Minersoft: Software Retrieval in Grid and Cloud Computing Infrastructures

MARIOS D. DIKAIAKOS, University of Cyprus
ASTERIOS KATSIFODIMOS, LRI Universite Paris-Sud XI and INRIA, Saclay
GEORGE PALLIS, University of Cyprus

One of the main goals of Cloud and Grid infrastructures is to make their services easily accessible and attractive to end-users. In this article we investigate the problem of supporting keyword-based searching for the discovery of software files that are installed on the nodes of large-scale, federated Grid and Cloud computing infrastructures. We address a number of challenges that arise from the unstructured nature of software and the unavailability of software-related metadata on large-scale networked environments. We present Minersoft, a harvester that visits Grid/Cloud infrastructures, crawls their file systems, identifies and classifies software files, and discovers implicit associations between them. The results of Minersoft harvesting are encoded in a weighted, typed graph, called the Software Graph. A number of information retrieval (IR) algorithms are used to enrich this graph with structural and content associations, to annotate software files with keywords and build inverted indexes to support keyword-based searching for software. Using a real testbed, we present an evaluation study of our approach, using data extracted from production-quality Grid and Cloud computing infrastructures. Experimental results show that Minersoft is a powerful tool for software search and discovery.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms: Design, Performance, Management

Additional Key Words and Phrases: Cloud computing, Grid computing, software search engine, resource management

ACM Reference Format:
Dikaiakos, M. D., Katsifodimos, A., and Pallis, G. 2012. Minersoft: Software retrieval in grid and cloud computing infrastructures. ACM Trans. Internet Technol. 12, 1, Article 2 (June 2012), 34 pages.
DOI = 10.1145/2220352.2220354 http://doi.acm.org/10.1145/2220352.2220354

1. INTRODUCTION

A growing number of large-scale Grid and Cloud infrastructures are in operation around the world, providing production-quality computing and storage services to numerous users from a wide range of scientific and business fields. Recent surveys report that by 2020 most people will employ software running on Cloud computing

This work was supported in part by the European Commission under the Seventh Framework Program through the SEARCHiN project, Marie Curie Action, contract FP6-042467, and the Enabling Grids for Escienc Eproject, contract INFSO-RI-222667. The work of A. Katsifodimos was done at the University of Cyprus.
Authors' addresses: M. D. Dikaiakos, Department of Computer Science, University of Cyprus, Cyprus; A. Katsifodimos, LRI, Universite Paris-Sud XI and INRIA, Saclay; G. Pallis, Department of Computer Science, University of Cyprus, Cyprus.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2012 ACM 1533-5399/2012/06-ART2 $15.00
DOI 10.1145/2220352.2220354 http://doi.acm.org/10.1145/2220352.2220354

infrastructures [Anderson and Rainie 2010]. One of the main goals of Cloud and Grid infrastructures is to make their services easily accessible and attractive to a wide range of users (Grid/Cloud application users/developers/administrators) [Armbrust et al. 2010; Dikaiakos et al. 2009]. Cloud computing is enabling companies and researchers to virtualize and externalize their data, providing them efficiency and flexibility in running their IT systems, and making it possible to deploy new services at low costs. To achieve this goal, it is important to establish advanced tools for software search and discovery in order to help users locate application software suitable to their needs and encourage software reuse [Bass et al. 2008; Mohagheghi and Conradi 2008], software investigation [Robillard 2008], clone detection [Gabel et al. 2008] and computational resources selection.

Software retrieval is an important element of software development and deployment in Grid/Cloud infrastructures, where the use and reuse of software components and libraries represent a major element of application development and infrastructure adoption. The need for software retrieval is a well-recognized problem in the computer software engineering literature and practice [Toch et al. 2007], but has been addressed primarily in the context of controlled software repositories [Grechanik et al. 2010] or, more recently, in the context of Web search engines [Coyle and Smyth 2007; Xue et al. 2008]. In the Grid and Cloud context, the problem is equally motivated but presents a different set of challenges: the software repositories are essentially the file systems of the different computing nodes of the infrastructures; these nodes comprise very large numbers of unclassified files belonging to a variety of types: source codes, binary executables, software libraries, software manuals, and numerous unrelated files. All these files are not attached to Web servers and are not accessible to Web search engine crawlers. Thus, existing Web search engines cannot be used for software retrieval in these infrastructures, since access to such a software cannot be gained through HTTP, the common transfer protocol of the Web. Furthermore, all these files do not contain explicitly specified links that could capture their interrelationships. Software installed in Grid/Cloud infrastructures is unstructured and software-related metadata or software descriptions in natural language are typically poor or unavailable. Also, typical file systems do not maintain metadata about the semantics of files and software. Consequently, software files are not easily amenable to information-extraction techniques used in information retrieval (IR) or semantic search techniques.

Nevertheless, adopting a full-text-based search paradigm for locating software seems like an obvious choice, given that full-text search is the dominant paradigm for information discovery [Li et al. 2008]. To motivate the importance of such a tool, let us consider a researcher who is searching for graph-mining software deployed on a Grid/Cloud infrastructure. Unfortunately, the manual discovery of such software is a daunting, nearly impossible task. Taking the case of EGEE/EGI,[1] one of the largest production Grids currently in operation, the software developer would have to gain access and search inside 300 sites, several of which host well over 1 million software-related files. The situation is equally challenging in emerging Cloud infrastructures: for example, the Amazon Elastic Cloud service provides access to a growing repository with more than 8,500 virtual computational servers (Amazon Machine Images, AMIs), with each AMI comprising over 14,000 files, including application and utility software. Therefore, the software developer would have to spawn some AMIs, connect to them, and search manually for installed software files.

Alternatives to manual search for software are limited. Although Grid infrastructures comprise centralized Grid information registries that can be queried to provide

[1]Enabling Grids for E-SciencE project:http://www.eu-egee.org/; European Grid Infrastructure:http://www.egi.eu/.

information about the configuration and status of Grid files, these registries typically contain scarce and poorly maintained tags about installed software [Dikaiakos et al. 2006]. The lack of well-organized and properly maintained information about software is due to the intrinsic characteristics of software management across large-scale, federated infrastructures: software installation and maintenance is performed by various actors in an uncoordinated manner and does not follow a common standard for software packaging and description. Similar problems arise in the context of Clouds, as they grow larger and more diverse. Currently, Cloud providers like Amazon, support only the capability to search for AMIs on the basis of their names and not on their contents.

Envisioning the existence of a software search engine, the software developer would submit a query to the search engine using some keywords (e.g., "graph mining" or "pajek"). In response to this query, the engine would return a list of software matching the query's keywords, along with additional textual descriptions and a listing of computational resources where this software was located. Thus, the software developer would be able to identify the providers hosting an application suitable to her needs and would accordingly prepare and submit jobs to them, thus, saving considerable effort. To meet this vision, we need a new methodology that will (i) automatically discover software-related files installed in large file systems; (ii) extract structure and meaning from those files; and (iii) discover and exploit implicit relationships between them. Also, we need to develop methods for effective querying and for deriving insights from query results. The provision of full-text search over large, distributed collections of unstructured data has been identified among the main open research challenges in data management that are expected to bring a high impact in the future [Agrawal and et al. 2008]. Searching for software falls under this general problem, since file-systems treat software files as unstructured data and maintain very little if any metadata about installed software.

Following this motivation, we developed the *Minersoft* software search engine. A prototype implementation of the Minersoft is available online.[2] Unlike desktop search, which is designed to assist users in locating specific files, Minersoft supports searching not only for source codes but also for executables and libraries stored in binary format, and metadata (software versions, timestamps, permissions). To the best of our knowledge, Minersoft provides the first full-text search facility for the retrieval of software installed in large-scale Grid and Cloud infrastructures. Minersoft visits a computational resource, crawls the file systems of Grid and Cloud computing sites, identifies software files of interest (binaries, libraries, documentations, etc.), assigns type information to these files, and discovers implicit associations between them. Subsequently, it extracts text and keywords from the context that surrounds software files in order to tag the software-files with descriptions amenable to full-text search.

To achieve these tasks, Minersoft invokes file-system utilities and object-code analyzers, implements heuristics for file-type identification and filename normalization, and performs document analysis algorithms on software documentation files and source-code comments. The results of Minersoft harvesting are encoded in the *software graph* (SG), which is used to represent the context of discovered software files. We process the software graph to annotate software files with metadata and keywords, and use these to build an inverted index of software. Indexes from different computational resource providers in Grids and Clouds are retrieved and merged into a central inverted index, which is used to support full-text software retrieval.

Our study is focused on software search in Grids and Clouds. Nowadays, software retrieval in Grids and Clouds has become an important element in software development,

[2]http://grid.ucy.ac.cy/minersoft.

where software components and libraries are used extensively in order to harness the capabilities of these infrastructures. The present work continues and improves upon the authors' preliminary efforts in Katsifodimos et al. [2009] and Pallis et al. [2009] focusing on developing a software search engine. In this article (i) we extend Minersoft architecture to support efficient software search not only on Grids but also on Cloud computing infrastructures. To be able to locate computing resources in different infrastructures, Minersoft uses a unique URI for each resource; the technical aspects are discussed in Section 3. (ii) We elaborate the SG and Minersoft's software retrieval algorithm to support queries for the metadata of files (e.g., version-specific queries); and (iii) we present an extended evaluation study of Minersoft's software retrieval engine. The main contributions of this work can be summarized as follows.

—We present the design, architecture, and implementation of the Minersoft system.
—We introduce the *Software Graph*, a typed, weighted graph that captures the metadata and content of software files found in a file system, along with structural and content associations between them (e.g., directory containment, library dependencies, documentation of software). We present a Software-Graph construction algorithm, which comprises techniques for discovering structural and content associations between software files that are installed on the file systems of large-scale distributed computing environments.
—We implement and apply this algorithm to retrieve and represent software files installed in three large-scale, production-quality distributed computing infrastructures: the EGEE Grid [Bird et al. 2009; EGEE 2010], Amazon EC2 Cloud [AMAZON 2009], and Rackspace [RACKSPACE 2009].
—We provide a test data set that can be used to evaluate software retrieval systems. This data-set collection contains software files installed in Grid and Cloud infrastructures and their relevance judgments for a set of keyword queries.[3]
—We conduct an experimental evaluation of Minersoft on real, large-scale Grid and Cloud testbeds, exploring performance issues of the proposed scheme and using the aforementioned data set. We also demonstrate the effectiveness of the Software Graph as a structure for annotating software files with descriptive keywords and for supporting full-text search for software. Results show that Minersoft achieves high search efficiency.

The contributions of this work can be used from researchers and software practitioners to harness the capabilities of Grids and Clouds, locating software files suitable to their needs and encouraging software investigation, software reuse, clone detection, and computational resource selection.

The remainder of this article is organized as follows. Section 2 presents an overview of related work. In Section 3, we introduce the concepts of the Grid and Cloud model and describe infrastructures indexed by Minersoft. In Section 4, we provide the definitions for software files, software package, and Software Graph, and describe the proposed algorithms to create the annotated Software Graph. Section 5 describes the Minersoft architecture, implementation and its performance evaluation. Section 6 presents the software retrieval evaluation. We conclude in Section 7, outlining the contributions that this work makes to the Internet technology.

2. RELATED WORK

A number of research efforts [Linstead et al. 2009; Zaremski and Wing 1997] have investigated the problem of software-component retrieval in the context of language-specific software repositories and CASE tools (a survey of recent work can be found

[3]We have made this data-set available through the Web at http://www.grid.ucy.ac.cy/minersoft/.

Table I. Existing Tools for Software Retrieval

Approach	Searching paradigm	Corpus	Software files Retrieval			
			Binaries/ scripts	Source code/ libraries	Software-description documents	Binary libraries
GURU	keyword-based	Repository		✓		
Suade	software elements (fields & methods)	Repository		✓		
Exemplar	APIs	Repository	✓	✓		✓
Marakatu	Keyword-based	Repository		✓		
SEC+	Keyword-based	Repository		✓		
Wumpus	Keyword-based	Repository			✓	
Extreme Harvesting	Keyword-based	Web		✓		
SPARS-J	Keyword-based	Internet repositories		✓		
Sourcerer	Keyword-based	Internet repositories		✓		
Koders	Keyword-based	Internet repositories		✓		
Google Code	Keyword-based	Web		✓		
Krugle	Keyword-based	Web		✓		
Portfolio	Keyword-based	Repository		✓		
Minersoft	Keyword-based	Grid, Cloud, Cluster, Repository	✓	✓	✓	✓

in Lucrédio et al. [2004]). One of the key distinguishing characteristics of these approaches is the corpus upon which the search is conducted. Table I presents a list of tools for software retrieval and summarizes their key attributes.

2.1. Searching in Software Repositories

Perhaps the first effort to establish a keyword-based paradigm for the retrieval of source code residing inside software repositories was presented in Maarek et al. [1991], with the GURU system. GURU adopted the cosine similarity metric to match queries with documented software files, and used probabilistic modeling (quantity of information) to map documented software to terms. GURU provided results that included full and partial matches. Like GURU, Exemplar [Grechanik et al. 2010] retrieves relevant software components from different repositories by linking API help pages whose information is of higher quality than ad hoc descriptions of components. Similar approaches have been followed by Antoniol et al. [2002], Marcus and Maletic [2003], and Lucia et al. [2007]. All these works exploit source-code comments and documentation files, representing them as term-vectors and using similarity metrics from information retrieval (IR) to identify associations between software files. Results showed that such schemes work well in practice and are able to discover links between documentation files and source codes.

