Ten Things You Can Do To Make InterBase Scream

Bill Karwin

Inprise Conference, August 1998

Come to this technical presentation to hear the best ways to increase the performance of your InterBase databases and applications. After you hear this presentation, you will know how to speed up the typical client/server system at least tenfold.

Introduction

I have been reading and collecting messages from InterBase users on email list servers and newsgroups for three years. The most consistent issue I hear developers ask about is performance. No matter how far advancements in computer hardware take us toward faster CPUs and storage, the demands of modern information users are greater than the speed by which IT applications deliver data.

Borland (now INPRISE Corporation) produces tools to enable Rapid Application Development. If one examines this phrase, it has ambiguous meaning. It could mean:

- To *develop software applications rapidly*;
  to finish one's software implementation project in less time.

But it could also mean:

- To *develop rapid applications*;
  to ensure that the software we write accomplishes its tasks as fast as possible, or at least faster than any other comparable program.

This is a nice idea, that Borland tools somehow facilitate software engineering by giving us abstract and simple components for program design, at the same time as the tools write the code for us to perform with optimal efficiency exactly what the programmer has in his or her mind.

But this is not the case. The nature of software development tools is to be flexible and general enough to allow us to write almost any program we can imagine. But by being so general, a software development tool cannot specialize or analyze our software design. Any experienced software developer knows that computers do what we tell them, often with a painfully literal interpretation. There is no "do what I mean" function in programming. A software development tool gives us the rope with which to invent the killer app, or by which to hang ourselves. As with any other engineering discipline, there is no substitute for experience and careful analysis.

Following are ten principle areas of analysis for writing high-performance client/server database

systems using InterBase.

#10: Design of queries

Correlated subqueries and joins

Subqueries are SELECT statements which are included as a clause or expression within another statement. They are typically used to generate a value or result set that are used in conditions of the superior query.

A correlated subquery is one in which the conditions of the subquery are different for each row in the parent query, because they depend on values that vary from row to row. InterBase executes the subquery many times, once for each row in the parent query. Evaluating each row has a large cost in performance relative to a non-correlated subquery. InterBase optimizes non-correlated subqueries out of the loop, executes once, and uses the result as a fixed dataset.

Example as correlated subquery:

```
SELECT *
FROM DEPARTMENT D
WHERE EXISTS
    (SELECT *
    FROM EMPLOYEE E
    WHERE E.EMP_NO = D.MNGR_NO
    AND E.JOB_COUNTRY = ‘England’)
```

Example as join:

```
SELECT D.*
FROM DEPARTMENT D JOIN EMPLOYEE E ON D.MNGR_NO = E.EMP_NO
WHERE E.JOB_COUNTRY = ‘England’
```

Query execution plan

The plan describes the way the optimizer has chosen to execute a query. For certain types of queries, the optimizer might not select the truly optimal plan. A human can analyze different alternate plans and specify a plan overriding the optimizer's analysis. The result can be amazing improvements in performance for some types of queries. In some dramatic cases, this has been used to reduce a 15 minute query to three seconds.

The elements of plan selection are:

- Assigning indexes
- Combining indexes
- Determining join order

- Generating rivers
- Cost estimation
- Sort merges

Syntax has been added to the SELECT expression in GPRE and DSQL/ISQL to allow the user to specify the PLAN for a query. The syntax should work for SELECT statements in the body of a view, a stored procedure, or a trigger.

It is beyond the scope of this paper to describe in detail the syntax of the PLAN clause for specifying the execution plan, or techniques for analyzing queries manually. For more details, see http://www.interbase.com/tech/docs/manage.html

Prepared queries and query parameters

InterBase supports parameterized queries in DSQL, for cases when a given statement is to be executed multiple times with different values. For example, loading a table with data might require a series of INSERT statements with values for each record inserted. Executing parameterized queries has a direct performance benefit, because the InterBase engine keeps the internal representation and optimization of the query after preparing it once.

