated with query execution; and (*iii*) DF services derive a total order of the result sets produced by target sources with respect to different scoring functions and content statistics.

The framework sets out to support a wide range of DIR strategies. For example, a CSD service may base source descriptions on term statistics derived from full-text content indices, while another may do so using partial indices derived directly from the content through query-based sampling techniques [21]. Similarly, a DF service may rely on a round-robin algorithm to merge results, another may be biased by the output of a CSS service, and yet another may employ non-heuristic techniques and leverage the output of a CSD service to ‘consistently’ re-rank results with respect to global content statistics.

Though pairwise different, strategies of description, selection, and fusion share a common architecture which the framework attempts to capture within an extensible design. In it, services are stateful and the state of their RIs is comprised of source descriptions (CSD) or sets thereof (CSS, DF) which are generated, accessed, and updated through distinguished web services. In particular:

– *access services* expose the state of RIs to clients, or otherwise consume it on their behalf. CSS *selectors* and DF *mergers* consume sets of descriptions to select over content sources and merge results which emanate from them, respectively. CSD *descriptors* expose descriptions through fine-grained interfaces suitable for selective access, or through file-transfer facilities suitable for coarse-grained access. Selectors choose sources based on cut-off points within rankings of their descriptions, where cut-offs are either specified by the client or else are derived from upper bounds on the number of results to be retrieved, also indicated by clients; in the latter case, selectors return an estimate of the number of results to be retrieved from selected sources. To promote responsiveness and optimal resource consumption, mergers process streams of results, using facilities provided the ResultSet service. Streaming is also supported in output, if the merging strategy allows it; in this case, a callback mechanism allows result merging on demand within configurable timeouts.
– *monitor services* observe the external environment for changes which may trigger an update to the state of RIs. CSD monitors react to changes to indices of content sources, for which they subscribe with instances of Index services; the regeneration of a description is governed by *update policies* based on a configurable combination of time and space criteria (i.e. every so often and/or whenever indices have changed of a given proportion). CSS and DF monitors react instead to changes to descriptions, for which they subscribe with CSD descriptors. In all cases, subscriptions are brokered by services in the CL, so as to achieve resilience to the redeployment of notification producers.
– *factory services* generate the state of RIs. The separation of factories and access services distinguishes two phases in the lifetime of RIs: in the *operative phase*, clients interact with access services to act upon state; in the *generative phase*, state is created, derived, updated or otherwise materialised locally to the instances. While the two phases may interleave during the lifetime of instances, their exact time of occurrence is ultimately determined by client strategies. Further, state generation may occur in either a passive or a proactive mode: in the *passive mode*, clients trigger generation by interacting with factories at any point in the lifetime of RIs; in the *proactive mode*, the instances create resources autonomously by reacting to the observation of key events in the environment, starting from their very deployment of the instance on a node (bootstrapping).

State persistence, service publication, Grid-based file exchange mechanisms, streamed processing, lifetime management, and other forms of interactions with the infrastructure are handled transparently. Specialisation is supported by: (*i*) extensive use of declarative configurations; (*ii*) domain-specific libraries for inverted index management, update policy, and best-effort discovery strategies; (*iii*) design patterns which allow pluggable algorithms for selection, fusion, and object bindings of XML serialisations of descriptions and results. Specific CSD, CSS, and DF services which make use of these facilities are described in Section 3.

3 The Search Framework in Action: Searching the Earth Science Information Space

Earth Science is a discipline that well represents the complex nature of e-Science activities and may thus gain tremendous benefits from the DILIGENT infrastructure. Earth Science scientists need to access data and tools within a multi-institutional, heterogeneous and large-scale context. The analysis and the generation of objective facts on the Earth status, i.e. Earth Observation (EO), require integration of specific data products, handling of information in multiple forms and use of storage and computing resources in a seamless, dynamic and cost effective way. The typical desiderata about the retrieval paradigm consists in mixing bugs of keywords, either free terms or ontology extracted, with geo-location features aiming to capture the area of pertinence.

The building of *periodical environmental reports*, for example, is a typical EO activity where the DILIGENT search infrastructure proves its appropriateness. These complex information objects, mostly built as aggregation of other information objects, require a lot of existing information, coming from worldwide distributed heterogeneous sources. This information has to be properly discovered and uniformly accessed. The so collected information has often to be coherently integrated with pertinent information generated on-demand through procedures that often need to access and process huge amount of data. This scenario has been termed *Implementation of Environmental Conventions* (IMPECT) and details on its implementation are provided in the follow.

The DILIGENT infrastructure to serve IMPECT consist of "external" content providers, such as the NASA CEOS IDN initiative[4], the European Environment Agency[5], and Medspiration[6], plus a pool of three community specific data sources, all placed at the ESA's European Space Research Institute (ESRIN), namely: the EO ESA web portal[7] documents and data, the EO Grid on demand system[8], and EO catalogue together with relevant databases and archives.