The use of folksonomy concepts has been investigated in the context of the Maracatu system [Vanderlei and et al. 2007]. Folksonomy is a cooperative classification scheme where users assign keywords (called tags) to software files. A drawback of this approach is that it requires user intervention to manually tag software. Finally, the use of ontologies is proposed in Khemakhem et al. [2007]; however, this work provides little evidence on the applicability and effectiveness of its approach. SEC+ is a more recent search engine paradigm for discovering software components [Khemakhem et al. 2010]; an ontology is used to identify the software components. Consequently, the efficiency of SEC+ is affected by how well the ontology describes the software packages. However, the development of a well-defined ontology for software packages remains an open issue, since there is a large number of “rules” in order to define which software resources constitute a software package.

The search for software can also benefit from extended file systems that capture file-related metadata and/or semantics, such as the Semantic File System [Gifford et al. 1991], the Linking File System (LiFS) [Ames et al. 2005], or from file systems that provide extensions to support search through facets [Koren et al. 2007], contextualization [Soules and Ganger 2005], desktop search (e.g., Confluence [Gyllstrom et al. 2007], and Wumpus [Yeung et al. 2007]). In the same context, SmartScan [Liu et al. 2010] is a metadata crawl tool that exploits patterns in metadata changes to improve the efficiency of support for file-system-wide metadata querying (e.g., like "which directory subtrees consume the most space?").

Although Minersoft could easily take advantage of the above file systems offering advanced metadata support, in our current design we assume that we can exploit only the metadata available in traditional Unix and Linux systems, which are common in most Grid and Cloud infrastructures.

2.2. Software Retrieval on the Web

The Web has been used as a platform for storing and publishing software repositories. Consequently, a number of research efforts and tools have focused on supporting topical Web searches that target software files [Susan et al. 2010]. Hummel and Atkinson [2004], described an approach for harvesting software components from the Web. The basic idea is to use the Web as the underlying repository and to utilize standard search engines, such as Google, as the means for discovering appropriate software assets. Other approaches have developed software-specific crawlers that crawl CVS and software repositories published through the Web, in order to build source-code search engines (e.g., SPARS-J [Matsushita 2005] and Sourcerer [Linstead et al. 2009]). At the time of this writing, four such software search engines are in operation on the Web: *Google Code*,[4] *Krugle*,[5] *Koders*[6] and Portfolio.[7] Google Code Search is for developers interested in Google-related open-source development; its users can search for open source-code and for Google services that support public APIs. Koders and Krugle support searching for open source code published on the Web. They enable software developers to easily search and browse source code in thousands of projects posted at hundreds of open source repositories. Portfolio [McMillan et al. 2011] is a code search system that retrieves highly relevant functions and projects from a large archive of C/C++ source code repositories.

Software applications can also be accessed and executed via the Web based on Web services. To this end, Web services infrastructure rely primarily on WSDL (Web services description language); SOAP (simple object access protocol); and UDDI (universal description and discovery interface). Currently, UDDI is the main standard for Web service discovery. However, it does not account for semantic information. To address this, several approaches have been proposed to support semantic search for Web services (METEOR-S,OWL-S, WOOGLE) [Brogi et al. 2008; Li et al. 2009; Toch et al. 2007]. The concept of Web services discovery is different from searching software files in Grids and Clouds, since Web services provide to end-users an invocation model that is different from Grid infrastructures or Cloud IaaS services. Specifically, the invocation model of Web services relies on standards like UDDI, WSDL, SOAP, and provide special-purpose services through these standards. On the other hand, software files installed on a Cloud/Grid infrastructure lie in the file-systems of the computational

[4]Google Code search engine: http://code.google.com/.
[5]Krugle: http://www.krugle.com.
[6]Koders search engine: http://www.koders.com.
[7]Portfolio:http://www.searchportfolio.net/.

resources, are unstructured, and can be part of a software project, or even be executed (binary files) in the target computational resource (e.g., a Cloud server).

2.3. Minersoft vs. Existing Approaches

Although we are not aware of any work that provides keyword-based searching for software files on large-scale Grid/Cloud infrastructures, our work overlaps with prior work on software retrieval [Antoniol et al. 2002; Grechanik et al. 2010; Vanderlei and et al. 2007]. These works mostly focus on developing schemes that facilitate the retrieval of software source files using the keyword-based paradigm. Minersoft differs from these works in a number of key aspects.

—Minersoft supports searching for software installed in the file systems of distributed computing infrastructures (Grids, Clouds, clusters), as well as in software repositories. The technical aspects are discussed in the next section.
—Minersoft supports searching not only for source codes but also for executables and libraries stored in binary format, and metadata (software versions, timestamps, permissions);
—Minersoft does not presume that file systems maintain metadata (tags, etc.) to support software search; instead, the Minersoft harvester generates such metadata automatically by invoking standard file-system utilities and tools and by exploiting the hierarchical organization of file systems;
—Minersoft introduces the concept of the *Software Graph*, a weighted, typed graph that represents software files and their associations in a single data structure amenable to further processing.
—Minersoft addresses a number of additional implementation challenges that are particular to Grid and Cloud infrastructures.
  (i) Software management is a decentralized activity; different machines in Grids and Clouds may fall under different policies in software installation, directory naming, and so on. Also, software entities on such infrastructures often come in a wide variety of packaging configurations and formats. Therefore, solutions that are language-specific or tailored to some specific software-component architecture are not applicable in the Minersoft context.
  (ii) Harvesting the software files found in Grid and Cloud infrastructures is a computationally demanding task. Therefore, this task should be distributed to the computational resources available in the infrastructure, achieving load balancing and reducing data communication overhead between the search engine and Grid or Cloud sites.
  (iii) The users of a distributed computing infrastructure do not have direct access to its servers. Therefore, a software harvester has to be either part of middleware services (something that would require the intervention to the middleware) or to be submitted for execution as a normal job through the middleware. In the Minersoft architecture and implementation, we adopt the nonintrusive approach, which facilitates the deployment of the system on different Grids and Clouds.

3. GRIDS AND CLOUDS

In this section we provide a brief overview of Grid and Cloud infrastructures where we perform software retrieval.

3.1. Grid Infrastructures

Grid infrastructures typically comprise large numbers of heterogeneous resources (computing, storage), distributed across multiple administrative domains (sites) and interconnected through an open network. Coordinated sharing of resources that span

multiple sites is made possible in the context of virtual organizations [Foster et al. 2001]. A virtual organization (VO) provides its members with access to a set of central middleware services, such as resource discovery and job submission. Through those services, the VO offers some level of resource virtualization, exposing only high-level functionality to Grid application programmers and end-users. The conceptual architecture of a Grid system consists of four layers: fabric, core middleware, user-level middleware, and Grid applications. The Grid fabric layer consists of the actual hardware and local operating system resources. The core Grid middleware provides services that abstract the complexity and heterogeneity of the fabric layer (i.e., remote process management, storage access, information registration and discovery). The user-level Grid middleware utilizes the interfaces provided by the low-level middleware so as to provide higher abstractions and services, such as resource and storage managers, schedulers, and application environments. Finally, the Grid applications layer utilizes the services provided by user-level middleware so as to offer engineering and scientific applications and software toolkits to Grid users.

Envisioning the existence of a software search engine for the EGEE infrastructure, the software developer would submit a query for the linear algebra package using the keyword "lapack" (a software for mathematical computation that stands for Linear Algebra PACKage). In response to this query, the engine would search inside 300 Grid sites (several of which host well over 1 million software-related files) and return software that matches the query's keywords, along with additional textual descriptions and a listing of computational resources where this software was located.

3.2. Cloud Computing

Cloud computing describes a recent trend in information technology (IT) that moves computing and data away from desktop and portable PCs into large data centers that provide on-demand services through the Internet on a "pay as you go" basis [Armbrust et al. 2010]. The computing nodes of a Cloud, called *Cloud virtual servers*, are managed by a single administrative domain. Typically, the service offerings of Cloud service providers (*Cloud providers*) comprise access to computing and storage capacity to software platforms for developing and deploying applications and to actual applications. User access to Cloud services is achieved via SSH calls, Web services, or batch systems.

The main technical underpinnings of Cloud computing infrastructures and services include elasticity, virtualization, service-oriented software, Grid computing technologies, management of large facilities, power efficiency, and so on Cloud service consumers purchase Cloud services in the form of infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), or software-as-a-service (SaaS) and sell value-added services (e.g., utility services) to end-users. Within the Cloud, the laws of probability give the service provider great leverage through statistical multiplexing of workloads and easier management, since a single software installation can cover the needs of many users.

Envisioning the existence of a software search engine for Cloud infrastructure, the user would submit a query for identifying the Cloud virtual servers that have been Hadoop-installed using the keywords "hadoop map-reduce." In response to this query, the engine would search inside virtual computational servers (e.g., 8,500 Amazon Machine Images, where each AMI comprises over 14,000 files) and return a list of software that matches the query's keywords, along with additional textual descriptions and a listing of the Cloud virtual servers where this software was located.

3.3. Examples of Minersoft-Crawlable Infrastructures

Minersoft's harvester crawls and locates software files lying on the filesystems of Grid and Cloud infrastructures that provide services in an infrastructure-as-a-service manner. Examples of such infrastructures are (i) Cloud infrastructures such as: Rackspace

Cloud, Amazon EC2, Linode, etc; (ii) grid infrastructures that include (but are not limited to) EGEE/EGI, Grid5000, TeraGrid et al.; (iii) large-scale Enterprise Grids/Clouds based on different batch systems (e.g. Condor, Blah). All the aforementioned infrastructures provide computing resources whose filesystem is accessible either (i) directly, with a remote login access method (e.g., ssh, rsh) or (ii) through a job submission protocol (e.g., Condor, gLite WMS, Blah). For example, in Amazon EC2, the computing resources' access protocol is ssh. On the other hand, in EGEE/EGI, the computing resources are accessible through the gLite WMS submission protocol. In both cases, Minersoft gains access to the file systems of the computing resources. To hide the access protocols that have to be used in each of the different infrastructures, Minersoft harvester builds an abstraction layer on top of them. To be able to locate computing resources in different infrastructures, Minersoft uses a unique URI for each resource. The URI's first part denotes the access protocol, and second denotes the name of the computing resource that needs to be accessed. For example, the URI "egee://ce101.grid.ucy.ac.cy" refers to the computing node ce101.grid.ucy.ac.cy, which can be accessed by the EGEE's gLite WMS submission protocol.

4. SOFTWARE GRAPH CONSTRUCTION AND INDEXING

A key responsibility of the Minersoft harvester is to construct a Software Graph (SG) that represents the software installed on each computing site. In this section, we provide a formal definition of the Software Graph and present the SG construction algorithm.

4.1. Definitions

Definition 4.1. A *Software file* is a file that is installed on a machine and belongs to one of the following categories: (i) *executable* (binary or script); (ii) software *library*; (iii) *source code* written in some programming language; (iv) *configuration file* required for the compilation and/or installation of code (e.g., makefiles); (v) unstructured or semi-structured *software-description document* that provides human-readable information about software, its installation, operation, and maintenance (manuals, readme files, etc.).

Definition 4.2. A *Software package* is a collection of associated software files that are installed and configured in order to function as a single entity that can accomplish a computation or a group of related computations.

The identification of a software file and its categorization into one of the categories mentioned above can be done heuristically by human experts (system administrators, software engineers, advanced users). Human experts can also recognize the associations that establish the grouping of software files into a software package. Normally, these associations are not represented through some common, explicit metadata format maintained in the file system. Instead, they are expressed implicitly by location and naming conventions or hidden inside configuration files (e.g., makefiles, software libraries). Therefore, the automation of software-file classification and grouping is a nontrivial task. To represent the software files found in a file system and the associations between them, we introduce the concept of the *Software Graph*.

Definition 4.3. The *Software Graph* is a weighted, metadata-rich, typed graph $G(V, E)$. The vertex-set V of the graph comprises vertices that represent software files found inside the file system of a computer (file vertices) and vertices that represent directories of the file system (directory vertices). The edges E of the graph represent structural and content associations between graph vertices.

Software Resouce #44243

METADATA

Name: mysql

Path: /usr/lib/libmysql.so.5.0

Type: library

Version: 5.0

Timestamp: 12 jul 2009 15:21:18 UTC

Permissions: ***owner***=root ***group***=root mask=655

Sites: egee://ce101.grid.ucy.ac.cy, rackspacecloud://ubuntu-9.04

FIELDS

Weight	#Terms	Field
0.5	18054	**Fulltext**: fetch column option field charset select usage outfile interactive connection directory serv...
0.1	3	**Path**: usr lib libmysql.so.5.0
0.2	4645	**Man Page**: mysql command line tool interactive query result set output format compress database ...
0.05	10107	**Readme**: release mysql dual license database gpl website documentation features bugs develop co...
0.05	1059	**Readme**: red hat install mysql server enterprise libtool building pstack innodb rpm download myis...
0.1	445	**Other**: recommend install mysql server instructions debian package copyright client exception obey g...

Fig. 1. Software file metadata.

The Software Graph is "typed" because its vertices and edges are assigned to different types (classes). Each vertex v of the Software Graph $G(V, E)$ is annotated with the following metadata attributes that describe its content and context:

—*name*(v) is the normalized name of the software file represented by v;
—*type*(v) denotes the type of v; a vertex can be classified into one of a finite number of types (more details on this are given in the next sections);
—*site*(v) denotes the computing site where file v is located;
—*path*(v) is a string that represents the path-name that a software file v has in the file system of *site*(v);
—*version*(v) denotes the version of v, where applicable;
—*timestamp*(v) denotes the last time that the file v was accessed; and
—*permission*(v) denotes the access permissions of file v.

