RESOURCE CONSUMPTION AND PERFORMANCE CHARACTERISTICS OF ISYS

ISYS Search Software Worldwide

United States
ISYS Search Software Inc
8775 East Orchard Road
Suite 811
Englewood, CO 80111
USA

Phone: +1 303 689 9998
Email: info-us@isys-search.com
Fax: +1 303 689 9997

Australia
ISYS Search Software Pty Ltd
Suite 102,10-12 Clarke St
Crows Nest NSW 2065
Australia

Phone: +61 2 9439 5800
Email: info-au@isys-search.com
Fax: +61 2 9439 8569

United Kingdom
ISYS Search Software (UK) Ltd
The Steam Mill
Steam Mill Street
Chester CH3 5AN
United Kingdom

Phone: +44 (0) 1244 313 216
Email: info-uk@isys-search.com
Fax: +44 (0) 1244 313 003

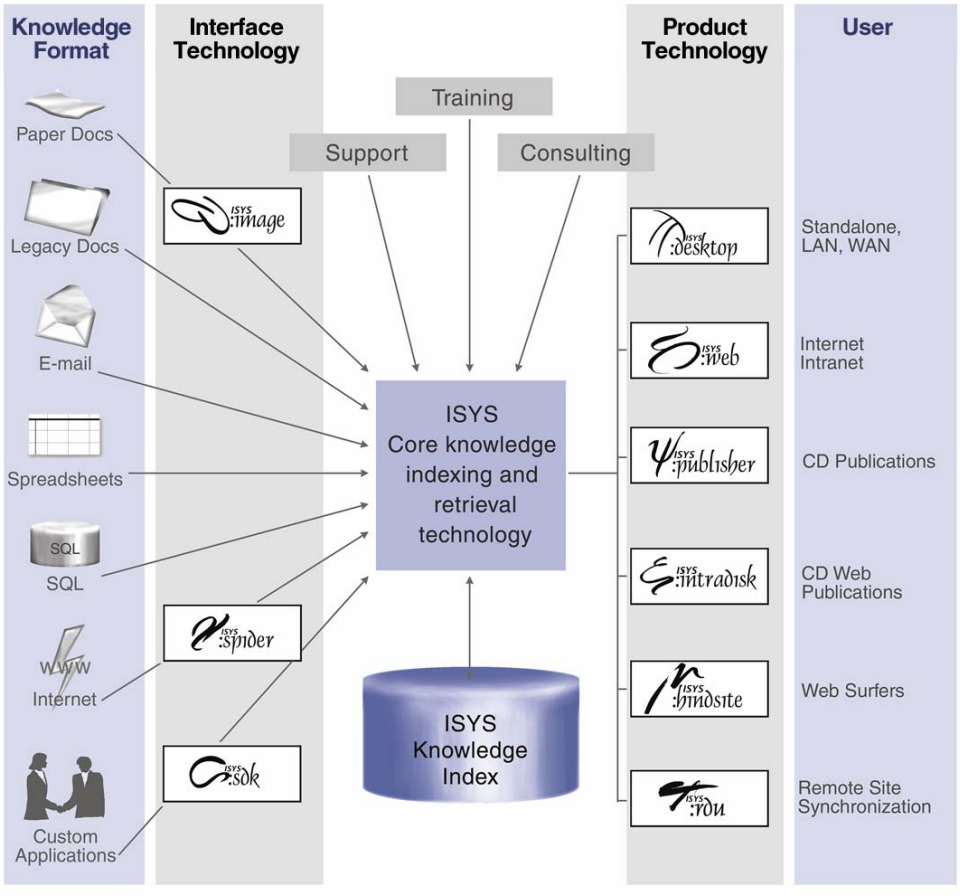
CONTENTS

INTRODUCTION

Organizations planning an ISYS deployment often wish to know more about the architecture, performance and scalability factors of ISYS in order to better prepare themselves. Frequently, new customers are concerned about bandwidth issues, CPU loading or the like. This document seeks to explain more about the technical implementation details of ISYS so that administrators will better understand how ISYS can deal with large data volumes and still maintain performance.

PRODUCT RANGE ARCHITECTURE

The ISYS product range consists of a series of interlocking products which can be utilized depending on the type of knowledge you want to manage, and the scenario in which each will be employed, as shown by the following diagram.

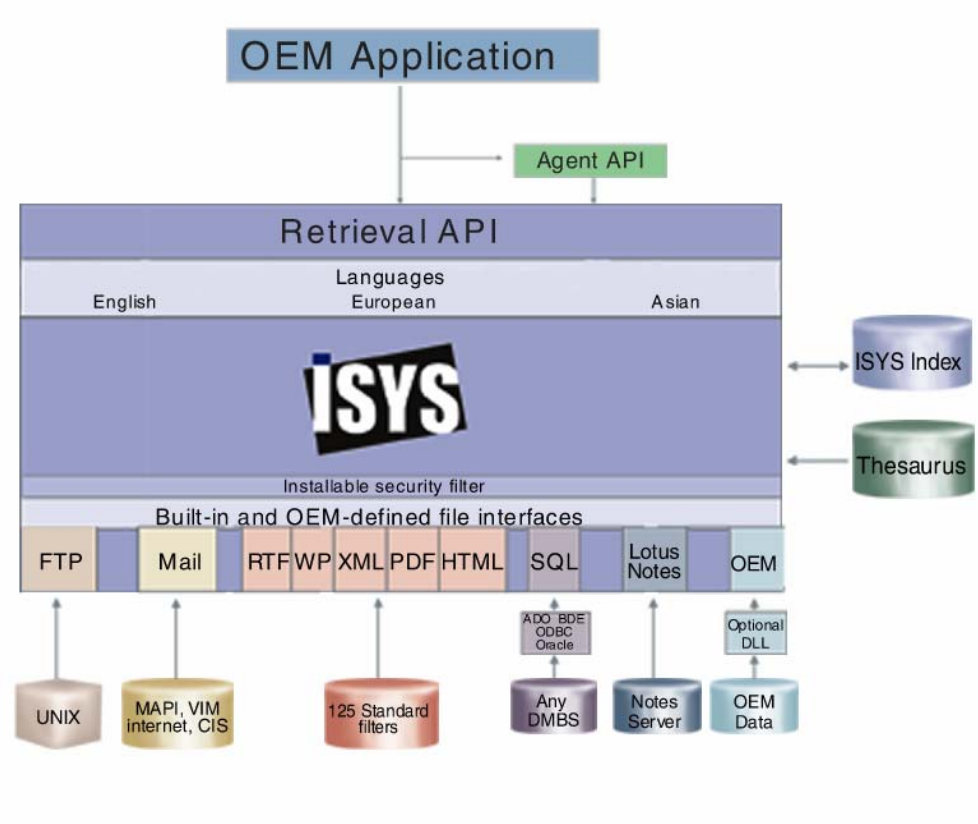


All products are based on the core ISYS engine, and are thus fully inter-operable. Additionally, OEM customers may choose to license the core ISYS engine directly for use in their own in-house or commercial applications.

