**Noname manuscript No.**
(will be inserted by the editor)

# Searching for Software on the EGEE Infrastructure

**George Pallis · Asterios Katsifodimos · Marios D. Dikaiakos**

Received: date / Accepted: date

**Abstract** Grid infrastructures are in operation around the world, federating an impressive collection of computational resources and a wide variety of application software. In this context, it is important to establish advanced software discovery services that could help end-users locate software components suitable to their needs. In this paper, we present the design, architecture and implementation of an open-source keyword-based paradigm for the search of software resources in large-scale Grid infrastructures, called *Minersoft*. A key goal of Minersoft is to annotate automatically all the software resources with keyword-rich metadata. Using advanced Information Retrieval (IR) techniques, we locate software resources with respect to users queries. The results of Minersoft harvesting are encoded in a weighted, typed graph, named the Software Graph. Experiments were conducted in EGEE, one of the largest Grid production services currently in operation. Experimental results show that Minersoft is a powerful tool achieving high search efficiency.

**Keywords** Software retrieval · Knowledge Grids · Management Resources

This work was supported in part by the European Commission under the 7th Framework Programme through the SEARCHiN project (Marie Curie Action, contract number FP6-042467) and the Enabling Grids for E-sciencE project (contract number INFSO-RI-222667) and makes use of results produced with the EGEE (www.eu-egee.org) Grid infrastructure. The authors would like to thank EGEE users that provided characteristic queries for evaluating Minersoft.

George Pallis
Department of Computer Science
University of Cyprus, Nicosia, 1678, Cyprus
tel: 0035722892700
fax: 0035722892701
E-mail: gpallis@cs.ucy.ac.cy

Asterios Katsifodimos
Department of Computer Science
University of Cyprus, Nicosia, 1678, Cyprus
E-mail: asteriosk@cs.ucy.ac.cy

Marios D. Dikaiakos
Department of Computer Science
University of Cyprus, Nicosia, 1678, Cyprus
E-mail: mdd@cs.ucy.ac.cy

## 1 Introduction

Currently, a number of large-scale Grid infrastructures are in operation around the world, federating an impressive collection of computational resources and a wide variety of application software [1,2]. These infrastructures provide production-quality computing and storage services to thousands of users that belong to a wide range of scientific and business communities. In the context of large-scale Grids, it is important to establish advanced software discovery services that can help end-users locate software components that are suitable for their computational needs. Earlier studies have shown that the difficulty in discovering software components was one of the key inhibitors for the adoption of component technologies and software reuse [8]. Therefore, the provision of a user-friendly tool to search for software is expected to expand the base of Grid users substantially.

Adopting a keyword-based search paradigm for locating software seems like an obvious choice, given that keyword search is currently the dominant paradigm for information discovery [26]. To motivate the importance of such a tool, let us consider a biologist who is searching for drug discovery software deployed on a Grid infrastructure. Unfortunately, the manual discovery of such software is a daunting, nearly impossible task. Taking the case of EGEE (Enabling Grids for E-SciencE), 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.

Existing alternatives to manual search for software are limited. Although Grid infrastructures comprise centralized Grid information registries that can be queried to provide information about the configuration and status of Grid resources, these registries typically contain scarce and poorly maintained tags about installed software [14]. 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 based on their name and not on their contents.

It should be noted that existing Web search engines cannot be used for the retrieval of software residing in Grid infrastructures, whose context is fundamentally different from the World-Wide Web: in contrast to Web pages, access to installed software cannot be gained through a common protocol (HTTP). Also, most software files are not amenable to traditional information extraction techniques used in Information Retrieval: their content is often binary encoded and/or contains little textual descriptions. Last, but not least, software is stored in file systems along with numerous other files of different kinds. Traditional file systems do not maintain metadata representing file semantics and distinguishing between different file types. Also, there are no hyperlinks explicitly linking software-related files. Consequently, the software-search problem cannot be addressed by traditional IR approaches or semantic search techniques.

Envisioning the existence of a Grid software search engine, a biologist would submit a query to the search engine using some keywords (e.g. "docking proteins biology," "drug discovery," or "autodock"). In response to this query, the engine would return a list of software matching the query's keywords, along with Grid sites where this software could be found. Thus, the user would be able to identify the sites hosting an application suitable to her needs, and would accordingly prepare and submit jobs to these sites. In another use-case, a researcher might need the Matlab software package in order to run experiments. The Matlab software package is large in size and the deployment of the software on each job submission is not realistic. In addition, licensing issues could prohibit such an action. Thus the researcher would have to locate sites where this software package is already installed. Instead of contacting site administrators of over 300 sites, the researcher would submit a query to the search engine searching for "Matlab" and the search engine would provide a list of sites that Matlab is already deployed. The user would then contact the site administrators of the sites returned by Minersoft, ensuring that he/she is licensed to use the Matlab software and he/she would prepare and submit jobs accordingly.

To meet this vision, we need a new methodology that will: i) discover automatically software-related resources installed in file systems that host a great number of files and a large variety of file types; ii) extract structure and meaning from those resources, capturing their context, and iii) discover implicit relationships among them. Also, we need to develop methods for effective querying and for deriving insight 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 [3]. Searching for software falls under this general problem since file-systems treat software resources as unstructured data and maintain very little if any metadata about installed software.

To address the software search challenge, we developed the *Minersoft* Grid harvesting system. Therefore, the main intention of Minersoft is to locate computing resources that contain software interesting for a user. Once the user has located the software that needs, he/she submits jobs to the resources found in order to use that software. This is important, since not all the software resources can be deployed by one job submission (i.e., large software packages like Matlab). Also, there are many licensing issues that do not permit distribution of software to third party resource providers (you have to find where the software is installed and run your experiments there, in case you have the right to use that software).

To the best of our knowledge, Minersoft provides the first full-text search facility for locating software resources installed in large-scale Grid infrastructures. Minersoft visits Grid sites, crawls their file systems, identifies software resources of interest (software, libraries, documentation), assigns type information to these resources, and discovers implicit associations between software and documentation files. Subsequently, it creates an inverted index of software resources that is used to support keyword-based searches. 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, which is used to represent the context of discovered software resources. We process the Software Graph to annotate software resources with metadata and keywords, and use these to build an inverted index of software. Indexes from different Grid sites are retrieved and merged into a central inverted index, which is used to support full-text searching for software installed on the nodes of a Grid infrastructure. The present work continues and improves upon the authors preliminary efforts in [22, 33]. The major contributions of this article are the following:

- We present the design, the architecture, and implementation of the Minersoft harvester.
- We provide a study about the installed software resources in EGEE [1] infrastructure.
- We introduce the *Software Graph*, a typed, weighted graph that captures the types and properties of software resources found in a file system, along with structural and content associations between them (e.g. directory containment, library dependencies, documentation of software).
- We present the Software Graph construction algorithm. This algorithm comprises techniques for discovering structural and content associations between software resources that are installed on the file systems of large-scale distributed computing environments.
- We conduct an experimental evaluation of Minersoft, on a real, large-scale Grid testbed - the EGEE infrastructure, exploring performance issues of the proposed scheme.

The remainder of this paper is organized as follows. Section 2 presents an overview of related work. In Section 3, we provide the definitions for software resources, software package and Software Graph. Section 4 describes the proposed algorithm to create a Software Graph annotated with keyword-based metadata. Section 5 describes the architecture of Minersoft. In Section 6 we present an experimental assessment of our work. We conclude in Section 7.

## 2 Related Work

A number of research efforts [27, 40] 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 in [29]). The distinguishing traits of these approaches are i) the searching paradigm; ii) the corpus upon which the search is conducted; iii) the access and the retrieval process of software resources. Table 1 presents a summary of the most indicative tools for software retrieval.

### 2.1 Searching paradigm

The keyword-based search paradigm for locating software is the dominant paradigm for software resources discovery. In [30], Maarek et. al. presented GURU, possibly the first effort to establish a keyword-based paradigm for the retrieval of source codes and software description documents. Other well-known keyword-based IR systems for software resources are the Maracatu [38], SEC [23], SPARS-J [32], Sourcerer [27], Google Code-Search Search[1] and Koders[2].

### 2.2 Corpus

Most software retrieval systems (GURU [30], Maracatu [38], SEC [23]) have been developed for the retrieval of source code residing inside software repositories or file systems (Wumpus [39]). Also, the Web has been used as a platform for storing and publishing software repositories. A number of research efforts and tools have focused on supporting topical Web searches that target software resources. Specifically, in [27], authors developed a keyword-based paradigm, called Sourcerer, for searching source-code repositories available on the Web. Google Code Search is for developers interested in open-source development. The user can search for open source-code and a list of Google services which support public APIs. Koders is a search engine for open source code. It enables software developers to easily search and browse source code in thousands of projects posted at hundreds of open source repositories. Finally, other researchers use as corpus publicly available CVS repositories, in order to build their own software search engines (e.g., SPARS-J) [32].