In addition to the attributes above, each vertex v of the Software Graph $G(V, E)$ is annotated with a set of *fields*: $field_l(v), l = 1, \ldots, f_v$ containing textual content (terms) associated to v. In particular: (i) $field_1(v)$ stores the terms extracted from v's own contents; (ii) $field_2(v)$ stores terms extracted from v's file-system path; (iii) $field_3(v), \ldots, field_{f_v}(v)$ store terms extracted from software documentation files associated to v. The number of these files $(f_v - 2)$ depends on the file-system organization of *site*(v) and on the algorithm that discovers such associations (see the subsequent section). Each $field_l(v)$ is assigned a weight g_l, such that $\sum_{l=1}^{f_v} g_l = 1$. Field weights are introduced to support weighted field scoring in the resolution of end-user queries. Figure 1 presents excerpts from the metadata fields produced by the Minersoft algorithm for a particular software library.

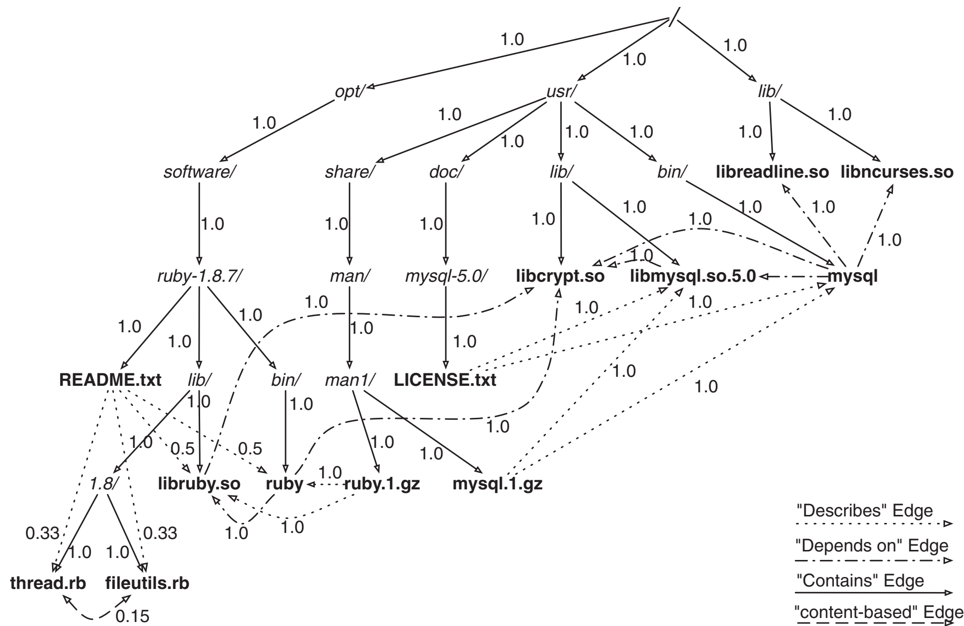

Fig. 2. Example of a software graph fragment.

The edges of the Software Graph represent structural and content associations between SG vertices. *Structural associations* correspond to "structural" properties among software-related files and file-system directories. These relationships are derived from file-system structure (e.g., directory containment); library dependencies, various other conventions (e.g., location and naming of documentation files); and from configuration files that describe the structuring of software packages (RPMs, tar files, etc.). *Content associations* correspond to relationships derived by text similarity between software files.

Each edge e of the graph has two attributes: $e = (type, w)$, where $type = \{structural, content\}$ denotes the type of association represented by e, and w is a real-valued weight ($0 < w \leq 1$) expressing the degree of correlation between the edge's vertices. The values of w are assigned during the structural dependency mining phase (see Section 4.3). Structural-based edges are further categorized as *describes*, *dependsOn*, and *contains*. *Contains* represents the relationship between a directory and its associated files; *describes* represents the relationship between software description documents and associated software files; *dependsOn* represents the dynamic dependencies that exist between binary executables and libraries. Content-based edges represent associations between vertices corresponding to source-code files with a high content similarity. Figure 2 presents a graphical representation of a sub-graph extracted from a SG produced by Minersoft.

4.2. Minersoft Algorithm

FST construction. Initially, Minersoft scans the file system of a site and creates a *file-system tree* (FST) data structure. The internal vertices of the tree correspond to directories of the file system; its leaves correspond to files. Edges represent containment relationships between directories and subdirectories or files. All FST edges are assigned a weight equal to one. During the scan, Minersoft ignores a *stop list* of files and directories that do not contain information of interest to software search (e.g., `/tmp`, `/proc`).

Labeling and pruning. Names and pathnames play an important role in file classification and in the discovery of associations between files. Consider the file /usr/lib/libruby.so.1.8.7. The fullpath of the file consists of two parts: the path of the directory in which the file is located (/usr/lib/) and the filename (libruby.so.1.8.7). The filename of the library consists of three parts: the prefix "lib," the main body "ruby," and the suffix ".1.8.7." Accordingly, Minersoft normalizes filenames and pathnames of FST vertices by identifying and removing suffixes and prefixes. The normalized names are stored as metadata annotations in the FST vertices. Subsequently, Minersoft applies a combination of system utilities and heuristics to classify each FST file-vertex into one of the following categories: binary executable, source code (e.g., Java, C++ scripts), library, software-description document (e.g., man-pages, readme files, html files), and irrelevant files. Initially, Minersoft tries to classify all files by their filenames, since the filename extensions usually denote the types of files. If this is not possible, it parses the output of the file command for all the unclassified files. If both classification methods fail, then the file is classified as irrelevant. For example, (by convention) the extension ".java" denotes that it is a source code file written in Java language. However, in many cases the file does not denote its type. For instance, the file */bin/bash* (and most of the binary executables in a Linux machine's filesystem) does not have a filename extension. To classify this file, we use the Linux file command. Running the file command against the file */bin/bash* returns: */bin/bash: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked*. Minersoft parses the output of the file command looking for a set of keywords assigned to each file category (e.g., "LSB executable" is associated with *Linux executable files* and "LSB shared object" is associated with *Linux libraries*). According to the keywords that are returned, Minersoft classifies each file accordingly. Minersoft prunes all FST leaves found to be irrelevant to software search, also dropping all internal FST vertices that are left with no descendants. This step results to a pruned version of the FST that contains only software-related file-vertices and the corresponding directory-vertices.

Structural dependency mining. Subsequently, Minersoft searches for "structural" relationships between software-related files (leaves of the file-system tree). Discovered relationships are inserted as edges that connect leaves of the FST, transforming the tree into a graph. The discovery of structural relationships is based on generalized rules that can be categorized as follows: (i) File naming and location conventions of Linux are used that represent expert knowledge about file-system organization. For instance, a set of file naming conventions link *man-pages* to the corresponding binary executables; *Readme* and *html* files are linked to related software files. (ii) Dynamic dependencies that exist between binary executables and libraries. These dependencies are represented in the headers of libraries and executables. Also, libraries may depend on other libraries. For instance, the library /usr/lib/libruby.1.8.7 depends on /lib/libpthread.so. To support the identification of such dependencies, Minersoft uses ldd, a Linux command that detects the dynamic dependencies of Linux executables/libraries. (iii) Dependencies extracted from package management tools. Specifically, Minersoft makes use of package information from the two major package managers, yum and apt, that contain information from RPM and DEB package files, respectively. Minersoft detects packages that are installed using those two package managers, and retrieves information about the files already found in the filesystem of a computational resource during crawling. If a software description file and a software resource (binary executable, source code or binary library) co-exist in the same package, Minersoft assumes that the software description file refers to the software resources. Then, Minersoft adds edges to the Software Graph connecting the software description file with each of the software resources. All aforementioned heuristic rules require no human intervention during the structural relationship mining process and have a low execution cost.

The structural dependency mining step produces the first version of the SG, which captures software files and their structural relationships. Subsequently, Minersoft seeks to enrich file-vertex annotation with additional metadata and to add more edges into the SG in order to better express content associations between software files.

Keyword scraping. In this step, Minersoft performs deep content analysis of each file-vertex of the SG in order to extract descriptive keywords. This is a resource-demanding computation that requires the transfer of all file contents from disk to memory to perform content parsing, stop-word elimination, stemming, and keyword extraction. Different keyword-scraping techniques are used for different types of files: for instance, in the case of source code, we extract keywords from the comments inside the source. Code mining techniques [McMillan et al. 2011] can be integrated in this step to extract valuable content from the source code files. Binary executable files and libraries contain strings that are used for printing out messages to the users, debugging information, logging, and so on. All this textual information can be used to extract useful features for these files. Hence, Minersoft parses the binary files byte-by-byte and captures the printable character sequences that are at least four characters long and are followed by an unprintable character. The extracted keywords are saved in the fields of the file-vertices of the SG.

Keyword flow. Software files (executables, libraries, source code) usually contain little or no free-text descriptions. Therefore, keyword scraping typically discovers very few keywords inside such files. To enrich the keyword sets of software-related file-vertices, Minersoft identifies edges that connect software-documentation file-vertices with software file-vertices, and copies selected keywords from the former into the fields of the latter.

Content association mining. Similar to Antoniol et al. [2002] and Marcus and Maletic [2003], we further improve the density of SG by calculating the cosine similarity between the SG vertices of source files. To implement this calculation, we represent each source-file vertex as a weighted term-vector derived from its source-code comments. To improve the performance of content association mining, we apply a feature extraction technique [Maarek et al. 1991] to estimate the quantity of information of individual terms and to disregard keywords of low value. Source codes that exhibit a high cosine-similarity value are joined through an edge that denotes the existence of a content relationship between them.

Inverted index construction. To support full-text search for software files, Minersoft creates an inverted index of software-related file-vertices of the SG. The inverted index has a set of terms, with each term being associated to a "posting" list of pointers to the software files containing the term. The terms are extracted from the fields and the metadata (e.g., software version) of SG vertices. The terms from all the fields of the vertices of the SG are tokenized, stemmed, and then passed to the full-text indexer. Metadata from SG vertices are extracted and passed to Lucene's full-text indexer [Lucene 2009]. Note that metadata is not stemmed nor tokenized because they have to remain intact. Every field of the vertices is stored in a Lucene field. More details on the implementation of full-text indexing are presented in Section 5.2.2.

In the subsequent sections, we provide more details on the algorithms for finding relationships between documentation and software-related files (Section 4.3), keyword extraction, and keyword flow (Section 4.4), software version extraction (Section 4.5), and content association mining between source-code files (Section 4.6).

4.3. Context Enrichment

During the structural dependency mining phase, Minersoft seeks to discover associations between documentation and software leaves of the file-system tree. These associations are represented as edges in the SG and contribute to the enrichment of the

context of software files. The discovery of such associations is relatively straightforward in the case of Unix/Javadoc online manuals, since, by convention, the normalized name of a file storing a manual is identical to the normalized file name of the corresponding executable. Minersoft can easily detect such a connection and insert an edge joining the associated leaves of the file-system tree. The association represented by this edge is considered strong and the edge is assigned a weight equal to 1.

In the case of *readme* files, however, the association between documentation and software is not obvious: software engineers do not follow a common, unambiguous convention when creating and placing readme files inside the directory of some software package. Therefore, we introduce a heuristic to identify the software files that are potentially described by a readme, and to calculate their degree of association. The key idea behind this heuristic is that a readme file describes its siblings in the file-system tree; if a sibling is a directory, then the readme-file's "influence" flows to the directory's descendants so that equidistant vertices receive the same amount of "influence," and vertices that are farther away receive a diminishing influence. If, for example, a readme-file leaf v^r has a vertex-set V^r of siblings in the file-system tree, then:

—Each *leaf* $v_i^r \in V^r$ receives an "influence" of 1 from vertex v^r.
—The descendant nodes f of each *internal node* $v_k^r \in V^r$ receive an "influence" of $1/(d-1)$, where d is the length of the FST path from v^r to f.

The association between software-file and readme-file vertices can be computed easily with a simple linear-time *breadth-first search* traversal of the FST, which keeps track of discovered readme files during the FST traversal. For each discovered association we insert a corresponding edge in the SG; the weight of the edge is equal to the association degree.

4.4. Content Enrichment

During the "keyword-flow" step, Minersoft enriches software-related vertices of the SG with keywords mined from associated documentation-related vertices. The keyword-flow algorithm is simple: for all software-related vertices v, we find all adjacent edges $e_d = (w, v)$ in the SG that are labeled "Describes". For each such edge e_d, we create an extra documentation field for v. Consequently, v ends up with an associated set of fields $field(v) = \{field_3^v, \ldots, field_{z_v}^v\}$, where $field_i^v, i = 3, \ldots, z_v$ correspond to keywords extracted from documentation vertices adjacent to v. Recall that $field_1^v$ corresponds to textual content extracted from v itself, and $field_2^v$ contains terms extracted from v's path-name.

Each field has a different degree of importance in terms of describing the content of the software file of v. Thus, we assign to each $field_i^v$ a different weight g_i, which is computed as follows: (i) For $i = 1$, namely for the field that includes the textual content extracted from v itself, we set $g_1 = \alpha_v$, where α_v is a constant $(0 < \alpha_v \leq 1)$. (ii) Similarly, for $i = 2$, namely for the field that includes the textual content extracted from pathname of file v itself, we set $g_2 = \beta_v$, where β_v is a constant $(0 < \beta_v \leq 1)$. (ii) For each remaining field of v $(i = 3 \ldots, z_v)$, g_i is set to α_v multiplied by the weight of the SG edge that introduced $field_i^v$ to v. The values of α_v and β_v are chosen so that $\sum_{i=1}^{z_v} g_i = 1$. Figure 1 presents the fields that the Minersoft algorithm produced for a particular software library. From this figure, we observe that the field, which includes the textual content of the file itself (g_1), has the largest value compared with the other fields. The intuition behind this is that the fields that explicitly describe (textual content field, pathname field) a file should take advantage from the other ones that describe it implicitly.

