Indexed vs. Unindexed Searching:
Distributed Searching • Email Filtering
Security Classifications • Forensics

Both indexed and unindexed searching have their place in the enterprise. Indexed text retrieval is typically more efficient for uses such as general information retrieval, distributed searching and security classification systems. But unindexed searching too has its place—in outgoing email filtering, searching of live data sources like RSS news feeds, and sometimes in forensics. This article will attempt to explain which search technique to use when, and why.

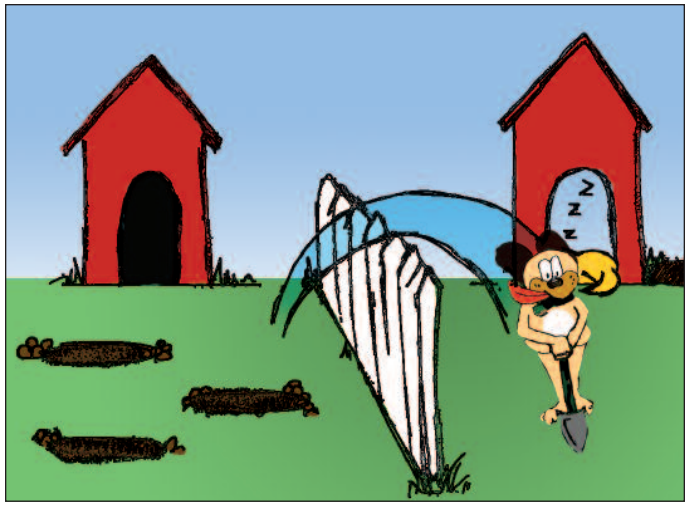
The classification filter relies on what dtSearch® calls a "SearchFilter object" for use while indexing. A search filter object is a mechanism to specify a subset of the documents in a collection for purposes of limiting a query to that subset. Conceptually, the SearchFilter object is similar to a list of document identifiers.

Overview: Indexed Text Retrieval

Indexing the inevitable millions of documents that any sizeable organization generates on shared file servers is the fastest way to facilitate data retrieval. An index will typically store each unique word in a document collection and its location within each document. Indexing also works with non-document data, e.g. for forensics search purposes.

After indexing, full-text search speed, even across

A dog digging in its own backyard to find bones that the dog remembers burying is performing an indexed search.



A dog digging in a neighboring backyard in the hopes of finding some of the neighboring dog's buried bones is performing an unindexed search.

Reprinted with permission of PC AI Online Magazine V. 18 #3
For more information about PC AI Online Magazine, visit www.pcai.com

This discussion uses dtSearch-based examples for illustrating distributed searching using XML as an interchange medium. This is a different use of XML than as a database storage format for holding Web-based and other fielded data ... The distributed search can then combine the streams of XML data, presenting a single, unified and flexible view.

millions of documents, is typically less than a second. While indexing a very large collection of documents for the first time may be time consuming, subsequent updates of the index are usually much faster. dtSearch, for example, simply checks the file modification dates of all indexed files, and only reindexes those files that have been added, deleted or changed since the last index update. (While the text retrieval terminology here relies on the dtSearch product line, the concepts in this article are generally applicable.)

In addition to enabling precision boolean searching, an index can also store such information as word positions, enabling word or phrase proximity searching. An index can also hold information about word frequency and distribution, enabling computation of natural language relevancy rankings across a document collection. If the company name appears in two million documents, it would get a low relevancy ranking. If the latest marketing terminology appears in only four documents, it would get a much higher relevancy rank. In that way, PR could, for example, enter a whole paragraph of proposed text for a press release as a natural language search, and zoom right in on the most relevant documents.

But full-text searching, whether boolean, natural language, or otherwise, is only part of the text retrieval answer. Suppose HR wants to limit its search to documents with an HR *executive* designation. This type of fielded data classification can result from fields or meta data inside a document, or from an overlaying document management-type application. With the latter, fielded data classification can rely on associated database entries, such as SQL or XML, or the addition of fields "on the fly" during the indexing process.

Adding in Security Classifications

Now suppose the goal is to enable searching organization-wide, but to keep the wrong documents out of the wrong hands. For example, suppose documents that bear certain HR designations might contain salary and other confidential information, and so should be kept out of the general information pool—yet still accessible for authorized HR employees who may need that information. Or suppose a company is doing government defense work, and needs, as part of that work, to implement a classified designation, so that only employees with a certain security clearance can review those documents.

One way to implement these types of document classifications is to build separate indexes for separate sets of documents that bear such classifications, and make the search indexes as well as the original documents available only from secure, limited-access servers. But while this type of method may work for very limited designations, it is unwieldy for implementing a complex document classification scheme.