### 2.3 Software Resources Retrieval

All the existing software-dedicated IR systems retrieve source files, while most of them retrieve also software-description documents. Regarding the mapping between queries and documented software resources, the cosine

[1] Google Code search engine: http://google.com/codesearch
[2] Koders search engine: http://www.koders.com/

similarity metric is mainly used. GURU uses probabilistic modeling (quantity of information) to map documents to terms providing results that include both full and partial matches. Similar approaches have also been proposed in [6,28,31]. All these works exploit source-code comments and documentation files. The methodology that they follow is to represent them as term-vectors and then they use similarity metrics from Information Retrieval (IR) to identify associations between software resources. Results showed that such schemes work well in practice and are able to discover links between documentation files and source codes [6,28,31].

In order to improve the representation of software resources, the use of folksonomy concepts has been investigated in the context of the Maracatu system [38]. Folksonomy is a cooperative classification scheme where the users assign keywords (called tags) to software resources. A drawback of this approach is that it requires user intervention to manually tag software resources. Finally, the use of ontologies is proposed in [23]; however, this work provides little evidence on the applicability and effectiveness of its solution.

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 [16], the Linking File System (LiFS) [5], or from file systems that provide extensions to support search through facets [24], contextualization [36], desktop search (e.g., Confluence [17], Wumpus [39]), etc. Although Minersoft could easily take advantage of the above file systems offering this kind of support, in our current design we assume that the file system provides the metadata found in traditional Unix and Linux systems that are common in most Grid and Cloud infrastructures.

Regarding the crawling process, in [18], authors 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 of discovering appropriate software assets.

### 2.4 Minersoft vs. existing approaches

Although we are not aware of any work that provides keyword-based searching for software resources on large-scale Grid infrastructures, our work overlaps with prior work on software retrieval [6,30,31,38]. 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 system and implementation aspects:

- **System aspects**:
  - Minersoft supports searching for software installed in the file systems of Grid and cluster infrastructures, as well as in software repositories;
  - Minersoft supports searching not only for source codes but also for executables and libraries stored in binary format;
- **Implementation aspects**:
  - 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. The Software Graph is used to represent software resources and associations thereof under a single data structure, amenable to further processing.
  - Minersoft addresses a number of additional implementation challenges that are specific to Grid infrastructures:
    - Software management is a decentralized activity; different machines in Grids may follow different policies about software installation, directory naming etc. 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.
    - Harvesting the software resources found in Grid infrastructures is a computationally demanding task. Therefore, this task can be distributed to the computational resources available in the infrastructure, achieving load balancing and reducing data communication overhead between the search engine and Grid sites.
    - The users of a Grid infrastructure do not have direct access to servers. Therefore, a 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. Minersoft is implemented to operate at the application level in the software stack of a Grid infrastructure so as to work on top of many different Grid infrastructures. Although, part of Minersoft's search facilities could be integrated in the information services of a Grid middleware,

such a service has been kept out of the Grid middleware stack to preserve middleware independency.

## 3 Background

In this section we provide some background about EGEE infrastructure and define software resource, software package and Software Graph, which are the main focus of this paper.

### 3.1 EGEE Infrastructure

The EGEE (Enabling Grid for E-sciencE) project brings together experts from over 50 countries with the common aim of building on recent advances in Grid technology and developing a production Grid infrastructure which is available to scientists. The project aims to provide researchers in academia and industry with access to major computing resources, independent of their geographic location.

The EGEE infrastructure[3] comprises 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 [15]. 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 EGEE 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 etc.). The user-level Grid middleware utilizes the interfaces provided by the low level middleware so as to provide higher abstractions and services, such as programming tools, resource brokers for managing resources and scheduling application tasks for execution on global resources. 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.

[3] EGEE Project Web site: http://project.eu-egee.org/

### 3.2 Definitions

**Definition 1 Software Resource**. A software resource is a file that is installed on a machine and belongs to one of the following categories: i) *executables* (binary or script), ii) software *libraries*, iii) *source codes* written in some programming language, iv) *configuration files* required for the compilation and/or installation of code (e.g. makefiles), v) unstructured or semi-structured *software-description documents*, which provide human-readable information about the software, its installation, operation, and maintenance (manuals, readme files, etc).

The identification of a software resource and its classification into one of these categories can be done by heuristics that have been addressed by human experts (system administrators, software engineers, advanced users). The heuristics used for classification are defined manually and are based on the file's filename, its placement in the filesystem and the output of the GNU Linux "file" command when executed against the file (the "file" command prints a description of the file that it runs against trying to describe it). For instance, two examples of classification rules are: i) a file described as an EFL/LSB executable by the "file" command is considered an "executable", ii) if a file is under a directory called man followed by a number between 1 and 9 (e.g. man6, man1) and its content is described as troff by the "file" command, then it is considered a "software-description file".

The heuristics used for classification are defined manually and are based on i) files' command descriptions, ii) filenames and iii) files' placements in the filesystem.

**Definition 2 Software Package**. A software package consists of one or more content or/and structurally associated software resources that function as a single entity to accomplish a task, or group of related tasks.

Human experts can recognize the associations that establish the grouping of software resources 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 non-trivial task. To represent the software resources found in a file-system and the associations between them we introduce the concept of the *Software Graph*.

**Definition 3 Software Graph**. Software Graph is a weighted, metadata-rich, typed graph $G(V, E)$. The ver-

| Approach | Searching paradigm | Corpus | Software Resources Retrieval | | | |
|---|---|---|---|---|---|---|
| | | | binaries/ scripts | source codes/ libraries | software-description documents | Binary libraries |
| GURU | keyword-based | Repository | | ✓ | ✓ | |
| Marakatu | keyword-based | Repository | | ✓ | | |
| SEC | keyword-based | Repository | | ✓ | | |
| Wumpus | keyword-based | File system search | | | ✓ | |
| Extreme Harvesting | keyword-based | Web | | ✓ | | |
| SPARS-J | keyword-based | CVS repositories | | ✓ | | |
| Sourcerer | keyword-based | Internet repositories | | ✓ | | |
| Koders | keyword-based | Internet open-source repositories | | ✓ | ✓ | |
| Google Code Search | keyword-based | Web | | ✓ | ✓ | |
| Minersoft | keyword-based | Grid, Cloud, Cluster, Repository | ✓ | ✓ | ✓ | ✓ |

**Table 1** Existing Tools for Software Retrieval

tex-set $V$ of the graph comprises: i) vertices representing software resources found on the file-system of a computing node (*file-vertices*), and ii) vertices representing directories of the file-system (*directory-vertices*). The edges $E$ of the graph represent structural and content associations between vertices.

*Structural associations* correspond to relationships between software resources and file-system directories. These relationships are derived from file-system structure according to various conventions (e.g., about the location and naming of documentation files) or from configuration files that describe the structuring of software packages (RPMs, tar files, etc). *Content associations* correspond to relationships between software resources derived by text similarity.

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 a number of associated metadata attributes, describing its content and context:

- $name(v)$ is the normalized name[4] of the software resource 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 set of terms derived from the path-name of software resource $v$ in the file system of $site(v)$.
- $zone_l(v), l = 1, \ldots, z_v$ is a set of zones assigned to vertex $v$. Each zone contains terms extracted from a software resource that is associated to $v$ and which contains textual content. In particular, $zone_1(v)$ stores the terms extracted from $v$'s own contents, whereas $zone_2(v), \ldots, zone_{z_v}(v)$ store terms extracted from software documentation files associated to $v$. The number $(z_v - 1)$ of these files depends on the file-system organization of $site(v)$ and on the algorithm that discovers such associations (see subsequent section). Each term of a zone is assigned an associated weight $w_i$, $0 < w_i \leq 1$ equal to the term's TF/IDF value in the corpus (software resources found on the file-system of a computing node). Furthermore, each $zone_l(v)$ is assigned a weight $g_l$ so that $\sum_{l=1}^{z_v} g_l = 1$. Zone weights are introduced to support weighted zone scoring in the resolution of end-user queries.

Each edge $e$ of the graph has two attributes: $e = (type, w)$, where $type$ denotes the 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 *Software Packages* are coherent clusters of "correlated" software resources in *Software Graph*. Next, we focus on presenting how the Software Graph can be

[4] Normalization techniques for filenames are presented in [35].

