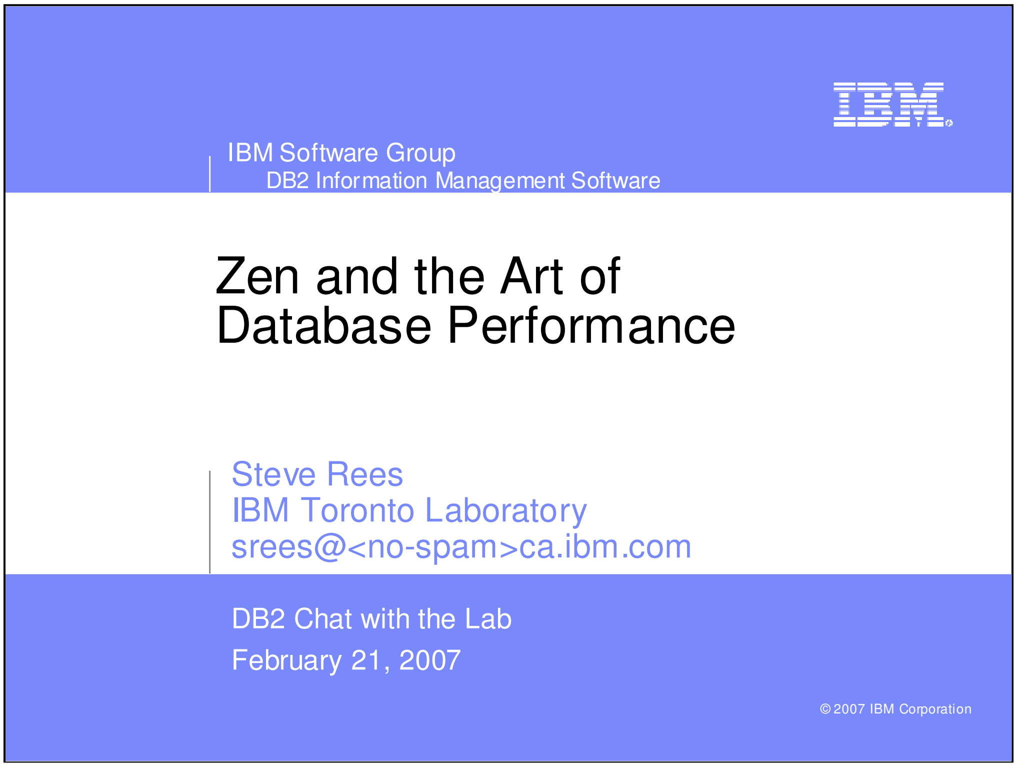


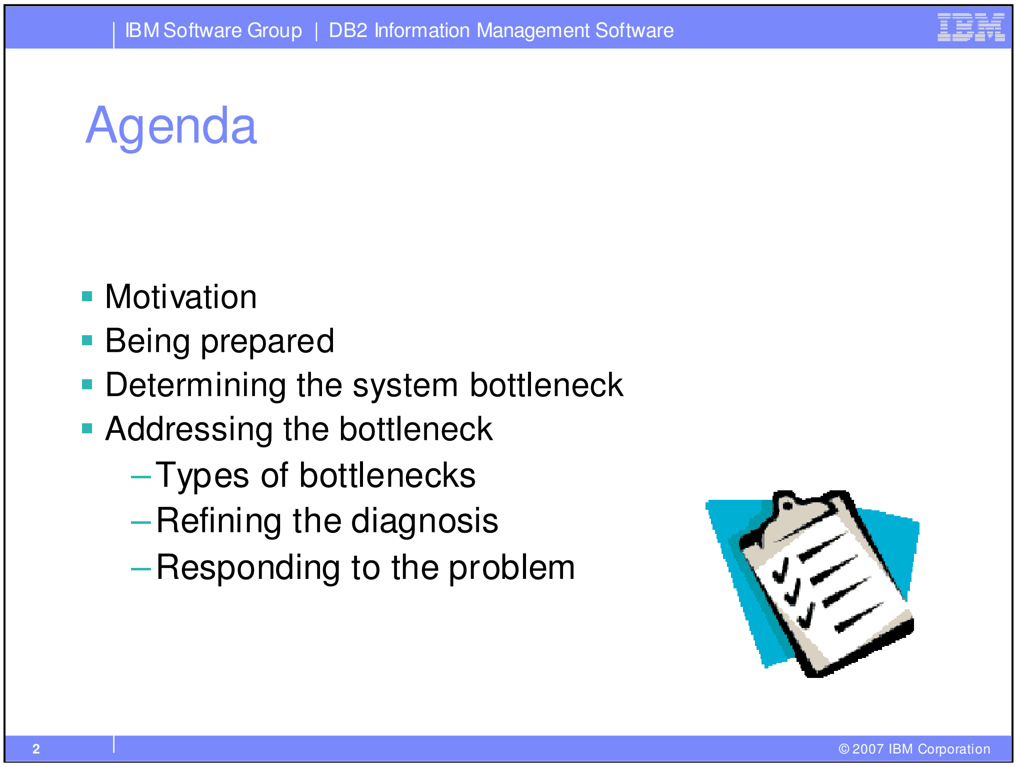
IBM Software Group
DB2 Information Management Software

Zen and the Art of Database Performance

Steve Rees
IBM Toronto Laboratory
srees@<no-spam>ca.ibm.com

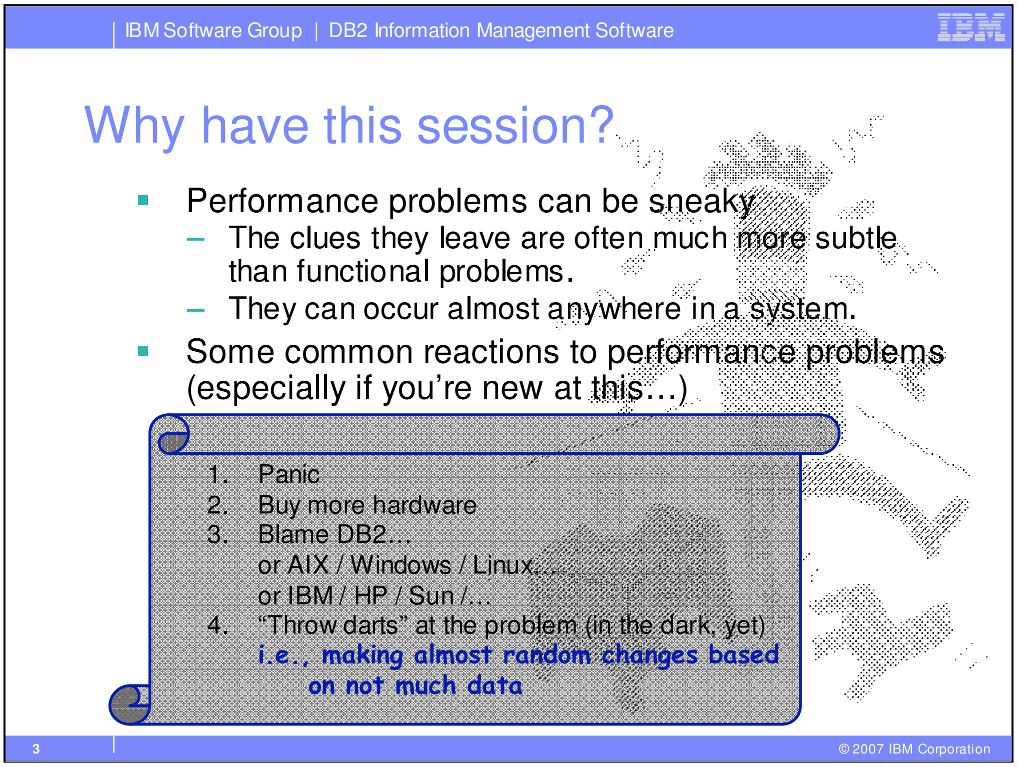
DB2 Chat with the Lab

February 21, 2007



Agenda

- Motivation
- Being prepared
- Determining the system bottleneck
- Addressing the bottleneck
  - Types of bottlenecks
  - Refining the diagnosis
  - Responding to the problem



IBM Software Group | DB2 Information Management Software

Why have this session?

- Performance problems can be sneaky
  - The clues they leave are often much more subtle than functional problems.
  - They can occur almost anywhere in a system.
- Some common reactions to performance problems (especially if you're new at this…)

1. Panic
2. Buy more hardware
3. Blame DB2…
   or AIX / Windows / Linux…
   or IBM / HP / Sun /…
4. "Throw darts" at the problem (in the dark, yet)
   i.e., making almost random changes based on not much data

3

© 2007 IBM Corporation

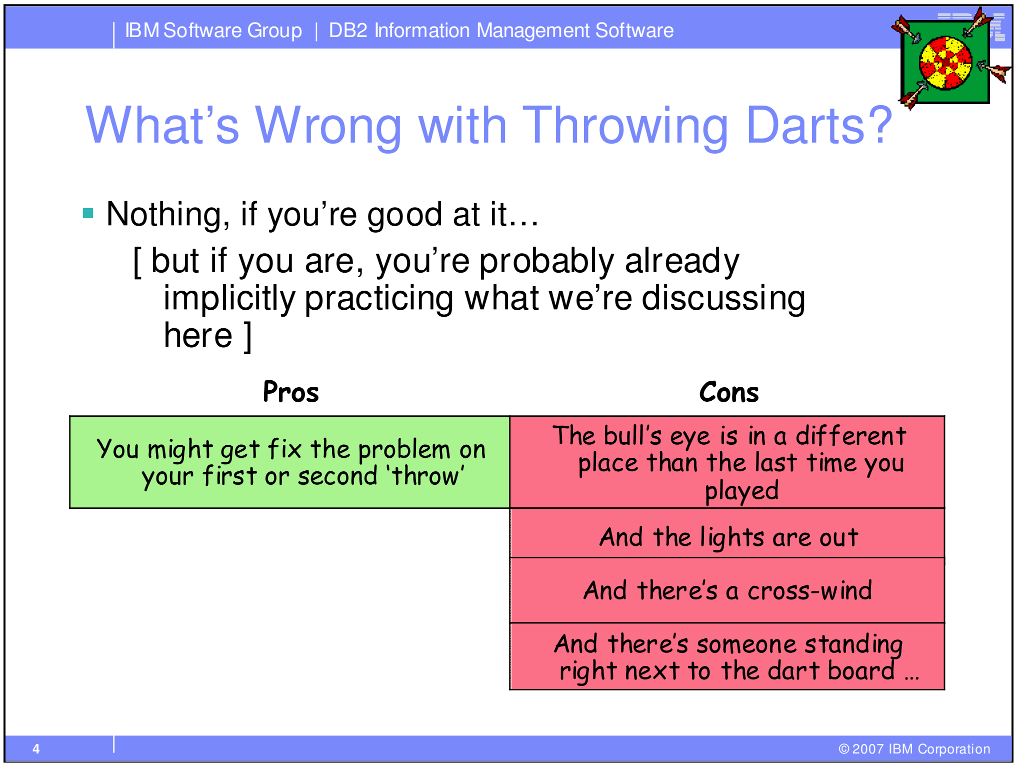
Debugging of functional problems is difficult enough – it's one of the toughest problems anyone involved with software may face. But debugging performance problems can be even worse.

- no messages

- no trap files

- no stack dumps

The clues we have are much more difficult to follow, and you have to know where to find them. If you've never faced one of these before, it can be pretty daunting…



What’s Wrong with Throwing Darts?

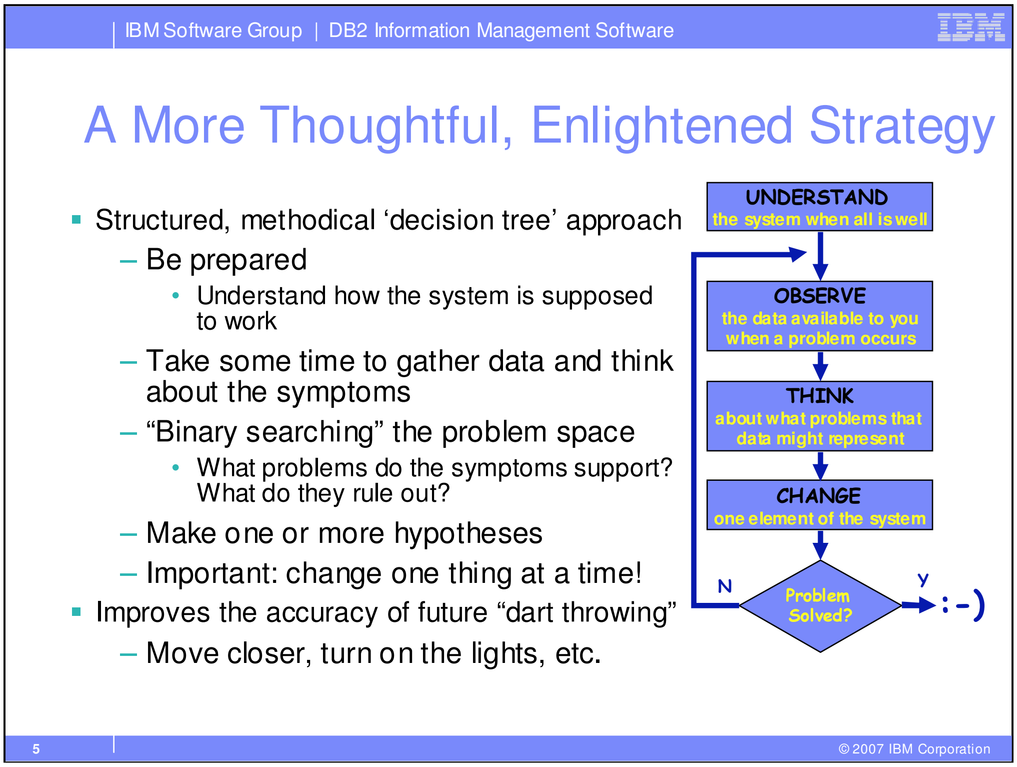
- Nothing, if you’re good at it…

  [but if you are, you’re probably already implicitly practicing what we’re discussing here]

Pros	Cons
You might get fix the problem on your first or second ‘throw’	The bull’s eye is in a different place than the last time you played
	And the lights are out
	And there’s a cross-wind
	And there’s someone standing right next to the dart board …

The ‘throwing darts’ method is about randomly trying different performance fixes without really knowing which ones really make sense to try in a given situation. Most people have quite a collection of these darts – from past experience, from colleagues, from conferences, from magazines. The trouble is that they don’t typically come with a user manual telling you when to use them. So often people will start to throw these ‘darts’ & hope that they’ll hit the target & solve the problem.

Now, you might get lucky – and might not even realize how lucky you are! But the odds aren’t great for repeated success at this – we need a better solution.



A More Thoughtful, Enlightened Strategy

- Structured, methodical 'decision tree' approach
  - Be prepared
    - Understand how the system is supposed to work
  - Take some time to gather data and think about the symptoms
  - "Binary searching" the problem space
    - What problems do the symptoms support? What do they rule out?
  - Make one or more hypotheses
  - Important: change one thing at a time!
- Improves the accuracy of future "dart throwing"
  - Move closer, turn on the lights, etc.

UNDERSTAND
the system when all is well

OBSERVE
the data available to you when a problem occurs

THINK
about what problems that data might represent

CHANGE
one element of the system

What we really want here is a proper strategy. Something scientific - think logically, change one thing at a time. If we really handle it right, we start off with tests / decisions / choices that rule out great swaths of potential problems with one stroke – in computer science terms, we're 'binary searching' the problem, creating a **decision tree**. In literary terms, we're Sherlock Holmes – ruling out the impossible until "whatever is left, however improbable, is the solution". Well, hopefully it's not as exhaustive as all that!

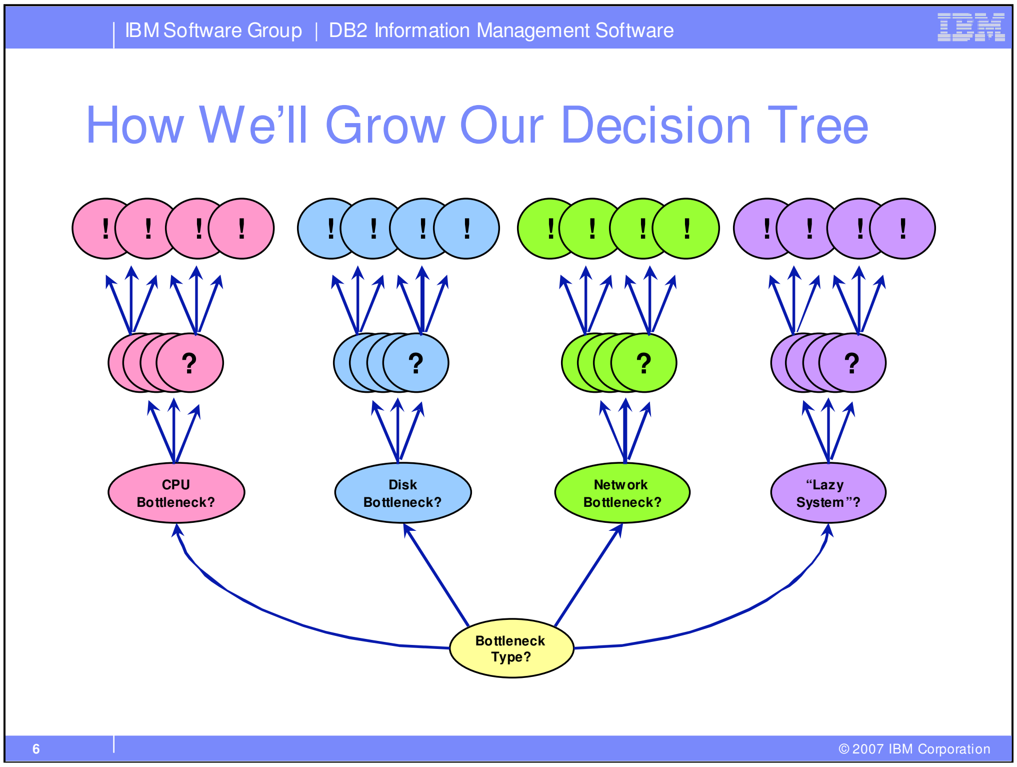
In terms of a good analogy – consider undoing a knot. You try not to just randomly pull at bits of string. If you can find a loop that connects to a end – that's the place to start! Pull on that, and reduce the 'complexity' of the problem by one knot – and continue, methodically and patiently.