4.5. Software Version Extraction

Often, different versions of the same software library are found in a distributed computing node. Minersoft tries to extract the library version information and enrich software-file metadata accordingly. According to the file naming conventions of Linux, every library has a special name called the "soname". The *soname* has the prefix "lib," followed by the name of the library, the suffix ".so," a period, and a version number that is incremented whenever the library's interface changes (e.g., libruby.so.1). A fully-qualified soname includes as a prefix the directory that it is in (e.g., /usr/lib/libruby.so.1); on a working system a fully-qualified soname is simply a symbolic link to the shared library's "real name". The real name adds to the soname a period, a minor number, another period, and the release number. The last period and release number are optional. The minor number and release number support configuration control by letting the user know exactly what version(s) of the library are installed. For example, ruby's interpreter library fully-qualified name is */usr/lib/libruby.so.1*, which is a symbolic link pointing to the "real name" of the library, that is, */usr/lib/libruby.so.1.8.7*. The real name of the ruby library denotes that the version number of the library is 1, the minor version number is 8, and the release number is 7.

Minersoft extracts the library version information of a software file v (of type "library" only) by inspecting its filename and adds that information to a metadata field ($version(v)$), thus incorporating it in the indexes and making it available to user queries. However, it is quite common that developers and/or software packagers do not respect the aforementioned file-naming conventions. In that case the filename of a library does not necessarily denote its version. Due to the fact that library versioning accuracy is very important to users, Minersoft extracts only version information from libraries that follow file-naming conventions. To the best of our knowledge, there is no other method for extracting version information about libraries without using platform-specific software package managers (e.g.. rpm, deb). However, even in the case that a package is used to install a library, there are cases where the package includes more than one library. In that case the package version does not reflect the version of each individual library in the package and makes it an unreliable source of version information.

4.6. Content Association Mining

Minersoft enriches the SG with edges that capture content association between source-code files in order to support, later on, the automatic identification of software packages in the SG.

To this end, we represent each source file s as a weighted term-vector $\overrightarrow{V}(s)$ in the Vector Space Model (VSM). We estimate the similarity between any two source-code files s_i and s_j as the cosine similarity of their respective term-vectors: $\overrightarrow{V}(s_i) \cdot \overrightarrow{V}(s_j)$. If the similarity score is larger than a specific threshold (for our experiments we have set the $threshold \geq 0.05$), we add a new typed, weighted edge to the SG, connecting s_i to s_j. The weight w of the new edge equals the calculated similarity score.

The components of the term-vectors correspond to terms of our dictionary. These terms are derived from $field_1^v$ and their weights are calculated using a TF-IDF weighing scheme. To reduce the dimensionality of the vectors and noise, we apply a feature-selection technique in order to choose the most important terms among the keywords assigned to the content fields of source-code files. Feature selection is based on the *quantity of information* $Q(t)$ metric that a term t has within a corpus, and is defined by the following equation: $Q(t) = -log_2(P(t))$, where $P(t)$ is the observed probability of occurrence of term t inside a corpus [Maarek et al. 1991]. In our case, the corpus is the union of all content fields of SG vertices of source files. To estimate the probability $P(t)$, we measure the percentage of content fields of SG vertices of source files wherein

t appears; we do not count the frequency of appearance of t in a content field, as this would create noise.

Subsequently, we dropped terms with the quantity of information value less than a specific threshold. The reason is that low-Q terms would be useful for identifying different classes of vertices. In our case, however, we already know the class to which each vertex belongs (this corresponds to the type of the respective file). Therefore, by dropping terms that are frequent inside the source-code class, we maintain terms that can be useful for discriminating between files inside a source-code class. The threshold is set dynamically by the system. Specifically, the terms are sorted with respect to the quantity of information value. We remove the 5% of the total terms that have the lowest $Q(t)$. For our experiments we remove the terms where $Q(t) < 3.5$. The feature-selection process improves the SG structure (avoids having extra edges between files) and the system's performance (reduces the term vectors).

5. MINERSOFT ARCHITECTURE, IMPLEMENTATION AND PERFORMANCE EVALUATION

Creating an information retrieval system for software files that can cope with the scale of emerging distributed computing infrastructures (Grids and Clouds) presents several challenges. Fast crawling technology is required to gather the software files and keep them up to date, without disrupting the normal operation of the infrastructure. Storage space must be used efficiently to store indices and metadata. The indexing system must process hundreds of gigabytes of data efficiently. For the efficient implementation of Minersoft in a Grid and Cloud setting, we take advantage of various parallelization techniques in order to do the following.

—Distribute the Minersoft computation to Grid and Cloud resource providers, exploiting their computation and storage power to speed-up the file retrieval and indexing processes in order to reduce the communication exchange between the Minersoft system and resource-provider sites, and to achieve Minersoft's scalability in the context of an increasing number of resource-provider sites. Minersoft tasks are wrapped as jobs that are submitted for execution to Grid and Cloud systems.
—Avoid overloading resource-provider sites by applying load-balancing techniques when deploying Minersoft jobs.
—Improve the performance of Minersoft jobs by employing multithreading to overlap local computation with input/output (I/O).
—Adapt to the policies put in place by different Grid and Cloud computing resource providers regarding their limitations, such as the number of jobs that can be accepted by their queuing systems, the total time that each of these jobs is allowed to run on a given Grid site, and so on.

In the remainder of this section, we describe the architecture of Minersoft, which is presented in Figure 3, its implementation and its performance evaluation.

5.1. Overview

Minersoft has a MapReduce-like architecture [Dean and Ghemawat 2004]; the crawling and indexing is done by several distributed multithreaded crawler and indexer jobs, which run in parallel for improved performance and efficiency. Each crawler and indexer is assigned for processing a number of *splits*, with each split comprising a different set of files. Initially, a crawler job scans the filesystem of a grid site/Cloud server in order to identify all the filenames of the files and divides the total list of files into splits. The Minersoft system comprises a number of key software components (see Figure 3).

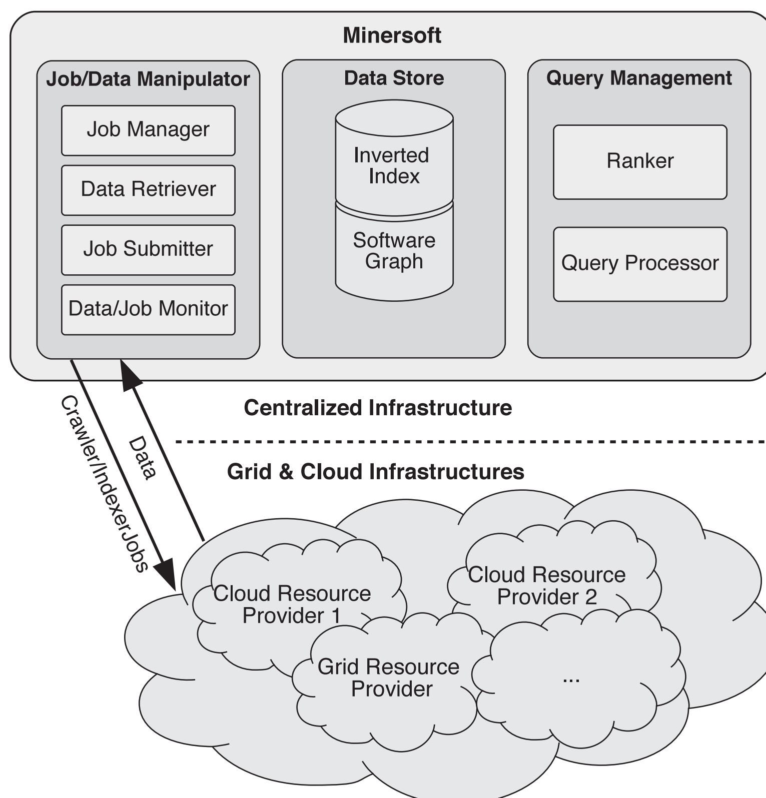

Fig. 3. Minersoft architecture.

(1) The *job manipulator* manages crawler and indexer jobs and their outputs. The manipulator comprises four modules: the job manager, the job submitter, the data retriever, and the monitor. The *job manager* undertakes the allocation of files into splits and the scheduling of crawler and indexer jobs. The *job submitter* handles the job submission and the *data retriever* retrieves the output of crawler and indexer jobs from the infrastructure. The *monitor* module maintains the overall supervision of Minersoft jobs. To this end, the *monitor* communicates with the job manager, the data-store, and the underlying infrastructure.
(2) The *datastore* module stores the resulting Software Graphs and the full-text inverted indexes.
(3) The *query engine* module is responsible for providing quality search results in response to user queries. The query engine module comprises the *query processor* and the *ranker*. The former receives search queries and matches them against the inverted indexes of Minersoft. The latter ranks query results in order to improve user-perceived accuracy and relevance of replies.

5.2. Distributed Crawling and Indexing

5.2.1. Minersoft Crawler. The crawler is a multithreaded program that performs FST construction, classification, and pruning, and structural dependency mining. To this end, the crawler scans the file-system of a computing site and constructs the FST, identifies software-related files and classifies them into the categories described earlier (binaries, libraries, documentation, etc.), drops irrelevant files, and applies the structural

dependency mining rules and algorithms described earlier. The crawler terminates after finishing with the processing of all splits assigned to it by the job manager.

The output of the crawler is the first version of the SG that corresponds to the site assigned to the crawler. This output is saved as a *metadata store file* comprising the file-id, name, type, path, size, and structural dependencies for all identified software files. The metadata store files are saved at the storage services associated with the computing site visited by the crawler, that is, at the local storage element of a grid site or at the storage service of a Cloud service.

5.2.2. Minersoft Indexer. The Minersoft indexer is a multithreaded program that reads the files captured in the *metadata store files* and creates full-text inverted indexes. To this end, the indexer performs keyword scraping, keyword flow, and content association mining in order to enrich the vertices of its assigned SG with keywords mined from associated documentation-related vertices. The Minersoft indexer can optionally stem terms before introducing them into its inverted indexes. The use of stemming has the advantage of increasing recall by retrieving terms that have the same roots but different endings. Also, stemming normally decreases the size of inverted indexes, since several terms are stemmed into one common root. Minersoft uses the Porter stemming algorithm [F 1997], which has been widely used in the literature.[8]

At the end of the indexing process, for each computing site (Grid site or Cloud virtual server) there is an inverted index containing a set of terms, with each term associated to a posting list of pointers to the software files containing the term. These terms are extracted from the metadata attributes and the fields of SG vertices. Specifically, after the keyword flow process, Minersoft creates an inverted index of the textual content of the vertices of SG. Each vertex is inserted as a document in the index by extracting the textual content of each of its fields and passing them to the indexer. Document field-weights inherit their value from the value of field-weight that was previously computed in the keyword scrapping and keyword flow steps.

5.2.3. Task Distribution and Load Balancing. The crawling and indexing of distributed computing infrastructures requires the retrieval and processing of large parts of the file systems of numerous sites. Therefore, this task needs to address various performance, reliability, and policy, issues. In the Grid context, a challenge for crawler and indexer jobs is to process all the software files residing within a Grid site without exceeding the time constraints imposed by site policies. Jobs that exceed the maximum allowed execution time are terminated by the site's batch system. The maximum wall-clock time usually ranges between 2 and 72 hours. Similarly, in the Cloud context, the challenge is to accomplish the crawling and indexing computation with the least possible cost spent on computing, storing, and transferring data.

To allow for a flexible management of its tasks, Minersoft decomposes the file system of each site into a number of *splits*, with the size of each split chosen so that the crawling can be distributed evenly and efficiently within the constraints of the underlying computing infrastructure. The splits are assigned to crawler/indexer jobs on a continuous basis: When a grid site or Cloud virtual server finishes with its assigned splits, the *monitor* informs the *job manager* in order to send more splits for processing. If a site becomes laggard, the *monitor* sends an alert message to the *job manager*, which cancels the running jobs and reschedules them to run when the workload of the site is reduced. Furthermore, if the batch system queue of a grid site is full and does not accept new jobs, the *monitor* sends an alert signal and the *job submitter* suspends the submission of new crawler/indexer jobs to that site until the batch system becomes ready to accept more.

[8]Porter stemming algorithm. http://tartarus.org/martin/PorterStemmer/.

Initially, Minersoft stores the data (indexes, SGs, etc.) on the storage facilities of the infrastructures that are being crawled. When the crawling process completes, Minersoft's *data retriever* module fetches the *metadata store files* from all machines and merges them into a *file index*. The *file index* comprises information about each software file and is temporally stored in the *datastore*. The *file index* will be used to identify the duplicate files during the indexing process; the duplication reduction policy is described in the following section. When the indexing has been completed, the *file index* is deleted. Then, the *data retriever* fetches the resulted inverted indexes and the individual SGs from all sites. Both the full-text inverted indexes and the SGs are stored in the *datastore*. This module is centralized and used for responding to users' queries.

5.2.4. Privacy and Accessibility Issues. Minersoft crawls the filesystem of a grid resource by running crawling threads submitted as regular computation jobs via a job submission service to that resource, and listing all readable files in its filesystem. The ownership of the files differs according to the local privacy policies of each grid site. For example, many grid sites make all the software files of a computational resource available to all the users of the infrastructure. However, there is software that is kept private because of licensing issues (e.g., matlab). That kind of software is not searchable, since the Minersoft harvester jobs have no permissions to access the software files and crawl them. In the case of Cloud virtual servers, Minersoft has full access to all the files that the virtual servers contained (as it instantiates Cloud virtual servers and logs in as root).