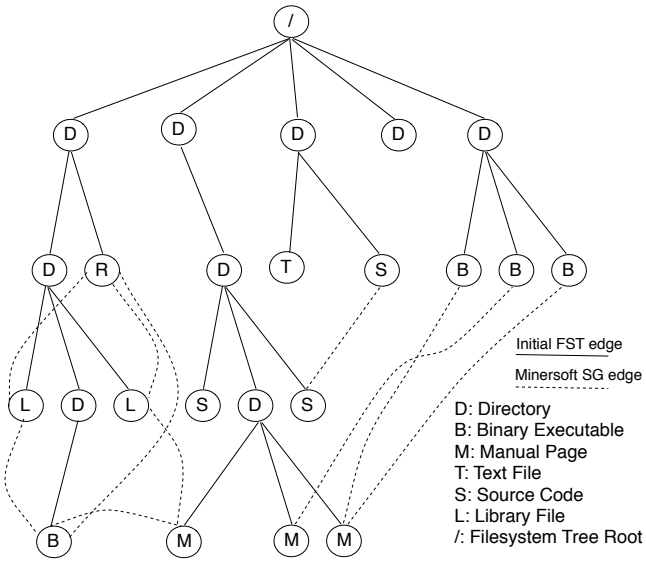


**Fig. 1** An example of a filesystem tree converted to a Software Graph

constructed (section 4), the architecture of Minersoft (section 5) and we evaluate its contribution (section 6).

## 4 Software Graph Construction and Indexing

### 4.1 Overview

A key responsibility of the Minersoft harvester is to construct a Software Graph (SG) for each computing site, starting from the contents of its file system. Figure 1 depicts an example of a filesystem that contains a set of software resources (binary executables, libraries, text files, source codes, readme files and manual pages) and it is converted to a Software Graph.

To this end, we propose an algorithm comprising of the following steps:

1. FST construction.
2. Classification and pruning.
3. Structural dependency mining.
   (a) Structural-context enrichment
4. Keyword scraping.
5. Keyword flow.
   (a) Content enrichment
6. Content association mining.
   (a) Content association
7. Inverted index construction.

The main objectives of the Minersoft algorithm are to i) discover software-related resources installed in file systems; ii) extract structure and meaning from those resources, capturing their context, iii) discover implicit relationships among them, and iv) enrich them with keywords. In the subsequent sections, we provide in detail these steps as well as the algorithms for finding relationships between documentation and software-related files (subsection 4.2.1), keyword extraction and keyword flow (subsection 4.2.2), and content association mining (subsection 4.2.3).

### 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 sub-directories 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`).

**Classification and pruning:** Names and pathnames play an important role in file classification and in the discovery of associations between files. Accordingly, Minersoft normalizes filenames and pathnames of FST vertices, by identifying and removing suffixes and prefixes [35]. 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 executables, source code (e.g. Java, C++ scripts), libraries, software-description documents (e.g. man-pages, readme files, html files) and irrelevant files. Minersoft prunes all FST leaves found to be irrelevant to software search (vertices that do not belong to any category), dropping also all internal FST vertices (directories) 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. Structural relationships can be identified by: i) Rules that represent expert knowledge about file-system organization, such as naming and location conventions. For instance, a set of rules link files that contain *man-pages* to the corresponding executables; *Readme* and *html* files are linked to related software files. ii) Dynamic dependencies that exist between libraries and binary executables. Binary executables and libraries usually depend on other libraries that need to be dynam-

ically linked during runtime. iii) Package management tools that exist in operating systems (i.e., Unix/Linux). These dependencies are mined from the headers of libraries and executables.

The structural dependency mining step produces the first version of the SG, which captures software resources 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 resources.

**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 only from the comments inside the source, since the actual code lines would create unnecessary noise without producing descriptive features. Binary executable files and libraries contain strings that are used for printing out messages to the users, debugging information, logging etc. All this textual information can be used to extract useful features for these resources. 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 zones 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 zones of the latter.

**Content association mining:** Similar to [6] and [31], 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 [30] 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 resources, 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 zones of SG vertices.

#### *4.2.1 Structural-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 resources. 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 $v^r$.
- Each leaf $f$ that is a descendant of an internal node $v_k^r \in V^r$, receives from $v^r$ 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 maintains a stack to keep 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.

*4.2.2 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, where $w$ is a documentation vertex. For each such edge $e_d$, we create an extra documentation zone for $v$. Consequently, $v$ ends up with an associated set of zones $zone(v) = \{zone_1^v, \ldots, zone_{z_v}^v\}$, where $zone_1^v$ corresponds to textual content extracted from $v$ itself and $zone_i^v, i = 2, \ldots, z_v$ correspond to keywords extracted from documentation vertices adjacent to $v$.

Each zone has a different degree of importance in terms of describing the content of the software file of $v$. Thus, we assign to each $zone_i^v$ a different weight $g_i$, which is computed as follows: i) For $i = 1$, namely for the zone that includes the textual content extracted from $v$ itself, we set $g_1 = \alpha_v$, where $0 < \alpha \leq 1$. ii) For each remaining zone of $v$ ($i = 2 \ldots, z_v$), $g_i$ is set to $\alpha_v$ multiplied by the weight of the SG edge that introduced $zone_i^v$ to $v$. The value of $\alpha_v$ is chosen so that $\sum_{i=1}^{z_v} g_i = 1$.

*4.2.3 Content Association*

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 comments found inside source-code files 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 zones 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 [30]. In our case, the corpus is the union of all content zones of SG vertices of source files. To estimate the probability $P(t)$, we measure the percentage of content zones of SG vertices of source files wherein $t$ appears; we do not count the frequency of appearance of $t$ in a content zone, as this would create noise.

Subsequently, we drop terms with quantity of information value less than a specific threshold (for our experiments we remove the terms where $Q(t) < 3.5$). 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 where each vertex belongs to (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.

## 5 Minersoft Architecture

Creating an information retrieval system for software resources 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 resources and keep them up to date. Storage space must be used efficiently to store indices and metadata. The indexing system must process hundreds of gigabytes of data efficiently. In this section, we provide a description of how the Minersoft architecture, depicted in Figure 2.

For the efficient implementation of Minersoft in a Grid setting, we take advantage of various parallelization techniques in order to:

- Distribute parts of the Minersoft computation to Grid resource providers. Thus, we take advantage of their computation and storage power, to speedup the file retrieval and indexing processes, to reduce the communication exchange between the Minersoft system and local Grid sites, and to sustain the scalability of Minersoft with respect to the total number of Grid sites. Minersoft tasks are wrapped as jobs that are submitted for execution to the Grid workload management system.
- Avoid overloading Grid sites by applying load-balancing techniques when deploying Minersoft jobs to Grid .
- Improve the performance of Minersoft jobs by employing multi-threading to overlap local computation with Input/Output (I/O).
- Adapt to the policies put in place by different Grid 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, etc.

### 5.1 Overview

Minersoft adopts a MapReduce-like architecture [13]; the crawling and indexing is done by several distributed multi-threaded crawler and indexer jobs, which run in parallel for improved performance and efficiency. The crawler and indexer jobs process a specific number of files, called *splits*. The key components of the Minersoft architecture are (see Figure 2):

1. The *job manipulator* manages crawler and indexer jobs and their outputs.
2. The *monitor* module maintains the overall supervision of Minersoft jobs. To this end, the *monitor* communicates with the job manager, the datastore, the LCG File Catalogs and the Logging and Bookkeeping services of EGEE infrastructure.
3. The *datastore* module stores the resulted Software Graphs and the full-text inverted indexes centrally.
4. The *query engine* module is responsible for providing quality search results in response to user searches. The query engine module comprises the *query processor* and the *ranker*. The former receives search queries and executes them against the inverted indexes of Minersoft. The latter ranks query results. To this end, a ranking algorithm is used to improve the accuracy and relevance of replies, especially when keyword-based searching produces very large numbers of "relevant" software resources. *Ranker* uses the Lucene relevance ranking[5]. In particular, Lucene provides a scoring algorithm that includes additional data to find best matches to user queries. The default scoring algorithm is fairly complex and considers such factors as the frequency of a particular query term with individual software resources and the frequency of the term in the total population of software resources.