Our process starts with understanding the system when all is well – be prepared!. We need to know what things look like when they're running well.

When a problem occurs, we observe the system – how does it differ from when all was well?

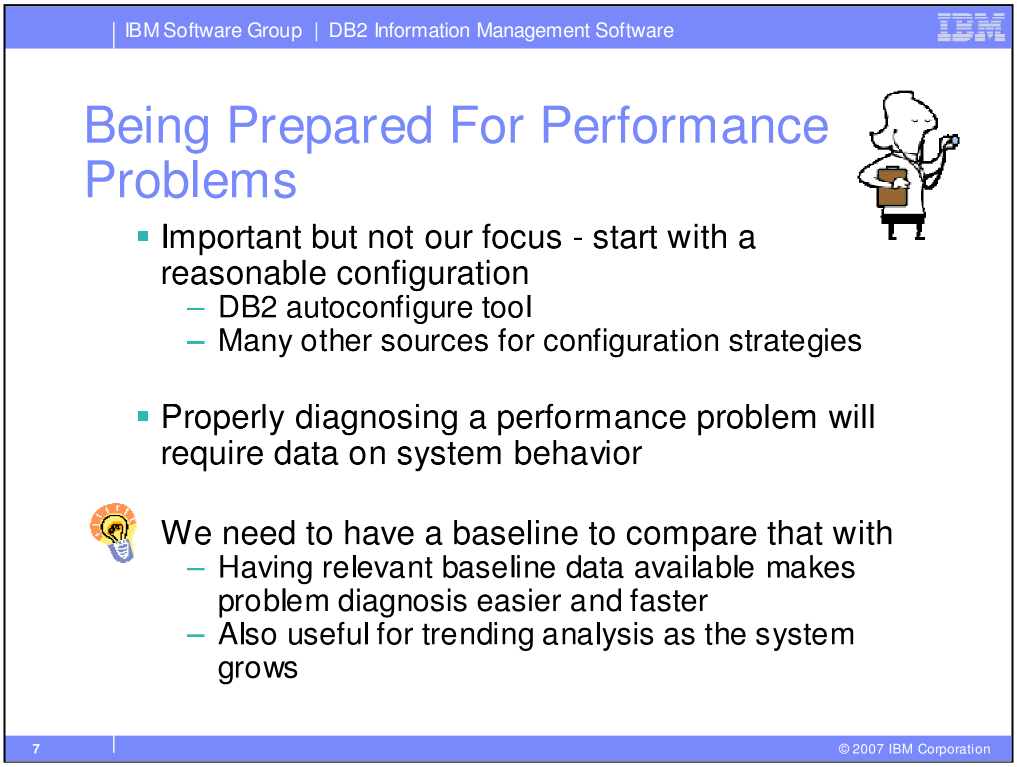
Then, what could cause the symptoms we're seeing? We don't want to waste time on trying to fix suspected problems that couldn't cause what we're seeing. We start with the most general possible causes (which tend to rule things out the quickest) and then move to more specific things.

When we've identified something that we might think is causing the problem, we need to be patient & methodical again – change one thing a time. This is the only way to be sure of what the real cause is, when things start to get better.



How We’ll Grow Our Decision Tree

We start out at a very basic level – is our bottleneck fundamentally a CPU problem, a disk problem, a network problem, or a ‘lazy system’ problem? And from there, we narrow down possible causes, knowing that we can eliminate (or at least de-prioritize) causes in other branches.



Being Prepared For Performance Problems

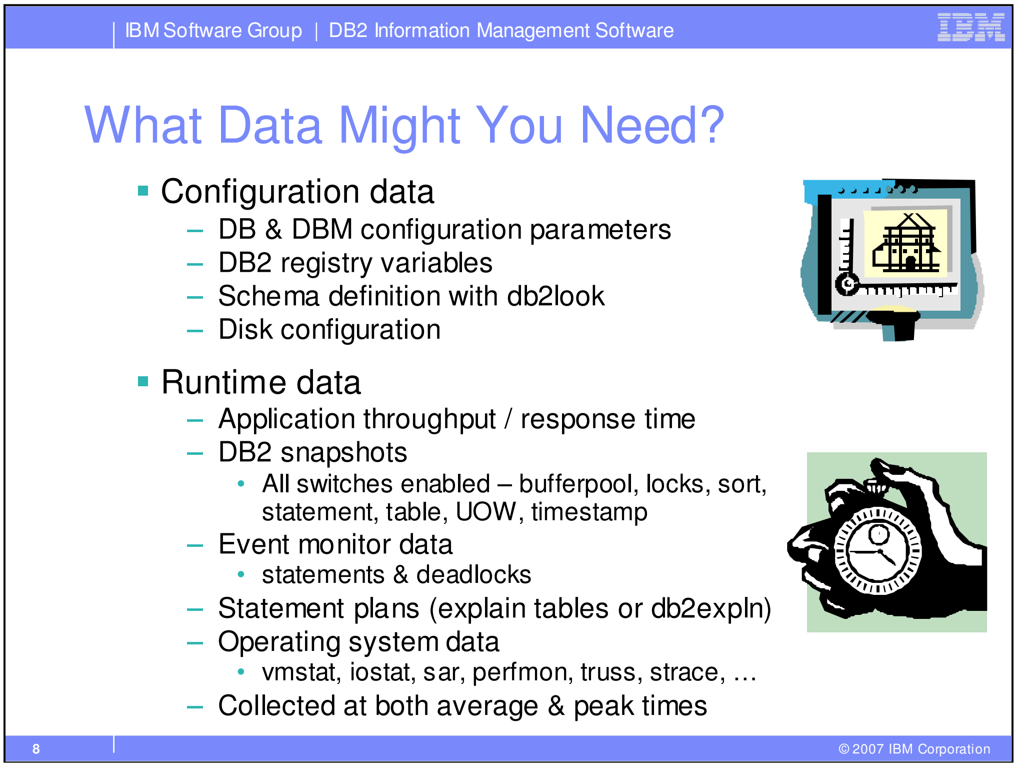
- Important but not our focus - start with a reasonable configuration
  - DB2 autoconfigure tool
  - Many other sources for configuration strategies

- Properly diagnosing a performance problem will require data on system behavior

- We need to have a baseline to compare that with
  - Having relevant baseline data available makes problem diagnosis easier and faster
  - Also useful for trending analysis as the system grows

We're not going to dwell on it here, but we do need to start with a decent initial configuration. There are lots of good presentations & documents on how to do this. One very sound approach is the 'db2 autoconfigure' tool.

The next thing we need to consider is having the right configuration / profile data available which reflects how things are when all is well. Being able to compare that with the system configuration / behavior when a problem occurs will make our lives much easier.



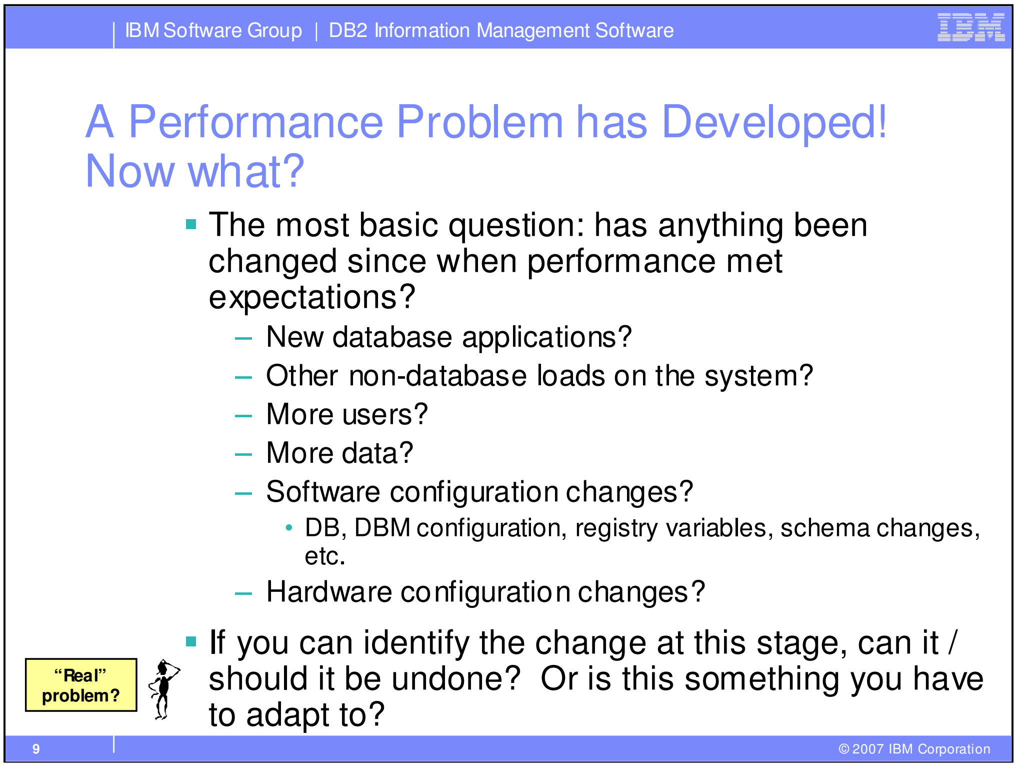
What Data Might You Need?

- Configuration data
  - DB & DBM configuration parameters
  - DB2 registry variables
  - Schema definition with db2look
  - Disk configuration
- Runtime data
  - Application throughput / response time
  - DB2 snapshots
    - All switches enabled – bufferpool, locks, sort, statement, table, UOW, timestamp
  - Event monitor data
    - statements & deadlocks
  - Statement plans (explain tables or db2expln)
  - Operating system data
    - vmstat, iostat, sar, perfmon, truss, strace, …
  - Collected at both average & peak times

Having a history of configuration changes and performance data is very useful, both in the case of when there's a problem, and also for trend analysis

Configuration to collect
- Schema & physical database design (db2look)
- Database & database manager configuration parameters
- environment variables
- DB2 registry variables

Performance data
- Collect averages but also peaks
- DB2 data:
  - snapshots
  - statement event monitors
  - db2pd
  - DB2 Performance Expert and Health Center also are very helpful in this area
- Operating system:
  - vmstat/iostat/sar/slashproc on UNIX
  - perfmon on Windows
  - hardware information – e.g. disk response times, network time, etc
- Application – performance as seen by the application is most relevant
  - throughput
  - response time



A Performance Problem has Developed! Now what?

- The most basic question: has anything been changed since when performance met expectations?
  - New database applications?
  - Other non-database loads on the system?
  - More users?
  - More data?
  - Software configuration changes?
    - DB, DBM configuration, registry variables, schema changes, etc.
  - Hardware configuration changes?
- If you can identify the change at this stage, can it / should it be undone? Or is this something you have to adapt to?

"Real" problem?

The very first thing we need to consider - has anything changed since when performance met expectations?

Often the trigger of a performance issue is a change in the requirements / load on the system

new applications?

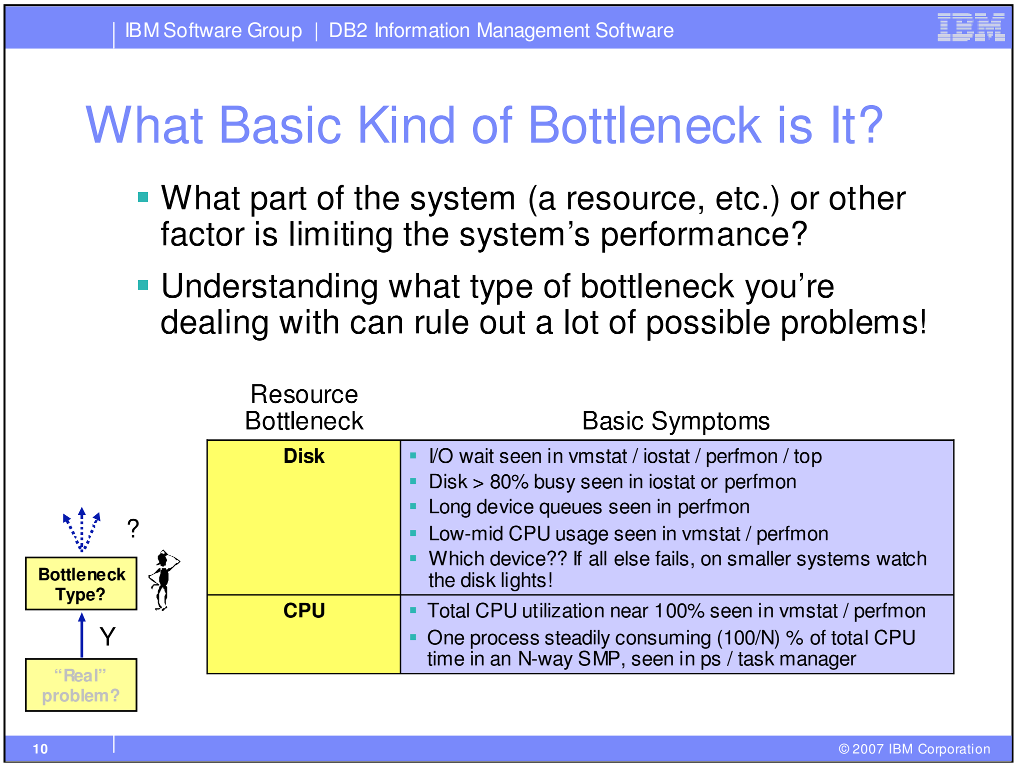
more users?

more data?

software upgrades or configuration changes?

hardware changes ?

If it was unintentional, or not critical, can it / should it be undone? Or is this something you have to live with?



IBM Software Group | DB2 Information Management Software

What Basic Kind of Bottleneck is It?

- What part of the system (a resource, etc.) or other factor is limiting the system’s performance?
- Understanding what type of bottleneck you’re dealing with can rule out a lot of possible problems!

?

Bottleneck Type?

Y

“Real” problem?

Resource Bottleneck	Basic Symptoms
Disk	▪ I/O wait seen in vmstat / iostat / perfmon / top ▪ Disk > 80% busy seen in iostat or perfmon ▪ Long device queues seen in perfmon ▪ Low-mid CPU usage seen in vmstat / perfmon ▪ Which device?? If all else fails, on smaller systems watch the disk lights!
CPU	▪ Total CPU utilization near 100% seen in vmstat / perfmon ▪ One process steadily consuming (100/N) % of total CPU time in an N-way SMP, seen in ps / task manager

10

© 2007 IBM Corporation

First question - what's my bottleneck?

(What is a bottleneck anyway? All systems are limited in the end. The ideal situation is to make all resource capacities match, at a level at or beyond what's required)

Bottleneck types:

resource shortage

Disk

symptoms

low CPU utilization

I/O wait time in vmstat, etc.

one or more devices showing up as > 80% utilized

long device queues (particularly in Windows)

which device?

iostat / lsof / perfmon can be helpful here

if all else fails, disk lights!

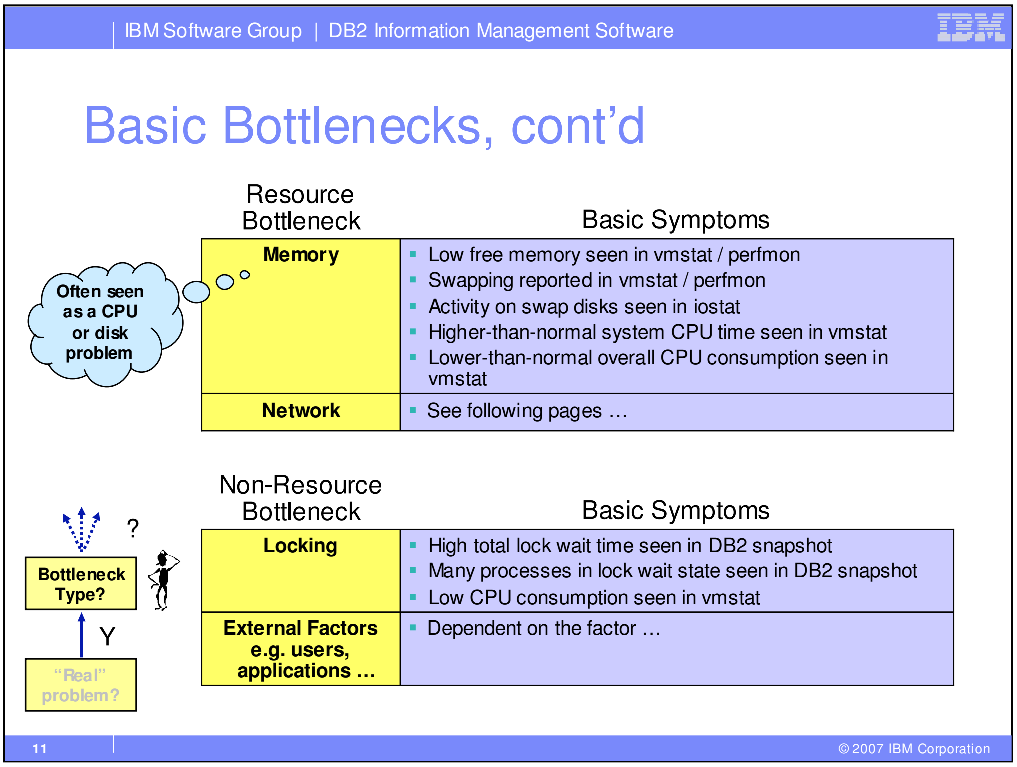
CPU

Symptoms

Total CPU utilization in vmstat, etc. at or near 100%

Or – can also be indicated by having one CPU saturated in an SMP system. For example, 25% CPU usage in a 4-way system can indicate a CPU bottleneck

if the workload is not parallelized.