5.2.5. Duplication Reduction Policy. Typically, popular software applications and packages are installed on multiple sites of distributed computing infrastructures. Minersoft tries to identify duplicate files in order to avoid duplication of indexing effort and improve its performance. To this end, the system classifies as duplicates those files that reside on different nodes but have the same name, path and size. Subsequently, the *job manager* uses a *duplicate reduction policy*, which comprises the following steps.

(1) The *file index* is sorted in ascending order with respect to the count of the distributed computing nodes where a file is found.
(2) Files that do not have any duplicate are directly assigned to the corresponding distributed computing node.
(3) If a file belongs to more than one Grid site or Cloud virtual server, the file is assigned to the site with the minimum number of assigned files.

5.3. Query Engine

The query engine computes a ranked result list for each user query. The engine comprises a query processor, which parses query expressions and translates a query into a set of stemmed, lower-cased terms. The query processor also supports date and version parsing and numeric formatting, as well as wildcard and phrase queries.

Ranking is especially important when queries result in large numbers of "relevant" software files. The query engine comprises the *ranker* module, which uses the Lucene relevance ranking algorithms [Lucene 2009]. The default scoring algorithms of Lucene take into account factors such as the frequency of query-term appearance in the fields of a software file and in the fields of all the software files inside the software graph. Specifically, the score of a query for a software file correlates to the cosine-distance between software file and query vectors in the vector space model (VSM) of information retrieval. A software file whose vector is closer to the query vector in that model is scored higher.

Minersoft users are not expected to be familiar with the file-system structure and/or the structuring of software in the nodes of a distributed computing infrastructure.

Therefore, we assume that a user inquiry has the form of a keyword-based query Q, defined as follows.

Definition 5.1. Let k be a keyword phrase (one or more keywords), then a *keyword query* Q is defined by the grammar $Q ::= k \mid k$ metadata:Q. The first construct allows a primitive concept in a query to be described by a set of one or more keywords (e.g., "statistical analysis software"). The second construct allows one to describe a software file in terms of its metadata (e.g., ruby *version* : 1.8.∗).

Minersoft inherits the query language of the underlying full-text search library, Lucene. Therefore, Minersoft supports queries that use Boolean operators (AND, OR, NOT) to support queries like "apache AND Lucene" (return all documents that contain both terms) or "apache OR Lucene" (return software files containing either of the terms). The language also supports wildcard queries. For example "apach* lucene NOT jakarta" returns all files that contain a term that starts with "apach," contain the term "lucene," and do not contain the term "jakarta." Minersoft supports fuzzy searches based on the edit distance algorithms. A fuzzy search matches files containing terms that are similar to a term of the query. For example, a query asking for "tex" can be used to search for files containing "latex" or "texlive". It also provides proximity queries (searching for terms that reside close to each other in a file).

5.4. Implementation and Deployment

The crawler is written in Python. The Python code scripts are put in a tar file and copied on a storage service node before job submission starts. The tar file is downloaded and untarred to the target resource-provider site before the crawler execution starts. Consequently, the size of the jobs' input sandbox is reduced, thus job submission is accelerated.

The indexer is written in Java and Bash and uses Apache's open-source Lucene library [Lucene 2009], which provides high performance full-text indexing and search functionality. Minersoft stores its terms as unigrams in its indexes using the Lucene's standard analyzer class. Similarly to the crawlers, the indexer jobs are deployed for execution to the distributed computing nodes of the Grid and Cloud infrastructures.

The *job manager* distributes the crawling and indexing workload before job submission begins. This is done by creating splits for each computing site that Minersoft has to crawl and index. The input file for each split resides on a storage service node associated to the target computing site, and is registered to a file catalog. The split input is then downloaded from the storage service and used to start the processing of files. The split input is a text file containing the list of files that have to be crawled or indexed. After execution, the jobs upload their outputs on the storage service and register the output files to a file catalog. The logical file names and the directories containing them in the file catalog are properly named so that they implicitly state the split number and the site that they came from or going to.

5.4.1. Cloud Infrastructures. The implementation of the *job manager* and *monitor* relies upon the Ganga system [Brochu et al. 2009], which is used to create and submit jobs as well as to resubmit them in case of failure. We adopted Ganga in order to have full control of the jobs and their respective arguments and input files.

In a Cloud infrastructure, users have access to different virtual machine images (VMIs). A VMI includes a filesystem that contains an operating system and (optionally) extra software that the creator of the VMI has selected to include. A VMI can be regarded as the hard disk of a computer. A hard disk can be connected to a computer and the operating system that is contained in the hard disk can be booted. Following the same paradigm, a VMI is instantiated (or booted or spawned) using the computational

and storage infrastructure of a Cloud service provider. For a VMI to be crawled, it first has to be instantiated (booted/spawned) so that Minersoft is able to access it. Cloud providers provide scripts for initiating machines. After its instantiation, Minersoft is able to log into the instantiated Cloud server and crawl/index its filesystem contents. Thus, to be able to search the VMIs for installed software, Minersoft has to instantiate all VMIs of interest and crawl/index them. Once crawling and indexing are over, the VMI instance can be shutdown. The fact that VMI instantiation is immediate, stresses the large difference in Grid and Cloud infrastructures. For a Grid infrastructure to be crawled/indexed, the machines have to become free so that Minersoft sends jobs to them, whereas in a Cloud infrastructure, crawling and indexing processes can be done simultaneously and on demand. To that end, we have implemented a Ganga plug-in that supports Cloud server instantiation as well as submission of jobs over SSH.

5.4.2. Grid Infrastructures. In a Grid infrastructure, the machines that reside in a Grid site are already booted. Thus, there is no need for their instantiation. The gLite Java API is used for submitting jobs in a Grid infrastructure. The EGEE gLite middleware is the de-facto standard for Grid job submission on all EGEE-related Grids. Minersoft proceeds directly to crawling and indexing of their filesystem contents. The *monitor* (through Ganga scripts) monitors the status of jobs after their submission and keeps a list of resource-provider sites and their failure rates. If there are sites with a very high failure rates, the *monitor* eventually puts them in a black list and notifies the *job manager* to stop submitting jobs to them.

5.5. Crawling and Indexing Performance Evaluation

In this section, we present a brief performance evaluation study of the crawling and indexing tasks of Minersoft. Our objective is to show that Minersoft works sufficiently well on the large-scale Grid and Cloud testbeds of our study. More details about the performance evaluation of Minersoft in Grid infrastructures can be found in Katsifodimos et al. [2009].

5.5.1. Evaluation Testbed. To evaluate the effectiveness of Minersoft, we deployed and operated the system on real production-rate testbeds. In particular, we used the Following:

—Ten sites of the EGEE infrastructure [EGEE 2010]. The EGEE (Enabling Grid for E-sciencE) infrastructure is one of the largest Grid production services currently in operation, and its objective is to provide researchers in academia and industry with access to major computing resources, independent of their geographic locations. In total, the Grid testbed used for Minersoft includes files with a total size of 1,5 terabytes.
—Six virtual servers of the Amazon Elastic Computing Cloud [AMAZON 2009] and four virtual servers of the Rackspace Cloud [RACKSPACE 2009]. Amazon and Rackspace are commercial providers that support scalable deployment of applications through Web service interfaces. These services enable customers to create, configure, and deploy virtual machines tailored to their computing needs.

For both testbeds, we perform data cleaning so as to remove the noisy data (e.g., temp directories, log files, etc.). This is a typical process in large-scale IR systems. Tables II and III present an overview of the file-system characteristics of Grid sites and Cloud virtual servers after preprocessing.

Table II. Grid Testbed

Grid Site	# of Files	Size (GB)
ce01.kallisto.hellasgrid.gr	3.541.403 (59%)	247,910(68%)
ce301.intercol.edu	97.906 (2%)	3,539 (1%)
grid-ce.ii.edu.mk	194.556 (3%)	3,731 (1%)
paugrid1.pamukkale.edu.tr	132.645 (2%)	10,347 (3%)
ce01.grid.info.uvt.ro	270.445 (5%)	2,693 (1%)
grid-lab-ce.ii.edu.mk	109.286 (2%)	18,634 (5%)
ce01.mosigrid.utcluj.ro	70.419 (1%)	61,880 (17%)
ce101.grid.ucy.ac.cy	1.278.851 (21%)	6,375 (2%)
ce64.phy.bg.ac.yu	150.661 (3%)	6,787 (2%)
testbed001.grid.ici.ro	125.028 (2%)	4,756 (1%)
Total	5.971.200	366,652

Table III. Cloud Testbed

Cloud Virtual Server	# of Files	Size (GB)
Amazon1	35.362 (10%)	0,644 (6%)
Amazon2	29.980 (9%)	0,656 (6%)
Amazon3	25.906 (8%)	1,020 (9%)
Amazon4	31.049 (9%)	1,327 (12%)
Amazon5	83.256 (24%)	2,102 (19%)
Amazon6	33.522 (10%)	0,789 (7%)
Rackspace1	24.035 (7%)	1,109 (10%)
Rackspace2	17.266 (5%)	0,661 (6%)
Rackspace3	13.589 (4%)	0,537 (5%)
Rackspace4	47.156 (14%)	2,306 (21%)
Total	341.121	11,156

5.5.2. Metrics and Evaluation Experiments. To evaluate the performance of crawler and indexer jobs, we use the following metrics:

—runtime: the average wall-clock time that a crawler/indexer job spends on a site including processing and I/O; this metric measures the average elapsed time that Minersoft needs to process (crawl or index) a split;
—file rate: the number of files that Minersoft crawls/indexes per second on a site;
—data rate: the size of files in bytes that Minersoft crawls/indexes per second on a site.

In Minersoft, the crawling and indexing process is done by the nodes of computing infrastructure (Grid/Cloud), bringing the computation close to the data. Thus, the communication exchange between the Minersoft system and resource-provider sites is reduced. In our experiments, each crawler and indexer job was configured to run with five threads. We also ran experiments with different numbers of threads (e.g., 1, 5, 9, and 13) and concluded that 5 threads per crawler/indexer job provide a good trade-off between crawling/indexing performance and server workload. Smaller or larger numbers of threads per crawler/indexer job usually result in significantly higher runtimes, due to poor CPU utilization or I/O contention. I the interest of space, we do not present the results. In the experiments presented here, the maximum number of files in a split is set to 100,000.

Figures 4 and 5 depict the *per-job average run time* for crawling and indexing Grid sites and Cloud virtual servers. From these diagrams, we observe that the run time of crawler jobs vary significantly across different sites. This imbalance is due to several factors, including the hardware heterogeneity of the infrastructure, the dynamic workload conditions of shared sites, and the dependence of crawler processing on site-dependent aspects. For example, the crawler performs expensive "deep" processing of binary and library files to deduce their types and extract dependencies. This is not

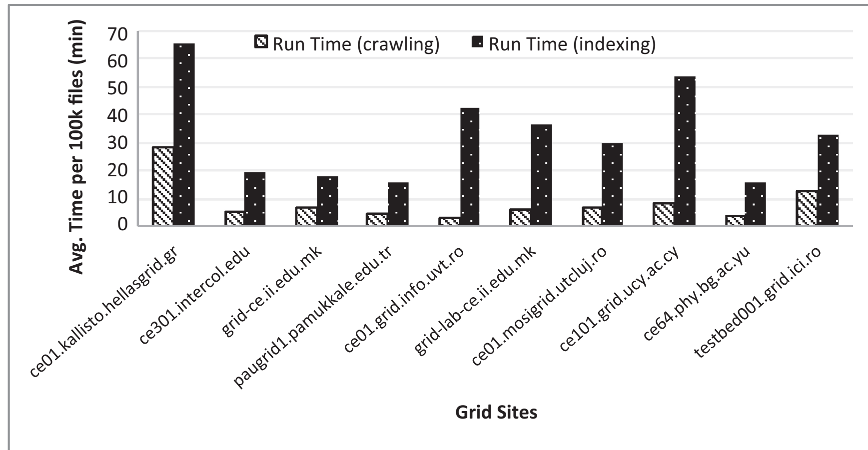

Fig. 4. Average times for jobs in Grid infrastructure.

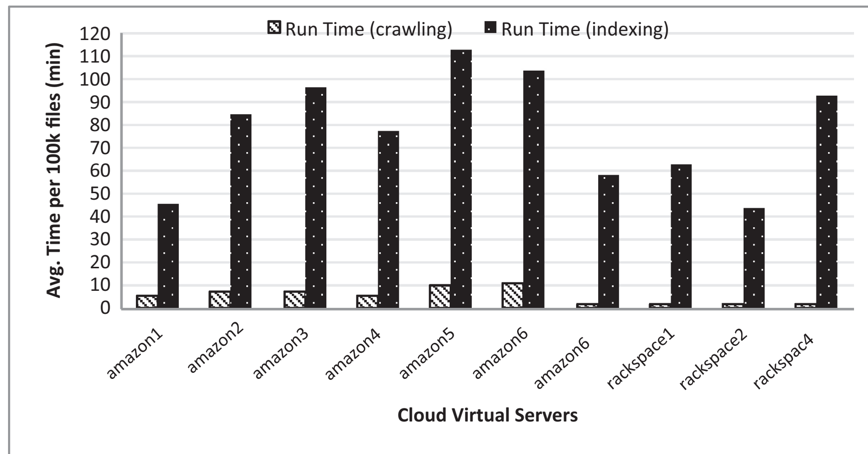

Fig. 5. Average times for jobs in Cloud infrastructure.

required for text files. Consequently, the percentage of binaries/libraries found in each site determines to some extent the corresponding crawling computation. Regarding the crawling throughput (file rate) of Minersoft, our results show that it ranges from 60 to just over 900 files per second. The data rate ranges from 4MB/sec to 23MB/sec.