### 5.2 Minersoft Crawler

The crawler is a multi-threaded 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

[5] Apache Lucene: http://lucene.apache.org/java/docs/

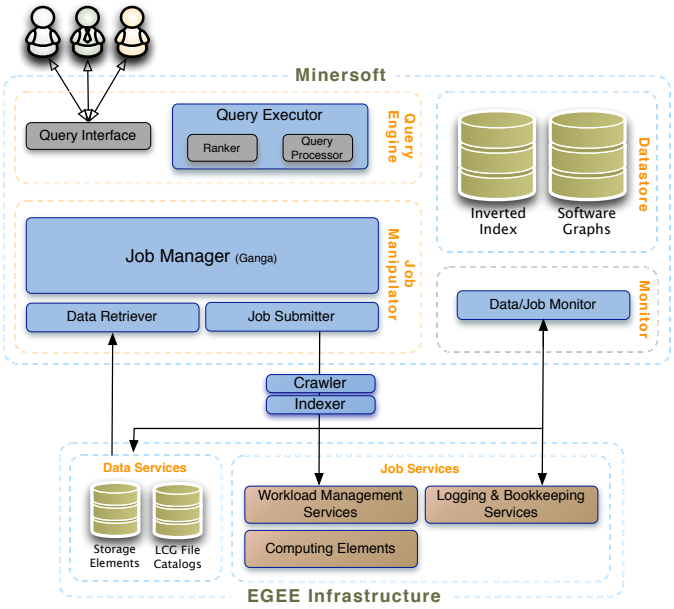


**Fig. 2** Minersoft architecture.

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 resources. 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.

### 5.3 Minersoft Indexer

The Minersoft indexer is a multi-threaded program that reads the files captured in the *metadata store files* and creates full-text inverted indexes. To this end, the indexer performs first 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. This results in enriching the terms and posting lists of inverted indexes with extra keywords. At the end of indexing process, for each Grid site 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. The terms are extracted from the zones of SG vertices.

### 5.4 Distributed Crawling and Indexing Process in EGEE Infrastructure

The crawling and indexing of EGEE Grid sites for software requires the retrieval and processing of large parts of the file systems of numerous sites. These tasks need to address various performance, reliability and policy issues. To address these issues, Minersoft undertakes the crawling and indexing of software resources installed in EGEE infrastructure in a distributed manner. The *job submitter* sends a number of multi-threaded crawler/indexer jobs to the Workload Management Services (WMSs) of Grid sites.

A challenge for crawler/indexer jobs is to process all the software resources residing within EGEE Grid sites, without exceeding the time constraints imposed by site policies. The jobs which run longer than the allowed time are terminated by the sites batch systems. The maximum wall clock time for an EGEE Grid site usually ranges between 2 and 72 hours.

In this context, the file-system of each EGEE Grid site is decomposed into a number of *splits*, where the size of each split is chosen so that the crawling can be distributed evenly and efficiently within the constraints of the underlying networked computing infrastructure. The number of splits is determined by the communication between the *job manager* and the *monitor*. The splits are assigned to crawler/indexer jobs on a continuous basis: When a Grid site finishes with its assigned splits, the *monitor* informs the *job manager* in order to send more splits for processing. If a Grid site becomes laggard, the *monitor* sends a message to *job manager*. Then, the crawler/indexer job is canceled and rescheduled to run when the Grid site's workload is reduced. Furthermore, if the batch system queue of a Grid site is full and does not accept new jobs, the *monitor* sends a signal and the *job submitter* suspends submitting crawler/indexer jobs to that Grid site until the batch system becomes ready to accept more.

When the crawling 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 resource and is temporally stored in the *Datastore*. The *file index* will be used in order to identify the duplicate files during the indexing process; the duplication reduction policy is described in the following subsection. 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*.

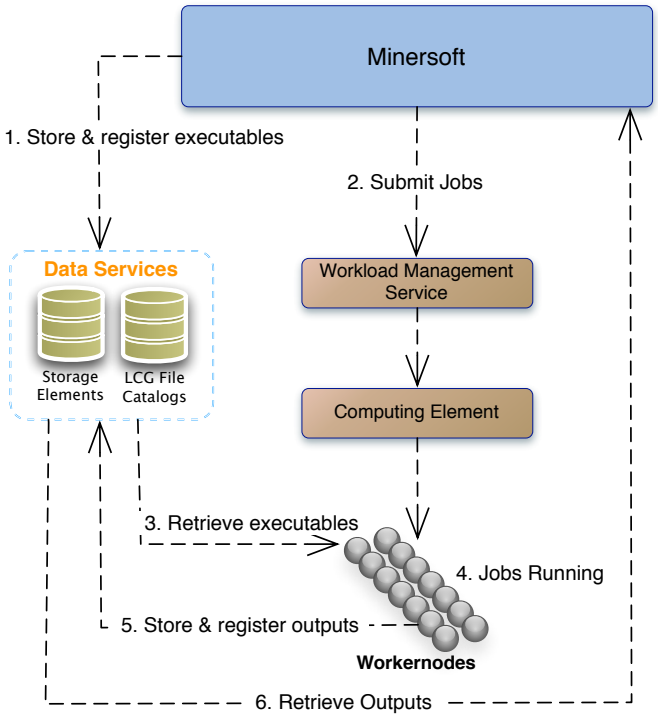


**Fig. 3** Minersoft crawler/indexer job's lifecycle.

### 5.5 Duplication Reduction Policy

Typically, popular software applications and packages are installed on multiple sites of distributed computing infrastructures. If we identify duplicates, we will be able to avoid indexing them multiple times. Consequently, the performance of indexing is improved.

To address this issue, the *job manager* uses a *duplicate reduction policy* to identify the exact duplicate files. According to our policy, a duplicate file is assigned to the Grid site which has the minimum number of assigned files that should be indexed. The key idea behind this policy is to avoid multiple indexing of duplicate software resources in Grid sites so as to prevent their overloading. In this context, for each Grid site/, the following steps take place:

1. The *file index* is sorted in ascending order with respect to the count of Grid sites that a file exists.
2. The files which do not have duplicates are directly assigned to the corresponding Grid site.
3. If a file belongs to more than one Grid sites, the file is assigned to the site with the minimum number of assigned files.

### 5.6 Minersoft Crawler/Indexer Job's Lifecycle in EGEE Infrastructure

Minersoft indexer and crawler jobs follow the same principles for file-staging and execution. The lifecycle of a crawler/ indexer job in EGEE infrastructure starts by copying its executable files to a Storage Element (SE) and terminates by downloading its output files (file-indexes, SGs, inverted file indexes) to the centralized Minersoft infrastructure. Their lifecycle is depicted in Figure 3. The details of each of the individual steps are described below:

1. *Store & register executables*: In order to submit and run crawler and indexer jobs efficiently, the executable files of these jobs are stored into SEs and registered to LCG File Catalogs (LFCs). Thus, the Workload Management System (WMS) and Minersoft avoid extra workload. Note that in most cases we have to submit a large number of jobs, where each job needs executables that their total size is about 3 MB.
2. *Submit jobs*: When the executables have been stored to the SEs, Minersoft submits the crawler/ indexer jobs using the WMS. The WMS moves the crawler/ indexer jobs to the Computing Elements (CEs) of EGEE infrastructure in order to be executed.
3. *Retrieve job executables*: The crawler/ indexer jobs are moved from the CE into one of its workernodes. The executable files of jobs are downloaded from the SEs.
4. *Jobs running*: When the executables have been downloaded, the crawler/indexer jobs start their execution.
5. *Store & register outputs*: When the jobs have successfully finished their execution, their outputs are stored in the SEs and registered to the LFCs making them available to Minersoft for further processing (SG processing, merging inverted indexes etc.).
6. *Retrieve outputs*: The *data retriever* module of Minersoft retrieves the outputs from the SEs and stores them to its centralized infrastructure.

### 5.7 Minersoft Implementation and Deployment

The implementation of the *job manager* and *monitor* relies upon the Ganga system [10], 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 this context, the *monitor* (through Ganga scripts) monitors the status of jobs after their submission and keeps a list of Grid sites and their failure rate. If there are Grid sites with a high failure rate, the *monitor* eventually puts them in a black list and notifies *job manager* so as to stop submitting jobs to them, thus excluding them from the current indexing/crawling session. Sites that are not fully crawled or indexed are not included in the search results.

The crawler is written in Python. The Python code scripts are put in a tar file and copied on a SE before job submission starts. The tar file is being downloaded and untarred to the target Grid site before the crawler execution starts. By doing that, the size of the jobs input sandbox is reduced, thus job submission is accelerated because the workload management system has to deal with much less files per job.