IBM Software Group | DB2 Information Management Software

Basic Bottlenecks, cont'd

Often seen as a CPU or disk problem

Resource Bottleneck	Basic Symptoms
Memory	▪ Low free memory seen in vmstat / perfmon ▪ Swapping reported in vmstat / perfmon ▪ Activity on swap disks seen in iostat ▪ Higher-than-normal system CPU time seen in vmstat ▪ Lower-than-normal overall CPU consumption seen in vmstat
Network	▪ See following pages …

Non-Resource Bottleneck	Basic Symptoms
Locking	▪ High total lock wait time seen in DB2 snapshot ▪ Many processes in lock wait state seen in DB2 snapshot ▪ Low CPU consumption seen in vmstat
External Factors e.g. users, applications …	▪ Dependent on the factor …

11 © 2007 IBM Corporation

Disk and CPU are the most common resource bottlenecks, but there are others.

A memory constraint or bottleneck is usually of a different nature than CPU or disk. Memory usage is typically more predictable than disk or CPU, since we set DB2 parameters which dictate how much will be consumed. So in other words, memory is an independent (or semi-independent) variable. Disk and CPU are utilized depending on what we ask the system as a whole to do – so they are dependent variables.

We can think of memory as the 'grease' between disk and CPU. If we don't have enough of it, it tends to put more stress on disk – and can thereby morph into a disk bottleneck. Are all disk bottlenecks attributable to a shortage of memory? Definitely not. The majority of databases are far too large to reside in memory, so the best the memory can do is provide us a window on the overall data. The more memory we have, the bigger the window – but – often even the biggest window we can afford is still a tiny fraction of the overall data volume. So in cases like that, we're potentially looking at a combination of both memory and disk bottlenecks.

Memory bottlenecks can show up in the following ways

low free memory

swapping

higher-than-normal system CPU time (because we're stressing the virtual memory management system)

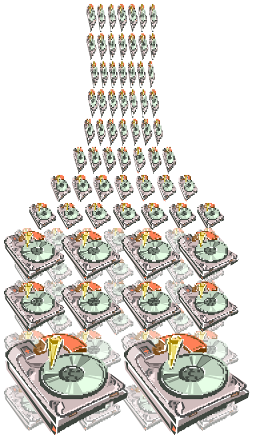
lower-than-normal overall CPU time (because we're waiting on disk)

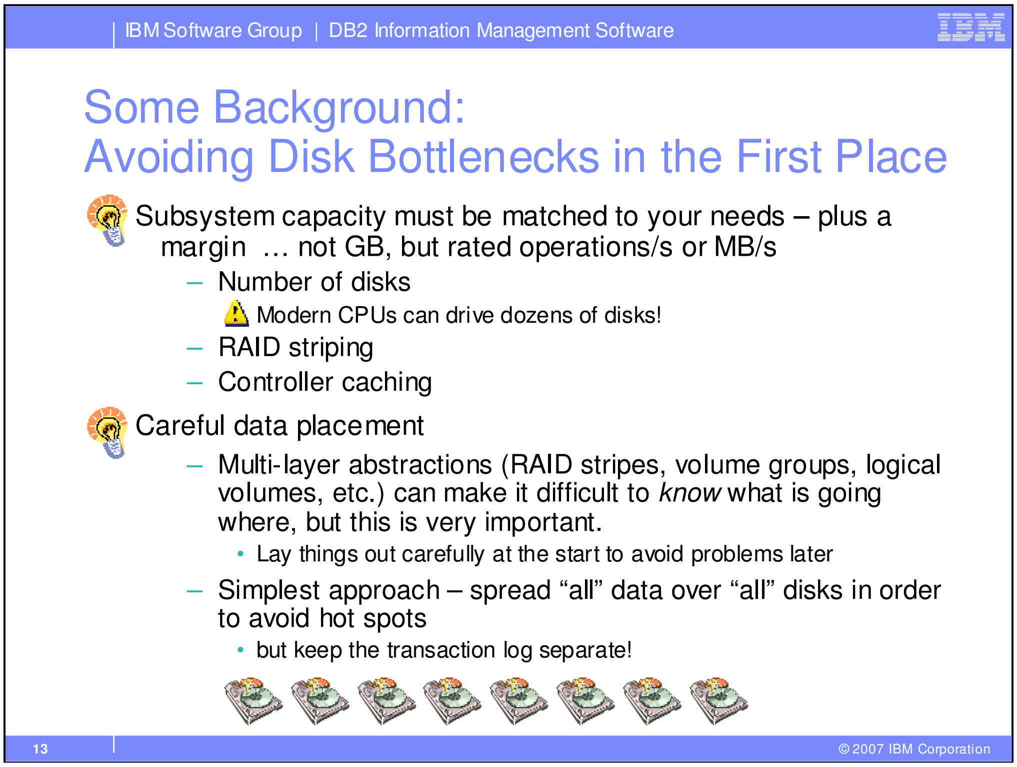
Network bottlenecks are a bit more involved to diagnose – but fortunately they're a bit more rare. We'll talk more about these later on.

Resource bottlenecks are the major type that we'll consider, but there are also bottlenecks which are less resource related. The obvious one is locking. Individual locks are resources, of course, but a bottleneck rarely develops on a single lock. It's much more normal for bottlenecks to develop on a network of lock dependencies, which can make them difficult to predict.
A lock bottleneck is most obviously seen in a DB2 lock snapshot – which shows lock wait time, and how many applications are waiting on locks at that time. As well, of course, if applications are waiting on locks, they're not doing anything else – so the CPU consumption in that case is typically low.

Last but not least are 'external factors' – such as user behavior, applications, etc. These can cause bottlenecks as well, but they're pretty well beyond the scope of what we're talking about here.

Basic Bottleneck #1 – Disk





IBM Software Group | DB2 Information Management Software

Some Background: Avoiding Disk Bottlenecks in the First Place

- Subsystem capacity must be matched to your needs – plus a margin … not GB, but rated operations/s or MB/s
  - Number of disks
    - ⚠ Modern CPUs can drive dozens of disks!
  - RAID striping
  - Controller caching
- Careful data placement
  - Multi-layer abstractions (RAID stripes, volume groups, logical volumes, etc.) can make it difficult to *know* what is going where, but this is very important.
    - Lay things out carefully at the start to avoid problems later
  - Simplest approach – spread “all” data over “all” disks in order to avoid hot spots
    - but keep the transaction log separate!

13

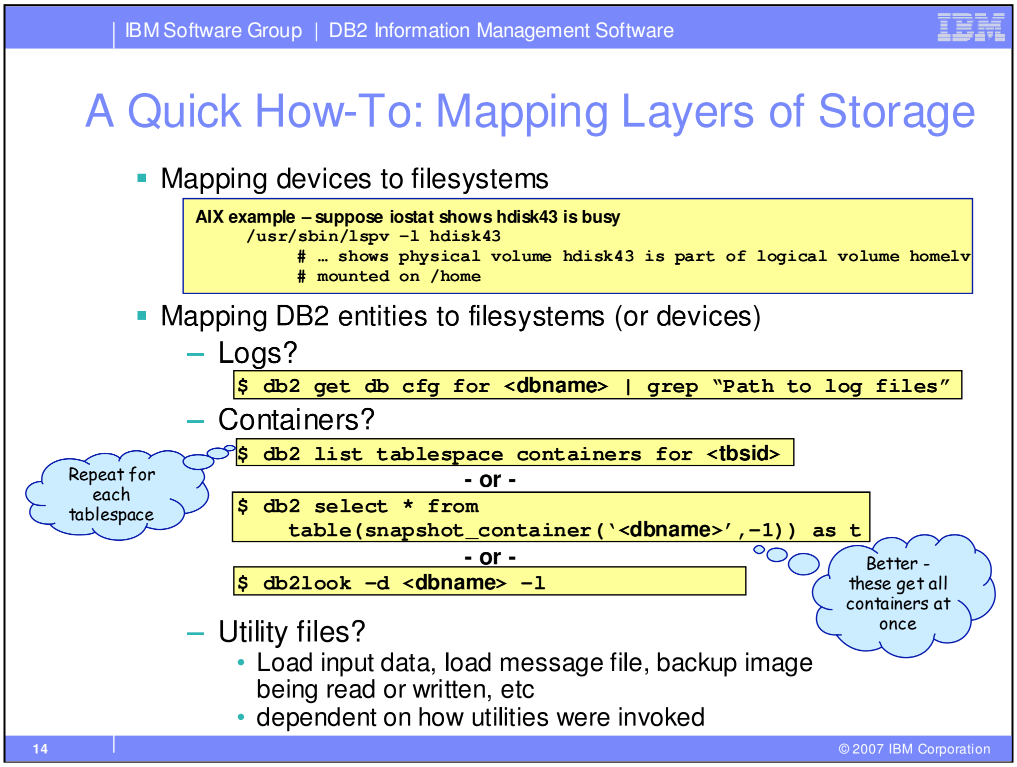
Two things we need to consider in order to try to avoid disk bottlenecks in the first place.

First, we need adequate capacity – but this is in terms of disk spindles, not GB. Disk capacity is increasing, but per-disk performance (seek time, transfer speed) is improving at a much slower rate. Consequently, it’s important to make sure the system has enough disks to support the required total throughput in I/Os per second or GB/s. 10-20 disks per CPU is not uncommon. A modern RAID controller or storage server also helps get the best performance and reliability out of a set of disks.

Second, data placement on disk is important. Tuning to the n-th degree is not generally necessary, but avoiding hot-spots is worth the effort. This means making sure that the hottest parts of your data aren’t accidentally residing on the same disks, and also making sure that transaction logs don’t share spindles with tablespace containers.

The main problem with data placement issues is that they don’t tend to cause problems until heavy load. By the time that scenario occurs, the problem may have become too well established to be easily fixable.

Since hot spots are the most common cause of disk bottlenecks, we should consider the easiest way to avoid them – which is to spread ‘everything over everything’ – a little bit of each tablespace on each disk. This is rarely 100% optimal, but it does usually offer a simple way to get “near optimal”



A Quick How-To: Mapping Layers of Storage

- Mapping devices to filesystems

```
AIX example – suppose iostat shows hdisk43 is busy
     /usr/sbin/lspv –l hdisk43
          # … shows physical volume hdisk43 is part of logical volume homelv
          # mounted on /home
```

- Mapping DB2 entities to filesystems (or devices)
  - Logs?

```
$ db2 get db cfg for <dbname> | grep "Path to log files"
```

  - Containers?

```
$ db2 list tablespace containers for <tbsid>
```

Repeat for each tablespace

- or -

```
$ db2 select * from
     table(snapshot_container('<dbname>',-1)) as t
```

- or -

```
$ db2look -d <dbname> -l
```

Better - these get all containers at once

  - Utility files?
    - Load input data, load message file, backup image being read or written, etc
    - dependent on how utilities were invoked

One practical challenge of administering database storage is determining how tablespace containers and other database entities map to devices. Operating systems and storage devices often provide multiple layers of abstraction between the real devices and the level at which they're specified.

Most operating systems provide ways to tunnel through these layers. One example is the lspv command on AIX, which lets us map a device name to a logical volume and a mount point.

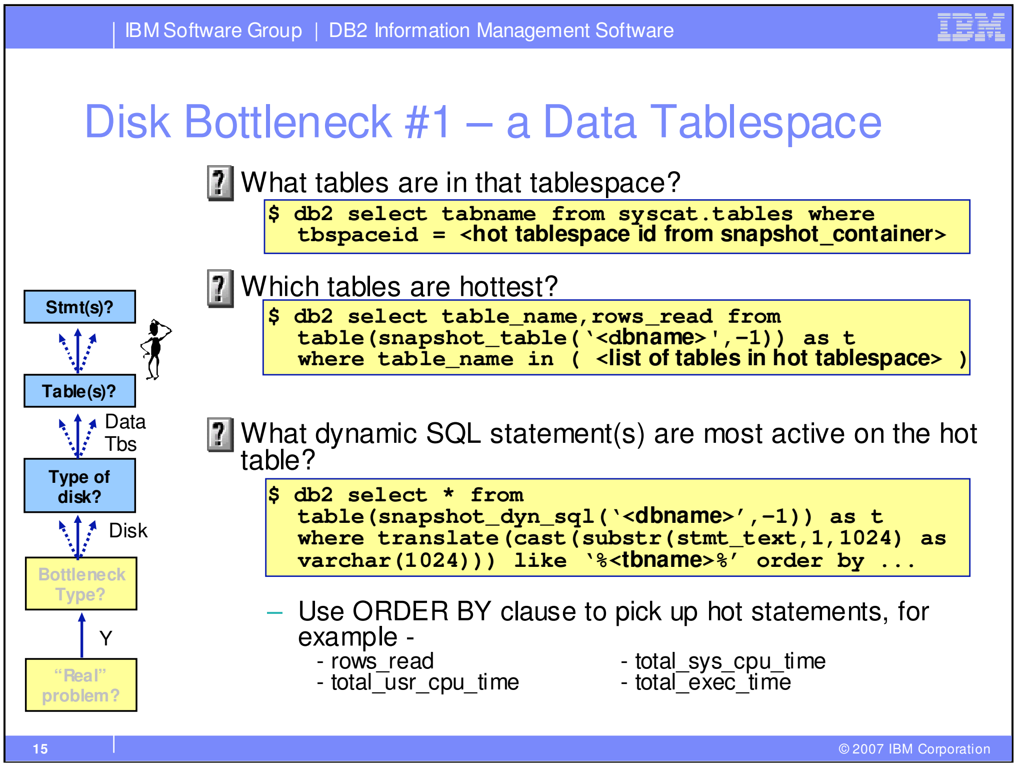
Finding the devices and filesystem paths that DB2 uses is straightforward. The log path is part of the database configuration. Container paths are deeper down inside the actual database schema, but we have multiple ways to get to that

- list tablespace containers (which we would need to issue multiple times)

- select from the table function wrapper 'snapshot_containers' (which gives us all containers in one go – very convenient)

- examining db2look output (which is a good thing to have on-hand anyway)

Note that logs and containers are obvious things to be aware of, in terms of their positioning on disk. A bit less obvious are utility files, such as load input files, backup images, etc. These are dependent on how utilities are invoked. 'db2 list utilities show detail' can provide some information here.



IBM Software Group | DB2 Information Management Software

Disk Bottleneck #1 – a Data Tablespace

Stmt(s)?

Table(s)?

Data Tbs

Type of disk?

Disk

Bottleneck Type?

Y

“Real” problem?

What tables are in that tablespace?

```
$ db2 select tabname from syscat.tables where
  tbspaceid = <hot tablespace id from snapshot_container>
```

Which tables are hottest?

```
$ db2 select table_name,rows_read from
  table(snapshot_table(‘<dbname>',-1)) as t
  where table_name in ( <list of tables in hot tablespace> )
```

What dynamic SQL statement(s) are most active on the hot table?

```
$ db2 select * from
  table(snapshot_dyn_sql(‘<dbname>’,-1)) as t
  where translate(cast(substr(stmt_text,1,1024) as
  varchar(1024))) like ‘%<tbname>%’ order by ...
```

– Use ORDER BY clause to pick up hot statements, for example -

- rows_read
- total_usr_cpu_time
- total_sys_cpu_time
- total_exec_time

15

© 2007 IBM Corporation

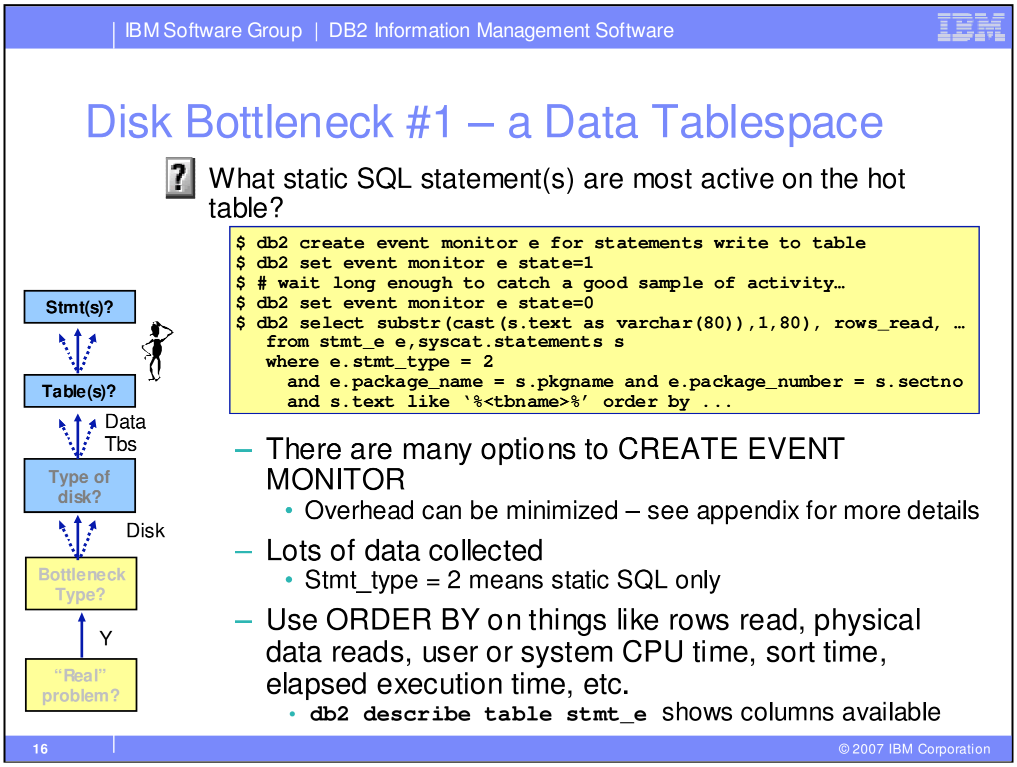
To this point, we’re drilling down from a general bottleneck, to a disk bottleneck on a particular device, and now to a bottleneck on a data tablespace which resides on that device. How do we determine what the real problem is? Why is it hot?