The resulting information space is tremendously heterogeneous, it contains classic documents like research studies and meteorological papers, satellite images, EO products like Chlorophyll-1 measure or vegetation indexes. To process such data effectively and efficiently appropriate customisation and instantiations of the DILIGENT Search framework have been needed but easy thanks to the possibility to plug-in new search operators in the search procedure.

With respect to the indexing facilities, the Geographical index is intended for such use. The web services used to implement this index use innovative techniques providing a highly dynamic search experience and allowing for post-index-creation addition of ad hoc query and sorting algorithms.

The Geographical index can be queried through an "index replication" represented by an instance of a GeoIndexLookupService. As these replications are based on a two dimensional R-Tree [22], the two step query processing normally used with R-Trees

[4] `http://idn.ceos.org`

[5] `http://www.eea.eu.int/`

[6] `http://www.medspiration.org/products/`

[7] `http://www.eoportal.org`

[8] `http://giserver.esrin.esa.int/`

[23] is also used in the Geographical index. This scheme calls for a filter step and a refinement step. In the filter step, standard MBR (Minimal Bounding Rectangle) queries are performed against the R-Tree producing a candidate set based on entries MBRs, which may have a number of false hits when considering the actual detailed geometry of both the query and entries [23]. In order to eliminate false hits, the candidate set is refined based on additional parameters, often but not necessarily relating to the actual geometry of the objects or queries. In between the standard two R-Tree query processing steps, the GeoIndexLookupService also implements a third step in order to rank and sort the refined results. The algorithms used in both the refinement and ranking steps greatly affect the functionality of the application and will vary for different use cases. By relying on the openness of the whole framework and on the Index, it is possible to introduce other refinement and sorting algorithms making the Geographical index adaptable to potentially any use case.

Two refinement operators and two ranking operators have been implemented. The TemporalRefiner and PolygonalRefiner refinement operators allow the user to refine the search results based on their timestamp or a specified polygonal shape. The TemporalRankEvaluator and ArealOverlapRankEvaluator ranking operators allow the results to be ranked based on their timestamp or based on the amount of overlap between their MBR and the query.

In case the refinement algorithm is very computing intensive and the candidate set returned from the filter step is large, low response times are upheld by exploiting the ResultSet Service mechanism. By only doing partial filtering [24] and ranking steps, and synchronising these with requests to the ResultSet Service, these steps are only performed for the entries returned to the user, saving a vast amount of computing cycles, and greatly lowering the GeoIndexLookupService's response time.

With respect to the Distributed Information Retrieval, the CSD service generates and maintains *term histograms* of textual sources, a coarse-grained form of index where containment relationships between terms and documents is intentionally abstracted over. The service interacts with the Index services to derive the histograms from full-text content indices and also to subscribe for point-to-point notifications of changes to such indices. The CSS service selects sources based on rankings produced with the standard CORI algorithm [25]; rankings are based on estimated relevance of content, and rely on term histograms staged from the CSD service prior to query submission. In particular, the service subscribes with the CSD service for changes to the staged histograms and updates them upon notification of such changes. Finally, the DF service merges query results based on either one of three techniques: a plain round-robin algorithm, a consistent merging algorithm, and a linear regression method based on source selection scores. The first offers the least effectiveness but acts as an upper bound on performance (results remain unparsed, output can be streamed). The second uses global statistics to give the best effectiveness but also the highest overhead (results are fully parsed, output cannot be streamed); in this case, the service interacts with the reference CSD service to gather histograms in advance of result submission. The third explores middle ground between the first two, and uses the output of the reference CSS service to heuristically normalise inconsistent result scores; as interaction with the CSS service must necessarily occur during query execution, deployment services in the CL layer are instructed to co-deploy both services on the same node.

4 Conclusion and Future Works

We have outlined the design of a service-based framework for large-scale distributed retrieval, built atop the computational facilities of a European Grid platform. While initial experience in the Earth Observation domain has built up some confidence around the capacity of the framework to accommodate the requirements of e-Science applications, there are a number of implementation and design improvements which are to be addressed in the immediate future, some of which we outline next.

Further abstraction over some particular service types, such as the information sources (e.g. indices, external search engines), will enhance opportunities for integration in search operations. Experimentation on query planning and optimisation by utilisation of domain-specific heuristics and content distribution statistics, based on non-linear regression techniques [26] is expected to yield faster and higher quality results. Improvements on the performance of the ResultSet transport mechanism are expected to be achieved through further exploitation of facilities provided by the underlying platform, such as GridFTP's parallel striped transfers. Partitioning of indices across DHNs, so as to exploit the resource pooling of the Grid for the distributed storage of very large indices, will improve availability on extremely large datasets. Finally, the support of uncooperative DIR strategies, such as query-based sampling techniques for generating term histograms or partial indices of external content sources, would allow us to support more advanced techniques of selection and fusion (e.g. the Semisupervised Learning method of data fusion [27] and the Unified Utility Maximisation method of resource selection [28]).