The indexer is written in Java and Bash and uses an open-source high performance, full-text index and search library (Apache Lucene). In order to execute the indexer jobs, we follow the same code-deployment scenario as with crawlers.

The *job manager* has to distribute the crawling and indexing workload before the job submission starts. This is done by creating splits for each Grid site that Minersoft has to crawl. The input file (list of files) for each split resides on a SE and is registered to a file catalog. The split input is then downloaded from a SE 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 SEs 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.

## 6 Evaluation

### 6.1 Testbed

The usefulness of the findings of any study depends on the realism of the data upon which the study operates. For this purpose, the experiments are conducted on 10 Grid sites of EGEE (Enabling Grid for E-sciencE). The EGEE 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 location. Totally, the Grid testbed includes 6 million files where their total size is over 366 GB. Table 2 presents the Grid that have been crawled and indexed by Minersoft.

Files in the workernodes of EGEE are not always readable by all users that are allowed to run jobs on

| Grid Site | # of Files | Size (MB) |
|---|---|---|
| ce01.kallisto.hellasgrid.gr | 3.541.403 | 259.953.246,2 |
| ce301.intercol.edu | 97.906 | 3.711.925,517 |
| grid-ce.ii.edu.mk | 194.556 | 4.981.889,552 |
| paugrid1.pamukkale.edu.tr | 132.645 | 3912.987,884 |
| ce01.grid.info.uvt.ro | 270.445 | 10.849.994,58 |
| grid-lab-ce.ii.edu.mk | 109.286 | 2.824.409,829 |
| ce01.mosigrid.utcluj.ro | 70.419 | 19.539.562,62 |
| ce101.grid.ucy.ac.cy | 1.278.851 | 64.886.738,85 |
| ce64.phy.bg.ac.yu | 150.661 | 6.685.600,57 |
| testbed001.grid.ici.ro | 125.028 | 7.117.152,75 |
| **Total** | 5.971.200 | 384.463.508,4 |

**Table 2** EGEE Testbed.

them. In some cases, access to files is only given to specific VOs because of licensing and/or site-policy reasons. In our experiments, Minersoft runs as a user from the South East Europe (SEE) VO and the files that it can read are all the files that are readable by that VO users in each EGEE site.

## 6.2 Crawling and Indexing Evaluation

In this section, we elaborate on the performance evaluation of the crawling and indexing tasks of Minersoft. Our objective is to show that Minersoft is a useful application for developers, administrators and researchers in EGEE infrastructure.

### *6.2.1 Examined metrics*

To assess the crawling and indexing in Minersoft, we investigate the performance of crawler and indexer jobs; recall that each job is responsible for a number of files (called splits) that exist on a Grid site. In this context, we use the following metrics:

- Run time: the average time that a crawler/indexer job spends on a Grid site, including processing and I/O; this metric measures the average elapsed time that Minersoft needs to process (crawl or index) a split.
- CPU time: the average CPU time in seconds spent by a crawler/indexer job while processing a split on a Grid site.
- File rate: the number of files that Minersoft crawls/ indexes per second on a Grid site.
- Size rate: the size of files in bytes that Minersoft crawls/indexes per second on a Grid site.

In our experiments, each crawler and indexer job was configured to run with five threads. We also ran experiments with different numbers of threads (from 1, 5, 9 to 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 to significantly higher run times, due to poor CPU utilization or I/O contention, respectively. Recall, that the crawler and indexer jobs process a specific number of files, called *splits*. In our experiments, the maximum number of files that a split can process is 100.000.

### *6.2.2 Crawling Evaluation*

Figure 4 depicts the *per-job average run-time* and *per-job average CPU-time* for crawling the Grid sites. The per-job CPU time takes into account the total time that all the job's threads spend in the CPU. The run-time values are significantly larger than the CPU times due to the system calls and I/O that each crawler performs while processing its file split. I/O is much more expensive in the case of Grid sites with shared file systems. Another observation is that the run-time and CPU-time of crawler jobs vary significantly across different Grid 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 the crawler processing on site-dependent aspects. For example, the crawler performs expensive "deep" processing of binary and library files to deduce their type and extract dependencies. This is not required for text files. Consequently, the percentage of binaries/libraries found in each site determines to some extent the corresponding crawling computation.

Table 3 depicts the throughput achieved by the Minersoft crawler on different Grid sites, expressed in terms of the number of files and the number of bytes processed. Results show that Minersoft achieves high crawling rates.

The files found by the crawlers to be irrelevant to software search are pruned from subsequent processing. Figure 5 presents the percentage of files that have been dropped in Grid sites. We observe that a large percentage of content in most Grid sites includes software resources. Specifically, on average 75% of total files' size that exist in Grid sites have been classified as software resources. These findings confirm the need to establish advanced software services in Grid infrastructures. The software-related files are categorized with respect to their type. From Table 4, we can see that most software-related files in the EGEE infrastructure are documentation files (man-pages, readme files, html files) and sources. Sources are files written in any programming language. Executable scripts (e.g. python, perl, bash) are also considered as sources (e.g. Java, C++). Table 5 presents the number of splits that have been created in order to crawl the files.

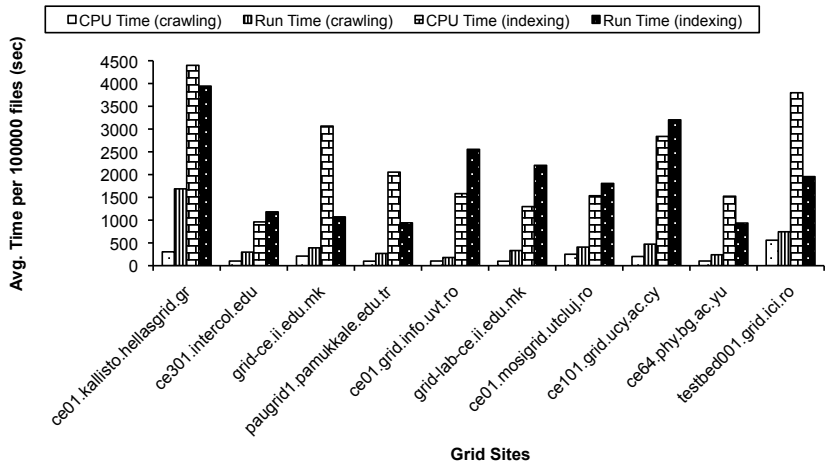


**Fig. 4** Average times for jobs in EGEE infrastructure.

| Grid Site | File rate (files/sec) | Size rate(MB/sec) |
|---|---|---|
| ce01.kallisto.hellasgrid.gr | 59,287 | 4,067 |
| ce301.intercol.edu | 335,773 | 6,085 |
| grid-ce.ii.edu.mk | 255,395 | 4,141 |
| paugrid1.pamukkale.edu.tr | 372,467 | 4,744 |
| ce01.grid.info.uvt.ro | 557,289 | 14,762 |
| grid-lab-ce.ii.edu.mk | 300,905 | 2,766 |
| ce01.mosigrid.utcluj.ro | 245,074 | 23,382 |
| ce101.grid.ucy.ac.cy | 211,604 | 9,577 |
| ce64.phy.bg.ac.yu | 417,031 | 9,075 |
| testbed001.grid.ici.ro | 134,300 | 3,111 |

**Table 3** Crawling rates in EGEE infrastructure.

| Grid Site | Binaries | Sources | Libraries | Docs | Irrelevant |
|---|---|---|---|---|---|
| ce01.kallisto.hellasgrid.gr | 41.990 | 1.407.701 | 142.873 | 1.672.246 | 276.593 |
| ce301.intercol.edu | 34.134 | 8.972 | 3.724 | 23.536 | 27.540 |
| grid-ce.ii.edu.mk | 16.869 | 69.915 | 8.080 | 61.469 | 38.223 |
| paugrid1.pamukkale.edu.tr | 7.383 | 47.388 | 7.935 | 43.861 | 26.078 |
| ce01.grid.info.uvt.ro | 8.999 | 40.442 | 3.778 | 42.652 | 174.574 |
| grid-lab-ce.ii.edu.mk | 7.703 | 46.116 | 2.983 | 37.333 | 15.151 |
| ce01.mosigrid.utcluj.ro | 17.828 | 12.475 | 2.310 | 18.091 | 19.715 |
| ce101.grid.ucy.ac.cy | 26.377 | 433.115 | 37.463 | 672.211 | 109.685 |
| ce64.phy.bg.ac.yu | 6.047 | 31.889 | 7.672 | 67.388 | 37.665 |
| testbed001.grid.ici.ro | 29.261 | 22.961 | 6.120 | 28.239 | 38.447 |
| **Total** | 196.591 | 2.120.974 | 222.938 | 2.667.026 | 763.671 |

**Table 4** Files Categories in EGEE infrastructure.

*6.2.3 Indexing Evaluation*

Figure 4 depicts the *per-job average run-time* and the *per-job CPU time* for indexing Grid sites. As expected, we observe that indexing is more computationally-intensive than crawling, since we need to conduct "deep" parsing inside the content of all files.