Finding out what tables are in the tablespace on the hot disk is fairly straightforward by querying the catalog. That gives us a list of tables which could be responsible – but which one is?

If it’s read IOs that are making the disk busy, we can look at the table snapshot to find out which of the tables in that tablespace have the greatest number of rows read. This is likely to be thte table driving the heavy read IO on the disk.

Once we figure out which tables are hot, we need to look at what kind of SQL activity might be driving all those IO’s. Which dynamic SQL statement is beating up that table? We can find that out by querying the dynamic SQL snapshot table function, looking for statements that refer to the table. We can then use other predicates or ORDER BY, etc. to focus on the hottest statements.

Note that we’ll get into much more detail on queries to these table functions, etc., in the 2nd half of the presentation.



Disk Bottleneck #1 – a Data Tablespace

Stmt(s)?
Table(s)?
Data
Tbs
Type of disk?
Disk
Bottleneck Type?
Y
“Real” problem?

? What static SQL statement(s) are most active on the hot table?

```
$ db2 create event monitor e for statements write to table
$ db2 set event monitor e state=1
$ # wait long enough to catch a good sample of activity…
$ db2 set event monitor e state=0
$ db2 select substr(cast(s.text as varchar(80)),1,80), rows_read, …
   from stmt_e e,syscat.statements s
   where e.stmt_type = 2
     and e.package_name = s.pkgname and e.package_number = s.sectno
     and s.text like ‘%<tbname>%’ order by ...
```

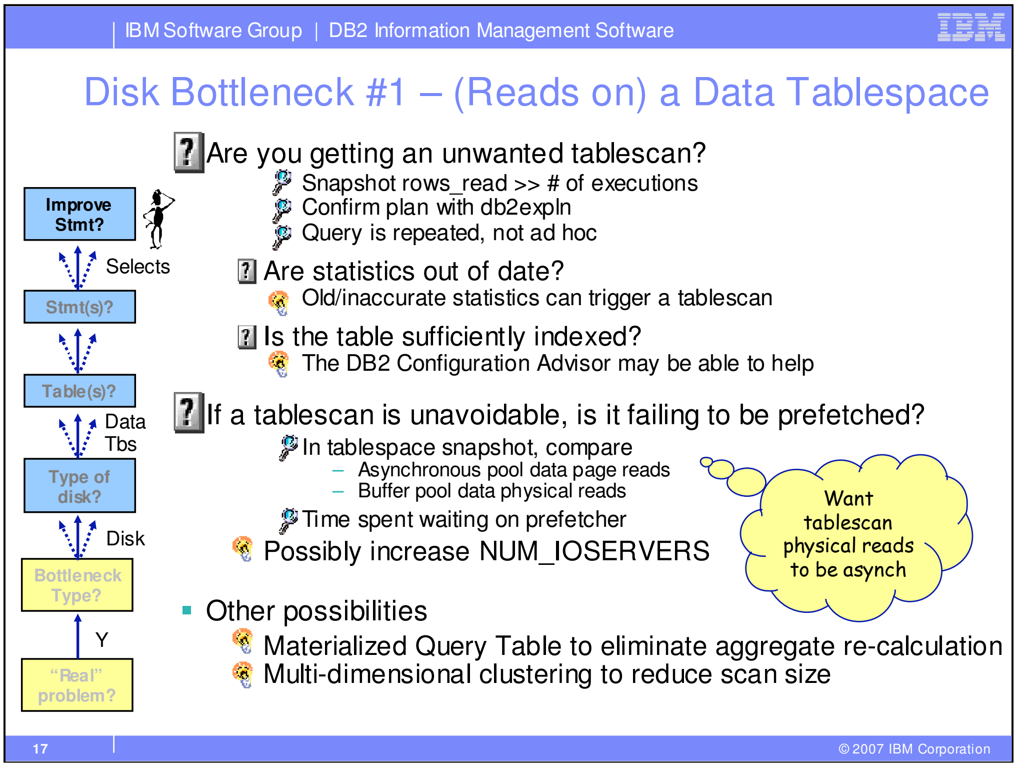
– There are many options to CREATE EVENT MONITOR
  - Overhead can be minimized – see appendix for more details
– Lots of data collected
  - Stmt_type = 2 means static SQL only
– Use ORDER BY on things like rows read, physical data reads, user or system CPU time, sort time, elapsed execution time, etc.
  - **db2 describe table stmt_e** shows columns available

It doesn’t have to be dynamic SQL that’s driving all the IO on the device – it could be static. But there’s no static SQL snapshot – so how do we find out what static SQL is beating up the table(s) in question?

The statement event monitor can provide us a wealth of data on execution activity within the DB2 engine. In particular, it records activity of static SQL statements, and also gives us a way to join event monitor records with the system catalogs, so that we can access the SQL statement text. DB2 is a powerful analysis tool – so let’s use it to summarize that data back into a form where we can easily find the hot static SQL statements.

Here what we do is join the statement event monitor table with syscat.statements, using pkg name & section number as join columns. That gives us access to the statement text, which is what we want to use to find the statements referencing one of our potentially hot tables. What we have excerpted above will put out all the operations (open & close of SELECT, INSERT, etc.) which are on a section that references the table we’re after – simple! Once we’re familiar with the basics, it’s easy to explore other data that’s available in these tables. We can filter & order by on CPU time, types of operations, etc.,

Note that statement event monitor overhead can be quite high, however there are lots of good techniques to reduce the overhead. We’ll talk about those in the 2nd half.



Disk Bottleneck #1 – (Reads on) a Data Tablespace

Improve Stmt?

Selects

Stmt(s)?

Table(s)?

Data Tbs

Type of disk?

Disk

Bottleneck Type?

Y

“Real” problem?

- Are you getting an unwanted tablescan?
  - Snapshot rows_read >> # of executions
  - Confirm plan with db2expln
  - Query is repeated, not ad hoc
- Are statistics out of date?
  - Old/inaccurate statistics can trigger a tablescan
- Is the table sufficiently indexed?
  - The DB2 Configuration Advisor may be able to help
- If a tablescan is unavoidable, is it failing to be prefetched?
  - In tablespace snapshot, compare
    - Asynchronous pool data page reads
    - Buffer pool data physical reads
  - Time spent waiting on prefetcher
  - Possibly increase NUM_IOSERVERS

Want tablescan physical reads to be asynch

- Other possibilities
  - Materialized Query Table to eliminate aggregate re-calculation
  - Multi-dimensional clustering to reduce scan size

Where are we on our process of narrowing down the problem? We’ve gone through only a few well-chosen steps and we’ve now narrowed down the problem to a particular statement. Supposing for a moment it’s a SELECT, where do we go from here?

One of the ways that such a SELECT could be over-taxing a disk is due to a tablescan. How do we tell if that’s what’s happening?

- if the ‘rows read’ statistic from the snapshot indicates a much larger value than the number of times the statement has been executed, that’s a good indication of a tablescan.

- once you suspect that, you can confirm with db2expln (good) or the explain tables and db2exfmt (better)

If the statement is one that gets run on a regular basis, then it may be worth optimizing .

Out of date or inaccurate statistics can trigger a tablescan. If the statement in question shouldn’t cause a tablescan, then the statistics are a good thing to check. Have you done runstats recently?

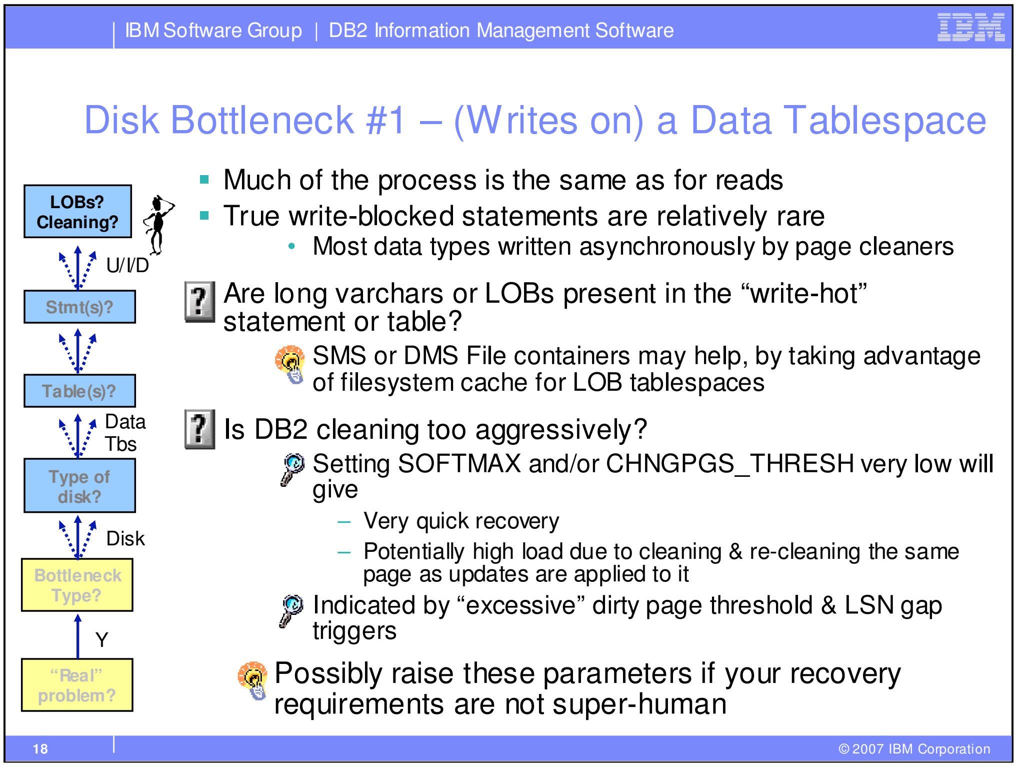
If stats aren’t the problem and the statement needs to be indexed, then the DB2 Design Advisor can help out there.

Sometimes a tablescan can’t be avoided though. In this case, we want to make sure the tablescan is as efficient as possible. Prefetching is important in that regard, since it helps streamline the whole process by letting the agent do real work instead of reading pages off disk. Also, prefetch IOs are typically larger and can be ‘easier on’ the IO subsystem.

To verify that prefetching is occurring, check the tablespace snapshot, and compare the number of ‘asynchronous pool data page reads’ (the amount the prefetchers have done) against ‘buffer pool data physical reads’ (the total number of physical data page reads from disk.)

If IOs are not almost all asynchronous, or ‘time spent waiting for prefetch’ is very high, you may want to consider increasing the number of prefetchers, or the prefetch size. Note – if you already have prefetchers processes (db2pfchr) which aren’t accumulating any CPU time, then you probably have enough of them already, and adding more might not help.

Once you’ve considered new indexes and prefetching / prefetch size, if there’s still an issue, you may want to look at MQTs or MDCs. Both of these technologies work by reducing the amont of data which has to be read to answer a query. They aren’t always applicable, but the Design Advisor does a good job of indicating which if either of these would help your workload.



Disk Bottleneck #1 – (Writes on) a Data Tablespace

LOBs?
Cleaning?
U/I/D
Stmt(s)?
Table(s)?
Data
Tbs
Type of
disk?
Disk
Bottleneck
Type?
Y
“Real”
problem?

- Much of the process is the same as for reads
- True write-blocked statements are relatively rare
  - Most data types written asynchronously by page cleaners

? Are long varchars or LOBs present in the “write-hot” statement or table?

- SMS or DMS File containers may help, by taking advantage of filesystem cache for LOB tablespaces

? Is DB2 cleaning too aggressively?

- Setting SOFTMAX and/or CHNGPGS_THRESH very low will give
  - Very quick recovery
  - Potentially high load due to cleaning & re-cleaning the same page as updates are applied to it
- Indicated by “excessive” dirty page threshold & LSN gap triggers

Possibly raise these parameters if your recovery requirements are not super-human

If our disk bottleneck is not caused by reads, it’s probably caused by writes…

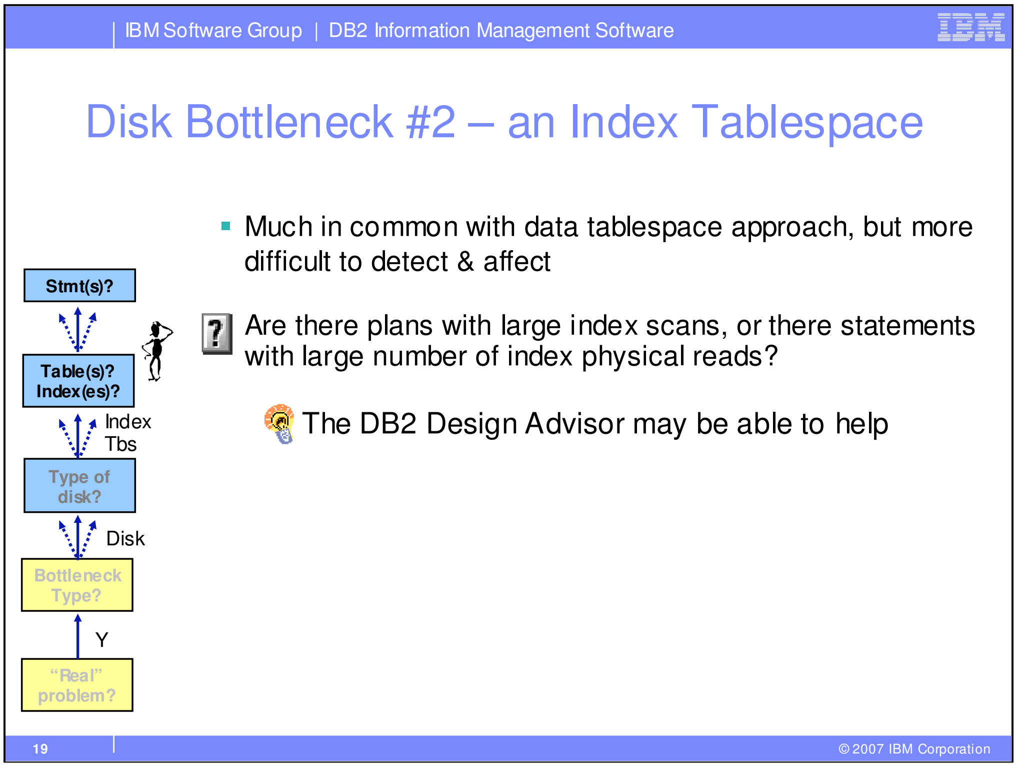
True write-blocked statements aren't that common since writes are asynchronous from statement
The write happens into bufferpool, 'dirty' page gets written out later by cleaner (usually) or by another agent (sometimes)

It is more common that the log bottlenecks before the tablespace does, since log writes are synchronous, and the log resides on a single path.
Problems with updates & inserts are more likely due to CPU & synchronization / locking issues than disk throughput capacity. This will lead us later to the ‘lazy system’ category.

Highly write-intensive tables are sometimes put in their own small bufferpool to make them 'drain' to disk more quickly and not impact other data by consuming an excessive amount of bufferpool pages. This can strain the I/O subsystem more, and might be the case in which you'd see this problem.

If DB2 is cleaning very aggressively due to low values in SOFTMAX and CHNGPGS_THRESH, this can lead to significant levels of write IO activity. If extremely fast recovery is not required, then reducing the rate of cleaning may provide some relief for an over-taxed disk subsystem.

LOBs and Long Varchars don’t go through the bufferpool, so often, putting these in SMS or DMS file tablespaces may help, since this will allow the operating system’s filesystem cache to help.



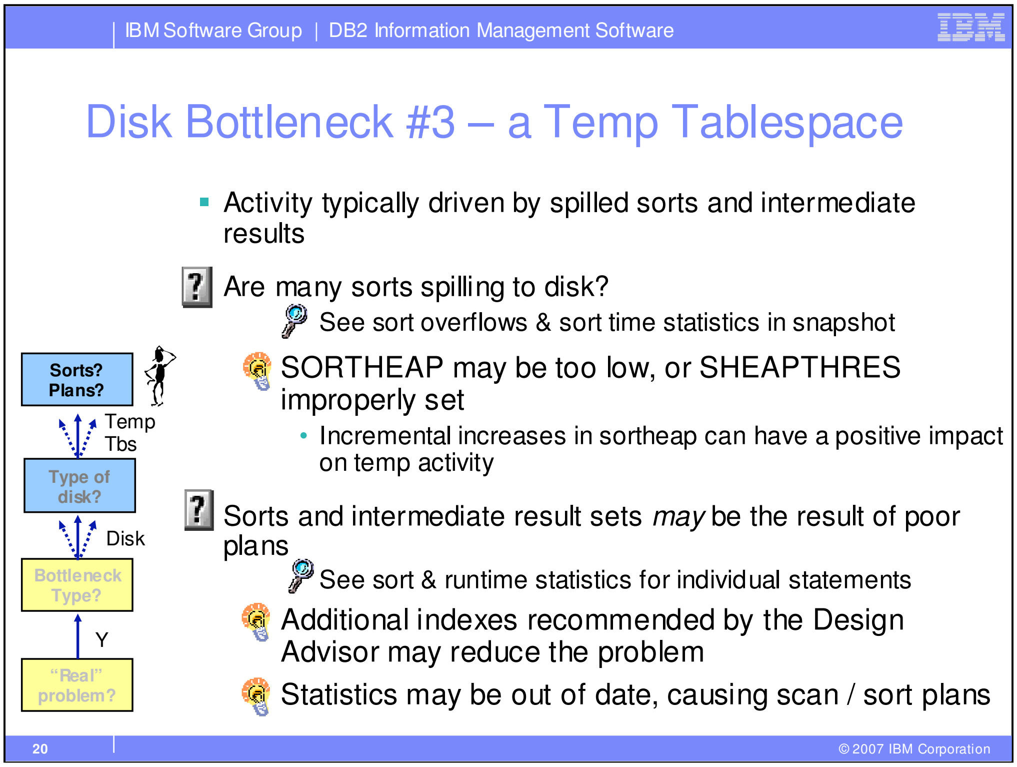
Disk Bottleneck #2 – an Index Tablespace

- Much in common with data tablespace approach, but more difficult to detect & affect

? Are there plans with large index scans, or there statements with large number of index physical reads?