Use parameterized DSQL queries in Delphi by following these steps:

1. Place a named parameter in the statement with the Delphi :PARAMETER syntax. in place of a *constant value* in a query. InterBase does not support parameters in place of anything other than constants; tables and column names cannot be parameterized.
2. *Prepare* the statement. Use the TQuery method Prepare. Delphi automatically prepares a query if it is executed without first being prepared. After execution, Delphi unprepares the query. When a query will be executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares.
3. *Specify parameters*. For example, with the TQuery component, use the ParamByName method to supply values for each parameter in the query.
4. *Execute* the statement. SELECT statements should use the Open method of TQuery. INSERT, UPDATE, and DELETE statements should use the ExecSQL method. These methods prepares the statement in SQL property for execution if it has not already been prepared. To speed performance, an application should ordinarily call Prepare before calling ExecSQL for the first time.
5. Repeat steps 3 and 4 as needed.
6. *Unprepare* the query.

In some real-world cases involving repetitive operations, using parameterized queries has increased performance 100%.

#9: Network protocol

InterBase supports three protocols: NetBEUI when connecting to a Windows NT server, IPX/SPX when connecting to a Novell NetWare server, and TCP/IP when connecting to any server.

NetBEUI and IPX/SPX

NetBEUI and IPX/SPX are network protocols designed for use on small local area networks. These protocols are commonly used for filesharing services. They are connectionless protocols, which means they broadcast packets to the entire network. This causes an growing amount of "noise" on a LAN. Noise, from the point of view of any given host, can be defined as network traffic that is not intended for the given host. On a LAN with a lot of hosts, enabling NetBEUI or IPX/SPX can overwhelm the network and reduce the available bandwidth for everyone to use. On most enterprise networks (including the INPRISE corporate), the IT staff discourages use of NetBEUI and IPX/SPX.

TCP/IP

TCP/IP is a connection-based protocol, which means packets are routed to the intended recipient. This reduces the saturation of the network and the load on individual hosts. There is effectively more bandwidth available to all hosts, and a much greater number of hosts can share the same network without as much of a performance penalty.

DNS

Each host on a TCP/IP network has a designated IP address, and TCP/IP traffic is routed to hosts by address. TCP/IP requires a means for clients to translate hostnames to their numeric addresses. This is done either on each client host in a file called HOSTS, or else on a central server using a protocol called DNS. The client requests that the DNS server resolve a hostname, and the server returns the IP address. Then the client can use the IP address to communicate directly with the intended destination.

Depending on the load on the network and the DNS server itself, hostname resolution can take several seconds. This translates directly into delays when making a network connection. You probably have noticed the message in a web browser, "Looking up host *name*…" followed by, "Connecting to host *name*…" The first message indicates the delay while querying a DNS server to resolve a hostname.

How can you speed up the hostname resolution step? Instead of relying on DNS, add host-to-address mappings to a HOSTS file on the client computer. Looking up a line in a local file is much faster and more reliable than querying a service running on another host over the network. This reduces the hostname resolution delay when initiating connections to hosts listed in the local HOSTS file.

DHCP

The drawback often mentioned for TCP/IP is that it is more difficult to administer than NetBEUI or IPX/SPX. All TCP/IP hosts on the network need to be allocated a unique IP address. This becomes a laborious task of information tracking and configuration management for IT staff. When workstations move from network to network, or when new workstations are added to the network, the installation and configuration needs to be very detail-oriented. Failure to correctly configure the IP address usually results in nonfunctional workstations.

Dynamic Host Configuration Protocol (DHCP) can help reduce the legwork for this task. The workstation queries a DHCP server and the server tells the workstation what IP address to use, out of a pool of unused addresses. The workstation uses that IP address for the duration of its session on the network. This is how most modem servers at Internet service providers work. But it can be used just as effectively on local networks.

InterBase works just as well when the client is using a DHCP-supplied IP address. As long as the database server keeps its name and the DNS service can provide the database server's IP address to clients, connectivity is unaffected.

#8: Database page issues

Page size

InterBase by default uses pages 1K in size. There is a 25-30% performance benefit in the majority of cases to using a page size of **4K**. General wisdom among customers and experts is that larger pages means better performance for reasons including:

- Fewer record fragments are split across pages

  It is common that records are defined to be larger than the default 1K page size. This means that records are fragmented and stored on multiple pages. Querying a given record requires multiple page reads from the database. By increasing the size of a page, the number of multiple page reads is reduced, and record fragments tend to be stored more contiguously.