Removing the duplicate files via the *duplicate reduction policy* leads to reducing the number of files. Consequently, this improves the indexing performance. Our results showed that about 11% of files belong to more than one Grid sites. In a previous study [22] we had also observed a large number of duplicate files in Grid infrastructure. So, the main conclusion of these findings is that there is a large number of duplicate files in Grid infrastructures. Table 5 presents the number of splits and the size of inverted file indexes in each Grid site of EGEE. In order to study the benefits of duplicate reduction policy, we also present the number of splits without performing duplication. From this table we observe that ce01.kallisto.hellasgrid.gr has 31 splits

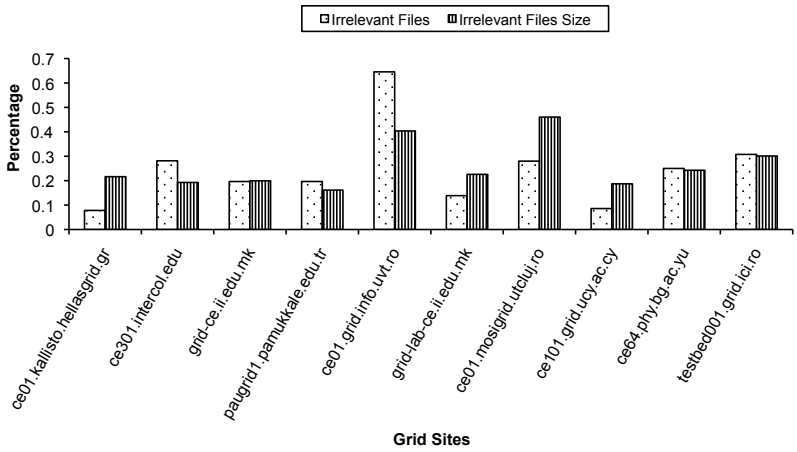


**Fig. 5** Percentage of irrelevant files in EGEE infrastructure.

| Grid Sites | Crawling Statistics | Indexing Statistics | | | |
|---|---|---|---|---|---|
| | # of splits | # of splits | # of splits including duplicates | Inverted Index size with stemming (MB) | Inverted Index size w/o stemming (MB) |
| ce01.kallisto.hellasgrid.gr | 37 | 31 | 33 | 58.129,359 | 60.988,246 |
| ce301.intercol.edu | 2 | 1 | 1 | 358,0312 | 395,089 |
| grid-ce.ii.edu.mk | 1 | 1 | 2 | 702,796 | 778,972 |
| paugrid1.pamukkale.edu.tr | 3 | 1 | 2 | 632,035 | 702,296 |
| ce01.grid.info.uvt.ro | 4 | 1 | 1 | 1.152,386 | 1.258,464 |
| grid-lab-ce.ii.edu.mk | 3 | 1 | 1 | 57,414 | 62,863 |
| ce01.mosigrid.utcluj.ro | 2 | 1 | 1 | 257,984 | 288,214 |
| ce101.grid.ucy.ac.cy | 14 | 10 | 12 | 13.588,468 | 14.073,417 |
| ce64.phy.bg.ac.yu | 3 | 1 | 2 | 871,449 | 964,621 |
| testbed001.grid.ici.ro | 3 | 1 | 1 | 646,671 | 715,148 |

**Table 5** Crawling & Indexing statistics in EGEE infrastructure.

instead of 33 splits (including duplicate files). Moreover, even if the number of splits is not reduced, it is reduced the number of files that have been assigned in a split. Consequently, the total indexing time is significantly reduced. Regarding the size of inverted indexes, we present the size of inverted indexes with and without performing stemming. We observe that stemming decreases the size of inverted indexes about 10%.

Finally, Table 6 depicts the throughput of the indexer expressed in terms of the number of files and the number of bytes processed per second in each Grid site of EGEE. The performance of indexing is affected by the hardware (disk seek, CPU/memory performance), file types, and the workload of each site.

To sum up, our experimentations concluded to the following empirical observations:

- Minersoft successfully crawled about 6 million valid files (384 GB size) and sustained high crawling rates.
- A large percentage of duplicate files exists in EGEE infrastructure. Identifying these files, the performance of indexing is significantly improved.
- The crawling and indexing in EGEE infrastructure is significantly affected by the hardware (local disk, shared file system), file types and the current workload of Grid sites.
- In most cases, more than 70% of files that exist in the workernodes file systems of Grid sites are software files. Advanced software discovery services in Grid infrastructures should be established.

### 6.3 Software Retrieval Evaluation

In this section, we evaluate the effectiveness of the Minersoft search engine for locating software on EGEE infrastructure. A difficulty in the evaluation of such a system is that there are not widely accepted any benchmark data collections dedicated to software (e.g., TREC, OHSUMED etc). In this context, we use the following methodology in order to evaluate the performance of Minersoft:

- *Data collection*: Our dataset consists of the software installed in 10 Grid sites of EGEE infrastructure

| Grid Site | File rate (files/sec) | Size rate (MB/sec) |
|---|---|---|
| ce01.kallisto.hellasgrid.gr | 25,359 | 1,363 |
| ce301.intercol.edu | 84,620 | 1,238 |
| grid-ce.ii.edu.mk | 93,297 | 1,211 |
| paugrid1.pamukkale.edu.tr | 106,283 | 1,135 |
| ce01.grid.info.uvt.ro | 39,143 | 0,618 |
| grid-lab-ce.ii.edu.mk | 45,387 | 0,323 |
| ce01.mosigrid.utcluj.ro | 55,363 | 2,850 |
| ce101.grid.ucy.ac.cy | 31,231 | 1,149 |
| ce64.phy.bg.ac.yu | 106,818 | 1,760 |
| testbed001.grid.ici.ro | 51,127 | 0,827 |

**Table 6** Indexing rates in EGEE infrastructure.

(Table 2). Table 4 presents the software resources that have been identified by Minersoft on those Grid sites.

- *Queries*: We use a collection of 26 keyword queries, which were extracted i) by EGEE users and ii) by real user-queries from the Sourcerer system [27]. These queries comprise either single- or multiple-keywords. Each query has an average of 2 keywords; this is comparable to values reported in the literature for Web search engines [37]. To further investigate the sensitivity of Minersoft, we have classified the queries into two categories: general-content and software-specific (see Table 7).
- *Relevance judgment*: A software resource is considered relevant if it addresses the stated information need and not because it just happens to contain all the keywords in the query. A software resource returned by Minersoft in response to some query is given a binary classification as either relevant or non-relevant with respect to the user information need behind the query. In addition, the result of each query has been rated at three levels of user satisfaction: "not satisfied", "satisfied", "very satisfied". These classifications have been done manually by EGEE administrators and experienced users and are referred to as the *gold standard* for our experiments.

| General-content queries | Software-specific queries |
|---|---|
| java virtual machine; statistical analysis software; ftp client; regular expression; sigmoid function; histogram plot; binary tree; zip deflater; pdf reader | imagemagick; octave numerical computations; lapack library; gsl library; boost c++ library; glite data management; xerces xml; subversion client; gcc fortran; lucene; jboss; mpich; autodock docking; atlas software; linear algebra package; fftw library; earthquake analysis |

**Table 7** Queries.

#### *6.3.1 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 resources ranked in the top 10 results that are labeled as relevant. The relevance of the retrieved results is determined by the *gold standard*. By default, we consider that the results are ranked with respect to the ranking function of Lucene, which is based on TF-IDF of software files and has extensively been used in the literature [7,12]. The maximum Precision@10 value that can be achieved is 1.
- NDCG (Normalized Discounted Cumulative Gain) is a retrieval measure devised specifically for evaluating user satisfaction [19]. For a given query q, the $top-K$ ranked results are examined in decreasing order of rank, and the NDCG 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", 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 [4,11]. In our evaluation metrics we do not consider the recall metric (the percentage of the number of relevant results). Such a metric requires to have full knowledge about all the relevant software resources with respect to a query. However,

such a knowledge is not feasible in a large-scale networked environment.