A more elegant security classification solution involves a document filter. Filtering is ultimately a sifting process, the computer equivalent of panning for gold, or separating the chaff from the wheat. The greater the volume of data that requires filtering, the more important it is to automate the filtering process, usually by looking for certain flags or keywords. Hence, the relationship between filtering and text searching.

In this context, the classification filter relies on what dtSearch calls a "SearchFilter object" for use while indexing. A search filter object is a mechanism to specify a subset of the documents in a collection for purposes of limiting a query to that subset. Conceptually, the SearchFilter object is similar to a list of document identifiers.

The SearchFilter object has the flexibility to integrate with more complex security settings. The security settings themselves

A text retrieval program like dtSearch typically receives live data streams, such as RSS news feeds, through a programmable data source interface. Searching this type of data usually involves scanning for a series of pre-established queries, much like email filtering. And like unindexed email filtering, finding the relevant key words usually triggers certain processes, such as sending an email notice after a "hit."

can exist on various levels. To use SearchFilters to implement security, an application would first create a series of SearchFilters, one for each security category, based on information in the database (or other repository) that specifies security rights. When a user submits a query, the application would select the SearchFilter that corresponds to the user's security category and attach it to the search. When the query executes, it will only return documents that the SearchFilter permits.

Filtering of Outgoing Emails

Email filtering typically scans for certain combinations of terms that represent knowledge that should not leave the organization. For example, suppose *projectx phase 2322* is a top secret item. Email filtering could flag any email body or document attachment that contains *projectx* or *2322*.

Since email filtering is a one-step operation—either the email proceeds or it does not—unindexed searching is usually the more efficient way to process the search. While unindexed searching is much slower than indexed searching, it is faster to do a single unindexed search than to build an index and then do a single search. And the more advanced relevancy ranking features that indexed-only searching provide tend to be less important in searching through individual emails and attachments than in searching through millions of documents in a document repository.

The most important special feature for email filtering is fuzzy searching to pick up typographical errors. In dtSearch, fuzzy searching works fully with unindexed searching as well as indexed searching. With fuzzy searching on, a search for *phase 2322* would also, for example, retrieve a possible misspelling in the form of *phase 2332*.

Typically an organization will embed unindexed searching capabilities into a custom application. Once the system flags an email, the system could trigger a warning to the sender. For example, the warning might say: did you really mean to send out an email mentioning *projectx phase 2322*?

The intent of the email filtering outlined above is not to catch willful abuse of company secrets, so much as to pick up casual misuse. Examples of such casual misuse might be an email sent in haste that could wind up in the wrong hands, or even the accidental attachment of the wrong file to a message. In other words, email filtering protects against ordinary human error. It does not, for example, protect against the employee who copies the *projectx phase 2322* files to portable media, and walks out with this under a jacket.

In addition to scanning outgoing emails and attachments, an organization may also want to search emails and attachments as archival records. In that case, indexing the emails, enabling the repeated instantaneous search of the knowledge stores that the emails and attachments represent, is the more efficient retrieval method.

Searching Data Outside the Enterprise Borders

Whereas the other levels above relate to data internal to an organization, this discussion relates to data outside the organization. An example is information that resides on the Internet, ranging from competitor Web sites, to regulatory Web sites. For such information, the answer is spidered indexed searching. The dtSearch Web spider, for example, can access text in any named Web site to X levels of depth, and even follow links off the site to related sites.