- Index B-trees are shallower

  Indexes are B-trees of pointers to data pages containing instances of specific indexed values, and if the index B-tree is larger than one page, additional database pages are allocated for the index tree. If the index pages are larger, fewer additional pages are needed to store the pointers. It is easier for the database cache to store the entire B-tree in memory, and indexed lookups are much faster.

- I/O is more contiguous

  It is fairly likely for successive records in a table to be requested in the same query. For example, this is done during a *table scan*, or query that returns or aggregates all

records in a table. InterBase stores records on the first page that is unused, rather than ensuring that they are stored near each other in the file. Doing a table scan can potentially require retrieval of data by seeking all over the database. Seeks take time just as reading data takes time.

Any given page can store only records from one table. This indicates that a larger page is certain to contain more data from the same table, and therefore reading that page returns more relevant data.

- Default number of cache buffers is a larger amount of memory

  The database cache is allocated in number of pages, rather than a fixed number of bytes. Therefore, by defining a database with a larger page size, the cache for the database implicitly contains more bytes from the database. A larger cache is more likely to have a better hit rate than a smaller cache.

- Most operating systems perform low-level I/O in 4096 byte blocks anyway

  A page read is performed at the OS level by reading in 4096 byte increments regardless of the size of the database page. Therefore, by defining the database with a page size of 4096, the database I/O matches the low-level I/O and this results in greater efficiency and better performance.

Changing the default page size when creating new databases and restoring databases results in an improvement in performance for InterBase without changing anything in the underlying architecture of the database. Applications require no change in their coding or design; the different page size is completely transparent to the client.

Although 4KB seems to be the best page size for most databases, this depends on the structure of the specific metadata and the way in which applications access the data. For this reason, you should not treat 4KB pages as a "magic bullet," and instead do some testing with your application and database under several different page sizes to analyze which configuration gives the best performance.

Compacting data on pages for read only databases

Data pages store multiple versions of data records, as data is updated. When a database is restored, the GBAK.EXE utility fills pages with data only up to 80% of the capacity of each page, to leave space for new record version deltas to be stored, hopefully on the same page with the original record. But in a database that is used mostly for reading data rather than updating it, the benefit of this 80% fill ratio is never used. In this case, it makes sense to restore data using the full capacity of each page. By storing 25% more data on each page, it reduces the amount of record fragmentation and increases the amount of data returned in each page read. You can specify the option to use all the space of every page for storing data during a database restore using the command:

```
GBAK -C -USE_ALL_SPACE backup_file.gbk database_file.gdb
```

#7: Transactions

InterBase requires that any query or other operation be associated with an active transaction. This is required by the *multi-generational architecture* of InterBase. Without a transaction, an operation has no context with which to maintain its *snapshot* of the database. Windows ISQL and BDE tools do a certain amount of automatic transaction management, but it is helpful for performance to manually start and finish transactions.

In the InterBase server engine, a snapshot is generated by making a copy of the state of all other transactions in the database. This snapshot is static for the current transaction. This means that any data committed to the database after the snapshot is created is not visible to operations using that snapshot. This is the *repeatable read* transaction mode. Two identical queries made at different times are guaranteed to get the same result set, even if other clients are updating data in the database.

Starting a transaction and making a snapshot data structure for the new transaction incurs some amount of overhead. This overhead is magnified when using automatic transaction-handling, because the typical automatic transaction behavior is to start a new transaction and commit it for every statement executed against the database!

Another mode—the default mode for BDE—is called *read committed*. In this mode, the snapshot is updated every time the state of any transaction changes. This allows operations in the current transaction to view or act on data that has been committed since the snapshot was created. Updating the snapshot also costs a little bit in performance, so it is recommended to always use the repeatable read mode in InterBase. To do this, configure BDE driver flags to the value 512 or 4608.

#6: Indexes

Defining indexes for some columns is part of designing a production database. Indexes dramatically improve performance of SELECT queries. The greater the number of rows in the table, the greater the benefit of using an index. Intelligently analyzing your database and defining indexes appropriately always improves performance.

An index in InterBase is a data structure inside the database file that provides a quick lookup mechanism for the location of specific values in a table. Queries make use of appropriate indexes automatically by means of the cost-based optimizer, which analyzes the tables and columns used in the query and chooses indexes which speed up the searching, sorting, or joining operations.

Indexes incur a small cost to maintain the index B-tree data structure during INSERT and UPDATE operations. Because of this cost, it is not recommended to be overly liberal with index definitions. Don’t create redundant indexes, and don’t make an index on every column as a substitute for database usage analysis.