The DB2 Design Advisor may be able to help

So far, we've been looking at high levels of read & write activity in DATA tablespaces. Less common but still known, are bottlenecks in index tablespaces. These are somewhat more difficult to detect, because we don't have an 'index snapshot' to let us know the levels of activity. The most obvious ways of seeing index activity are (1) highly active statements with index-only plans [dyanamic SQL snapshot; statement event monitor] (2) highly active tablespaces which contain only indexes [tablespace snapshot].



Disk Bottleneck #3 – a Temp Tablespace

- Activity typically driven by spilled sorts and intermediate results

? Are many sorts spilling to disk?

- See sort overflows & sort time statistics in snapshot
- SORTHEAP may be too low, or SHEAPTHRES improperly set
  - Incremental increases in sortheap can have a positive impact on temp activity

? Sorts and intermediate result sets *may* be the result of poor plans

- See sort & runtime statistics for individual statements
- Additional indexes recommended by the Design Advisor may reduce the problem
- Statistics may be out of date, causing scan / sort plans

Sorts? Plans?

Temp Tbs

Type of disk?

Disk

Bottleneck Type?

Y

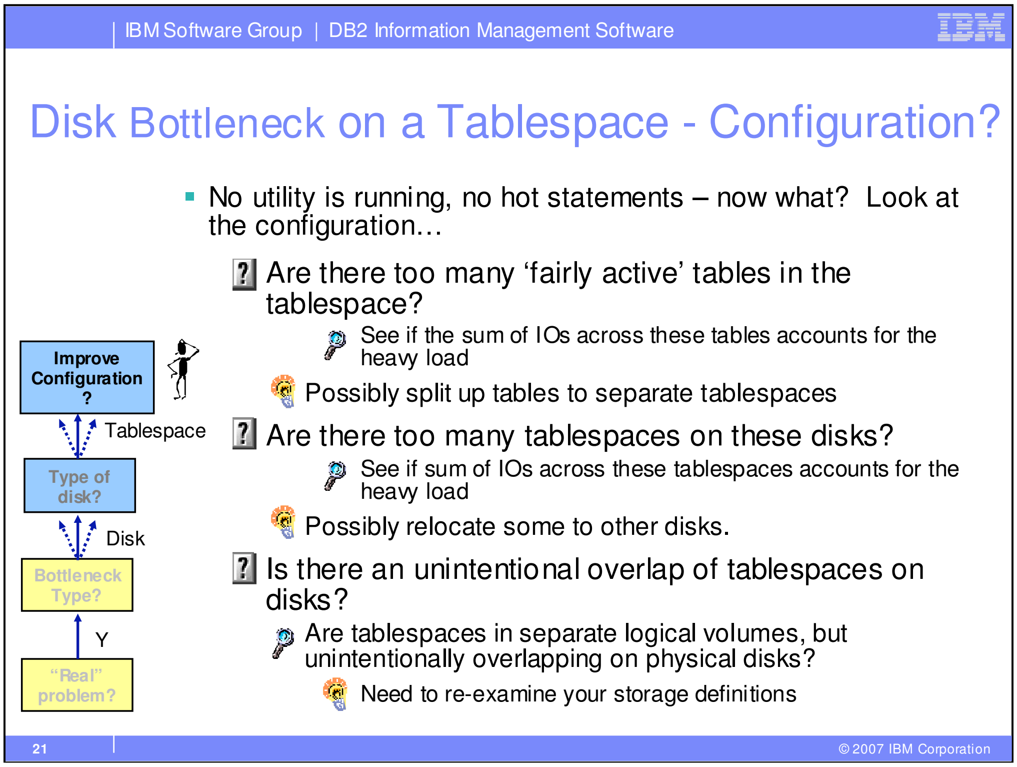
"Real" problem?

High levels of temp tablespace IO activity tend to go hand-in-hand with very large queries, with large result sets and/or lots of sorting.

High sort overflows [db snapshot], high sort time [dynamic SQL snapshot] both indicate potential trouble. We may be able to control this if there is still some headroom in SORTHEAP or SHEAPTHRES. Incremental increases in these can have a measurable influence on IO.

Opportunity for indexing to eliminate sort? (i.e., join or simple ORDER BY)

Index advisor can suggest/confirm



IBM Software Group | DB2 Information Management Software

Disk Bottleneck on a Tablespace - Configuration?

- No utility is running, no hot statements – now what? Look at the configuration…
  - Are there too many 'fairly active' tables in the tablespace?
    - See if the sum of IOs across these tables accounts for the heavy load
  - Possibly split up tables to separate tablespaces
  - Are there too many tablespaces on these disks?
    - See if sum of IOs across these tablespaces accounts for the heavy load
  - Possibly relocate some to other disks.
  - Is there an unintentional overlap of tablespaces on disks?
  - Are tablespaces in separate logical volumes, but unintentionally overlapping on physical disks?
    - Need to re-examine your storage definitions

21

If the statement plan is optimal (or at least reasonable - no tablescan, or tablescan because index not possible, etc.), or no statement really stands out, and not utility related

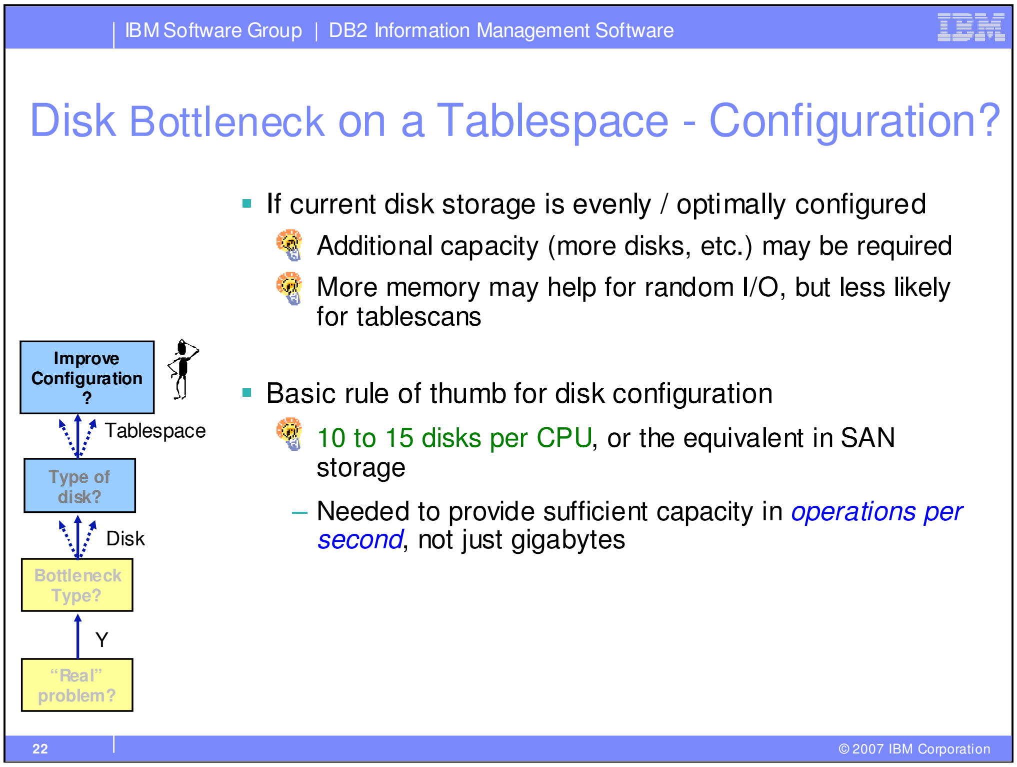
Then we may need to look at the physical database design. Check parallelism first, then faster pieces.

If it's not one table individually, it might be a collective effect of the activity on a number of tables, possibly in one tablespace. We may be able to break them up.

Similarly but at a higher level, it may be too many 'fairly active' tablespaces on one set of disks.

If it's nothing we planned – it might be something we didn't. Sometimes storage definitions – logical volumes, etc. – are quite opaque, and we can inadvertently put things on the same spindles

Can we spread out over more containers/disks? Can we add more containers/disks?



IBM Software Group | DB2 Information Management Software

Disk Bottleneck on a Tablespace - Configuration?

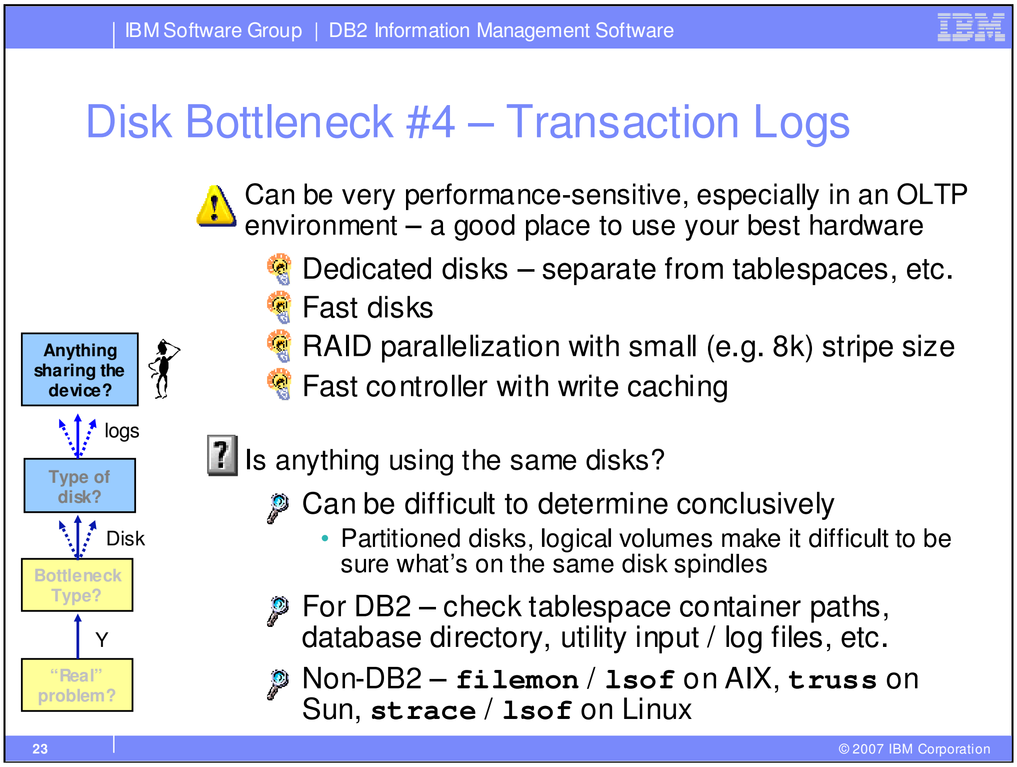
- If current disk storage is evenly / optimally configured
  - Additional capacity (more disks, etc.) may be required
  - More memory may help for random I/O, but less likely for tablescans
- Basic rule of thumb for disk configuration
  - 10 to 15 disks per CPU, or the equivalent in SAN storage
    - – Needed to provide sufficient capacity in *operations per second*, not just gigabytes

22

© 2007 IBM Corporation

If we've done all we can with what we've got – sometimes we just have to add more hardware. But the good thing is – we've come to this by way of careful consideration & due dilligence. We didn't just leap for the phone.

Don't forget – when we're talking about disk capacity for performance, it's in IO's per second. Modern disks are monsters in terms of size, but their operation speed hasn't kept up the same pace. So even if your disks are 140 GB each, you may still need 10-15 of them per CPU, if you don't want to be IO bound.



Disk Bottleneck #4 – Transaction Logs

Anything sharing the device?

logs

Type of disk?

Disk

Bottleneck Type?

Y

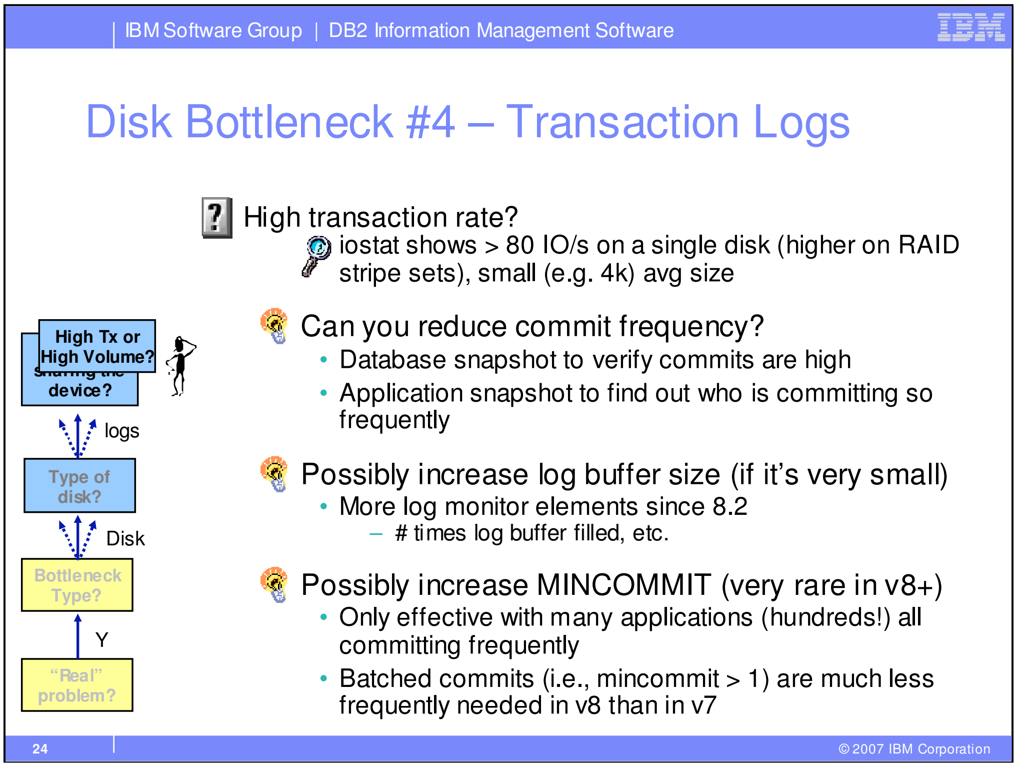
“Real” problem?

- Can be very performance-sensitive, especially in an OLTP environment – a good place to use your best hardware
  - Dedicated disks – separate from tablespaces, etc.
  - Fast disks
  - RAID parallelization with small (e.g. 8k) stripe size
  - Fast controller with write caching
- Is anything using the same disks?
  - Can be difficult to determine conclusively
    - Partitioned disks, logical volumes make it difficult to be sure what’s on the same disk spindles
  - For DB2 – check tablespace container paths, database directory, utility input / log files, etc.
  - Non-DB2 – **`filemon`** / **`lsof`** on AIX, **`truss`** on Sun, **`strace`** / **`lsof`** on Linux

Transaction logs are often the Achilles’ heel of a system, because they can often make the difference between a well-performing system and a poorly performing one. This is an important area to configure carefully.

First, never (if you’re concerned about performance) let the logs go to their default location, which is under the database directory. It’s generally advisable to keep them on their own spindles, and for high-transaction rate systems, use multiple disks in a RAID configuration.

If you suspect a log IO bottleneck (high IO wait time on the log devices, high log IO time in the snapshots), check to make sure nothing else is on the same disks. Start with examine the log path – but you may have to go as far as to get down to the device level. OS tools can help here.



Disk Bottleneck #4 – Transaction Logs

High Tx or High Volume?

Sharing the device?

logs

Type of disk?

Disk

Bottleneck Type?

Y

“Real” problem?

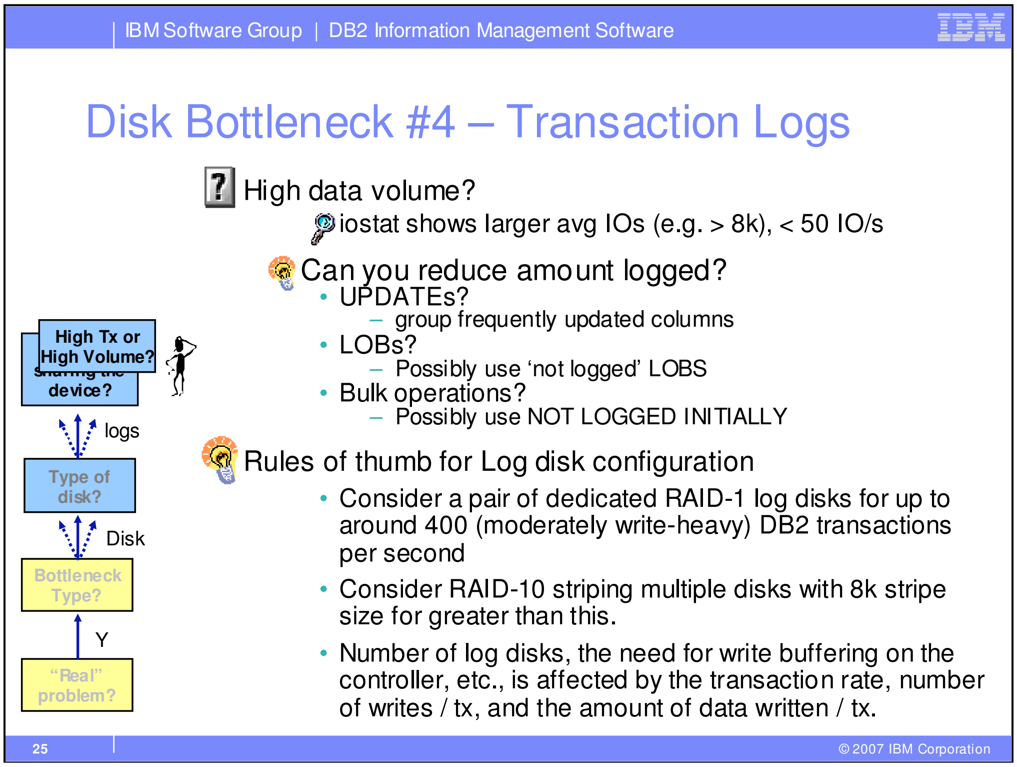
- High transaction rate?
  - iostat shows > 80 IO/s on a single disk (higher on RAID stripe sets), small (e.g. 4k) avg size
- Can you reduce commit frequency?
  - Database snapshot to verify commits are high
  - Application snapshot to find out who is committing so frequently
- Possibly increase log buffer size (if it’s very small)
  - More log monitor elements since 8.2
    - # times log buffer filled, etc.
- Possibly increase MINCOMMIT (very rare in v8+)
  - Only effective with many applications (hundreds!) all committing frequently
  - Batched commits (i.e., mincommit > 1) are much less frequently needed in v8 than in v7

Often the bottleneck on the log is due to an excessive number of small IOs, particularly in high Tx rate systems. In this case you might see log IO size around 4k in the snapshots.