Indexed searching—and XML as the interchange language for synthesizing

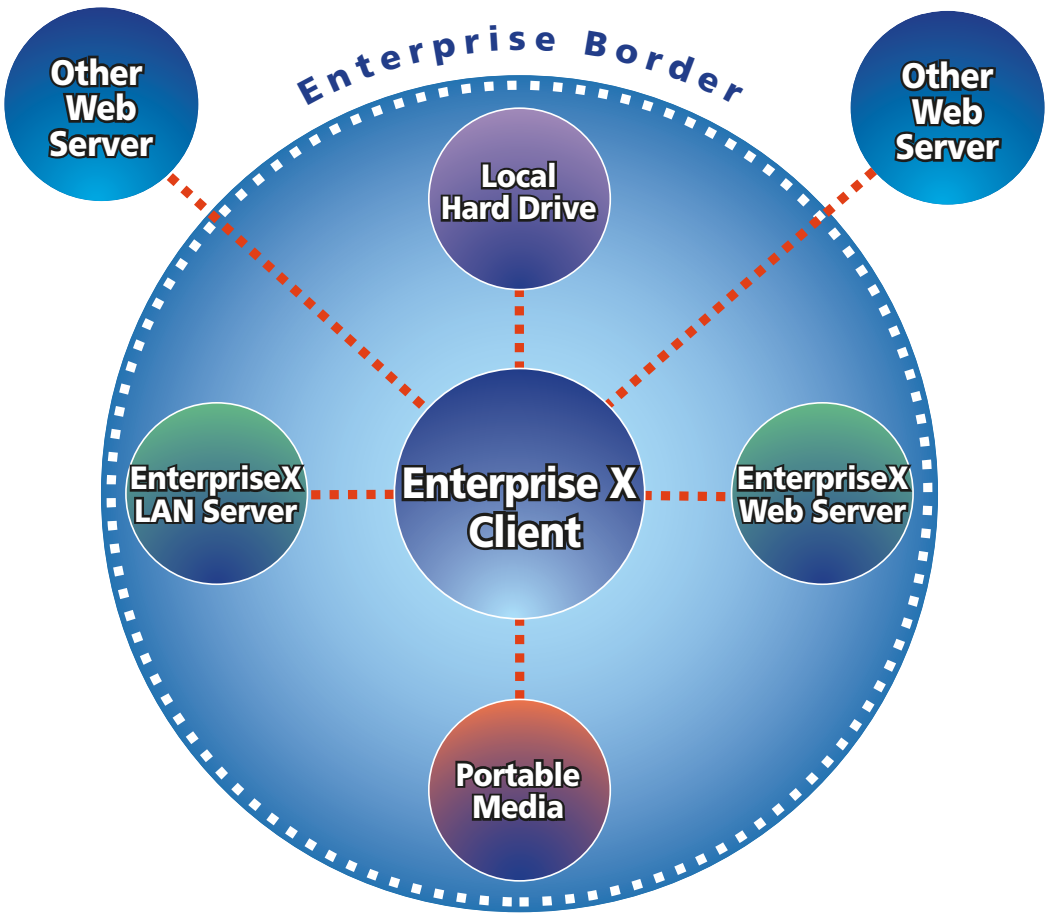
A good guideline for both indexed and unindexed searching of forensically retrieved data is to search twice: once with a file format filter on, and once with a file format filter off. In dtSearch, for example, file format filtering would represent the default for searching. And searching without file format recognition would correspond to a binary data search.

search results as in dtSearch—enables display of data with highlighted hits and links and images intact for popular Web-based formats, such as dynamic content ASP.NET, as well as PDF, HTML and Web-based XML. In fact, such searches can look to the end-user just like searching a local file server, and search results can even display internal and external retrieved content in a fully integrated way. See *Distributed (Indexed) Searching: Evolution to XML* (next page) for additional details on this usage of XML.

While this type of indexed searching does not, except for the spidering component, differ significantly from indexed searching of retrieved files inside an organization, another type does differ considerably in structure. Monitoring of live data streams such as RSS news feeds for relevant search queries would be an example of the latter type. In that case, there is no "file" to perform unindexed searching on, such as with email messages and attachments, or the above spidered Web site content example.

A text retrieval program like dtSearch typically receives live data streams, such as RSS news feeds, through a programmable

Distributed Searching



The distributed search returns to dtSearch multiple streams of XML data, which the application then combines and presents to the user in a single, unified view. Retrieved HTML, PDF, XML, ZIP, MS Office, etc. files appear with highlighted hits, as well as (for HTML, XML and PDF) images, formatting and links intact. A distributed search can also support hit-highlighted treatment of dynamically-generated content, like ASP.NET and SharePoint.

Reprinted with permission of PC AI Online Magazine V. 18 #3
For more information about PC AI Online Magazine, visit www.pcai.com

The most important special feature for email filtering is fuzzy searching to pick up typographical errors. In dtSearch, fuzzy searching works fully with unindexed searching as well as indexed searching. With fuzzy searching on, a search for *phase 2322* would also, for example, retrieve a possible misspelling in the form of *phase 2332*.

data source interface. Searching this type of data usually involves scanning for a series of pre-established queries, much like email filtering. And like unindexed email filtering, finding the relevant key words usually triggers certain processes, such as sending an email notice after a "hit."

Since scanning of live data sources is typically a one-pass filtering operation, unindexed searching is usually the more efficient search method. For subsequent, often repeat, "on demand" searching of a data repository of news feeds, the balance, however, shifts to indexed searching. And the same indexed searching considerations that apply to a general information management repository would likewise apply.

Searching Forensically Retrieved Data

When you delete a file, the data remains; the computer just marks it as deleted. Both "undelete" programs and forensic investigation tools work to recover such deleted files. But that, for retrieval of forensically-recovered data, is only the beginning. If an investigator is examining a stack of harddrives, and not sure whether one or more are even relevant to an

Distributed (Indexed) Searching: Evolution to XML

Conventional browser-based searching returns search results as an HTML stream. Returning results in XML, however, makes for much smarter search results.