### *6.3.2 Examined Approaches*

In order to evaluate the Minersoft efficiency, we evaluate the construction phases of Minersoft's inverted index. Specifically, we examine the following:

- *Full-text search*: Inverted index terms are only extracted from the full-text content of discovered files in the examined testbed infrastructure without any preprocessing. This approach is relevant to the desktop search systems (also known as file system search - e.g., Confluence [17], Wumpus [39]). *Full-text search* is used as a baseline for our experiments.
- *Path-enhanced search*: The terms of inverted index are extracted from the content and path of SG vertices. The irrelevant files are discarded.
- *Context-enhanced search*:The files have been classified into file categories. The irrelevant files are discarded. We also exclude the software-description documents from the posting lists. The terms of the inverted index are extracted from the content and path of SG vertices.
- *Software-description-enriched search*: The terms of inverted index are extracted from the content of SG vertices as well as from the zones of documentation files (i.e., man-pages and readme files) and the path of SG vertices.
- *Text-files-enriched search*: The terms of inverted index are extracted from the content, the path and the zones from the 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.

### *6.3.3 Evaluation*

Figures 6, 7, and 8 present the results of the examined approaches with respect to the query types for *Precision*@10, NDCG and NCG. Each approach is a step towards the construction of the inverted index that is implemented in Minersoft. For completeness of presentation, we present the average and median values of the examined metrics. The general observation is that Minersoft improves significantly both the *Precision*@10 and the examined cumulative gain measures compared with the baseline approach - *full-text search* - for both types of queries. Specifically, Minersoft improves the *Precision*@10 about 139%, the NDCG about 142% and the NCG about 135% with respect to the baseline approach. Another general observation is that Minersoft achieves quite similar performance for both software-specific and general-content queries.

Regarding the intermediate steps for the construction of SG, the highest improvement is observed at *Context-enhanced search*. This explained by the fact that the non-relevant software files have been removed and the searching is done only at software files. Our findings show also that the addition of metadata attribute of path in software resources makes Minersoft more effective. In particular, *Path-enhanced search* improves both the *Precision*@10 about 40% and the cumulative gain measures about 53% for NDCG and 44% for DCG with respect to the baseline approach. This is an indication that the paths of software files include descriptive keywords for software resources.

The enrichment of software-description documents increases the precision as well as user satisfaction. Specifically, *Software-description-enriched search* achieves higher *Precision*@10 (about 3%) and higher cumulative gain measures (on average about 3% for NDCG and 4.3% for NCG) than the *Context-enhanced search*. The improvements during the *Software-description-enriched search* step are affected by the number of the executables and software libraries that exist in the data set. The larger the number of these types of files exist in repositories, the better results are obtained. Another interesting observation is that most of queries indicate *Precision*@10 close to 1 (see median values), whereas the average *Precision*@10, NDCG and NCG values for all the queries are about 0.77, 0.71, 0.69 respectively.

Regarding the *text-files-enriched search*, we observe that this approach does not improve the general system's performance. This is explained by the fact the software developers use similar filenames in their software packages. However, taking a deeper look at the results, we observe *text-files-enriched search* improves user satisfaction about 2% for software-specific queries since more results are returned to users than the previous examined approach.

To sum up, the results show that Minersoft is a powerful tool since it achieves high effectiveness for both types of queries. Focusing on the query types, we observe that Minersoft presents high efficiency for both types of queries, achieving very high performance for software-specific queries. Specifically, our experimentations concluded to the following empirical observations:

- Minersoft improves the *Precision*@10 about 139% and Cumulative gain measures (NDCG, NCG) over 135% with respect to the baseline approach.
- The paths of software files in file-systems include descriptive keywords for software resources.
- Software developers use similar filenames for their software packages.

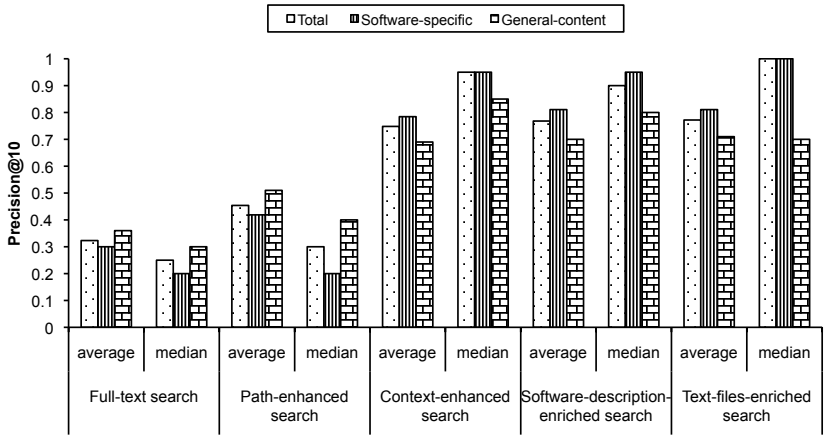


**Fig. 6** Precision@10 results.

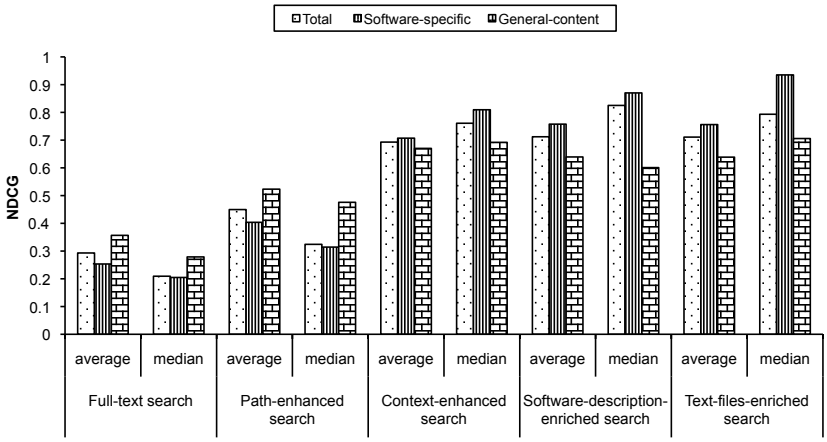


**Fig. 7** NDCG results.

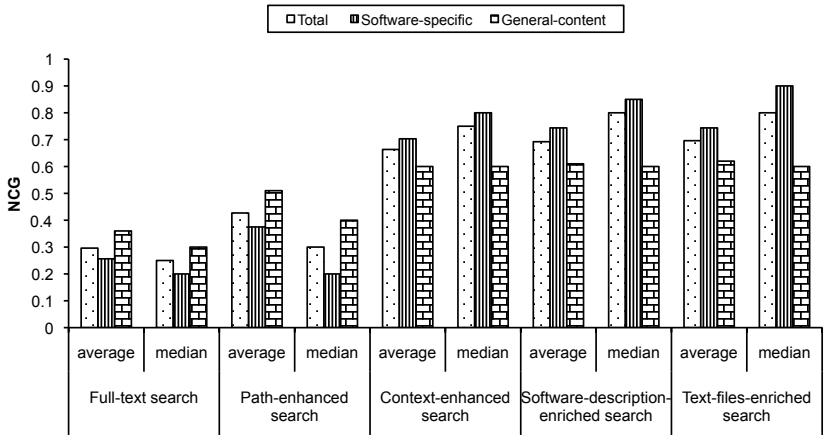


**Fig. 8** NCG results.

| 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 |

**Table 8** Software Graphs statistics in EGEE infrastructure.

*6.3.4 Software Graph Statistics*

Table 8 presents the statistics of the resulted SGs. Recall that Minersoft harvester constructs a SG in each Grid site. We do not present further analysis of the SGs since this is out of the scope of this work. Of course, a thorough study of the structure and evolution of SGs would lead to insightful conclusions in software engineering community. In the literature, a large number of dynamic large-scale networks have been extensively studied [25] in order to identify their latent characteristics.

Here, we briefly present the main characteristics of these graphs. Table 8 presents the edges that have been added due to structure dependency ($E_{SD}$) and content associations ($E_{CA}$). For completeness of presentation, the index size of each graph is presented. Based on these statistics, a general observation is that the SGs are not sparse. Specifically, we found that most of them follow the relation $E = V^{\alpha}$, where $1.1 < \alpha < 1.36$; note that $\alpha = 2$ corresponds to an extremely dense graph where each node has, on average, edges to a constant fraction of all 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 which are due to structure dependency associations.