All ISYS products are optimized to run on Intel-based (or compatible) computers running Windows family operating systems, but are able to manage knowledge stored on a wide variety of platforms, including Unix and mainframe.

ENGINE ARCHITECTURE

The ISYS engine is a highly-optimized, integrated indexing and search tool. It provides a homogenous interface to a heterogeneous world, and is fully polymorphic in its ability to perform identical search operations on completely disparate data sources. For example, Oracle databases, three-dimensional spreadsheets, PowerPoint presentations, external websites, mainframe print files and Microsoft Office documents can all be treated in the same way. Users (and applications) do not have to be aware of the underlying nature of the data unless they want to be.



GENERAL VERSUS SPECIFIC

For knowledge management products in general, and ISYS specifically, there are many, many variables that can affect any performance and scalability scenario. An example of these that affect the source data include:

- The breadth of vocabulary in the index
- The number of occurrences of each unique word in the index
- The shape of the curve if one were to graph the above relationship
- The minimum, average, maximum, and median size of individual documents
- The format of the documents involved, for example, ASCII versus Word
- Index granularity

An example of factors that affect the use of the index at query time include:

- The number of words involved in queries
- The number and types of operators (proximity, Boolean, etc) involved in queries
- The use of features such as fuzzy precompensation
- The presence and usage of synonyms or thesaurus
- Wild cards or tense conflation, wide or narrow
- The use of broad-scope queries versus narrow-scope (fielded) queries
- The size of final results
- The size of intermediate results
- The form in which results are displayed
- The proportion of found documents that are subsequently browsed
- The number of queries per hour, and the characteristics of those queries
- Hardware performance

In theory, if precise answers were known to all of the above factors (and more), then a precise resource consumption answer could be given for any scenario.

In practice, however, exact answers are rarely available. More often than not, this is because the *most unknown* factor is the behavior of users. Nowhere is this more true than with a new implementation where you are deploying a totally new information resource, and cannot predict the extent to which it will be taken up, and what type of queries and other behaviors will be involved.

The net effect of this is that the information contained in this document should be considered **indicative only**, and the details will depend on your own unique situation. The guidelines and issues discussed in this document are universally applicable, but your specific **mileage may vary**.

LANGUAGE HANDLING

ISYS normalizes all characters to a common form, based on the active code page of the machine. Unicode data, when encountered, is transparently converted to the common form so that searches succeed regardless of whether the target data was expressed in Unicode or single-byte.

ISYS provides built-in handling of all single-byte languages (English, German, French, Arabic, Greek, Dutch, Spanish, Vietnamese, Malaysian, Cyrillic, etc), as well as multi-byte languages (Korean, Japanese, Traditional and Simplified Chinese).

Language options are configured on an index-by-index basis, and automatically take appropriate defaults when a machine is detected as having language settings other than English. For European languages, indexes may be configured to either respect diacriticals (the default), or to search as though the diacriticals were not present.

CLIENT-SERVER VERSUS SINGLE-TIER

ISYS is available in both client-server (ISYS:web or ISYS:web.asp) and single-tier (ISYS:desktop) configurations. The ISYS:sdk may be used to create applications in either architecture.

The choice between configurations should be made in light of “bigger picture” considerations within your organization – whether your strategic preference is thin client or applications running directly on the desktop. Performance should rarely be a deciding architectural factor for the reasons discussed herein.

Client-server architecture offers many advantages for conventionally structured relational database systems. The key benefit is its ability to perform an application’s resource-intensive processing on a server in close proximity to the data. Consequently, a relatively small result set is sent from client to server, which minimizes network traffic.

For example, consider a traditional database application which needs to access a relational table containing 100,000 records, select 100 of those records, and summarize them into 20 lines. In a client-server environment, the client sends the request across the network to the server, the server does all the work, then just sends back the 20 summary records the client application really wants to see. In a non-client-server environment, there would be 100,000 records flowing across the network, almost all of which would be discarded by the client as unwanted records. In this conventional scenario, client-server computing offers clearly superior utilization of network resources.

For knowledge management applications, however, client-server architectures are not automatically superior in this respect. Unlike structured databases, knowledge management products such as ISYS typically do not perform large-scale searching or summarization. Certainly, processing is required to apply proximity searching and Boolean operators, but the amount of I/O is generally dramatically less than that performed by a relational query on a structured database.

Searching in a knowledge management system is confined to a relatively small index file, and is not a search of the text itself.

The ISYS index is designed so that, generally speaking, no data is read from the index which is not subsequently used. ISYS never has to *search* its index, rather the index is structured such that it always knows exactly where to go in the index to find a particular result.

Thus, although client-server architecture may often be desirable for other reasons, the performance issues associated with traditional relational database client-server architecture do not apply to fully inverted text indexes such as ISYS.

It is also worth noting that it's entirely valid to make use of both approaches, for example running ISYS:web and ISYS:desktop alongside each other.

NETWORK IMPACT EXPECTATIONS

When considering any aspect of ISYS network performance, there are really two quite distinct situations: maintaining the index versus querying the index.

Performance during index maintenance tends not to be the most important aspect, since maintenance occurs relatively rarely (compared to the frequency of queries), and there is often some flexibility as to the time of day in which it is performed.

The first law of knowledge management is that good performance during queries is paramount, because queries happen so often and because this is where the benefit of products such as ISYS accrues.

The guideline here is that ISYS imposes a much lower load on your network during searches than you probably expect. You may wish to refer to the document “Resource Consumption and Performance Characteristics” for details.

NETWORK TRAFFIC TUNING ESSENTIALS

The most important rule when implementing ISYS on any form of network is to be aware of and avoid the performance penalty for crossing a slow network segment.

There are two angles to this. One is from the ISYS engine to the user, which obviously depends on whether you are running the ISYS engine locally or as a remote server. The other is from the ISYS engine to the disk-based information it accesses. This section discusses the latter.

Typically, the ISYS engine (whether running locally or as a server) will be accessing information on a hard disk which may either reside in the same machine as the CPU on which the engine is running, or on a disk elsewhere on a LAN or WAN. Usually, the effective speed of the disk volume will *not* be clearly indicated in the means by which you access it. In other words, a drive may be accessed via a mapped drive letter (for example, “R:”), or via a UNC (“\\server\data”), and in either case may be a fast local disk or a much slower remote disk. The means of identification tells you nothing about the speed of access.

The trick, therefore, is to do as much I/O as possible on the local disk. At first glance, this may appear to be a highly complex task that is best left in the hands of the software vendor. In fact, there are a number of simple deployment and implementation factors that put these issues simply and firmly under your control.

INDEXING REMOTELY

Ideally, your data being indexed, your indexes, and the indexing process will all be running on the same machine with locally attached devices (not network shares). *See* Figure 1.

