Collaborative indexed search engine over SMB

Andrei Serea (andrei.serea@gmail.com)

Abstract. We propose an extended version of the indexed search that WindowsXP includes in SP3 and its main idea is that the search engine will use indexing and file digests based on the SHA-1 algorithm. The project, named EyeSkyHigh, will be able to search the local area network, indexing and hashing any desired remote resources available by collaborating with other workstations to balance the work load.

Keywords: index, hash, digest, search, LAN.

1. Introduction

It is well known that a standard file search (using a regular expression to match the file/directory name/path) in any operating system can take several minutes or more, depending on the regular expression used and the space searched.

The time required grows substantially when the search is done over an entire file server that typically has several terabytes of storage or, even worse, over the entire file sharing system of a LAN.

The application (which is completely user-space) will have two parts: a user interface to facilitate the search and a background process that will compute the digests for every file on the current machine and run the indexing process.

The main features are:

1. Fast searching in any location within the local machine's data storage unit;
2. Fast searching in any location accessible within the LAN of the current workstation;
3. Parameterized access to CPU for the indexing and hashing process (based on priority rules);
4. Ability to report duplicates in any search done.

The first feature is the one that the indexed search from WindowsXP covers so we won't get into details. The other 3 however are somewhat new.

The second feature implies the following:

- Access to file sharing mechanism;
- Communication between the background processes running on 2 different machines (to exchange the required search information);
- Distributed digests computing and indexing of the shared resources available on the LAN that reside on machines which do not have the application installed. In this case the indexing and digests will be done by other machines on the LAN that do have the application installed. The

workload will be balanced between them. A workstation may refuse to participate in such actions (when a search on the LAN has not been requested by itself);

The third feature must permit the user to limit the access of the background process to system resources. This is extremely important since this process is very expensive when it comes to CPU and memory requirements.

The last feature is a result of the hashing action. Because of this, the application is able to easily detect duplicate files, a very useful option when carrying out a search over an entire LAN.

Let us review the main functions that the application should cover: generate index or hash data for a location (remote or local), search a monitored location (that is indexed and, possibly, hashed), control background jobs priority and access to the system's processor.

To cover all these functions and also ensure extensibility, the application has been divided into 6 separate modules, each designed to fulfill a specific task:

- Indexing engine;
- Hashing engine;
- Search engine;
- Inter-workstation exchange module;
- Parallelism module;
- User interface module;

The job of the indexing engine module is to process requests like "Please create index data for files inside c:\My Documents". Of course the sender of these requests is of no interest to the indexing engine so this is made as transparent as possible.

The hashing engine module works and serves requests in a similar way with the indexing engine. The only difference is that it creates hash data for files found at the given location. The transparency issue applies here also.

Next, the search engine module answers to search requests typically formed by two elements: a search query for the file content and one for the file path and name. It uses information provided by the two data generation modules, the indexing and the hashing engine. The transparency issue applies here also.

As said above, all the engines presented so far (indexing, hashing and searching engines) should not be aware of the entity that sends them the requests, since these can come from two different places: either from the local user or from another workstation that requests index/hash data for a location on the current workstation or performs a search query that includes files on the current workstation.

The parallelism module is designed to serve communication between different workstations. Since this communication implies sending a large variation of requests/responses, the module implements communication at a low level, without having the capability of understanding the content of the messages it sends. This remains the job of the entities that use the module.

As specified in the introduction, the application is capable of searching all accessible LAN locations, as long as index and hash data are first generated. But since not all machines in a LAN must have the application installed, the jobs of indexing and hashing locations provided by workstations that do not have the application installed or, for some reason, refuse to answer to anything else than local requests, must be done by other, willing, workstations. This idea is also suggested by the name of the article itself. The parallelism module was designed to accomplish this task, allowing negotiation of the work balance between active workstations.

The last module's function (User Interface) is self explanatory.

4. Implementation

Each of the modules presented in the previous section gave a particular stage in the implementation process. In the following paragraphs we will discuss only the tricky issues that appeared due to the nature of the problem needed to be solved.

The first real problem is how to keep up-to-date the data generated for monitored locations (index and/or hash). Generating the initial data is not enough since files have the bad habit of changing from time to time. Therefore, for searches to remain accurate, index and hash data should be updated.

Second, this application implies multiple users on multiple workstations and this means concurrency, a delicate problem that should be handled with care.