Can you reduce commit frequency? (db snapshot to verify commits are high; app snapshot to find who is committing so much)

In rare cases, it might be useful to increase MINCOMMIT. This used to have a greater impact in v7, and its impact varies by platform (generally very low on AIX.) The most benefit would be seen in systems where there are hundreds of connections running very small transactions.

Do you need to increase log buffer size? (only relevant if it's quite small; see new snapshot element indicating ‘log buffer full’ condition.)



Disk Bottleneck #4 – Transaction Logs

High data volume?

iostat shows larger avg IOs (e.g. > 8k), < 50 IO/s

Can you reduce amount logged?

- UPDATEs?
  - group frequently updated columns
- LOBs?
  - Possibly use ‘not logged’ LOBS
- Bulk operations?
  - Possibly use NOT LOGGED INITIALLY

Rules of thumb for Log disk configuration

- Consider a pair of dedicated RAID-1 log disks for up to around 400 (moderately write-heavy) DB2 transactions per second
- Consider RAID-10 striping multiple disks with 8k stripe size for greater than this.
- Number of log disks, the need for write buffering on the controller, etc., is affected by the transaction rate, number of writes / tx, and the amount of data written / tx.

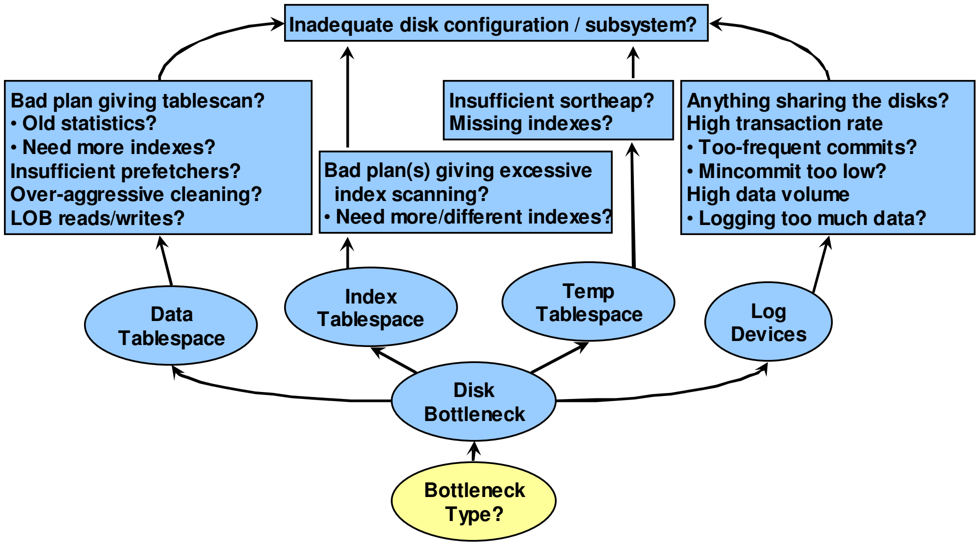
Less common is the case where we’re bottlenecked because of high throughput to the log. (average I/O size 'quite a bit larger' than 4k - 8k, 16k, etc.)

Can we reduce the amount logged? If we have freedom to change the schema, we might put columns to be updated together at beginning of row

If it’s due to a bulk operation - import, insert w. subselect, etc. – possibly look at NOT LOGGED INITIALLY

If it’s due to LOB operation, possibly look at not logged LOBs

Disk Bottleneck Diagnosis – 10,000 feet



Inadequate disk configuration / subsystem?

Bad plan giving tablescan?
- Old statistics?
- Need more indexes?

Insufficient prefetchers?
Over-aggressive cleaning?
LOB reads/writes?

Bad plan(s) giving excessive index scanning?
- Need more/different indexes?

Insufficient sortheap?
Missing indexes?

Anything sharing the disks?
High transaction rate
- Too-frequent commits?
- Mincommit too low?

High data volume
- Logging too much data?

Data Tablespace

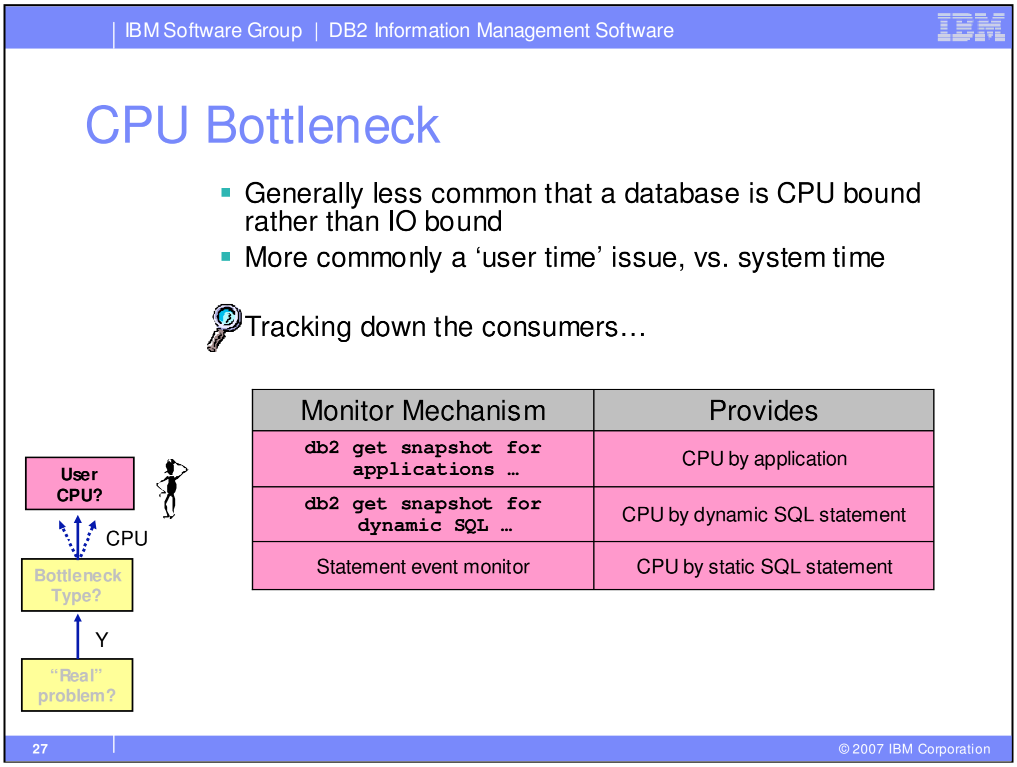
Index Tablespace

Temp Tablespace

Log Devices

Disk Bottleneck

Bottleneck Type?



IBM Software Group | DB2 Information Management Software

CPU Bottleneck

- Generally less common that a database is CPU bound rather than IO bound
- More commonly a 'user time' issue, vs. system time

Tracking down the consumers…

Monitor Mechanism	Provides
`db2 get snapshot for applications …`	CPU by application
`db2 get snapshot for dynamic SQL …`	CPU by dynamic SQL statement
Statement event monitor	CPU by static SQL statement

27

CPU bottleneck

It is less common that a well-tuned database system is CPU blocked than disk blocked (partly because CPU power is cheaper than disk power)

so, be suspicious if you're running out of CPU and haven't already tuned your disk carefully

We're looking for things consuming CPU that shouldn't be

What's using it up?

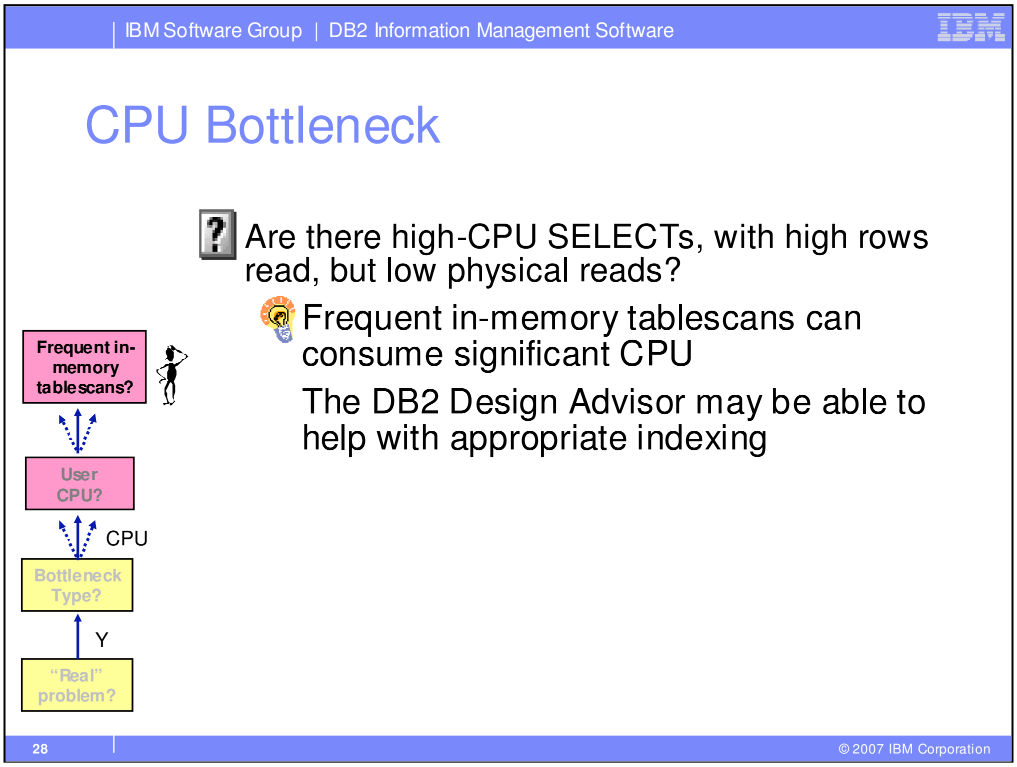
application snapshot - CPU consumption broken down by application

dynamic SQL snapshot - CPU consumption broken down by dynamic statement

statement event monitor - CPU consumption by static statement (sum up over all instances)

We usually associate tablescans with lots of IO, but an in-memory scan of a small table can be a CPU sponge …

If this occurs, see if an index could be applied.



IBM Software Group | DB2 Information Management Software

CPU Bottleneck

Are there high-CPU SELECTs, with high rows read, but low physical reads?

Frequent in-memory tablescans can consume significant CPU

The DB2 Design Advisor may be able to help with appropriate indexing

Frequent in-memory tablescans?

User CPU?

CPU

Bottleneck Type?

Y

"Real" problem?

28

© 2007 IBM Corporation

CPU bottleneck

It is less common that a well-tuned database system is CPU blocked than disk blocked (partly because CPU power is cheaper than disk power)

so, be suspicious if you're running out of CPU and haven't already tuned your disk carefully

We're looking for things consuming CPU that shouldn't be

What's using it up?

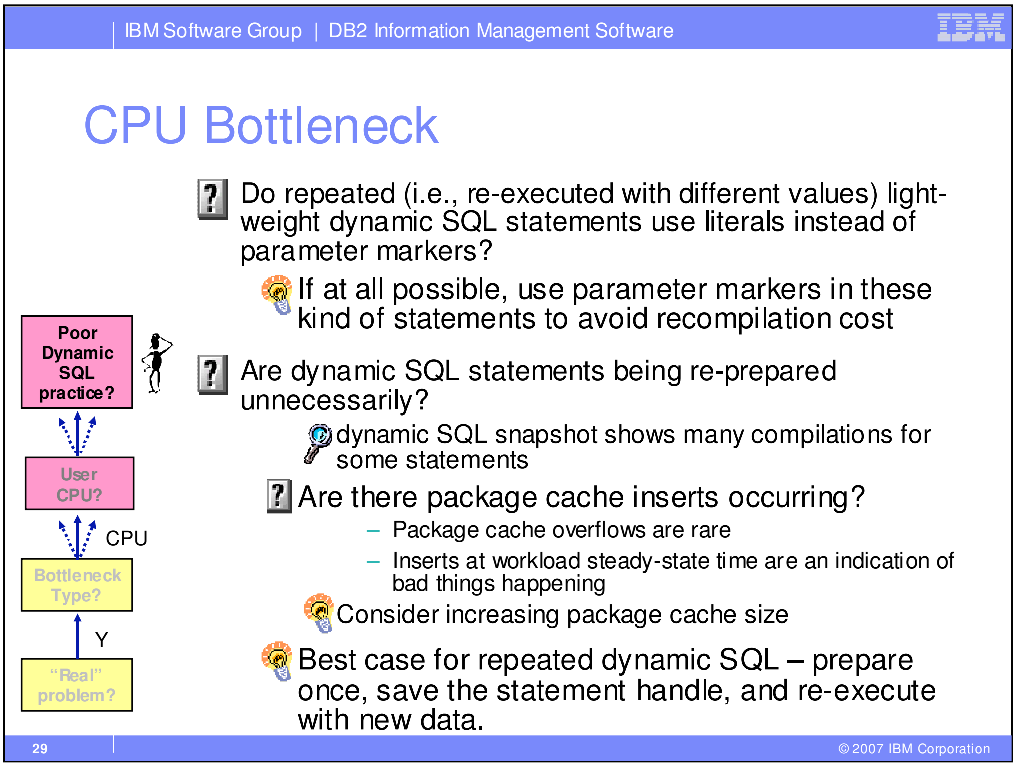
application snapshot - CPU consumption broken down by application

dynamic SQL snapshot - CPU consumption broken down by dynamic statement

statement event monitor - CPU consumption by static statement (sum up over all instances)

We usually associate tablescans with lots of IO, but an in-memory scan of a small table can be a CPU sponge …

If this occurs, see if an index could be applied.



CPU Bottleneck

Poor Dynamic SQL practice?

User CPU?

CPU

Bottleneck Type?

Y

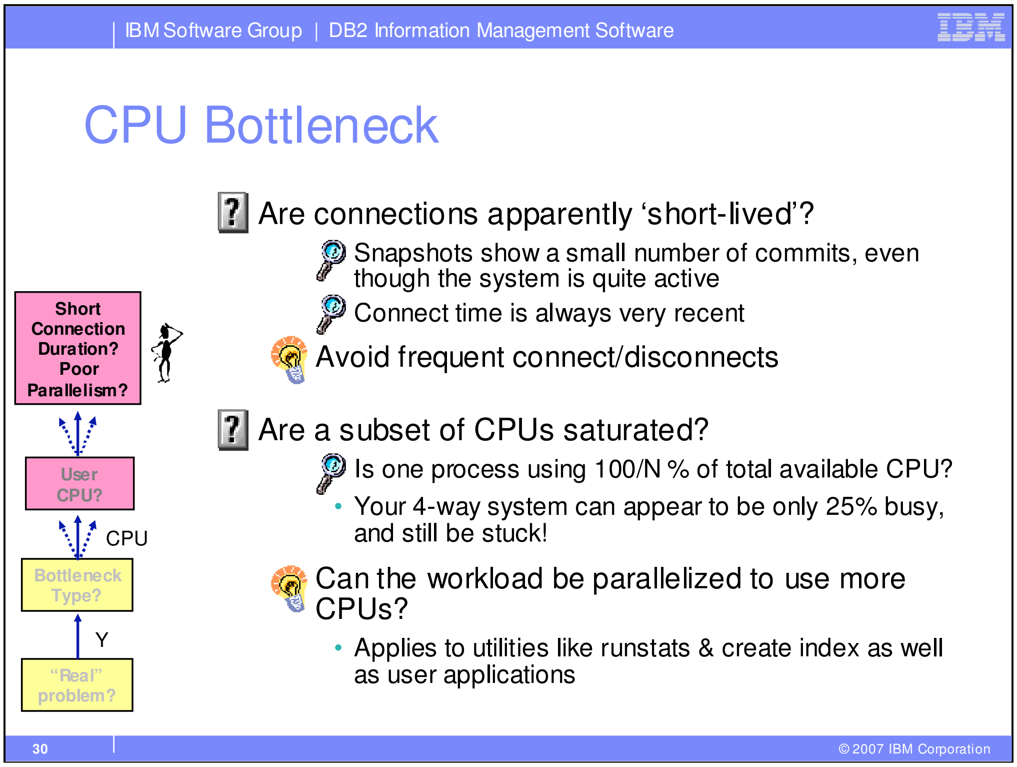
“Real” problem?

- ? Do repeated (i.e., re-executed with different values) light-weight dynamic SQL statements use literals instead of parameter markers?
  - If at all possible, use parameter markers in these kind of statements to avoid recompilation cost
- ? Are dynamic SQL statements being re-prepared unnecessarily?
  - dynamic SQL snapshot shows many compilations for some statements
  - ? Are there package cache inserts occurring?
    - Package cache overflows are rare
    - Inserts at workload steady-state time are an indication of bad things happening
  - Consider increasing package cache size
- Best case for repeated dynamic SQL – prepare once, save the statement handle, and re-execute with new data.

Recompiling simple SQL statements that differ only by literal values can be extremely expensive. Individual compilations are cheap, but they can really add up. Look for statements that differ only by parameter values in the dynamic SQL snapshot.

Another possibility - is the package cache large enough to hold all steady-state dynamic SQL statements (assuming there is a maximum steady-state number) - look for package cache inserts

In an iterative system with very simple statements (ie, that don’t make use of distribution stats), the best case - dynamic SQL applications prepare once, using parameter markers, and re-execute as necessary.