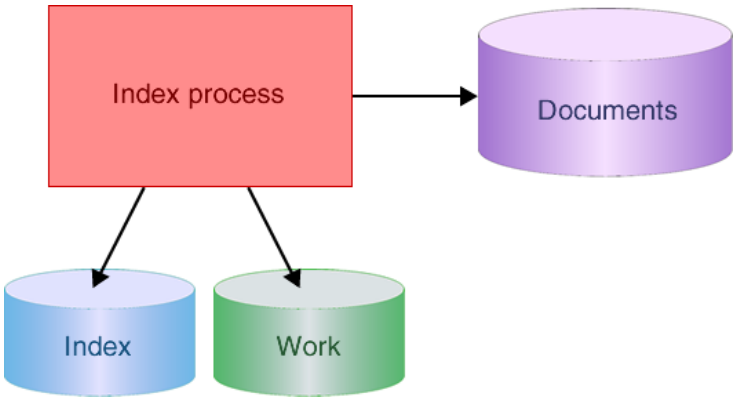


Figure 1: Ideal Performance.

In reality, such a simple arrangement is rarely the case. The processing dynamics of ISYS indexing are such that very different disk I/O load considerations are present for the different types of I/O. Specifically, I/O can be broken into the following categories

- Document I/O
- Index I/O
- Work file I/O

An ISYS indexing run consists of a period where the document files are read and the text is extracted, periods where extracted and transformed text is flushed to work files, and periods where the work files are used to update the ISYS index.

Typically, the document I/O is negligible compared to everything else. Even for very large documents, the I/O involved in reading is rarely a performance consideration because ISYS reads each document once, from start to finish, and that's all.

Work file I/O is far more intense because the work files are revisited during the indexing run and accessed non-sequentially. By default, indexing work files are created in the same directory as the index itself.

Index I/O is the most intense of all, particularly when an existing large index is having a large amount of new text added to it. For particularly large data volumes, the work files may be processed into the index several times during a large indexing run.

The guideline to derive from this is that it is far more important to have your indexes (and work files) on a fast device locally attached to the CPU running the indexing process, than it is your documents. Put another way, it is almost always fine to build an index of remote documents (Figure 2), but it is rarely acceptable to build a remote index of documents, be they local or remote (Figure 3).

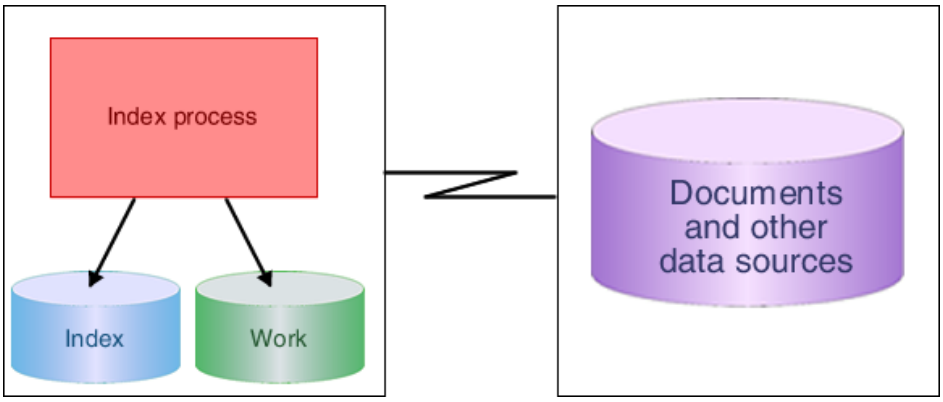


Figure 2: Very good performance

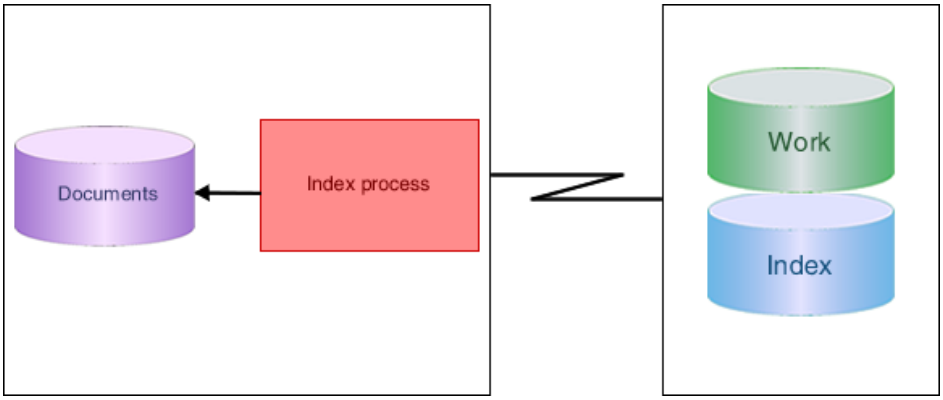


Figure 3: Very poor performance.

The solution, of course, if you really need to construct an index remotely, is to also run the indexing process remotely.

If the scenario is not one of remote indexing, but of merely building an index which resides on a slow volume, then great benefit can be reaped by leaving the index and indexing process where they are, and relocating the work files to a faster device.

INDEX SCHEME OVERVIEW

ISYS uses a fully-inverted index architecture, supplemented by a series of proprietary ancillary structures.

ISYS indexes are generally self-managing, in that for normal operation they do not require regular compaction or rebuild utilities to be run. They reuse their own space, and store information in a highly compact form.

There are four main files that comprise an ISYS index:

- IXC One fixed length entry per indexed document, containing document name and other details.
- IXD Optional fuzzy precompensation index.
- IXB Word and document index, performance-aiding information, concordance pointers.
- IXA Index into the IXB.

The relative sizes of these files will depend on your exact details, for example, your average document size and vocabulary spread. Growth characteristics are described in the document “Resource Consumption and Performance Characteristics”.

The powerhouse is the IXB file, which contains an index of unique document names, unique words, a series of bookmark entries to assist in navigating large documents efficiently, and the concordance pointers which represent the inverted index. The IXB file is logically a self-managing, variable length, direct access block file.

Each concordance pointer is a three-part hierarchical pointer, holding document number, paragraph number within document, and word number within paragraph. The concordance pointers are run encoded, and then compressed. The unit of space allocation is the nybble (4 bits). Each pointer is of variable length, and each subfield of the pointer is of variable length. Concordance pointers may vary in size from 1.5 bytes (12 bits) up to 9.5 bytes (76 bits). For any given index, there would be a great diversity of pointer sizes, but an average would be in the range of 3 to 5 bytes.

Concordance pointers are blocked into the IXB, and stored backwards so that new documents can be placed on the near end of the chain rather than the far end. A key performance indicator for inverted index search engines is the number of word references which can be retrieved per physical I/O. For ISYS, this is typically in the range of 300-400. In other words, when you search for a particular term, ISYS can retrieve details of around 400 occurrences of that term, no matter where they appear, for each physical I/O.

Note that retrieval time is a function of how many concordance pointers are to be retrieved for a given search term, not total index size.

EFFECT OF CD OR READ-ONLY OPTIMIZATION

When an index has been processed by ISYS:publisher into a CD Optimized index, the size of the IXC file is reduced by a fixed percentage, and the IXA file size is reduced to a placeholder. This is because the machinery within the index to self-manage and re-use space has been removed.