Indexes are actually detrimental to performance when created on a column that has few unique values. The classic example is a SEX column on a large table; the only values are `male`, `female`,

and perhaps `unspecified`. Maintaining such indexes is expensive, and lookups are often more expensive than if the lookup were done without using an index.

What uses an index:

- Primary & Foreign keys
- Sort keys, including DISTINCT and GROUP BY
- Search criteria (WHERE)

What doesn’t use an index:

- Search criteria for CONTAINING, LIKE, <>
- Columns used in aggregate functions, like COUNT()
- Other expressions, like UPPER()

Directional indexes

Indexes are defined as either ASCENDING or DESCENDING. To sort in both directions, you need one index of each type. This is also very important if you are using a scrolling list in a Delphi form.

Tuning indexes

The *selectivity* of an index is an indicator of its uniqueness. The optimizer uses selectivity in its cost-based analysis algorithm when deciding whether to use a given index in a query execution plan. If the selectivity is out of date and doesn't accurately represent the state of the index, the optimizer might use or discount the index inappropriately. This doesn’t usually have a great performance penalty unless the selectivity is highly out of date. To recompute an index selectivity:

SET STATISTICS *name;*

Periodically, a B-tree data structure might become imbalanced, or it might have some values in the tree that have been deleted from the database (this should not happen in InterBase 5.0 and later, thanks to index garbage collection). You should periodically rebuild an index:

ALTER INDEX *name* INACTIVE;
ALTER INDEX *name* ACTIVE;

#5: Database cache

The IBSERVER.EXE process running on an InterBase server maintains a cache in memory of recently used data and index pages. Like any cache, it depends on repeated use of data on a given page to help speed up subsequent access. In InterBase 4.2 and later (that is, Superserver

implementations), the cache is shared by all clients connected to the database.

By default, InterBase allocates enough memory for 256 database pages. If the page size of the current database is 1 kilobyte, then 256K of memory is used. If the page size is 4 kilobytes, then 1MB of memory is used. The InterBase API provides a means for any individual client to request that the size of the cache be higher. In InterBase 5.0 and later, you can set a property on an individual database that establishes a different default cache size when any client connects to that database:

GFIX.EXE -BUFFERS 5000 *DATABASE.GDB*

The default of 256 is designed to be a lean configuration for smaller-memory systems that want InterBase to refrain from excessive memory use. Using more memory for cache is beneficial to performance. It is highly recommended to raise the cache size property for a database if you have enough memory to accommodate it.

Don’t raise the cache size so high that the memory used by IBSERVER.EXE starts to page into virtual memory. That defeats the benefit of caching data from disk in memory!

Don’t raise the cache size higher than the number of pages in the database (which you can view with the Database Statistics in the InterBase Server Manager, or with the GSTAT.EXE command-line program). There’s no benefit to this, since any given page from disk occupies only one page in the cache, and isn’t duplicated.

One block of memory is allocated for cache per database. If clients connect to two separate databases on one server, the IBSERVER.EXE process maintains two separate cache areas of memory.

You should experiment with larger cache sizes and analyze the performance improvements. At some point, you will observe diminishing returns. But you can possibly achieve up to 30% improvement from this.

#4: I/O Buffering

InterBase on Wintel implements a *write-through cache* by default. Every write operation to a page in cache is immediately written out to the operating system’s disk I/O (which itself might have a cache, but sometimes it doesn’t).

By contrast, a *write-back cache* defers flushing of the contents of a given cache page until a later time. Multiple writes to a cache page are performed in memory before the page is written out to disk, resulting in better response time for the majority of write operations. Write-back cache consolidates I/O efficiently, and therefore it is much faster than write-through cache.

InterBase offers write-back cache as the default on UNIX versions, and as an option on Wintel. This can be configured at a database level by GFIX.EXE -WRITE ASYNC or by clearing the "Enable Forced Writes" checkbox under the Database Properties dialog in the InterBase Server Manager.

The real benefit of using asynchronous writes (write-back cache) is about 4x performance in the typical case. Some users have reported up to 20x performance gain merely from configuring asynchronous writes, in applications that make heavy use of write operations (INSERT, UPDATE, DELETE). The more writing an application does to the database—including write operations spawned by triggers—the more benefit the application gains.