IBM Software Group | DB2 Information Management Software

CPU Bottleneck

Short Connection Duration? Poor Parallelism?

User CPU?

CPU

Bottleneck Type?

Y

“Real” problem?

? Are connections apparently ‘short-lived’?

- Snapshots show a small number of commits, even though the system is quite active
- Connect time is always very recent

Avoid frequent connect/disconnects

? Are a subset of CPUs saturated?

- Is one process using 100/N % of total available CPU?
- Your 4-way system can appear to be only 25% busy, and still be stuck!

Can the workload be parallelized to use more CPUs?

- Applies to utilities like runstats & create index as well as user applications

30

Building and tearing down connections can be very expensive, especially relative to lighter-weight transactions.

Check out application snapshot - Do agents have a small number of commits? Is the connect timestamp very recent?

More conclusively, the statement event monitor will report a connection event every time a new connection is made.

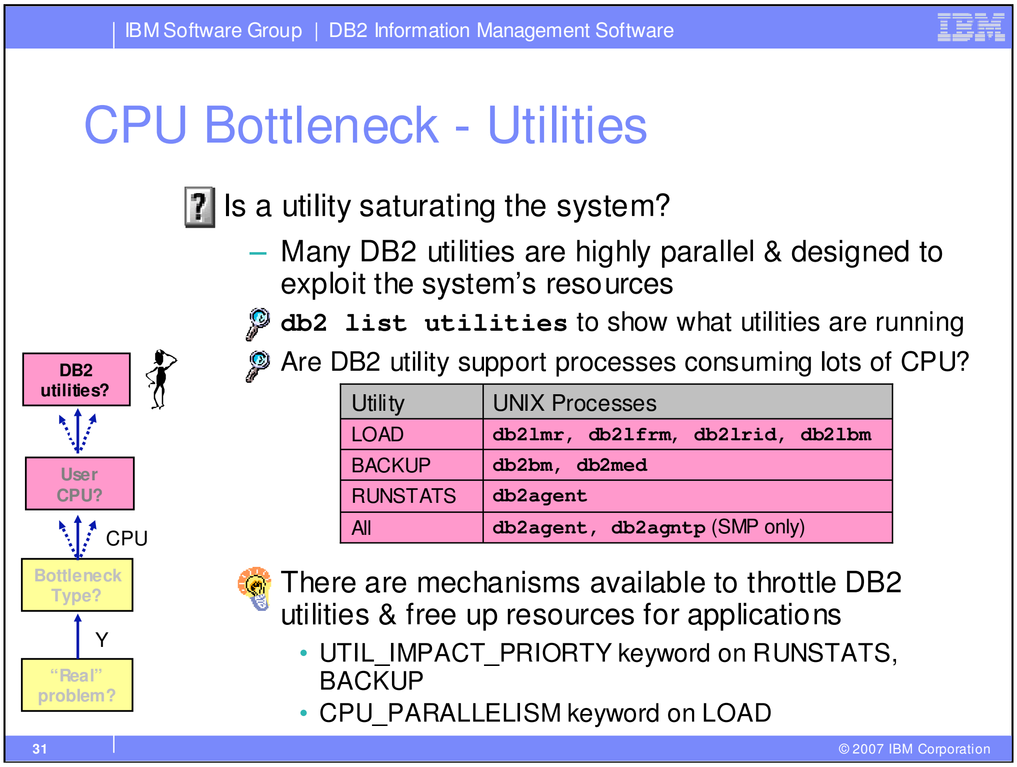
A CPU bottleneck doesn’t have to saturate the whole system - it can also be on a single CPU

if CPU utilization tops out at 100/(# CPUs) % with one dominant agent/process, there may be an opportunity to improve performance by parallelizing the workload

creating multiple application threads, eg for batch jobs processing date ranges (care needs to be paid to locking issues)

concurrent utilities

multiple loads or runstats



CPU Bottleneck - Utilities

? Is a utility saturating the system?

- Many DB2 utilities are highly parallel & designed to exploit the system's resources

`db2 list utilities` to show what utilities are running

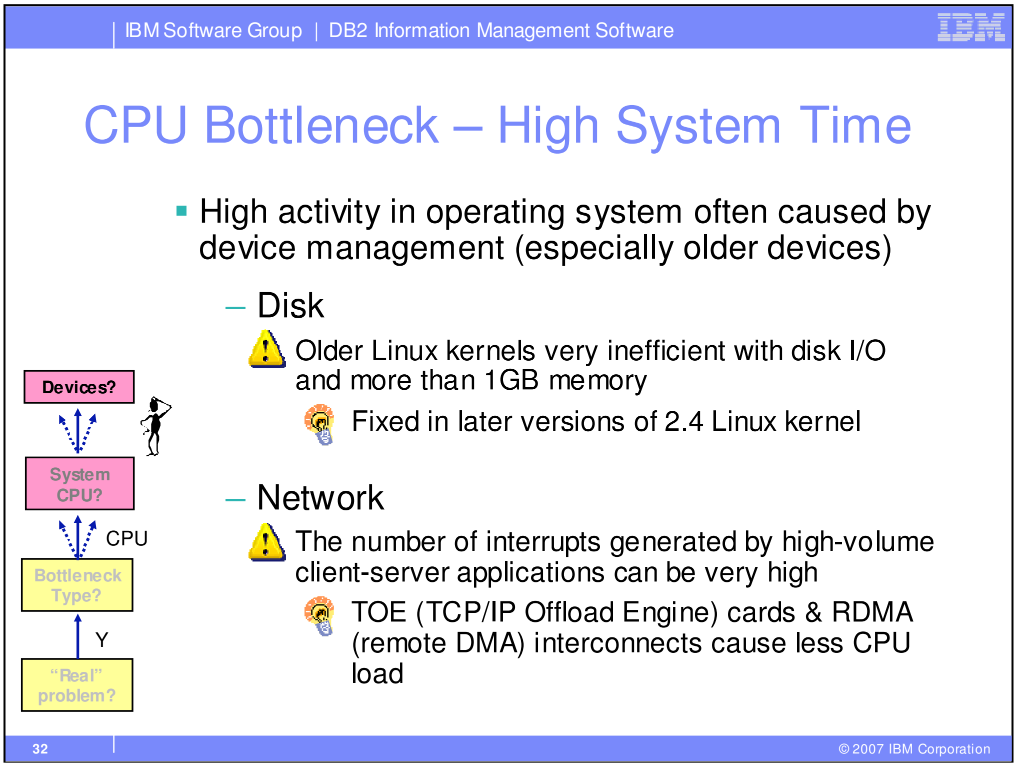
Are DB2 utility support processes consuming lots of CPU?

Utility	UNIX Processes
LOAD	**`db2lmr, db2lfrm, db2lrid, db2lbm`**
BACKUP	**`db2bm, db2med`**
RUNSTATS	**`db2agent`**
All	**`db2agent, db2agntp`** (SMP only)

There are mechanisms available to throttle DB2 utilities & free up resources for applications

- UTIL_IMPACT_PRIORTY keyword on RUNSTATS, BACKUP
- CPU_PARALLELISM keyword on LOAD

Well-run utilities may use a significant amount of resource (they're intended to scale well, so they will often use a lot of CPU). This can create a problem if it denies CPU to other applications Look at utility throttling.



CPU Bottleneck – High System Time

- High activity in operating system often caused by device management (especially older devices)
  - Disk
    - ⚠ Older Linux kernels very inefficient with disk I/O and more than 1GB memory
      - 💡 Fixed in later versions of 2.4 Linux kernel
  - Network
    - ⚠ The number of interrupts generated by high-volume client-server applications can be very high
      - 💡 TOE (TCP/IP Offload Engine) cards & RDMA (remote DMA) interconnects cause less CPU load

Most cases of high CPU usage map to high user time. High system time is more rare.

In general this means lots of activity in things the operating system is responsible for, such as device management, especially with older (stupider) devices

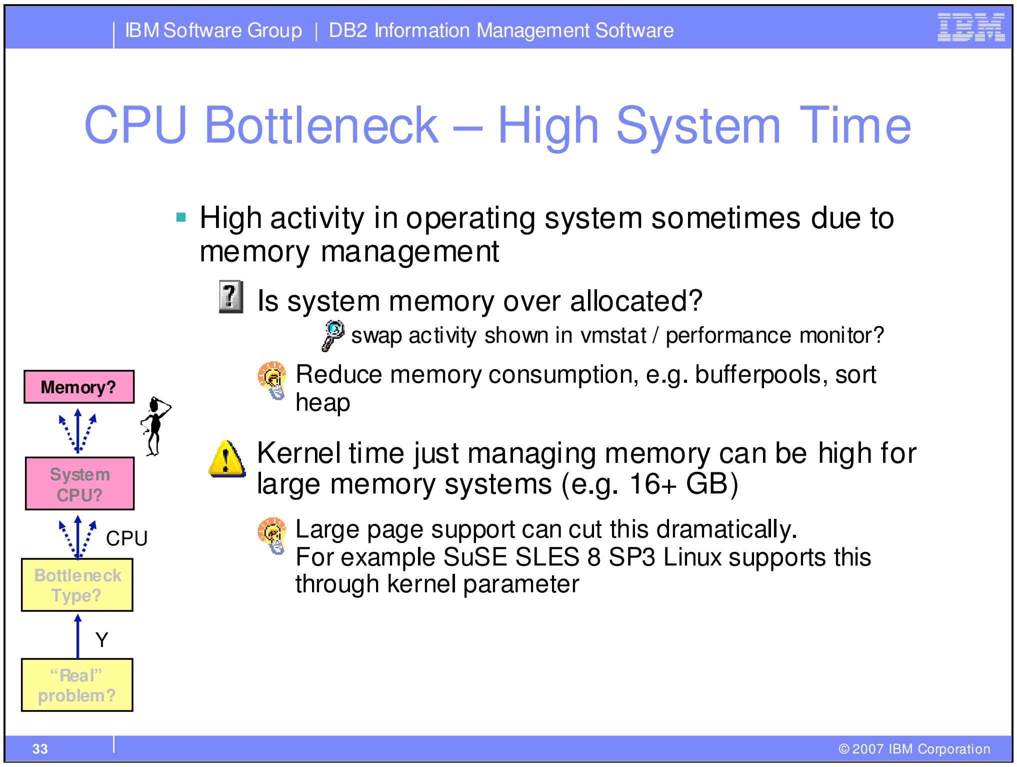
disk I/O

DMA & smart controllers help alleviate load from the CPU

Linux especially - older kernel levels did internal copying for disk ops when > 1Gb real memory available.

Network I/0

TOE cards implement much of the TCP/IP stack in firmware, off-loading the CPU

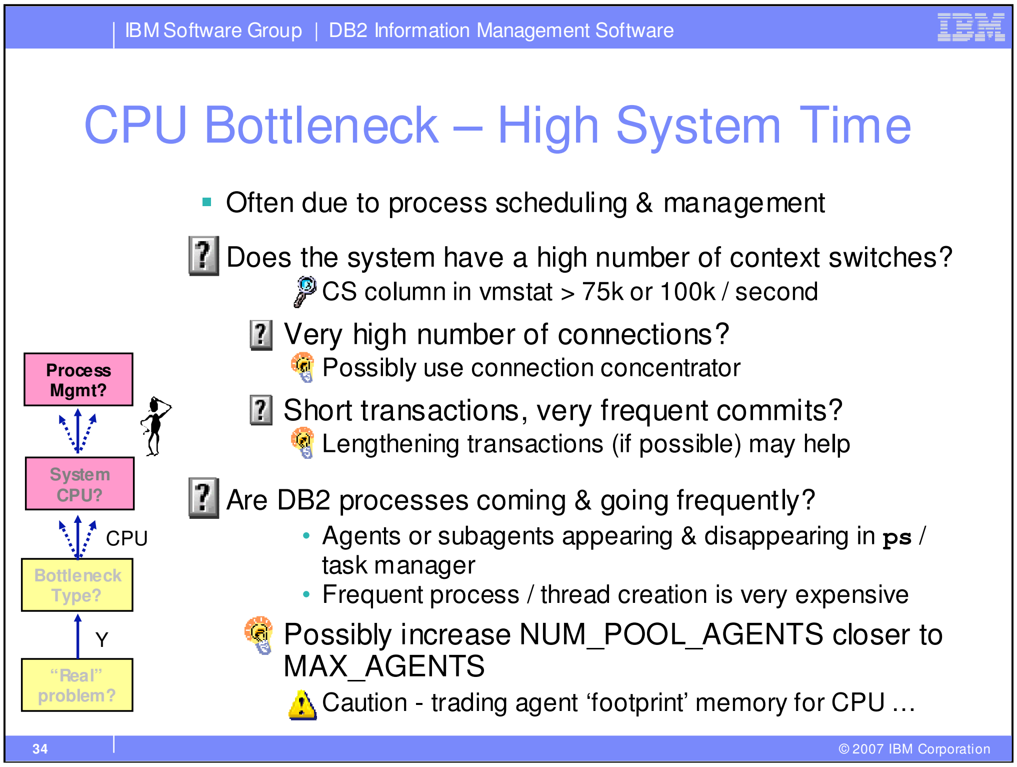


CPU Bottleneck – High System Time

- High activity in operating system sometimes due to memory management
  - ? Is system memory over allocated?
    - swap activity shown in vmstat / performance monitor?
  - Reduce memory consumption, e.g. bufferpools, sort heap
  - ⚠ Kernel time just managing memory can be high for large memory systems (e.g. 16+ GB)
  - Large page support can cut this dramatically.
    For example SuSE SLES 8 SP3 Linux supports this through kernel parameter

Memory over allocation is another potential culprit, although it would often drive high IO with disk swapping activity.

Also, monster memory configurations of 64GB or larger may make good use of ‘large memory pages’ provided by the operating system. Usually the OS handles memory pages at a resolution of a k or two. With so much memory, the page tables the O/S has to handle are huge. Using large memory pages can significantly reduce memory management cost.



CPU Bottleneck – High System Time

- Often due to process scheduling & management

? Does the system have a high number of context switches?
  - CS column in vmstat > 75k or 100k / second
  - ? Very high number of connections?
    - Possibly use connection concentrator
  - ? Short transactions, very frequent commits?
    - Lengthening transactions (if possible) may help

? Are DB2 processes coming & going frequently?
  - Agents or subagents appearing & disappearing in **ps** / task manager
  - Frequent process / thread creation is very expensive
  - Possibly increase NUM_POOL_AGENTS closer to MAX_AGENTS
    - ⚠ Caution - trading agent ‘footprint’ memory for CPU …

Process Mgmt?

System CPU?

CPU

Bottleneck Type?

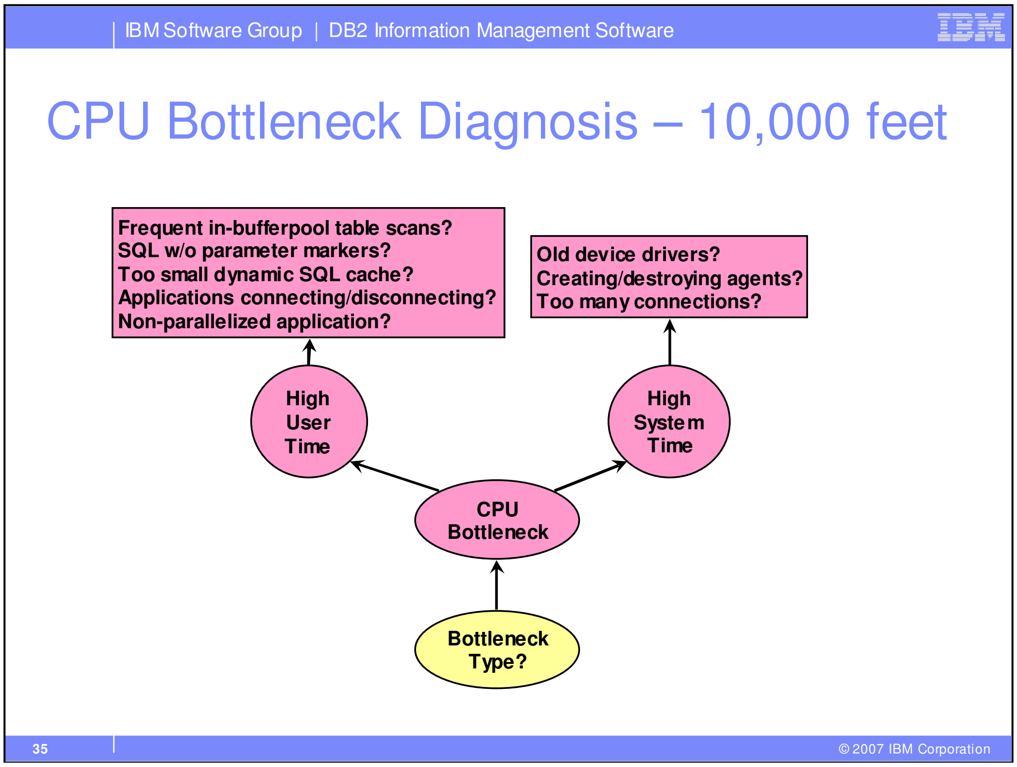
Y

“Real” problem?

Another fairly common cause of high system CPU is context switching, particularly in an environment with an extremely high number of connections. Using the connection concentrator will support the same number of application connections, but reduce the number of agents in the system. This will reduce memory footprint, and may help reduce context switches as well. Note that this works best for short transactions, since the concentrator can only move switch between servicing connections on UOW boundaries.

Sometimes high context switches are driven by very high commit rates – e.g. autocommit – due to the requirement of getting the transaction logger involved. Increasing transaction size may help.

Rapid creation/destruction of DB2 processes (db2 agents in the case of frequent connect/disconnect; subagents in the case of SMP or MPP parallelism) can show up as high system CPU as well, since process creation/destruction is one of the more expensive operations an OS does.



CPU Bottleneck Diagnosis – 10,000 feet

Bottleneck Type?