In the case of a CD Optimized and compressed index, the discrete IXB blocks are subject to individual compression. Three separate compression schemes are employed depending on the class of IXB block involved, and the schemes are adaptive.

Additionally, optimized indexes are rearranged to make the logical and physical block clustering coincide, thus minimizing changes of direction in disk head movement.

INDEXING PROCESS OVERVIEW

When an ISYS index is created, various configuration parameters are read from their respective files and loaded into the index files. These include:

- Significant and insignificant character lists
- Common words
- Intelligent date, number and dot handling

These items are initialized into the index files so that if the configuration is altered after the index is created, the settings used will be those that were in effect at index creation time. For these settings, it is crucial that they do not alter during the life of the index, as they affect the way existing data should have been interpreted. Configurations may be changed during the life of the index, but require a simple REINDEX in order to take effect.

For ISYS:sdk usage, there are a variety of methods of updating the index:

- Using rule-based indexing, and performing an IDB_Function("UPDATE"). This method automatically reconciles the state of the indexes to the current document set, and is the method used by ISYS:desktop and ISYS:web.
- Using transactional indexing, and applying a journaled transaction file by calling Process_Transactions.
- Using the low-level indexing API to supervise index activities at a very detailed level.

The details contained herein primarily relate to the first (rule-based) method. When an index update is performed, the processing consists of three parts:

- Enumerating the document space and comparing what is found against what is in the index, in light of the rules which describe what should be in the index. This produces a list of documents to be added to, updated in, or removed from the index.
- Opening each document, reading the words, and effectively performing a matrix transposition of the word/location relationship. A document is a list of words presented in the order they occur, and in this phase, ISYS transforms this into a list of words presented in word order, along with positional information. This information is collated into a compressed form in memory, and written to disk-based work files as required.

- The final phase involves opening the index files for write purposes, and updating the concordance chains based on the contents of the work files. This may be highly I/O intensive.

The first and second phases are interruptable, that is, they may be terminated without any implications. The final phase is atomic, and must be allowed to fully complete in order to obtain a logically consistent index. The ISYS engine ensures that this process runs to completion, barring apocalyptic events.

De-indexing is a separate matter, and is outside the scope of this document.

INDEXING EFFICIENCY

Essentially, indexing performance is a function of the amount of information to be indexed, the existing index size, the vocabulary characteristics, and the format of the data.

In the simplest case of ISYS indexing ASCII files, there is no additional layer of software between ISYS and the text. In the case of a Word document, for example, not only must ISYS navigate through the complexities of a Word binary file, but also call Windows OLE to access the underlying storage. Similarly, with SQL-based data, there is a DBMS between ISYS and the text. The overhead imposed is variable, and may be a factor of two, depending on your details.

Indexing performance is best measured in words per minute. This is because some document formats are more efficient than others. For example, a megabyte ASCII file contains rather a lot of text, whereas a megabyte Word document could potentially contain very little.

Performance depends on many factors, including vocabulary and indexing options selected. As an indication, on 2Ghz machine, ISYS has been measured reading and indexing four million words per minute. For ASCII files, this is in the order of 25Mb/min. For Word documents, this might be more like 100-150Mb/min.

For ISYS, indexing efficiency comes down to economies of scale. ISYS is designed to work better with large data volumes. For example, it is better to index 10,000 documents at once than it is 100 batches of 100 documents. It's even better to index 1,000,000 documents at once. This is because, after the matrix transposition is performed, effectively all documents are being processed at the same time.

The effect of this is so extreme, that you may consider that every additional document you want to index is being done at no cost. Obviously this is not really the case, but the relative cost of indexing an extra document in a large indexing run versus starting a new indexing run is so pronounced that this is the correct mindset to adopt.

INDEX VOLUME AND SCALING

Each ISYS index may contain up to 32 million documents, each of up to 65,535 paragraphs, each of up to 65,535 words. Orthogonally, this implies a maximum document size of two billion words. If a single paragraph exceeds 65,535 words, then ISYS will insert its own virtual paragraph

marker. If a single document exceeds 65,535 paragraphs, then information will still be fully indexed, but proximity searching may be compromised.

Up to 64 indexes may be chained at once for simultaneous searching, which allows for a maximum of two billion documents being covered by a single search, providing for a theoretical maximum of four quadrillion words (four thousand billion words) in a single search. In practical terms you are unlikely to have the maximum number of documents, all exactly of the maximum allowable size.

Chains of chains may also be chained, as long as the total number of indexes does not exceed 64. ISYS automatically resolves loops, recursive whirls and duplicates where complex chains of chains are involved. Chaining imposes a small fixed overhead proportional to the number of indexes in the chain, not the number of documents involved. The overhead of resolving the chain is sufficiently small as to be insignificant for most practical purposes.

Indexes maintain very good retrieval performance as size increases, due to the factors described previously in this document. Retrieval time remains a function of the number of references found, not the total amount of information in the index.

Index maintenance performance will degrade slowly as index size increases, until it reaches a plateau. Updating in larger batches helps avoid the effects of performance decline as indexes grow, because the critical measure is the size of the index at the *beginning* of the update batch, not the end. Appropriate use of index granularity, leveraged off the knowledge of the specifics of your scenario can help to keep index maintenance performance fast.

INDEX GRANULARITY

While it might seem simplest to have one monolithic index containing all your information, in practice it is simplest to implement a level of index granularity that reflects the natural granularity of your data. This aids data management issues, and also assists with performance.

For example, consider the scenario of new documents being created on a daily basis which need to be collected and searched en-masse. This might be capturing information from a news feed or indexing reports manually created by field officers.

The best arrangement, rather than to place all your text in a single subdirectory holding hundreds of thousands of files, would be to have a directory structure that reflects the time-based nature of the data. For example, one directory per year, with one directory per month underneath, and perhaps one directory per day underneath that. You would arrange for new data to be created in the appropriate directory as time passes, and this would let you more easily perform backups and archiving.

In this sample scenario, your ISYS index structure would reflect the same architecture to some degree. For example, you might have one ISYS index per year (even though your data granularity is by month or day). ISYS will only update one index at a time, but can easily search multiple indexes simultaneously by chaining (statically or dynamically). Alternatively you might have one ISYS index per month. You may even have one ISYS index per month for the current year, but one index per year for past years.

The advantages of appropriate index granularity are:

- Because each index is smaller, they are more convenient to move around, backup and generally work with from a file management perspective.
- If you want to maintain only a "window" of data (for example, the last five years), then "dropping off" old data simply consists of archiving to tape and no longer referencing that index in the chain. If you used a monolithic index, then you would have to perform a large amount of work to actively de-index the old material.
- Because each index is smaller, incremental updates will be processed faster.
- If you do want to search some ancient data archived to tape, then the data and the appropriate portion of index travel together.
- If an index becomes corrupt for any reason, you only have to rebuild the corrupted portion, which will take less time.

Even if your situation does not involve chronological data then granularity should still be considered. Most data has a natural granularity, be it geographic, functional, chronological or other. Working with your natural granularity makes life easier. Working against it makes life harder.