As mentioned earlier, indexing entails "deep" parsing inside the content of all files. This requires substantial I/O operations and renders indexing a more time-consuming process than crawling. This observation is illustrated by Figures 4 and 5. Regarding the duplicate reduction, our experiments show that, indeed, a large degree of software-file duplication exists across the files of Grid and Cloud infrastructures, with about 11% of files belonging to more than one Grid site and 32% of files belonging to more than one Cloud Virtual Server.

Another interesting observation is that the duplicate reduction policy reduces significantly the total indexing time. This is achieved because the number of files that have been assigned in a split is reduced. In some cases, the number of splits is also reduced.

Compared with crawling, we observe that fewer splits are required for indexing than for crawling, since the irrelevant files have already been deleted.

We also studied the throughput achieved by the Minersoft indexer on different Grid sites and Cloud virtual servers. In the case of Grids, results showed that the performance of indexing is affected by the hardware (disk seek, CPU/memory performance), file types, and the workload of each site. In the case of Clouds, indexing is mainly affected by file types and the current site workload. Specifically, the indexing throughput (file rate) of Minersoft ranges from 16 to 100 files per second.

To sum up, our experiments resulted in the following empirical observations.

—Minersoft successfully crawled the Grid and Cloud infrastructure with crawling throughput from 60 to just over 900 files per second and indexing throughput from 16 to 100 files per second.
—The performance of indexing was improved by about 11% in the Grid infrastructure and 32% in Clouds by identifying duplicates and avoiding their processing.
—The crawling and indexing performance varies significantly across different sites, and is affected by the hardware (local disk, shared file system), file types found, and other workloads running simultaneously with Minersoft.

5.5.3. Open Issues for Research. The experimental results showed that Minersoft crawls and indexes Grid sites/Cloud virtual servers in an efficient way. Despite this fact, the crawling and indexing rates of Minersoft can be further improved by investigating novel and efficient policies in terms of the following.

—Determining the size of splits. As mentioned earlier, the files of each computing node are organized into a number of splits. The size of each split is chosen to ensure that the crawling and indexing can be distributed evenly and efficiently. Considering that in such an infrastructure the execution time and workload cannot be determined in advance using historical data, the size of splits should be adapted to the state of the computing nodes. To meet this challenge, the *monitor* should have a global view of the system's workload so as to dynamically rearrange the size of splits as well as schedule them to Grid sites/Cloud virtual servers.
—Determining the number of threads per crawler/indexer jobs. To improve the efficiency of crawling and indexing, Minersoft should enhance self-adaptive mechanisms in order to assign a sufficient number of threads to Grid/Cloud resources. Grid/Cloud monitoring systems provide information about resource utilization (CPU utilization, memory utilization, disk utilization, etc.), and network connectivity in computing nodes. Thus, if the current status of a computing node changes, the *job manipulator* should modify the number of threads per crawler/ indexer jobs in the node. The system should also support a failure mechanism that would inform the job manager regarding the cause for failure.
—Detecting duplicate software files. Files that are exact duplicates of each other can be identified by heuristics or check-summing techniques. In order to prevent duplicate files, crawler jobs need to periodically communicate and coordinate with each other. However, this communication may result in overhead. Can we minimize this communication overhead while maintaining the effectiveness of the crawler job? Cho and Garcia-Molina [2002] dealt with this problem in the context of the Web. Another issue is the identification of near-duplicate files. If we could successfully identify these files, we could improve the performance of indexing, since a percentage of files will be deleted. Manber [1994] has developed algorithms for near-duplicate detection to reduce storage in large-scale file systems.
—Determining politeness in Grid infrastructures. Minersoft jobs should not obstruct the normal operation of Grid sites. Minersoft should adhere to strict rate-limiting

Table IV. Files Categories in Grid Sites of EGEE Infrastructure

Grid Site	Binaries	Sources	Libraries	Docs	Irrelevant
ce01.kallisto.hellasgrid.gr	41.990 (1%)	1.407.701 (40%)	142.873 (4%)	1.672.246 (47%)	276.593 (8%)
ce301.intercol.edu	34.134 (35%)	8.972 (9%)	3.724 (4%)	23.536 (24%)	27.540 (28%)
grid-ce.ii.edu.mk	16.869 (9%)	69.915 (35%)	8.080 (4%)	61.469 (32%)	38.223 (20%)
paugrid1.pamukkale.edu.tr	7.383 (6%)	47.388 (26%)	7.935 (6%)	43.861 (32%)	26.078 (20%)
ce01.grid.info.uvt.ro	8.999 (3%)	40.442 (15%)	3.778 (1%)	42.652 (16%)	174.574 (66%)
grid-lab-ce.ii.edu.mk	7.703 (7%)	46.116 (42%)	2.983 (3%)	37.333 (34%)	15.151 (14%)
ce01.mosigrid.utcluj.ro	17.828 (25%)	12.475 (18%)	2.310 (3%)	18.091 (26%)	19.715 (28%)
ce101.grid.ucy.ac.cy	26.377 (2%)	433.115 (34%)	37.463 (3%)	672.211 (52%)	109.685 (9%)
ce64.phy.bg.ac.yu	6.047 (4%)	31.889 (21%)	7.672 (5%)	67.388 (45%)	37.665 (25%)
testbed001.grid.ici.ro	29.261 (23%)	22.961 (18%)	6.120 (5%)	28.239 (23%)	38.447 (31%)
Total	196.591 (3%)	2.120.974 (36%)	222.938 (4%)	2.667.026 (45%)	763.671 (12%)

Table V. Files Categories in Amazon and Rackspace Cloud Providers

Cloud Virtual Server	Binaries	Sources	Libraries	Docs	Irrelevant
Amazon1	2.442 (7%)	6.358 (18%)	1.000 (3%)	19.546 (55%)	6.016 (17%)
Amazon2	2.328 (7%)	7.351 (25%)	1.075 (4%)	12.474 (41%)	6.752 (23%)
Amazon3	2.344 (9%)	5.708 (22%)	1.259 (5%)	8.066 (31%)	8.529 (33%)
Amazon4	3.229 (10%)	3.534 (11%)	1.480 (5%)	11.925 (38%)	10.881 (36%)
Amazon5	5.098 (6%)	25.076 (30%)	1.458 (2%)	27.777 (33%)	23.847 (29%)
Amazon6	1.562 (5%)	11.348 (34%)	1.087 (3%)	13.216 (39%)	6.309 (19%)
Rackspace1	2.900 (12%)	2.859 (12%)	963 (4%)	4.488 (19%)	12.825 (53%)
Rackspace2	2.498 (14%)	1.880 (11%)	798 (5%)	4.134 (24%)	7.956 (46%)
Rackspace3	2.347 (17%)	2.047 (15%)	684 (5%)	3.491 (26%)	5.020 (37%)
Rackspace4	2.963 (6%)	10.425 (22%)	1.950 (4%)	19.997 (42%)	11.821 (26%)
Total	27.711 (8%)	76.586 (22%)	11.754 (3%)	125.114 (37%)	99.956 (30%)

policies when accessing poorly provisioned (in terms of workload) Grid sites. To address this issue, Minersoft should implement a flexible policy that would avoid running multiple crawler jobs to overloaded Grid sites.

6. SOFTWARE RETRIEVAL EVALUATION

In this section, we present an evaluation study that examines the effectiveness of the Minersoft search engine in software retrieval tasks. A difficulty in the evaluation of such systems is that there are no widely accepted data collections dedicated to benchmarking software search engines (like TREC and OHSUMED for text retrieval). Therefore, we use the following methodology in order to evaluate the performance of Minersoft:

—*Data collection and filtering*: Our dataset consists of the software installed in 10 Grid sites of the EGEE infrastructure (Table II) and 10 Cloud Virtual Servers from Amazon Elastic Computing and Rackspace Cloud providers (Table III). The files found by the crawlers to be irrelevant to software search are pruned from subsequent processing. Our experiments show that a large percentage of content in most Grid sites/Cloud virtual servers corresponds to software files. Specifically, on average 75% (75% of total file size) and 70% (77% of total file size) of total files that exist in Grid sites and Cloud virtual servers, respectively, are categorized as software files. These findings confirm the need to establish advanced software retrieval in Grid and Cloud infrastructures. Software-related files are further categorized according to their detected type. We present a synopsis of this categorization in Tables IV and V. As we can see from these tables, most software-related files in our testbed are documentation files (man-pages, readme files, html files) and source-code programs written in various programming or scripting languages.

—*Queries*: We use a collection of *220 queries*, which were either suggested by 50 participants (EGEE users, software engineers, programmers), or extracted from real user queries from the Sourcerer system [Linstead et al. 2009]. The dataset is available through the Web at http://www.grid.ucy.ac.cy/minersoft/. To further investigate the sensitivity of Minersoft, we classified the queries into three categories: *informational* (72 queries), *navigational* (117 queries), and *versional* (31 queries). Informational queries represent searches for software programs based on short descriptions of their functionality (e.g., log analyzer, rendering text, linear algebra package, Web services). Navigational queries represent searches for a specific file or library based on their names (e.g., jboss, rails ruby, mpich, autodock docking), whereas, versional queries (e.g., spread library version:2.* libncurses version:5.4, libxml version:2.*) represent searches for software with a particular version.

—*Relevance judgment*: A software file is considered relevant if it addresses the stated information need and not because it just happens to contain all query keywords. A software file returned by Minersoft in response to some query is given a binary classification as either relevant or nonrelevant with respect to the user information need for the query. Furthermore, each query result is rated according to three levels of user satisfaction: "not satisfied," "satisfied," and "very satisfied". These classifications have been made manually by two expert administrators and software engineers. The two judges evaluated relevance independently and their agreement was measured afterwards. It turned out that the experts agreed on the relevance status of 95% of the retrieved software (a total of 9,416 software files). The corresponding Kappa statistic, which factors out the expected (chance) agreement, is 0.91. Consequently, the interjudge agreement of relevance is high, and these classifications are referred to as the *gold standard* for our experiments (relevance judgements for all queries are available at http://www.grid.ucy.ac.cy/minersoft/).

6.1. Performance Measures

The effectiveness of Minersoft should be evaluated on the basis of how much it helps users achieve their software searches efficiently and effectively. In this context, we used the following performance measures.

—*Precision*@10 reports the fraction of software files ranked in the top 10 results that are labeled as relevant. The relevance of the retrieved results is determined by the *gold standard*. The results are ranked with respect to the ranking function of Lucene [Lucene 2009], which is based on the TF-IDF metric, and has been used extensively in the literature for the ranking of Web-search results [Bao et al. 2007; Cohen et al. 2008]. In our case, the TF-IDF calculation is based on the fields associated by Minersoft to software files. The maximum Precision@10 value that can be achieved is 1.

—MRR (Mean reciprocal rank) is the average of the reciprocal ranks over a set of queries, where the reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer. The maximum MRR value that can be achieved is 1.

—NDCG (normalized discounted cumulative gain) is a retrieval measure devised specifically for evaluating user satisfaction [Järvelin and Kekäläinen 2002]. For a given query q, the $top - K$ ranked results are examined in decreasing order of rank, and the NDCG value is computed as

$$NDCG_q = M_q \cdot \sum_{j=1}^{K=10} \frac{2^{r(j)} - 1}{\log_2(1 + j)},$$

where each $r(j)$ is an integer relevance label (0="not satisfied," 1="satisfied," and 2="very satisfied") of the result returned at position j and M_q is a normalization constant calculated so that a perfect ordering would obtain NDCG of 1.

—NCG (normalized cumulative gain) is the predecessor of NDCG, and its main difference is that it does not take into account the position of the results. For a given query q, the NCG is computed as

$$NCG_q = M_q \cdot \sum_{j=1}^{K=10} r(j).$$

A perfect ordering would obtain NCG of 1.

Cumulative gain measures (NDCG, NCG) and precision complement each other when evaluating the effectiveness of IR systems [Al-Maskari et al. 2007; Clarke and et al. 2008]. In our evaluation metrics we do not consider the recall metric (the percentage of the number of relevant results). Such a metric requires having full knowledge about all the relevant software files with respect to a query. However, such knowledge is not feasible in a large-scale networked environment.

6.2. Evaluation Scenarios

We calculate the metrics mentioned above for the 220 queries, with the searches being performed against a set of different indexes representing alternative optimizations implemented inside Minersoft. In particular, we estimate the effectiveness of Minersoft in the following scenarios.

—*Full-text*. Inverted index terms are extracted from the full-text content (field 1) of software-related files which are discovered in the examined testbed. Irrelevant files are discarded from the index and posting lists. *Full-text* is used as a baseline for our experiments.

—*Path-enhanced*. With *path-enhanced search* we study the impact of file-paths in software retrieval. The terms of the inverted index are extracted from the content and path fields of SG vertices (fields 1 and 2). Irrelevant files are discarded from the index and posting lists.

—*Context-enhanced*. The files are categorized into file categories. Irrelevant files and software-description documents are discarded from the index and posting lists. The terms of the inverted index are extracted from the content and path fields of SG vertices (fields 1 and 2) that correspond to software files *only* (libraries, sources, and binaries). *Context-enhanced* shows the influence of file categorization in the software retrieval process.

—*Doc-enhanced*. The terms of the inverted index are extracted from the content and path of SG vertices (fields 1 and 2) as well as from the fields of linked documentation files (i.e., man-pages and readme files, that is, fields 3 and beyond). *Doc-enhanced* highlights the effectiveness of enriching software files with keywords extracted from related documentation in the software retrieval process.

—*Text-file enhanced*. The terms of the inverted index are extracted from the content, the path, related documentation files, and other text files of SG vertices with the same normalized filename. Recall that Minersoft normalizes filenames and pathnames of SG vertices by identifying and removing suffixes and prefixes.

Table VI summarizes the evaluation scenarios.