Acknowledgments. This work is partially funded by the European Commission in the context of the DILIGENT project, under the 2nd call of FP6 IST priority.

References

1. Blair, D.C.: The data-document distinction revisited. SIGMIS Database 37, 77–96 (2006)
2. Sanderson, R.: Srw: Search/retrieve webservice. Public Draft (2003)
3. Callan, J.: 5 Distributed Information Retrieval. In: Advances in Information Retrieval, pp. 127–150. Kluwer Academic Publishers, Hingham, MA (2000)
4. Kobayashi, M., Takeda, K.: Information retrieval on the web. ACM Comput. Surv. 32, 144–173 (2000)
5. Risson, J., Moors, T.: Survey of research towards robust peer-to-peer networks: search methods. Comput. Networks 50, 3485–3521 (2006)
6. Atkinson, M., Crowcroft, J., Goble, C., Gurd, J., Rodden, T., Shadbolt, N., Sloman, M., Sommerville, I., Storey, T.: Computer Challenges to emerge from eScience (e-Science vision document)
7. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organization. The International Journal of High Performance Computing Applications 15, 200–222 (2001)
8. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum (2002)
9. Globus Alliance: The Globus Alliance Website, `http://www.globus.org/`

10. EGEE: Enabling Grids for E-sciencE. INFSO 508833, `http://public.eu-egee.org/`
11. Atkins, D.E., Droegemeier, K.K., Feldman, S.I., Garcia-Molina, H., Klein, M.L., Messerschmitt, D.G., Messina, P., Ostriker, J.P., Wright, M.H.: Revolutionizing science and engineering through cyberinfrastructure. Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure (2003)
12. Larson, R.R., Sanderson, R.: Grid-based digital libraries: Cheshire3 and distributed retrieval. In: JCDL '05: Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries, pp. 112–113. ACM Press, New York (2005)
13. GRACE: GRid seArch & Categorization Engine (2005), `http://www.grace-ist.org`
14. Banks, T.: Web Services Resource Framework (WSRF) - Primer. Committee draft 01, OASIS (2005), `http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-01.pdf`
15. Niblett, P., Graham, S.: Events and service-oriented architecture: The oasis web services notification specification. IBM Systems Journal 44, 869–886 (2005)
16. Kossmann, D.: The state of the art in distributed query processing. ACM Computing Surveys 32, 422–469 (2000)
17. Ioannidis, Y.E.: Query optimization. ACM Computing Surveys 28, 121–123 (1996)
18. Stonebraker, M., Aoki, P., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C., Yu, A.: Mariposa: A Wide-Area Distributed Database System. The VLDB Journal 5, 48–63 (1996)
19. Chen, C., Roussopoulos, N.: Adaptive selectivity estimation using query feedback. In: 1994 ACM SIGMOD International Conference on Management of data, pp. 161–172 (1994)
20. Simeoni, F., Azzopardi, L., Crestani, F.: An application framework for distributed information retrieval. In: Sugimoto, S., Hunter, J., Rauber, A., Morishima, A. (eds.) ICADL 2006. LNCS, vol. 4312, pp. 192–201. Springer, Heidelberg (2006)
21. Callan, J.P., Connell, M.E.: Query-based sampling of text databases. Information Systems 19, 97–130 (2001)
22. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data, pp. 47–57. ACM Press, New York (1984)
23. Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A., Theodoridis, Y.: R-Trees: Theory and Applications. In: Advanced Information and Knowledge Processing, Springer, Heidelberg (2006)
24. Martínez, C.: Partial Quicksort. In: The First Workshop on Analytic Algorithmics and Combinatorics (ANALCO04), New Orleans (2004)
25. Callan, J.P., Lu, Z., Croft, W.B.: Searching distributed collections with inference networks. In: SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 21–28. ACM Press, New York (1995)
26. Sun, W., Ling, Y., Rishe, N., Deng, Y.: An instant and accurate size estimation method for joins and selections in a retrieval-intensive environment. In: SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data, pp. 79–88. ACM Press, New York (1993)
27. Si, L., Callan, J.: A semisupervised learning method to merge search engine results. ACM Trans. Inf. Syst. 21, 457–491 (2003)
28. Si, L., Callan, J.P.: Unified utility maximization framework for resource selection. In: Grossman, D., Gravano, L., Zhai, C., Herzog, O., Evans, D.A. (eds.) CIKM, pp. 32–41. ACM, New York (2004)