ENUMERATION EFFICIENCY

In most scenarios, indexes will be updated by the simple approach of using ISYS Utilities or by issuing an IDB_Function("UPDATE") call, as opposed to a more specific transactional or low-level approach. In these situations, as described in the section *Indexing Process Overview* (page 12), ISYS begins by enumerating the document name space (be it disk files, email, SQL, etc) in order to determine what changes need to be assimilated into the index.

In most situations, the time spent enumerating the document name space will be brief compared to the time spent updating the indexes. However, you may have a scenario in which the enumeration time becomes a substantial proportion of total processing time, for example if you have a very large number of documents, very few of which are new or altered each update run.

The default settings of ISYS will give perfectly acceptable performance, but if you wish to leverage your situational knowledge to enhance performance, there are four avenues open to you:

- Where possible, use explicit indexing rules rather than AUTO. AUTO requires ISYS to read the first kilobyte of each file to ascertain its format and this I/O, although small, adds up over a large number of files.
- Where AUTO is still being used, make judicious use of BYPASS rules to let ISYS know which files it doesn't need to consider. For example, if you have a mixture of Wordperfect and Word documents spread through a directory tree with no naming conventions, plus a bunch of ".XYZ" files which do not contain text, then use an AUTO rule for the documents but precede it with a BYPASS rule. This is true even for things like TIF files, because following a blanket AUTO rule ISYS will inspect the TIF file just to make sure it's not a document with an unusual extension. If you can confidently state that everything with a TIF extension is definitely not a document, then tell ISYS.
- Where large portions of your documents are known to be static, make use of the STATIC directive in your ISYS.CFG to completely short-circuit enumeration of those sub-trees during UPDATEs.

- Where appropriate, use index granularity to ameliorate the situation.

As is usually the case with ISYS, most anticipated performance problems never arise, so it is best to take the simplest approach and only use these techniques if and when required.

ISYS AND VIRUS SCANNERS

Running a virus scanner will most likely have a dramatic negative impact on ISYS indexing performance. Most virus scanners intercept file open requests and automatically step in and scan the file to ensure it does not contain a virus. An ISYS indexing run is likely to open many files in order to index them, and the added overhead of your virus scanner could have a substantial impact on performance.

This is especially true where AUTO is being used and where your indexing rules cause ISYS to inspect a large number of files which end up not being included in the index (non-document files), since ISYS has to open each file to determine its contents. The guidelines in the above section on Enumeration Efficiency will help you prevent ISYS from accessing files it doesn't need to, and the benefit reaped from this will be even greater where virus scanning software is being used.

The best approach, however, is to configure your virus scanning software to not check for viruses upon file access while an ISYS indexing run is proceeding.

DOCUMENT GRANULARITY

It is a recurring theme in this document that the details of the performance you will see depend on the specifics of your situation. One very basic measure is that of document granularity. Taken to extremes, this refers to the difference between having an index that consists of a single mammoth document, versus having hundreds of millions of documents containing only one or two words each. The vast bulk of normality lies somewhere in between.

The nature of your data will dictate a natural granularity, and it usually pays to go along with the natural granularity rather than impose an artificial one.

Granularity has a number of effects on ISYS, the main one being the searchability of documents and navigability of results. For example, ISYS likes to find related terms in the same document. If your granularity is too large or too small, then you subvert ISYS' ability to use related terms to find high quality documents. Similarly, when ISYS produces a result list, its first step is to display a result of all the documents found. If this list only ever consists of the same, single mammoth document, then you are not getting best value from ISYS. ISYS will work regardless of the granularity of your documents, but keeping to a granularity that accurately reflects the nature of your documents and the way in which you want to use them will give best results.

The other means by which granularity effects ISYS is through performance and scalability. Again, the best approach is to stick with the natural granularity of your data, and to not fight it. However, be aware that extreme divergence at either end of the curve may result in atypical performance or resource consumption. For example, in a "normal" index the IXC file (which contains one fixed length record per document) is a fraction of the size of the IXB file. If your data consisted entirely

of documents containing only a single word, then you might find that your IXC file would become disproportionately large relative to your IXB.

Everything will work fine, even at the extremities of the envelope – you should just be aware that the performance you observe may be different to other, more common, scenarios.

EFFICIENCY OF VERY LARGE DOCUMENTS

Except in some unusual circumstances, ISYS never loads an entire document into memory. ISYS has the ability to navigate randomly through exceedingly large documents, going directly from one hit to the next without accessing the parts of the document in between. It achieves this by recording "bookmark" information or checkpoints into the index, which facilitates reading a document commencing from any point within it, even for complex formats such as Microsoft Word or Excel, and for compound file structures such as CHM or XML.

This means that you do not have to artificially break documents into small pieces for fear of overloading machines with small amounts of memory. Extremely large documents can be opened with impunity on modestly configured machines and across slow networks. ISYS is regularly used with individual documents hundreds of megabytes in size.

ISYS SEARCHING VERSUS SQL SEARCHING

A key strength of ISYS is its ability to index SQL relational databases, either on their own or in conjunction with disparate external data sources. A single query can span multiple SQL databases on different DBMSs, local Microsoft Office files, Lotus Notes, email, spreadsheets, external websites, and so on.

Searching SQL databases with ISYS is fundamentally different to the way you would do it using normal SQL methods.

Furthermore, ISYS can completely satisfy a query request without placing any load on your SQL server whatsoever. The query is completely handled by ISYS and uses no DBMS resources.

SQL is good at searching for a specific value, or range of values within a specific column. It is very bad at searching for a particular value which may occur in *any* column, and it is similarly very bad at searching for a *fragment* which may occur in a specific column.

Consider a table which contains three textual columns and the desire to find a given word anywhere in any of those columns. This could be done using the following SQL statement

```
SELECT *
FROM MyTable
WHERE (Column1 LIKE '%CAT%')  OR  (Column2 LIKE '%CAT%')  OR
(Column3 LIKE '%CAT%')
```

This is an exceedingly poor approach. Not only does it leave the SQL optimizer with no choice but to perform a laborious full table scan which guarantees unacceptable performance as data volumes grow, but it also may miss information or produce false matches because SQL has no

understanding of the concept of a “word”. For example, it would match on “caterpillar”, even though that is not the intention. Altering the LIKE comparison to include a space on either side (LIKE ‘% CAT %’) would solve this problem, but then fail to match where the target word appears at the start of the column, the end of the column, or followed by punctuation such as a comma or full stop.

In short, for searching text there is no substitute for a tool which properly understands the nature of text and can do appropriate parsing, as well as add value though language specific features such as tense conflation and synonyms.

Further, you obtain the benefit of search performance that is proportional to the size of the result, whereas misusing SQL as in the above example produces performance which is proportional to the amount of data being searched and thus *cannot scale*.

Some SQL database products do include full text style search capabilities, but these are usually an afterthought and lack the feature richness of professional search tools. Furthermore, they are usually restricted to searching only within the SQL database, and cannot step outside that environment into external Word documents, etc.