Table VI. Evaluation Scenarios

	Keyword-sources for the index				Files included in the postings lists	
	File contents	Path name	Software descriptions	Text files	Software files	Rest of files
Full-text	√				√	√
Path-enhanced	√	√			√	√
Context-enhanced	√	√			√	
Doc-enhanced	√	√	√		√	
Text-file enhanced	√	√	√	√	√	

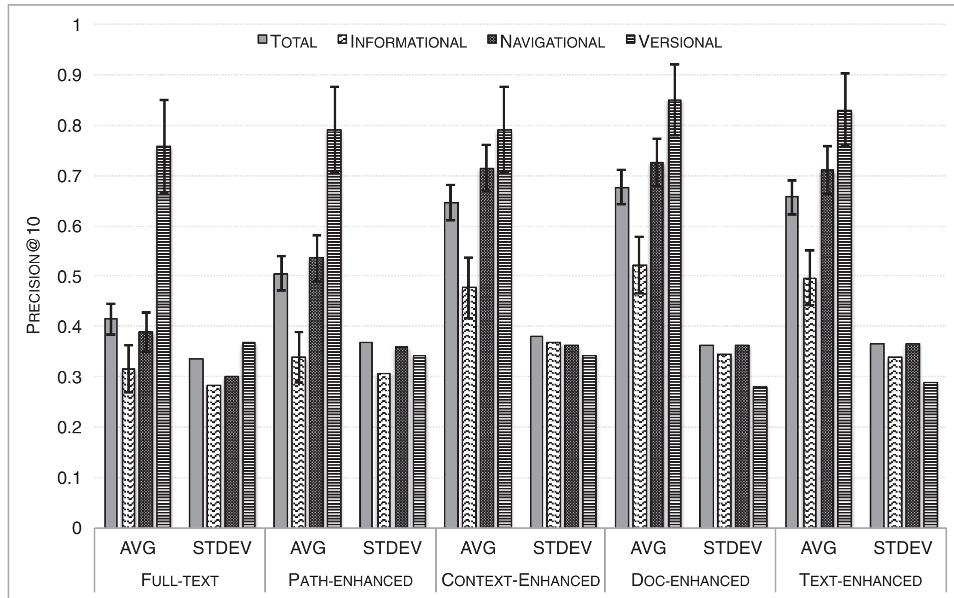

Fig. 6. Precision@10 results.

6.3. Evaluation

Each evaluation scenario corresponds to a step towards the construction of Minersoft's inverted index (Section 4). Through this step-by-step construction and evaluation of the inverted index, we illustrate the benefits of using the concept of fields to enrich the textual content found inside software files. In our evaluation experiments, we examine both average and median values of our metrics. Figures 6, 7, 8, and 9 present the average results of the examined approaches with respect to the query types for Precision@10, MRR, NDCG, and NCG. For completeness, we present the confidence intervals (confidence level 0.95) and standard deviation values. The confidence interval generates a lower and upper limit for the observed mean values. The interval estimate gives an indication of how much uncertainty there is in our estimate of the true mean. The narrower the interval, the more precise our estimate. Regarding their diversity, we observe that the data for *full-text*, *path-enhanced*, and *context-enhanced* approaches are spread out over a large range of values, whereas for the other examined approaches, the data tends to be close to the average values. The highest diversity is observed for versional queries. Median values are omitted for the sake of brevity, as they do not present significant differences from the averages.

The general observation is that Minersoft significantly improves the Precision@10, the MRR, and the examined cumulative gain measures compared with the baseline

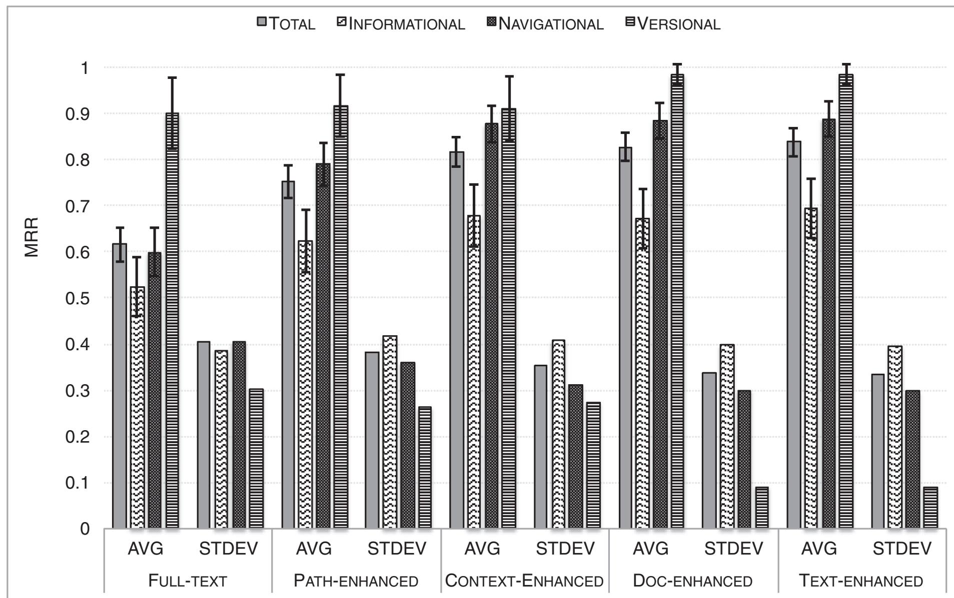

Fig. 7. MRR results.

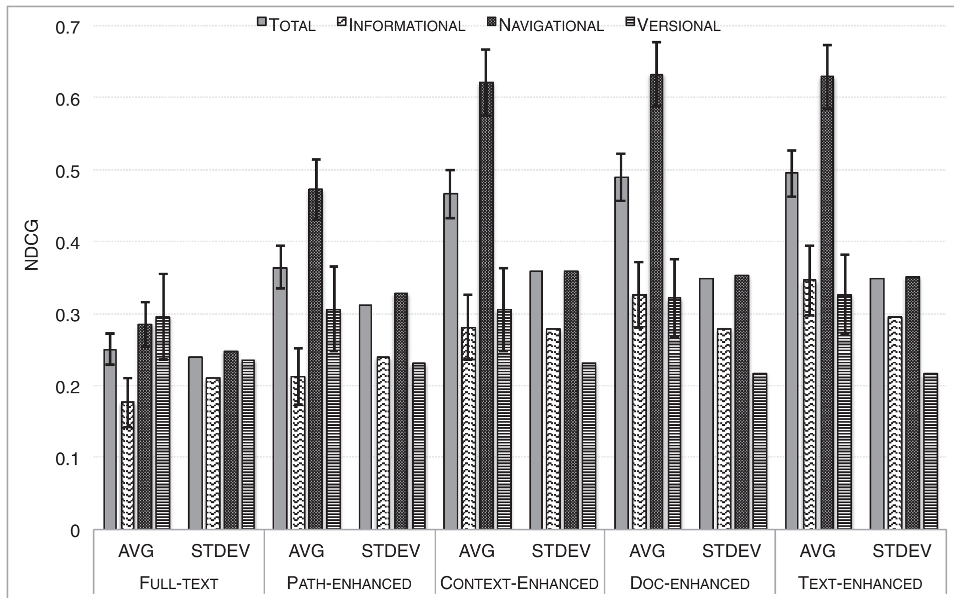

Fig. 8. NDCG results.

approach—*full-text*—for all types of queries. Specifically, Minersoft improves Precision@10 about 60%, MRR about 36%, and the cumulative gain measures (NDCG, NCG) about 97% for NDCG and 84% for NCG with respect to the baseline approach.

Regarding the intermediate steps for the construction of SG, the greatest improvement is observed with *context-enhanced*, explained by the fact that Minersoft has

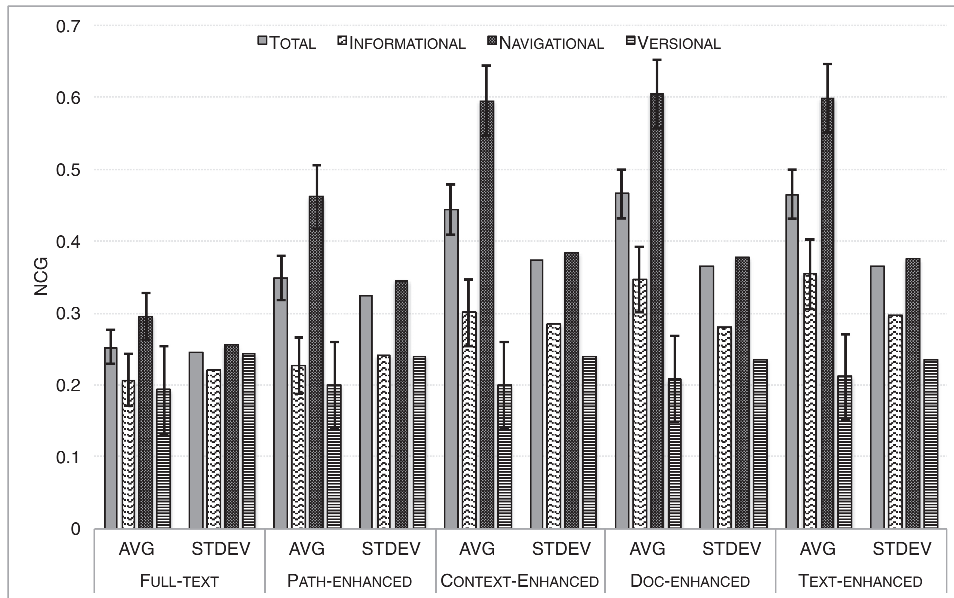

Fig. 9. NCG results.

classified the files into categories with respect to their types and that searching is done only in these categories of files. Our findings also show that taking into account the path name of software files results in improved search performance. In particular, under *path-enhanced* search we see an improvement in the *Precision*@10 and MRR of about 20% and the cumulative gain measures (NDCG, NCG) of about 40% with respect to the baseline approach. This is an indication that the paths of software files include descriptive keywords for software files.

The enrichment of software files with keywords from software description documents through the keyword flow process increases precision as well as user satisfaction. Specifically, *doc-enhanced* achieves higher *Precision*@10 and MRR (about 5% and 3% respectivelly) and higher cumulative gain measures (on average about 5% for NDCG and NCG) than the *context-enhanced* does. The improvements during the *doc-enhanced* step are affected by the number of the executables and software libraries that exist in the data set. Recall that this step enriches the executables and software libraries with extra keywords. Consequently, the larger the number of these types of files, the better the results that are obtained. Regarding types of queries, the major improvement is observed for versional and navigational queries. Another interesting observation is that Minersoft achieves high MRR for all types of queries. Specifically, for navigational and versional queries, the MRR is close to 0.9 and 1, respectively, whereas for informational queries, the MRR is about 0.7. The higher the position in which the result is found, the higher the score for MRR. Regarding informational queries, *Precision*@10 is about 0.5. Although half of the retrieved results are relevant, MRR is close to 1, which means that the first result is relevant to the user's information needs.

Regarding the *text-file enhanced*, we observe that this approach slightly improves search performance. Specifically, it achieves higher cumulative gain measures (on average about 2% for NDCG and NCG) than the *doc-enhanced* does while *Precision*@10 and MRR remain stable. Taking a deeper look at these results, we observe that *text-file enhanced* search improves results for informational queries, whereas for versional and

Table VII. Software Graph Statistics in Grid Sites

Grid Sites	V	E (total edges)	E_{SD}	E_{CA}
ce01.kallisto.hellasgrid.gr	3.264.810	1.291.884.123	9.540.597	1.282.343.526
ce301.intercol.edu	70.366	150.033	96.922	53.111
grid-ce.ii.edu.mk	156.333	1.659.309	322.495	1.336.814
paugrid1.pamukkale.edu.tr	106.567	1.195.702	223.529	972.173
ce01.grid.info.uvt.ro	95.871	1.465.779	199.537	1.266.242
grid-lab-ce.ii.edu.mk	94.135	179.127	158.733	20.394
ce01.mosigrid.utcluj.ro	50.704	158.451	86.249	72.202
ce101.grid.ucy.ac.cy	1.169.166	97.967.442	2.117.300	95.850.142
ce64.phy.bg.ac.yu	112.996	987.759	201.950	785.809
testbed001.grid.ici.ro	86.581	772.005	225.591	546.414
Total	5.207.529	1.396.419.730	13.172.903	1.383.246.827

navigational queries there is no improvement. This is explained by the fact that software developers use similar file-names in their software packages, and, consequently, the *text-file enhanced* does not improve the retrieval process. Another interesting observation is that although Minersoft performs very well (*Precision*@10 and MRR in *doc-enhanced* and *text-file enhanced*) in responding to highly selective queries (queries that require specific versions of software), the values of NCG and NDCG are lower. This is explained by the fact that the results returned are not always 10 (in most cases, 2 to 5 results are returned to the queries).

Finally, we study the effects of stemming in software files. Although stemming typically improves recall and reduces index size, its use leads to a decrease in precision and user satisfaction. What is the trade-off between stemming and no stemming in software retrieval? Our findings show that stemming deteriorates Minersoft's search performance about 4%. Regarding the intermediate steps for the construction of SG, the general observations of the evaluation scenarios are similar with those for no stemming. In terms of storage, stemming decreases the size of inverted indexes by about 6%. Specifically, the total indexing with and without stemming is 77.114GB and 81.916GB, respectively. For brevity, we do not present these figures.

To sum up, the results show that Minersoft is a powerful tool because it is highly effective for both types of queries. Specifically, our experiments conclude with the following empirical observations.