The downside of asynchronous writes is that some data might be lost if the server has a power loss, or if IBSERVER.EXE exits abnormally for any reason. Write-through cache protects against data loss, at some performance cost. If the server host and client/server application is tested thoroughly and it isn't susceptible to crashes, then it is highly recommended to use asynchronous writes.

#3: The active server

InterBase provides *active metadata* to allow the database server to enforce in a centralized way:

- Business rules
- Security
- Data integrity
- Less network consumption

InterBase features for active metadata

- Triggers
- Stored Procedures
- SELECT Procedures
- Cascading Declarative Referential Integrity (PRIMARY KEY & FOREIGN KEY)
- SQL Privileges
- User-Defined Functions (UDFs)
- User-Defined Exceptions
- Events

See http://www.interbase.com/tech/docs/trig_sp.html
http://www.interbase.com/tech/docs/udfs.html for more information.

#2: BDE settings

Change default BDE driver options in the BDE Administrator.

Driver flags

The recommended value for the DRIVER FLAGS is 4608.

By adding 512 to the DRIVER FLAGS in BDE Config tool, you specify that the default transaction mode should be repeatable read transactions. This reduces the overhead that automatic transaction control incurs (see #7: Transactions).

By adding 4096 to the DRIVER FLAGS, you specify that the InterBase SQL Links driver should use soft commits. Soft commits are a feature of InterBase that let the driver retain the cursor when committing changes. Soft commits improve performance on updates to large sets of data. When using hard commits, the BDE must refetch all the records in a dataset, even for a single record change. This is less expensive when using a desktop database, because the data is transferred in core memory. For a client/server database like InterBase, refreshing a dataset consumes the network bandwidth and degrades performance radically. With soft commit, the cursor is retained and a refetch is not performed.

DRIVER FLAGS	Isolation level	Commit type
0	Read committed	hard commit
512	Repeatable read	hard commit
4096	Read committed	soft commit
4608	Repeatable read	soft commit

Caveat: soft commits are never used in explicit transactions started by BDE client applications. This means that if you use explicit transaction start and commit, then the driver flag for soft commit is not used.

SQL Passthru Mode

The recommended value for this property is SHARED NOAUTOCOMMIT

SQLPASSTHRU MODE specifies whether the BDE and passthrough SQL statements can share the same database connections. In most cases, SQLPASSTHRU MODE is set to SHARED AUTOCOMMIT. If however, you want to pass SQL transaction control statements to your server, you must use the SQL Explorer to set the BDE SQLPASSTHRU MODE to NOT SHARED. There are some reports of 10X performance when using this setting, though depends on quantity of data.

Use explicit transaction control and avoid autocommitted statements. Use the following methods:

TDatabase.StartTransaction

TDatabase.Commit

SQL Query Mode

Set to SERVER to allow the InterBase engine to interpret and execute SQL statements, not BDE.

#1: VCL component properties

TQuery

- CachedUpdates = False

  Allow the server to handle updates, deletes, and conflicts

- RequestLive = False

  There are some reports from Delphi users that setting RequestLive to False can prevents the VCL from keeping a client-side copy of rows; this has a benefit to performance because less data must be sent over the network

In a client/server configuration, a "fetch-all" is the nadir of performance, because it forces a refresh of an entire dataset over the network. Here are some instances in which cause a TQuery to perform a fetch-all:

- The **Locate** method; you should use Locate only on local datasets
- The **RecordCount** property; it’s nice to get the information on how many records are in a dataset, but calculating the RecordCount itself forces a fetchall.
- The **Constraints** property; let the server enforce the constraint.
- The **Filter** property; use a WHERE clause and let the server do the filtering before sending the dataset over the wire.
- Commit when BDE driver flags does not include 4096, or when using explicit transaction control.

TTable

The TTable is designed for use on relatively small tables in a local database, accessed in core memory. TTable gathers information about the metadata of the table, and tries to maintain a cache of the dataset in memory. TTable refreshes its client-side copy of data when you issue the TTable.post method or the you TDatabase.rollback method. This incurs a huge network overhead for most client/server databases, which have much larger datasets and are accessed over a network. You can observe the activity of TTable with the SQL Monitor tool. This reports all calls to the BDE and InterBase API.