SQL INDEXING CONCEPTS

A single ISYS index may reference a single or multiple SQL data sources, and a mixture of external data if desired. The ISYS index does not import or duplicate the SQL data, but is merely an index to the original data in its original location, which may be any ADO, BDE or ODBC data source, residing on any hardware or software platform.

Conceptually, each SQL data source is described to ISYS by means of an arbitrary SQL SELECT statement. The full power of the SELECT statement is available to you, so you can:

- Span multiple tables by means of JOINs
- Select a subset of rows or columns
- Reorder rows or columns
- Perform arbitrary pre-processing of information

ISYS uses the SELECT statement to enumerate the name space, and considers each row returned as a separate ISYS “document”. This means the granularity of the SQL database is at the row level, which maximizes search and result usefulness.

As well as using the SELECT statement to fully enumerate the name space, ISYS also uses SELECT to retrieve individual rows for indexing or browsing purposes. It does this by adorning the SELECT with a WHERE clause specifying the record required. Note that this only happens during indexing or when a particular record is opened at browse time. An entire ISYS query can be resolved without accessing the underlying SQL data at all, and thus imposes zero load on your DBMS.

ISYS formats each resultant row into a “document” containing field/value pairs. An example document is shown below:

```
EMPLOYEE_ID: 567
LNAME: SMITH
FNAME: JOHN
ADDRESS: 23 HICKS ST, SMITHVILLE
DOB: May-23-65
NOTES: John was interviewed by Dave Hicks from HR upon
commencement, and was found to have an excellent work
attitude. His hobbies included breeding of cats, dogs and
```

This is the default textual rendition of the SQL row. Each column from the table is considered a paragraph for ISYS purposes, which allows the proximity search operators of ISYS to be used to good advantage. Each paragraph begins with the column name, and then is followed by the data for that column. ISYS does not actually convert the data or save it to disk in any way – this textual rendition is only a view of the underlying SQL data being accessed directly.

Using the ISYS Rich SQL feature, you can arrange to format the document in any way you like. Individual columns can be placed anywhere, font varied, graphics inserted, fields re-labeled, selected fields bounded and wrapped. The mechanism for doing this is an HTML template, thus the full richness of HTML is available to you in formatting the document.

The indexing and searchability of the row is always done according to the underlying document image as exemplified above, not the presentation template. Hit highlighting is automatically transferred from the simple textual rendition to the richly formatted form.

In some applications, you may not wish to use ISYS to browse documents at all – you may wish to only use ISYS to return the primary keys of the found rows, and access the rows in your own application. This is absolutely fine, but ISYS will still index a virtual textual view of the document and search accordingly, so in order to have a good understanding of how the search functionality works, it's important to keep this model in mind.

SQL PRIMARY KEY FIELDS

It is a fundamental tenet of relational databases that every row must be uniquely identified by a primary key, and this also applies to the table generated by the SELECT statement that defines the SQL data ISYS is to index.

Part of your SQL configuration within ISYS involves nominating a primary key for ISYS to use. It is this key which becomes adorned to the SELECT statement through a WHERE clause to access specific rows at indexing and at browse times. For most straightforward single table scenarios, this will simply be a matter of nominating the table's primary key field. A little care is required to make sure you nominate a field you know to be unique. For example, in the above scenario EMPLOYEE_ID would be the unique identifier, not LNAME.

ISYS has no way of verifying that the field you nominate is a good choice. For example, if you incorrectly specified LNAME everything would work fine until a non-unique last name was encountered. Also, on small data volumes performance may well be acceptable, whereas on larger data volumes performance would likely start to degrade.

In the case of a SELECT which joins multiple tables together, you need to be especially aware because a key which is unique for a single table, after a one-to-many join, is no longer unique. This is not a situation brought about by ISYS – it's simply how relational databases work. In these cases, you would need to specify a compound key which is usually the primary key from the first table, and a supplemental key from the second table. If the join is a one-to-one, then the original single key will be sufficient. Non-unique primary keys are diagnosed at the end of the indexing run as critical errors.

In general, the process is one of looking at your data schema, understanding the data relationships, and specifying the correct key or combination of keys to ensure you have a suitable unique identifier for the result of the SELECT. If in doubt, consult your DBA.

SQL EFFICIENCY

As described above, ISYS can fully resolve a query without involving the SQL DBMS in any way. This means that absolutely no traffic is sent to the DBMS during query evaluation, which implies that you can run your ISYS server on a different machine to spread load if you wish.

ISYS always fully executes the SELECT in order to enumerate the document name space at the beginning of an UPDATE. It subsequently re-executes the SELECT with an additional WHERE clause to access individual records for indexing or browsing purposes. It only does this when it needs to access that particular record – never for searching purposes.

It is important that the primary key field(s) you specify are efficient in an SQL sense. In other words, the SQL optimizer needs to be able to generate some efficient way of accessing individual records, preferably via an SQL index. It is very important to note the difference between an ISYS index, which is a word-oriented, full text index of the entire SQL record, versus an SQL index which exists on a single field and has been created using an SQL "CREATE INDEX" command. The SQL index allows the optimizer to efficiently access records via nominated primary and secondary keys.

A classic symptom of an inappropriate primary key is that your indexing run starts proceeding at a reasonable pace, but gets steadily slower as it progresses. For example, the first thousand rows might be indexed in five seconds, but after a few hundred thousand rows it might be taking fifteen seconds to index a thousand rows. Later it might grow to thirty seconds per thousand.

If you see a steady, arithmetic degradation in SQL indexing speed then it is almost certain there is no efficient way to access individual records based on your key field expression. Review your key field expression and your available SQL indexes. Consult your DBA. This situation is an SQL issue, not an ISYS one.

SQL FIELD-LEVEL SEARCHING

It has been mentioned above that ISYS searching is fundamentally different to the conventional SQL searching to which you may be accustomed. In particular, SQL is best at finding an exact value in a nominated column – it can do other things, but those are not its strength. ISYS is best at finding words or phrases in combinations anywhere in the whole SQL table. ISYS can also do field-level searching for exact values, but typically if you want to perform that sort of searching all

the time, although ISYS can do it, SQL can probably do it better. If you want to combine approaches, then ISYS is the way to go.

Put another way, traditional SQL databases are very good at structured searching, and very poor at unstructured searching. ISYS is best at unstructured searching but also performs well in structured searching.

The way to perform an ISYS search for a given word across an entire SQL record is just to pose the word as a query, for example:

```
SMITH
```

Likewise, to perform a Boolean or more complex search across an entire record:

```
SMITH AND HICKS
```

To search for a given term in a particular field, perform a labeled paragraph search:

```
SMITH IN NOTES
```

Or you can get slightly more sophisticated:

```
(HICKS IN NOTES) and (JOHN IN FNAME) and (AFTER 01-Jan-62 IN DOB)
```

Due to its polymorphic searching ability, ISYS is obliged to offer a surprising level of flexibility in its fielded searching. A given field does not have to occur in the same place in each record, nor does not even have to occur, or may occur more than once in a given record. For SQL data, a certain amount of regularity can be assumed, but don't forget that a single search may be covering multiple disparate tables, as well as XML and ASCII file dumps, etc.