—Minersoft improves significantly different metrics that assess search performance. In particular, the Minersoft algorithm results in improvining of *Precision*@10 by about 60%, for MRR by about 20%, and for cumulative gain measures (NDCG, NCG) by more than 84% with respect to the baseline approach.
—Path-names of software-related files include descriptive keywords that carry semantic meaning with respect to the functionality of the software. Therefore, taking keywords extracted from path-names into account leads to improved software search.
—Stemming deteriorates about 4% software-search performance. On the other hand, it decreases the size of inverted indexes about 6%.
—File-name similarity is not a good indication of relevance between software and text files. Therefore, keywords extracted from text files to enrich software descriptions do not result in improved software-search performance.

6.4. Software Graph Statistics

Tables VII and VIII present the statistics for the resulting SGs. Recall that Minersoft harvester constructs a SG in each Grid site/virtual Cloud server. We do not present further analysis of the SGs since it is out of the scope of this work. Of course, a thorough study of the structure and evolution of SGs could lead to useful insights

Table VIII. Software Graph Statistics in Cloud Virtual Servers

Grid Sites	V	E (total edges)	E_{SD}	E_{CA}
Amazon1	29.346	79.636	49.368	30.268
Amazon2	23.228	74.180	41.107	33.073
Amazon3	17.377	62.032	34.919	27.113
Amazon4	20.168	61.688	45.811	15.877
Amazon5	59.409	583.198	102.534	480.664
Amazon6	27.213	96.513	47.484	49.029
Rackspace1	11.210	30.236	24.221	6.015
Rackspace2	9.310	23.959	20.714	3.245
Rackspace3	8.569	21.181	17.968	3.213
Rackspace4	35.335	157.574	68.185	89.389
Total	241.165	1.190.197	452.311	737.886

for the software engineering community. In the literature, a large number of dynamic large-scale networks have been extensively studied [Leskovec et al. 2007] in order to identify their latent characteristics.

Here, we briefly present the main characteristics of these graphs. Tables VII and VIII present the edges that have been added due to structural dependencies (E_{SD}) and content associations (E_{CA}). Based on these statistics, a general observation is that the SGs are not sparse. Specifically, we found that in the case of Grids most of them follow the relation $E = V^{\alpha}$, where $1.1 < \alpha < 1.36$, whereas in case of Clouds, most of Cloud virtual servers follow the relation $E = V^{\alpha}$, where $1.1 < \alpha < 1.31$; note that $\alpha = 2$ corresponds to an extremely dense graph where each node has on average a number of adjacent edges equal to a constant fraction of all SG nodes. Another interesting observation is that most of the edges are due to content associations. However, most of these edges have lower weights ($0, 05 \leq w < 0, 2$) than the edges that are due to structure-dependency associations.

7. CONCLUSION

In this article we present Minersoft—a tool that enables keyword-based searches for software installed on Grid/Cloud computing infrastructures. The software design of Minersoft enables the distribution of its crawling and indexing tasks to large-scale networked environments. The results of Minersoft harvesting are encoded in a weighted, typed graph, called the SG. The SG is used to automatically annotate the software files with keyword-rich metadata. Using a real testbed, we present the performance issues of crawling and indexing. Experimental results showed that SG represents in an efficient way the software files, improving the search of software packages in large-scale networked environments.

ACKNOWLEDGMENTS

The authors would like to thank EGEE users and administrators, Paris Ionas and Andreas Papadopoulos, who provided characteristic queries and judgements for evaluating Minersoft.

REFERENCES

AGRAWAL, R. ET AL. 2008. The Claremont report on database research. *SIGMOD Rec. 37*, 3, 9–19.

AL-MASKARI, A., SANDERSON, M., AND CLOUGH, P. 2007. The relationship between IR effectiveness measures and user satisfaction. In *Proceedings of SIGIR '07*. ACM, New York, 773–774.

AMAZON. 2009. Amazon Elastic Compute (EC2) Cloud. http://aws.amazon.com/ec2.

AMES, A., MALTZAHN, C., BOBB, N., MILLER, E. L., BRANDT, S. A., NEEMAN, A., HIATT, A., AND TUTEJA, D. 2005. Richer file system metadata using links and attributes. In *Proceedings of MSST '05.* IEEE, Los Alamitos, CA, 49–fi60.

ANDERSON, J. AND RAINIE, L. 2010. The future of Cloud computing. Tech. rep., Pew Internet and American Life Project, http://www.pewinternet.org/Reports/2010/The-future-of-cloud-computing.aspx.

ANTONIOL, G., CANFORA, G., CASAZZA, G., LUCIA, A. D., AND MERLO, E. 2002. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng. 28*, 10, 970–983.

ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., AND ZAHARIA, M. 2010. A view of cloud computing. *Comm. ACM 53*, 4, 50–58.

RACKSPACE. 2009. The Rackspace Cloud. http://www.mosso.com/rackspace.jsp.

BAO, S., XUE, G., WU, X., YU, Y., FEI, B., AND SU, Z. 2007. Optimizing web search using social annotations. In *Proceedings of WWW'07*. ACM, New York, 501–510.

BASS, L., CLEMENTS, P., KAZMAN, R., AND KLEIN, M. 2008. Evaluating the software architecture competence of organizations. In *Proceedings of WICSA'08*. 249–252.

BIRD, I., JONES, B., AND KEE, K. F. 2009. The organization and management of grid infrastructures. *Computer 42*, 1, 36–46.

BROCHU, F., EGEDE, U., ELMSHEUSER, J., ET AL. 2009. Ganga: A tool for computational-task management and easy access to Grid resources. *Comput. Phys. Comm.* http://ganga.web.cern.ch/ganga/documents/index.php.

BROGI, A., CORFINI, S., AND POPESCU, R. 2008. Semantics-based composition-oriented discovery of web services. *ACM Trans. Internet Technol. 8*, 4, 1–39.

CHO, J. AND GARCIA-MOLINA, H. 2002. Parallel crawlers. In *Proceedings of the 11th International Conference on the World Wide Web* (*WWW'02*). ACM, New York, 124–135.

CLARKE, C. L. ET AL. 2008. Novelty and diversity in information retrieval evaluation. In *Proceedings of SIGIR'08.* ACM, New York, 659–666.

COHEN, S., DOMSHLAK, C., AND ZWERDLING, N. 2008. On ranking techniques for desktop search. *ACM Trans. Inf. Syst. 26*, 2, 1–24.

COYLE, M. AND SMYTH, B. 2007. Supporting intelligent web search. *ACM Trans. Internet Technol. 7*.

DEAN, J. AND GHEMAWAT, S. 2004. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation* (*OSDI'04*). Usenix Association, 137–150.

DIKAIAKOS, M. D., KATSAROS, D., MEHRA, P., PALLIS, G., AND VAKALI, A. 2009. Cloud computing: Distributed Internet computing for IT and scientific research . *IEEE Internet Comput. 13*, 5 , 10–13.

DIKAIAKOS, M. D., SAKELLARIOU, R., AND IOANNIDIS, Y. 2006. *Information Services for Largescale Grids*: A *Case for a Grid Search Engine*. American Scientific Publishers, 571–585.

EGEE. 2010. Enabling grids for E-sciencE (EGEE). http://www.eu-egee.org/.

FOSTER, I., KESSELMAN, C., AND TUECKE, S. 2001. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. Supercomput. Appl. 15*, 3, 200–222.

GABEL, M., JIANG, L., AND SU, Z. 2008. Scalable detection of semantic clones. In *Proceedings of the 30th International Conference on Software Engineering* (*ICSE '08*). ACM, New York, 321–330.

GIFFORD, D. K., JOUVELOT, P., SHELDON, M. A., AND O'TOOLE, J. W. 1991. Semantic file systems. In *Proceedings of SOSP'91.* ACM, New York, 16–25.

GRECHANIK, M., FU, C., XIE, Q., MCMILLAN, C., POSHYVANYK, D., AND CUMBY, C. 2010. A search engine for finding highly relevant applications. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering* (*ICSE '10*), ACM, New York, 475–484.

GYLLSTROM, K. A., SOULES, C., AND VEITCH, A. 2007. Confluence: Enhancing contextual desktop search. In *Proceedings of SIGIR'07.* ACM, New York, 717–718.

HUMMEL, O. AND ATKINSON, C. 2004. Extreme harvesting: Test driven discovery and reuse of software components. In *Proceedings of the IEEE International Conference on Information Reuse and Integration*. IEEE, Los Alamitos, CA, 66–72.

JÄRVELIN, K. AND KEKÄLÄINEN, J. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst. 20*, 4, 422–446.

KATSIFODIMOS, A., PALLIS, G., AND DIKAIAKOS, D. M. 2009. Harvesting large-scale grids for software resources. In *Proceedings of CCGRID'09.* IEEE, Los Alamitos, CA,

KHEMAKHEM, S., DRIRA, K., AND JMAIEL, M. 2007. Sec+: An enhanced search engine for component-based software development. *SIGSOFT Softw. Eng. Notes 32*, 4, 4.

KHEMAKHEM, S., DRIRA, K., AND JMAIEL, M. 2010. An integration ontology for components composition. *Int. Jo Web Portals 2*, 3, 35–42.

KOREN, J., LEUNG, A., ZHANG, Y., MALTZAHN, C., AMES, S., AND MILLER, E. 2007. Searching and navigating metabyte-scale file systems based on facets. In *Proceedings of PDSW'07.* 21–25.

LESKOVEC, J., KLEINBERG, J., AND FALOUTSOS, C. 2007. Graph evolution: Densification and shrinking diameters. *ACM Trans Knowl. Disav. Data 1*, 1.

LI, G., OOI, B. C., FENG, J., WANG, J., AND ZHOU, L. 2008. Ease: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *Proceedings of SIGMOD'08.* ACM, New York, 903–914.

LI, Q., LIU, A., LIU, H., LIN, B., HUANG, L., AND GU, N. 2009. Web services provision: Solutions, challenges and opportunities. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication (ICUIMC'09)*. ACM, New York, 80–87.

LINSTEAD, E., BAJRACHARYA, S., NGO, T., RIGOR, P., LOPES, C., AND BALDI, P. 2009. Sourcerer: Mining and searching internet-scale software repositories. *Data Mining. Knowl. Discov 18*, 2, 300–336.

LIU, L., XU, L., WU, Y., YANG, G., AND GANGER, G. R. 2010. SmartScan: Efficient metadata crawl for storage management metadata querying in large file systems. Tech. rep.CMU-PDL-10-112, Parallel Data Lab., Carnegie Mellon University.

LUCENE. 2009. Apache Lucene. http://lucene.apache.org/core/.

LUCIA, A. D., FASANO, F., OLIVETO, R., AND TORTORA, G. 2007. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Trans. Softw. Eng. Methodol. 16*, 4, 13.

LUCREDIO, D., DO PRADO, A. F., AND DE ALMEIDA, E. S. 2004. A survey on software components search and retrieval. In *Proceedings of the 30th Euromicro Conferenc.* 152–159.

MAAREK, Y. S., BERRY, D. M., AND KAISER, G. E. 1991. An information retrieval approach for automatically constructing software libraries. *IEEE Trans. Softw. Eng. 17*, 8, 800–813.

MANBER, U. 1994. Finding similar files in a large file system. In *Proceedings of the USENIX Winter Technical Conference*. USENIX Association, Berkeley, CA, 2.

MARCUS, A. AND MALETIC, J. 2003. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of ICSE2003.* 125–135.

MATSUSHITA, M. 2005. Ranking significance of software components based on use relations. *IEEE Trans. Softw. Eng. 31*, 3, 213–225.

MCMILLAN, C., GRECHANIK, M., POSHYVANYK, D., XIE, Q., AND FU, C. 2011. Portfolio: Finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering* (*ICSE'11*). ACM, New York, 111–120.

MOHAGHEGHI, P. AND CONRADI, R. 2008. An empirical investigation of software reuse benefits in a large telecom product. *ACM Trans. Softw. Eng. Methodol. 17*, 3, 1–31.

PALLIS, G., KATSIFODIMOS, A., AND DIKAIAKOS, D. M. 2009. Effective keyword search for software resources installed in large-scale grid infrastructures. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence.* ACM, New York.

ROBILLARD, M. P. 2008. ROBILLARD, M. P. 2008. Topology analysis of software dependencies. *ACM Trans. Softw. Eng. Methodol. 17*, 4, 1–36.

SOULES, C. A. N. AND GANGER, G. R. 2005. Connections: using context to enhance file search. *SIGOPS Oper. Syst. Rev. 39*, 5, 119–132.

SPÄRCK JONES, K. AND WILLETT, P. 1997. *Readings in Information Retrieval.* Morgan Kaufman, San Francisco.

SUSAN, S., MEDHA, U., SUKANYA, R., AND CHRISTINA, L. 2010. How well do search engines support code retrieval on the Web? *ACM Trans. Softw. Eng. Methodol*.

TOCH, E., GAL, A., REINHARTZ-BERGER, I., AND DORI, D. 2007. A semantic approach to approximate service retrieval. *ACM Trans. Internet Technol. 8*.

VANDERLEI, T. ET AL. 2007. A cooperative classification mechanism for search and retrieval software components. In *Proceedings of SAC'07.* ACM, New York, 866–871.

XUE, X.-B., ZHOU, Z.-H., AND ZHANG, Z. M. 2008. Improving web search using image snippets. *ACM Trans. Internet Technol. 8*, 21–28.

YEUNG, P. C., FREUND, L., AND CLARKE, C. L. 2007. X-site: A workplace search tool for software engineers. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (*SIGIR'07*). ACM, New York, 900.

ZAREMSKI, A. M. AND WING, J. M. 1997. Specification matching of software components. *ACM Trans. Softw. Eng. Methodol. 6*, 4, 333–369.

Received May 2011; revised February 2012; accepted April 2012