Don’t use TTable for Client/Server— use TQuery instead

Though TTable is very convenient for its RAD methods and its abstract data-aware model, it should never be used with InterBase. TTable was not designed to be used for client/server applications. The following statement is From the Delphi 3.0 VCL Object and Component Reference Help:

> "TQuery is of particular importance to the development of scalable database applications. If there is any chance that an application built to run against local databases will be scaled to a remote SQL database server in the future, use TQuery components from the start to ensure easier scaling later."

Alternatives to SQL-Links

- IB Objects

  Jason Wharton authored a set of VCL components for use with Delphi 2 and 3. It is designed to provide very sophisticated data component technology that is optimized for use with InterBase. The demo product can be downloaded from www.ibobjects.com.

- Free IB Components

  Greg Deatz authored a set of data-aware VCL components for use with the TDataSet architecture introduced in Delphi 3. Free IB Components has some catching up to do with respect to the component implementation of IB Objects, but it has promise to become a better integrated component set in Delphi 3 and future versions. It can be downloaded from www.interbase.com/download/components.html.

- InterBase Data Access Components

  Based on Greg Deatz' Free IB Components, InterBase Software Corp. is in the process of engineering a more fully-featured set of data-aware VCL components for use with the TDataSet architecture. ISC plans to make this interface available in InterBase 6.0.

Bonus tips

The following sections describe additional techniques for performance enhancement, but they are not necessarily covered in the presentation accompanying this paper.

Screen savers

Many people overlook the impact of screen savers on their database server. Because servers are often set aside in a machine room, it's easy for the performance impact of a screen saver to be out of sight, out of mind. It should not be taken lightly, however. Screen savers demand a surprising amount of CPU resources to run, and these programs run continually, 24 hours a day.

The tricky thing about screen savers is their ability to disappear when the database administrator tries to log in to the console to diagnose a mysterious drop in performance. All seems normal to the DBA as soon as she touches the server, but the speed will degrade soon after she leaves it.

Not all screen savers are created equal. The Windows NT **OpenGL** screen savers do continuous floating-point computations to draw three-dimensional shaded shapes in real time. They demand up to 90% of the system CPU, and cause InterBase and other services to slow to *one-tenth* their normal speed!

The Marquee screen saver is by some reports the least demanding one, especially when it is configured to pass text across the screen slowly. Some system administrators like to configure a Marquee on each screen in the machine room, to display the respective machine's hostname. This becomes a machine-name label, in raster form.

It is preferable to disable the screen saver feature altogether. A server might go untouched for days or weeks at a time, especially if you use InterBase, the "DBA-less" database server. A screen saver can also be entertainment, but in a machine room, who is going to enjoy it?

If you must have phosphor-burn protection for a monitor that is left on, get an Energy Star monitor that has a power conservation mode. This mode blackens the screen after a configurable period of idleness. This not only protects against phosphor burn, but conserves power. This is like a simple "black screen" screen saver, but it is handled in the electronics of the monitor, instead of in software.

The best option is to simply turn off the monitor. This saves the phosphors, saves power, and decreases the amount of heat in the machine room.

Console logins

Don't leave the console logged in on a Windows NT database server. Even if the desktop is idle, it could be using as much as 30% of the machine's CPU resources just maintaining the interface. Log out. InterBase includes the Server Manager for Windows, which allows most InterBase maintenance and monitoring tasks to be performed from another workstation, without logging in at the server's console.

Fast I/O

A multi-user database server's hard drives are no place to be thrifty, especially in today's market of inexpensive storage. The performance gains of a relatively high-end I/O system is a very cost-effective way to get more bang for the buck.

Slow disk subsystems are often the weak link in an otherwise high-performance server machine.

The top-rated CPU and maximum memory helps. But if a cheap disk I/O interface limits the data transfer rate, then the money spent on the expensive components is wasted.

It's not appropriate for this paper to state exactly what the best I/O system is. The technology changes so quickly that any recommendation here would be useless. Suffice to say that when specifying the machine for a server platform, research the best hardware solution available. General recommendations for I/O technology are:

- Use SCSI; SCSI technology does not have significantly superior throughput to EIDE (PC Magazine, http://www.zdnet.com/pcmag/pclabs/report/r960702a.htm); but bus-mastering SCSI takes less CPU resources
- Use a disk controller with built in cache memory
- Disk striping (included in RAID levels 0, 3, or 5) gains parallel I/O across multiple disks; up to 10x by some reports
- Don't assume all disks of a given size perform equally; research performance ratings

Dedicated server

Using a server for both workgroup file and print services and a database server is like letting another user play a video game on your workstation. It detracts from the performance of the workstation, and it's not the intended use for the machine.