The first step for ISYS in doing a field-level search is to identify where the fields occur. This process may actually be more resource consumptive than locating the target words themselves, because there may be many more occurrences of the field label than the target word, and ISYS performance is related to result size.

You can leverage your knowledge of your SQL data and the fact that each field occurs only once in each record, and always at the same place (if indeed it does), to provide ISYS with the information it needs to dramatically increase fielded search performance. The technique is simply to address the search to a particular field *ordinal* (which is always at a fixed location), rather than a field name which may occur anywhere. For example:

```
FIELD 5 (HICKS)
```

Sharing your knowledge of the regular structure of your data with ISYS like this can result in a performance boost which is breathtaking – several orders of magnitude. This is of particular use for ISYS:sdk applications, or with ISYS:web or ISYS:web.asp, where the actual syntax behind a fielded search can be hidden from the user.

ISYS AND XML

ISYS can either process XML files as large, uninterpreted monolithic text files, or can process the XML intelligently.

When processing XML files as XML, the paradigm is akin to that of SQL data sources – the XML file is seen as a series of rows and columns, and each record within the XML file is indexed as a separate ISYS "document".

XML records are formatted into textual renditions, similar to SQL records. They can be viewed directly, or via a template to add powerful formatting. The searchability factors are the same as for SQL. Note, however, that ISYS' ability to do field-level searching and be tolerant of fields that occur more than once per record is particularly useful for XML.

ISYS accesses the records in the XML file directly, not via a DOM. This means that ISYS provides blindingly fast access, even on extremely large XML files. In benchmarks against competitors, ISYS has provided sub-second access to individual records in XML files so large that attempts to access them via the standard DOM simply caused the machine to freeze for several minutes and then reboot. ISYS provides unparalleled scalability for large XML files.

RELEVANCE RANKING

ISYS relevance ranking is based on a computed expression of the relevance of each document compared to the other documents in the same result set. The calculation is based on hit clustering, cluster proximity, document density, word weight and other factors.

The algorithm is fuzzy in nature, and automatically provides weighting, for example:

- Hits in the meta data are considered more relevant than hits elsewhere
- Hits near the top of a document are considered more relevant than hits near the end
- Notwithstanding the above, a large mass of hits near the end of a document will rank as more relevant than a document with few hits near the top.
- Search terms that occur near each other are considered more relevant than, for example, a ten page document with a hit on "Cat" on the first page, and a hit on "Dog" on the last. This is because although a document may contain both terms, if they occur far apart, they may not be related in a meaningful sense. If they occur near each other, it is more likely that the subject matter connects them in some way.
- If all else is equal, a given quality of hits in small documents will rank better than in large ones.

Relevance is a comparative measure amongst the documents in the same result set.

ISYS:WEB SERVER BANDWIDTH CONSIDERATIONS

There are two aspects when considering server bandwidth. The first is local network traffic to indexes and documents, and has been adequately covered previously in this document. The second is bandwidth out to the browser.

In terms of browser bandwidth, the main thing to remember is that ISYS offers a range of options that let you make an appropriate choice for your situation. ISYS:web provides three levels of fidelity in rendering documents:

- Speed mode – a highly efficient text-only rendition
- Rich HTML – an on-the-fly conversion of the document to HTML, preserving graphics and formatting
- Native application – the original document rendered via a browser plug-in or native application

Of the three, Speed Mode is usually the best option. However, when dealing with particularly large documents or in very high traffic situations, Outline Browse may also be used. Outline Browse enables you, for documents larger than a certain size, to deliver only those portions of the document which may be of interest. Further options let the user expand up and down around rendered portions to see more text if desired.

Outline Browse provides superb bandwidth utilization, even on the largest documents, and even on the slowest networks. Outline browse is a substantial scalability tool for dealing with oversized documents in a practical way.

REAL-TIME INDEXING

ISYS has been used to good effect when real-time indexing is required. In one application using the ISYS:sdk, ISYS was used to process a peak of seven news stories per second, and run 3000 custom queries against each one.

The key to using ISYS in a real-time scenario is gain leverage from the power of batching and the beneficial effects of queuing theory. Attempting to deal with a flow of incoming documents in real-time means you must always be able to cope with the peaks. This can prove to be a difficult scaling problem, and should traffic exceed your expected peak momentarily, then traditionally you must implement some form of queuing mechanism, and take advantage of an anticipated “quiet time” in the near future to process through the backlog.

With ISYS, such complexities are not necessary. The best method is to record your real-time data into a transaction file and have a separate process which takes the transaction file and applies it to the ISYS index (by calling Process_Transactions). When the Process_Transactions call completes, the process simply repeats the procedure. In times of little traffic, each transaction file will contain only a single item of data. In busy periods, the transaction file may contain several items of data. Remember the mindset for indexing scaling: “second and subsequent documents can almost be thought of being processed for free”. This means that if the real-time traffic load were such that the transaction file had grown to three entries while the last one was being applied, then those three will be applied next time through in almost the same time it would have taken to apply one.

This architecture creates a system that automatically optimally balances near real-time updates versus throughput. The structure effectively automatically adjusts the granularity in a self-balancing way, and is far simpler to implement than a system that tries to keep pace and then queues.

TAXONOMIES AND CATEGORIZATION

Broadly speaking, taxonomies can be of two types:

- Fully manual – humans create the structure of the taxonomy and also decide which documents belong to which taxonomic category
- Fully automatic – software generates the structure and populates it.

Research has shown that the manual construction of a taxonomy quickly reaches a point of diminishing return. Yet at the other extreme, we've all seen computer-generated categorizations produce quite meaningless and bewildering categories.

ISYS takes a hybrid approach that optimizes the reward for effort equation. With ISYS, a human generates the structure of the taxonomy, and the software populates the categories with documents. Moreover, this populating takes place dynamically at execution-time, so new information is automatically categorized upon arrival, with no manual intervention. The ISYS term for this is a Concept Tree. ISYS:desktop, ISYS:web and ISYS:sdk all support Concept Trees, and the trees are interchangeable between products.

The ISYS approach offers substantial benefits in scalability, because the manual portion (the preparation of the structure) is of a relatively fixed size. In other words, once you prepare your structure your work is largely done, even as your document volume doubles and redoubles. You only need revisit your structure should the nature of the information take a substantial twist. Further, because of its searching performance characteristics, ISYS can very efficiently slot documents into the categories, even on very large data volumes.

DISTRIBUTED INDEX AND DATA SCENARIOS

Performance issues aside, there are other considerations that impact index and document topology.