HTML tells how to display the data, not what the data is. XML tell what the data is, not how to display it. While HTML simply paints a picture of what search results look like, XML can include numeric values, such as number of hits and word offsets of hits, indicating where hits are in each document.

To illustrate, assume a distributed search of six servers, some local, some remote. With HTML, a distributed search returns six different search results, with no method for combining them. With XML, a distributed search returns six streams of XML data.

The distributed search can then combine the streams of XML data, presenting a single, unified and flexible view. At the user's request, the XML data can instantly resort search results from, for example, ascending date order to descending hit number. Retrieved Web-based files appear just as they would in a Web browser, i.e. including all embedded links and images, and the addition of highlighted hits.

A retrieved PDF file would look similar to other retrieved Web-based files, including all existing embedded images with highlighted hits. In the case of the PDF file, however, the server transmission includes an XML component along with the underlying file. The server first sends links to the original PDF file, followed by links to an XML file describing where the hits are as page and character offsets. Adobe Reader, operating through the browser, downloads the XML file with the hit offsets for highlighting hits.

Reprinted with permission of PC AI Online Magazine V. 18 #3
For more information about PC AI Online Magazine, visit www.pcai.com

Indexed searching—and XML as the interchange language for synthesizing search results as in dtSearch—enables display of data with highlighted hits and links and images intact for popular Web-based formats, such as dynamic content ASP.NET, as well as PDF, HTML and Web-based XML.

investigation, unindexed searching is often a good first-pass tool. Maybe the PC from the dumpster really did contain only soup recipes, and has no bearing on the matter at hand.

An initial unindexed search can scan the PC for certain key words to determine threshold relevance. Often, however, forensic investigators are searching not for just a couple of terms, but for hundreds of terms. For query strings consisting of hundreds of terms, building an index and then searching is generally more efficient than scanning the files for all of these terms in an unindexed search. In any case, following a determination or even a suspicion of threshold relevance—so the PC was not just for soup recipes!—the balance clearly shifts towards indexed searching.

Whether indexed or unindexed, searching generally needs to apply a file format filter to most current computer data. Popular file formats such as PDF, MS Word, and MS Excel store text in such a way as to make the text often unrecognizable in raw form. For example, *abcdefghijklmnopqrstuvwxyz* could look as follows in a PDF file: *8 0 obj<</Length 57/Filter/FlateDecode/L 69/S 38>>stream xÚb```c``α`....*

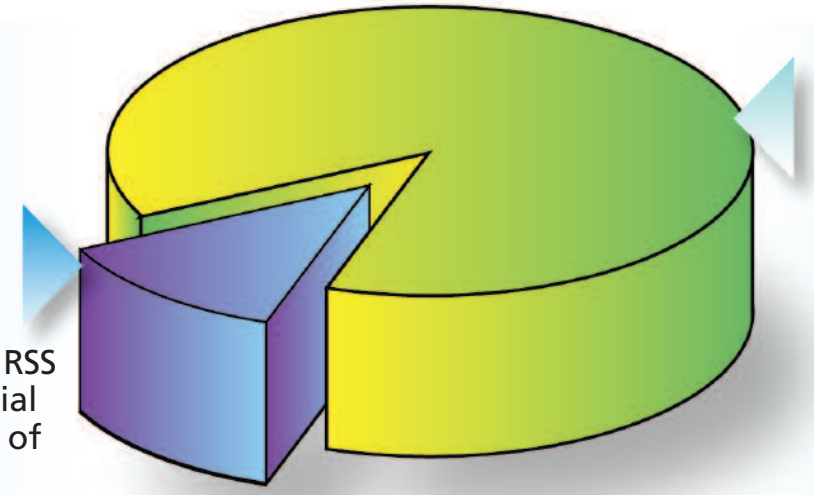
Without a file format filter, therefore, a search would almost certainly miss a lot of data. But just searching data applying a file format filter can miss data that appears only in raw form. For example, "slack" space, such as the space between an end of a file and the end of the allocated sector, can hide data. Hence, a good guideline for both indexed and unindexed searching of forensically retrieved data is to search twice: once with a file format filter on, and once with a file format filter off.

In dtSearch, for example, file format filtering would represent the default for searching. And searching without file format recognition would correspond to a binary data search. An intermediate step in dtSearch, called "filtered binary," would, while viewing all retrieved data as binary, attempt to programmatically sift out the text, and leave behind what looks like formatting data.

Please visit dtSearch online at www.dtsearch.com

Unindexed Searching

Works best with single-pass operations, like:
- email filtering
- searching live data feeds like RSS
- making an initial determination of relevance in forensics



Indexed Searching

Use with everything else

Reprinted with permission of PC AI Online Magazine V. 18 #3
For more information about PC AI Online Magazine, visit www.pcai.com