## 7 Conclusion

In this paper, we present Minersoft - a tool which enables keyword-based searches for software installed on Grid computing infrastructures. The software design of Minersoft enables the distribution of its crawling and indexing tasks to large-scale network environments. The results of Minersoft harvesting are encoded in a weighted, typed graph, called the SG. The SG is used to annotate automatically the software resources 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 resources, improving the searching of software packages in large-scale network environments.

Minersoft can be easily extended to support search on Cloud infrastructures like Amazon's EC2. There are two main issues that Minersoft has to overcome in order to support searching in Cloud infrastructures: i) job submission protocol: Minersoft has to be provided access to machines on an infrastructure through a job submission protocol or other means of access; ii) data storage facilities: Minersoft needs storage space available in order to store the software graph and index files as well as a data access protocol to manipulate them. In order to overcome the job submission protocol issues, Minersoft uses the Ganga system to submit and monitor jobs. Ganga is a middleware/infrastructure which is independent and it can be extended to support many protocols of job submission to different infrastructures through plugins (e.g., it can submit jobs through the SSH protocol). Cloud infrastructures that provide virtual servers on demand (like Amazon's EC2 and Rackspace's Cloud Servers) can be searched from Minersoft by changing the underlying job submission protocol in Ganga to use SSH (or any connectivity tool that is provided by the Cloud service provider). As far as data storage is concerned, instead of storing files in Grid Storage Elements, file storage services, such as the Amazon S3 and Rackspace Cloud Files can be used, given the APIs of the respective data access protocols.

For future work, we plan to further exploit the SG so as to be able to identify software packages. In the literature, a wide range of algorithms have been proposed towards to this goal [21]. The identification of coherent clusters of software resources is also beneficial in terms of locating relevant individual softwares, classifying and labeling them with a set of tags [34]. Last but not least, the SG may contribute in improving the ranking of query results [20]. Ranking is an integral component of any information retrieval system. In the case of software search in large-scale network environments the role of ranking the results becomes critical. To this end,

the SG may offer a rich context of information which is expressed through its edges. A ranking function can be built by analyzing these edges. Kleinberg, and Brin and Page have built upon this idea by introducing the area of link analysis ranking on the Web, where hyperlink structures are used to rank Web pages [9].

## References


1. Enabling Grids for E-SciencE project. http://www.eu-egee.org/ (last accessed February 2010).
2. teragrid. http://www.teragrid.org/index.php (last accessed February 2010).
3. R. Agrawal and et al. The claremont report on database research. *SIGMOD Rec.*, 37(3):9–19, 2008.
4. A. Al-Maskari, M. Sanderson, and P. Clough. The relationship between ir effectiveness measures and user satisfaction. In *SIGIR '07*, pages 773–774, New York, NY, USA, 2007.
5. A. Ames, C. Maltzahn, N. Bobb, E. L. Miller, S. A. Brandt, A. Neeman, A. Hiatt, and D. Tuteja. Richer file system metadata using links and attributes. In *MSST '05*, pages 49–60, Washington, DC, USA, 2005. IEEE Computer Society.
6. G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, 2002.
7. S. Bao, G. Xue, X. Wu, Y. Yu, B. Fei, and Z. Su. Optimizing web search using social annotations. In *WWW '07*, pages 501–510, New York, NY, USA, 2007. ACM.
8. L. Bass, P. Clements, R. Kazman, and M. Klein. Evaluating the software architecture competence of organizations. In *WICSA '08*, pages 249–252, 2008.
9. A. Borodin, G. O. Roberts, J. S. Rosenthal, and P. Tsaparas. Link analysis ranking: algorithms, theory, and experiments. *ACM TOIT*, 5(1):231–297, 2005.
10. F. Brochu, U. Egede, J. Elmsheuser, and K. H. et al. Ganga: a tool for computational-task management and easy access to Grid resources. *Computer Physics Communications (submitted)*, 2009. http://ganga.web.cern.ch/ganga/documents/index.php.
11. C. L. Clarke and et al. Novelty and diversity in information retrieval evaluation. In *SIGIR '08*, pages 659–666, New York, NY, USA, 2008. ACM.
12. S. Cohen, C. Domshlak, and N. Zwerdling. On ranking techniques for desktop search. *ACM TOIS*, 26(2):1–24, 2008.
13. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of OSDI '04: 6th Symposium on Operating System Design and Implemention*, pages 137–150. Usenix Association, December 2004.
14. M. D. Dikaiakos, R. Sakellariou, and Y. Ioannidis. *Information Services for Large-scale Grids: A Case for a Grid Search Engine*, chapter Engineering the Grid: status and perspectives, pages 571–585. American Scientific Publishers, 2006.
15. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3):200–222, 2001.
16. D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. James W. O'Toole. Semantic file systems. In *SOSP '91*, pages 16–25, New York, NY, USA, 1991. ACM.
17. K. A. Gyllstrom, C. Soules, and A. Veitch. Confluence: enhancing contextual desktop search. In *SIGIR '07*, pages 717–718, New York, NY, USA, 2007. ACM.
18. O. Hummel and C. Atkinson. Extreme harvesting: Test driven discovery and reuse of software components. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, IRI - 2004, Las Vegas Hilton, Las Vegas, NV, USA*, pages 66–72, 2004.
19. K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM TOIS*, 20(4):422–446, 2002.
20. H.-Y. Kao and S.-F. Lin. A fast pagerank convergence method based on the cluster prediction. In *WI '07*, pages 593–599, Washington, DC, USA, 2007. IEEE.
21. D. Katsaros, G. Pallis, K. Stamos, A. Vakali, A. Sidiropoulos, and Y. Manolopoulos. Cdns content outsourcing via generalized communities. *IEEE TKDE*, 21(1):137–151, 2009.
22. A. Katsifodimos, G. Pallis, and D. M. Dikaiakos. Harvesting large-scale grids for software resources. In *CCGRID '09*, Shanghai, China, 2009. IEEE Computer Society.
23. S. Khemakhem, K. Drira, and M. Jmaiel. Sec+: an enhanced search engine for component-based software development. *SIGSOFT Softw. Eng. Notes*, 32(4):4, 2007.
24. J. Koren, A. Leung, Y. Zhang, C. Maltzahn, S. Ames, and E. Miller. Searching and navigating petabyte-scale file systems based on facets. In *PDSW '07*, pages 21–25, 2007.
25. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM TKDD*, 1(1), 2007.
26. G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD 2008*, pages 903–914, New York, NY, USA, 2008. ACM.
27. E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi. Sourcerer: mining and searching internet-scale software repositories. *Data Min. Knowl. Discov.*, 18(2):300–336, 2009.
28. A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Trans. Softw. Eng. Methodol.*, 16(4):13, 2007.
29. D. Lucrédio, A. F. do Prado, and E. S. de Almeida. A survey on software components search and retrieval. In *Proceedings of the 30th Euromicro Conference*, pages 152–159, 2004.
30. Y. S. Maarek, D. M. Berry, and G. E. Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE Trans. Softw. Eng.*, 17(8):800–813, 1991.
31. A. Marcus and J. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *ICSE 2003*, pages 125–135, May 2003.
32. M. Matsushita. Ranking significance of software components based on use relations. *IEEE Trans. Softw. Eng.*, 31(3):213–225, 2005.
33. G. Pallis, A. Katsifodimos, and D. M. Dikaiakos. Effective keyword search for software resources installed in large-scale grid infrastructures. In *2009 IEEE/WIC/ACM International Conference on Web Intelligence*, Milano, Italy, 2009.
34. D. Ramage, P. Heymann, C. D. Manning, and H. Garcia-Molina. Clustering the tagged web. In *WSDM '09*, pages 54–63, New York, NY, USA, 2009. ACM.
35. D. Robinson, I. Sung, and N. Williams. File systems, unicode, and normalization. In *Unicode '06*, 2006.
36. C. A. N. Soules and G. R. Ganger. Connections: using context to enhance file search. *SIGOPS Oper. Syst. Rev.*, 39(5):119–132, 2005.
37. J. Teevan, E. Adar, R. Jones, and M. A. S. Potts. Information re-retrieval: repeat queries in yahoo's logs. In *SIGIR '07*, pages 151–158, New York, NY, USA, 2007. ACM.
38. T. Vanderlei and et. al. A cooperative classification mechanism for search and retrieval software components. In *SAC '07*, pages 866–871, New York, NY, USA, 2007. ACM.
39. P. C. Yeung, L. Freund, and C. L. Clarke. X-site: a workplace search tool for software engineers. In *SIGIR '07*, New York, NY, USA, 2007. ACM.
40. A. M. Zaremski and J. M. Wing. Specification matching of software components. *ACM TOSEM*, 6(4):333–369, 1997.