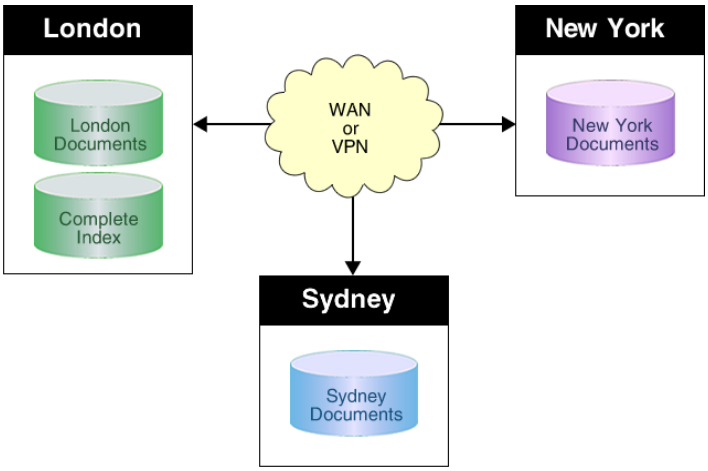
In a simple case of a single physical location and a reasonably fast LAN, documents and indexes can be located without too much thought. However, where an organization has multiple offices with slow data links, some design may be in order.

Consider a legal firm with offices in London, New York and Sydney. Each office has its own high speed LAN, but is connected via a relatively slow VPN. The goal is to create a company-wide brief bank.

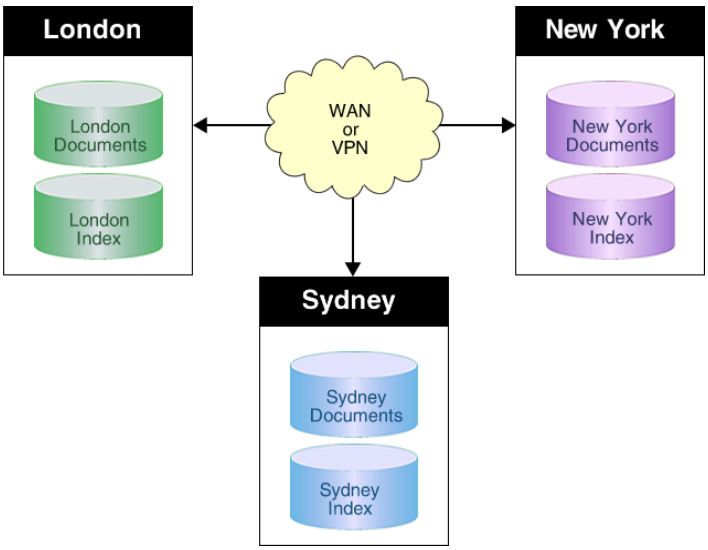
There are three basic models for data and index distribution.

One approach is to store the text documents throughout the WAN. For example, documents could simply be stored on the LAN upon which they are created. A comprehensive index of all the documents would then be maintained on a single, arbitrary file server. Users at any office could then search the index to find information residing on any server within the WAN, subject to LAN security, of course. This approach works best if the documents can be stored on the LAN that is most likely to retrieve them. Because the index is centrally located, it is easier to maintain and manage. As discussed, the index is a relatively small file and imposes minimal WAN traffic when

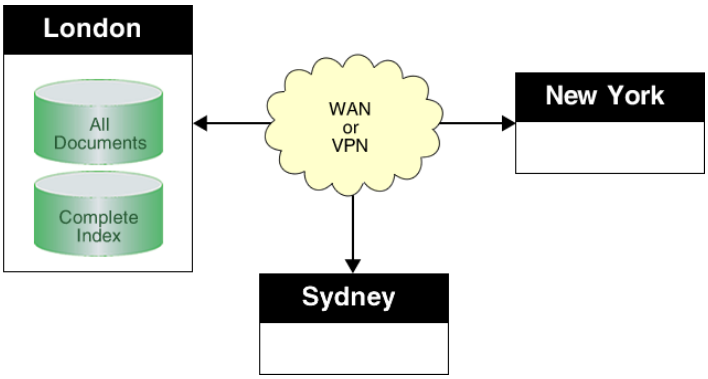
used for query purposes remotely. The documents, whether large or small, are located locally and so in most cases there is minimal overhead required to retrieve them, assuming each office tends to most frequently access its own information, and more rarely needs the other offices' information for reference.



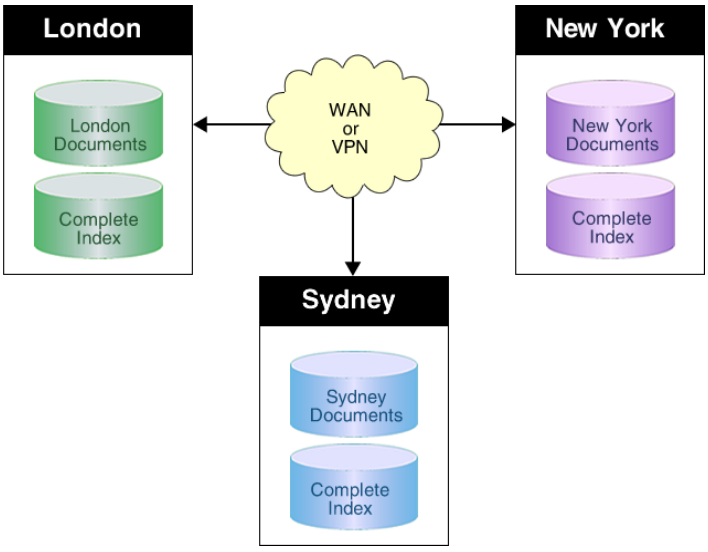
Using another approach, each LAN could have its own documents and indexes. Users on other LANs could search the indexes of other selected sites. Although not attractive from an aesthetic point of view, all local queries would be completely processed without any WAN traffic whatsoever, and each office would be self-sufficient in the event of WAN outage.



In a third scenario, a central document collection and index could be maintained on a single LAN. Users at other sites could search the central database regardless of their physical location. This is the neatest approach, but potentially the least effective, least convenient, and least fault tolerant.



A combination of these approaches can also be implemented, for example the central index could be periodically replicated locally. ISYS could then use the locally replicated index for searching but the centrally maintained documents for browsing. This would have the advantage of minimal network traffic and a high degree of fault tolerance. The WAN would only be utilized when people needed to actually view documents domiciled in other locations.



If WAN reliability were very low (or perhaps even non-existent, for example with a very remote workforce), and if uptime were paramount, then you may even choose to replicate both indexes and documents across all locations. ISYS Search Software's ISYS Replication and Distribution Utility product can assist with replication issues, if this is a path you wish to take.

Inevitably, the best approach will depend on your exact circumstances, and among the factors to consider are:

- What proportion of a location's document access will be of its own documents, versus other locations' documents ?
- How slow is your long haul network compared to local access?
- What is the likelihood of the long-haul connectivity failing?
- What are the ramifications when it does fail?
- How acceptable is it for remote documents to be slower to open than local documents?

OPPORTUNISTIC LOCKING

Opportunistic Locking is a feature of Microsoft Windows NT and 2000 which attempts to provide enhanced disk performance at the expense of data integrity.

Opportunistic Locking is on by default, which means that data loss may occur.

Like many vendors, ISYS Search Software recommends that Opportunistic Locking should be turned off if problems occur, particularly where large data volumes or intensive I/O loads are involved. Microsoft Knowledge Base Article Q296264 describes how this can be done.