Use a hand-me-down server as the file and print server, and a new machine for the database server. Or vice-versa, depending on the relative priority of these tasks—the database server benefits from having a dedicated machine, even if it is not the fastest model available. Whatever is the most important service should be given the best machine as dedicated hardware.

If performance is a high priority, spend money more effectively by buying a dedicated machine instead of trying to increase resources such as RAM on a machine that is providing another competing service. Compare the cost of the hardware with the cost of having less than maximum performance. Is it worth it?

Along the same lines, it is best to put a database on a dedicated drive, so that the database I/O doesn't compete with the operating system virtual memory paging file or other operating system I/O.

Windows NT optimization for network applications

Jung Vu (jungv@knowledgeweb.com) writes:

> "Setting the NT server to "Optimized for Network Applications" (under Network/Server property in NT Control Panel) greatly improve the performance of IB server. The peak CPU usage by IB which used to occur a few seconds every minute are gone."

Multiprocessor hardware

Many users have reported only a modest performance improvement by using multiprocessor

hardware. The InterBase server engine is certified to work on SMP hardware, but does not implement parallel execution features yet (this is a long-term technology enhancement, planned for a future release).

The InterBase lock manager is a single-threaded section of code, so database requests tend to serialize in order to acquire locks. This usually isn't a severe bottleneck, because lock management is a high-throughput operation, compared to physical I/O.

SMP systems do benefit the InterBase server in that additional CPUs can take the load of other processing for the server, such as network services, desktop, and other processes. The amount of performance improvement in this case depends on the demands of other processes relative to the InterBase server process. Based on some reports from users, this is between 5% and 20% performance improvement. On the other hand, our technical support experience tells us that SMP actually tends to *decrease* performance of InterBase on Windows NT.

Blob segment size

A Blob is a datatype with an unbounded size. They can be many megabytes in size, much larger than any database interface can handle in a single I/O transfer. So Blobs are defined as a series of segments of uniform size, and the I/O interface transfers Blobs one segment at a time. By default, InterBase Blobs have a segment size of 80 bytes.

It is advantageous to define a Blob with a segment size equal to the page size. If both the page size and the Blob segment size are 4096, queries of large Blobs can achieve a data transfer rate of up to 20MB per second! InterBase ceases to be any kind of bottleneck in this situation; it is more likely that the hardware I/O bus, the network bandwidth, or the middleware are the limiting factors for throughput.

Avoid Windows NT

UNIX and Linux are better as server platforms than Windows NT. In scalability, security, stability, and especially performance, UNIX and Linux contain more mature and proven technology. In all these areas, UNIX and Linux are demonstrating their superiority over Microsoft's resource-hungry server operating system. Some of the areas in which the deficiencies of Windows NT result in performance inferiority include:

- NT makes poor use of memory and virtual memory
- NT has a poor multiprocess prioritization model
- NT has bugs in its process scheduling on SMP hardware
- NT requires much more CPU and memory resources to match UNIX/Linux performance

The IT industry is realizing that the "ease of use" of Windows NT is more than offset by the high cost of ownership of that platform. Downtime, excessive hardware resource requirements, frustrating bugs, undocumented features, and mysterious performance problems continue to plague Windows NT. Someday perhaps Windows NT will be better as a server operating system, but today UNIX and Linux are clearly better. Use Windows NT for workstations, to use its

strengths to best advantage. NT and UNIX/Linux computers can easily share a network.

InterBase was created on UNIX platforms, and has recently released version 5.1.1 on the popular Linux operating system. The InterBase versions for Linux and SCO OpenServer are priced the same as the version for Windows NT, and run on the same Intel-based hardware.

Bill Karwin is Manager of Technical Publications at InterBase Software Corp. His email address is bkarwin@interbase.com.
Bill welcomes feedback, but please do not send Bill requests for individual Technical Support {link to tech_offer.txt}*; he cannot provide that service.*