Also, remember that the application can search any accessible locations via SMB? This means even locations found on workstations that might not have the app installed, therefore no possibility to generate on its own index and hash data or to answer to search queries. Such tasks must be fulfilled by the so-called "active workstations" – the ones that do have the application installed and running – and in such a manner that the workload is spread as equally as possible among them.

Finally, there is the priority issue. What should we do in an extreme situation when 10 other workstations request different data from station A and its user also wants to index 5 locations at the same time? Which of these jobs should be done first? And how do we prioritize them?

All these problems will be discussed and cleared out in this section. But since all the solutions reside heavily on the technologies used, we will first start by tacking a quick glimpse into this area.

4.1. Technologies

There are a couple of questions one must answer before starting to develop such a project.

First, because the application faces some complex problems such as concurrency, work balance, access to the file sharing system and to transport layer protocols, a platform that already covers some of these problems is obviously recommended to be used. Our choice is Eclipse and in the following paragraphs the advantages of this decision will be pointed out.

Second, because the core of this application is the ability to index and search using index data, a choice regarding the indexed search engine to use has to be made. Developing a new one from scratch is out of the question since this field is so heavily researched and covered by many free and open source projects.

Apache's Lucene was the choice best suited for this application and, just like for the previous question, this decision will be explained in the next paragraphs.

4.1.1. Eclipse

Eclipse is an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across its lifecycle.

It offers a wide range of plugins and frameworks that cover almost all possible needs when it comes to software development and all needs for this particular project. These include:

- Powerful and easy to use user interface;
- Management of the file systems (both local or remote);
- Support for concurrent jobs (functions like start/pause/schedule/prioritize);

Another great advantage in using Eclipse is that it is written on top of Java, just as Lucene, making our application cross-platform.

Discussing in detail the critical points reached during the implementation process will point out how Eclipse helped getting pass them.

4.1.2. Lucene

Lucene is a gem in the open-source world: a high-performance, highly scalable, full-featured fast text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. It delivers performance and is disarmingly easy to use.

The key advantages of Lucene that transformed it into the unbeatable solution for this project are:

- Open source project. Having been started in 2002, Lucene comes in package with a large open-source community that outruns the support of almost any commercial software.
- Developed on Java. This makes Lucene a cross-platform solution. Taking into consideration that Eclipse has the same advantage, these two transform our project into a platform-independent solution.
- Fast, stable and reliable. Lucene has proved its capabilities on and on through its years of existence, currently being used in a variety of surprising places: in discussion groups at Fortune 100 companies, in commercial issue trackers, in email search from Microsoft, in the Nutch web search engine (that scales to

billions of pages). It is used by diverse companies including Akamai, Overture, Technorati, HotJobs, Epiphany, FedEx, Mayo Clinic, MIT, New Scientist Magazine, and many others.

Being an open-source project (just like Eclipse), Lucene comes in package with a community that outruns the support.

4.2. Index Update

4.3. Concurrency

4.4. Work balance

To better understand this issue, let us presume that we have a LAN with 10 active workstations and, unfortunately, a very crucial file server that is not active, therefore it is not able to perform indexing, hashing and searching tasks on its own. Any workstation that wants to search in this fileserver must first do a lot of indexing work. But since it is a fileserver, one can affirm that with high probability, other workstations will try at a later time to search the fileserver too. It should be intuitive by now that a good decision in this situation is to split the work between the active workstations

If analyzed, the work balance resumes to only 3 small problems:

1. How to describe the workload of an active workstation, at a specific moment in time?
2. How should the workstations negotiate the "splitting"?
3. When should extra tasks be assigned to a workstation (given its workload)?

Every active workstation has, at a given moment in time, a couple of coefficients that describe its current workload and predict its evolution in the nearby future. The first gives an absolute value of its workload (on a scale from 1 to 100) and the second refers to the predicted decrease of the workload in the next 1000ms.

When a task such as "Index Locations x, y, and z on the fileserver" is created by workstation 1, it broadcasts the task on the LAN to see which other workstations are available to perform it or a part of it. Each active workstation sends a response containing its workload to the workstation 1. Finally, now that workstation 1 has all the workloads of all stations, it can choose the best suited to do the job, it splits it between them and then sends the corresponding part to each one.

A workstation is eligible for extra work (this does not include user requests that have the highest priority and are never refused) only when its workload coefficients do not show a greater than 50% activity for the 10 seconds. This is a factor that has to be chosen based on statistics and acts as a tuning factor for the search engine.

4.5. Job scheduling/priority

5. Performance tests/benchmarks

6. References