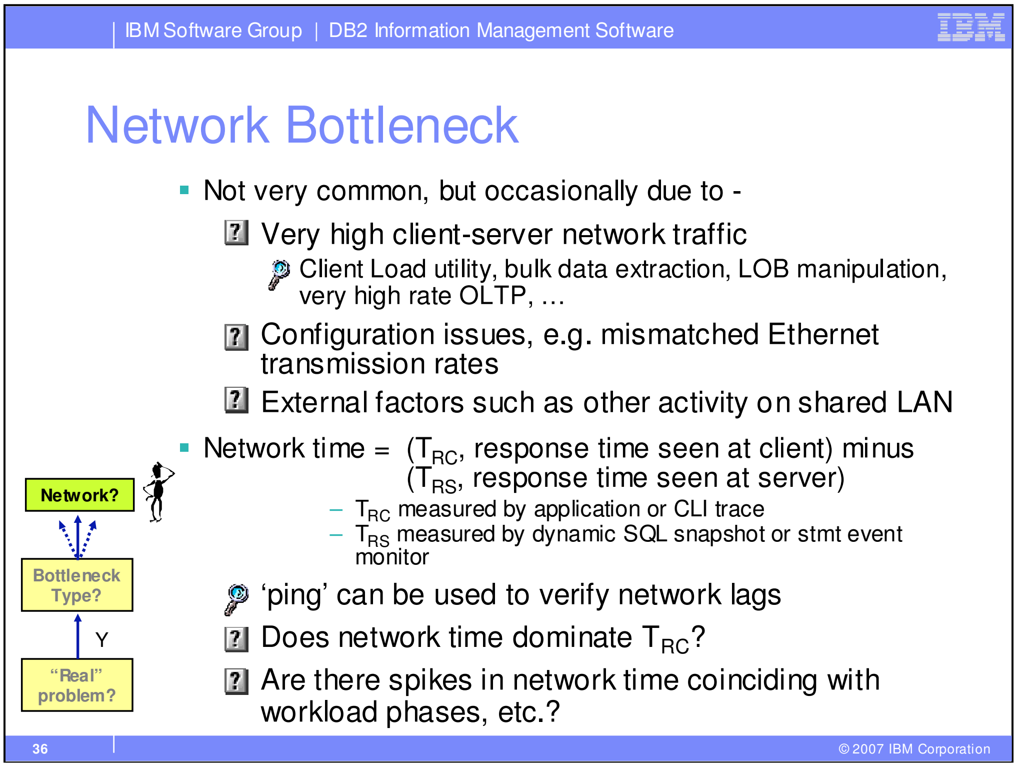
CPU Bottleneck

High User Time

Frequent in-bufferpool table scans?
SQL w/o parameter markers?
Too small dynamic SQL cache?
Applications connecting/disconnecting?
Non-parallelized application?

High System Time

Old device drivers?
Creating/destroying agents?
Too many connections?



Network Bottleneck

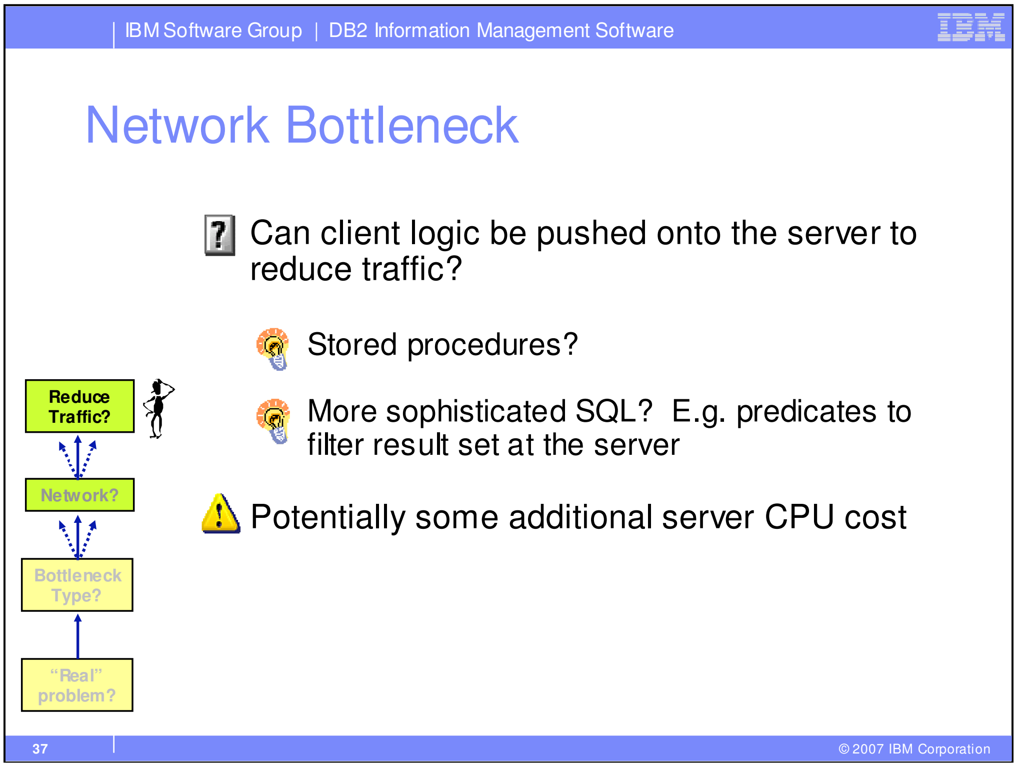
- Not very common, but occasionally due to -
  - Very high client-server network traffic
    - Client Load utility, bulk data extraction, LOB manipulation, very high rate OLTP, …
  - Configuration issues, e.g. mismatched Ethernet transmission rates
  - External factors such as other activity on shared LAN
- Network time = (T_{RC}, response time seen at client) minus (T_{RS}, response time seen at server)
  - T_{RC} measured by application or CLI trace
  - T_{RS} measured by dynamic SQL snapshot or stmt event monitor
  - 'ping' can be used to verify network lags
  - Does network time dominate T_{RC}?
  - Are there spikes in network time coinciding with workload phases, etc.?

Network bottlenecks typically arise from very high volumes of data being moved around – very large result sets, client load, etc. – although they can also arise from configuration problems, such as network cards accidentally being left set half-duplex, or an incorrect speed setting, etc. Of course, if the network is shared then the combined load of a bunch of moderate workloads can cause problems too.

To diagnose this, we need to look at the server response time (are statements running fast at the server? – measurable by snapshots & event monitors) and compare it with the response time seen at the client (measurable by the application, or by a CLI or Java trace). A significant difference generally indicates a network problem.

'ping' is a useful tool for monitoring network performance. It simply sends a small request to a remote machine (i.e. from client to server) and report how long the round-trip takes. A healthy, high-performance LAN should have ping times under 1ms.

Network loads are rarely constant, so it's important to measure the load during different phases of operation.



IBM Software Group | DB2 Information Management Software

Network Bottleneck

Reduce Traffic?
Network?
Bottleneck Type?
"Real" problem?

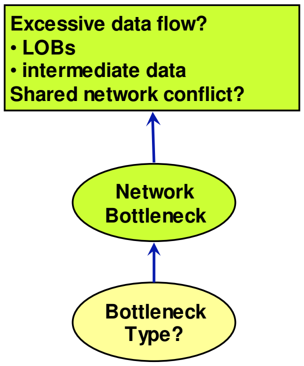
- Can client logic be pushed onto the server to reduce traffic?
  - Stored procedures?
  - More sophisticated SQL? E.g. predicates to filter result set at the server
- ⚠ Potentially some additional server CPU cost

37

One way to reduce data flow, particularly for large result sets, is to push the logic which deals with it down onto the server. This can be done with stored procedures, UDFs & even some of the more powerful SQL added to DB2. Many legacy applications rely on client-side processing to filter, join or sort data, etc., which often requires a great deal of data flow from server to client. DB2 can obviously do this kind of processing very efficiently.

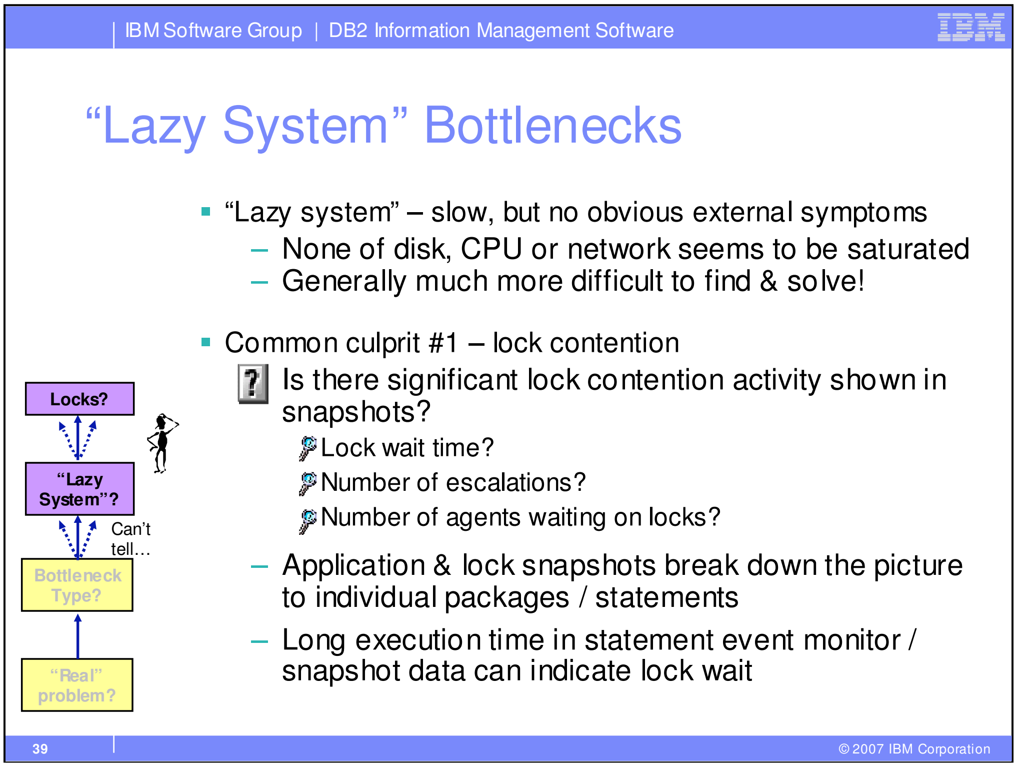
Of course, this is pushing more work onto the database server, so you need to consider how network & server resources are balanced on your system.

Network Bottleneck Diagnosis – 10,000 ft



Excessive data flow?
- LOBs
- intermediate data

Shared network conflict?

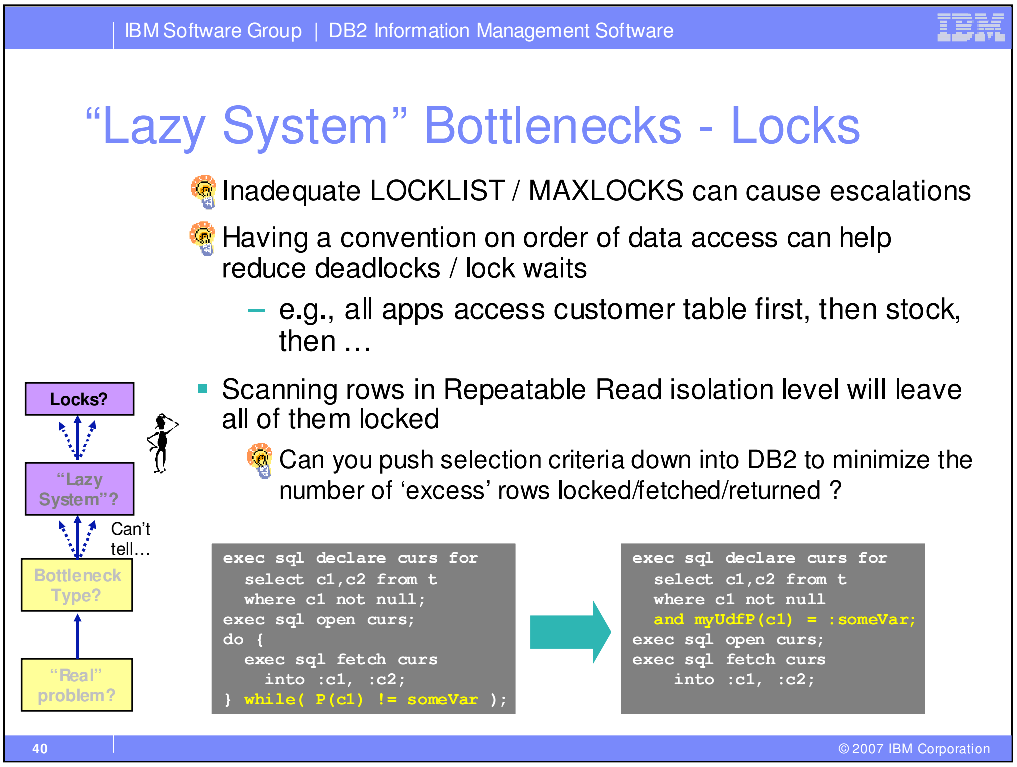


"Lazy System" Bottlenecks

- "Lazy system" – slow, but no obvious external symptoms
  - None of disk, CPU or network seems to be saturated
  - Generally much more difficult to find & solve!
- Common culprit #1 – lock contention
  - ? Is there significant lock contention activity shown in snapshots?
    - Lock wait time?
    - Number of escalations?
    - Number of agents waiting on locks?
  - Application & lock snapshots break down the picture to individual packages / statements
  - Long execution time in statement event monitor / snapshot data can indicate lock wait

"lazy system" bottlenecks are probably the most frustrating. You're be looking at your system, and no resource will see saturated, yet you can't make it run any faster. What's up?

Locking is the most common culprit. There can be long individual wait times, but in the case of a significant overall slowdown it's generally seen as a large number of agents waiting on locks. Lock snapshots are useful, and the snapshot_lockwait table function is great at showing lock wait dependencies.



“Lazy System” Bottlenecks - Locks

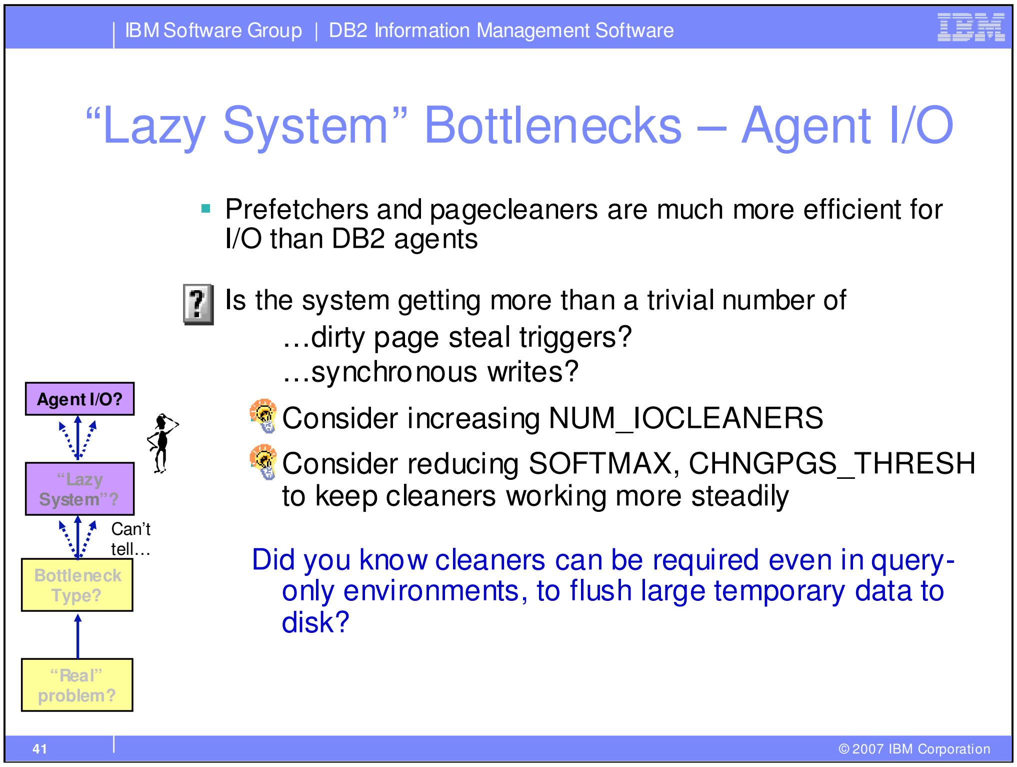
- Inadequate LOCKLIST / MAXLOCKS can cause escalations
- Having a convention on order of data access can help reduce deadlocks / lock waits
  - e.g., all apps access customer table first, then stock, then …
- Scanning rows in Repeatable Read isolation level will leave all of them locked
  - Can you push selection criteria down into DB2 to minimize the number of ‘excess’ rows locked/fetched/returned ?

```
exec sql declare curs for
  select c1,c2 from t
  where c1 not null;
exec sql open curs;
do {
  exec sql fetch curs
    into :c1, :c2;
} while( P(c1) != someVar );
```

```
exec sql declare curs for
  select c1,c2 from t
  where c1 not null
  and myUdfP(c1) = :someVar;
exec sql open curs;
exec sql fetch curs
    into :c1, :c2;
```

In bulk operations, try to commit reasonably often to avoid accumulating an excessive number of locks.

Locking also has CPU overhead - running standalone batch jobs at locksize table saves locking/unlocking cost for each row.



IBM Software Group | DB2 Information Management Software

"Lazy System" Bottlenecks – Agent I/O

- Prefetchers and pagecleaners are much more efficient for I/O than DB2 agents

? Is the system getting more than a trivial number of

…dirty page steal triggers?

…synchronous writes?

Consider increasing NUM_IOCLEANERS

Consider reducing SOFTMAX, CHNGPGS_THRESH to keep cleaners working more steadily

Did you know cleaners can be required even in query-only environments, to flush large temporary data to disk?

41

© 2007 IBM Corporation

Likely culprit #2 - DB2 agents doing I/O instead of cleaners & prefetchers

Case 1 - cleaners

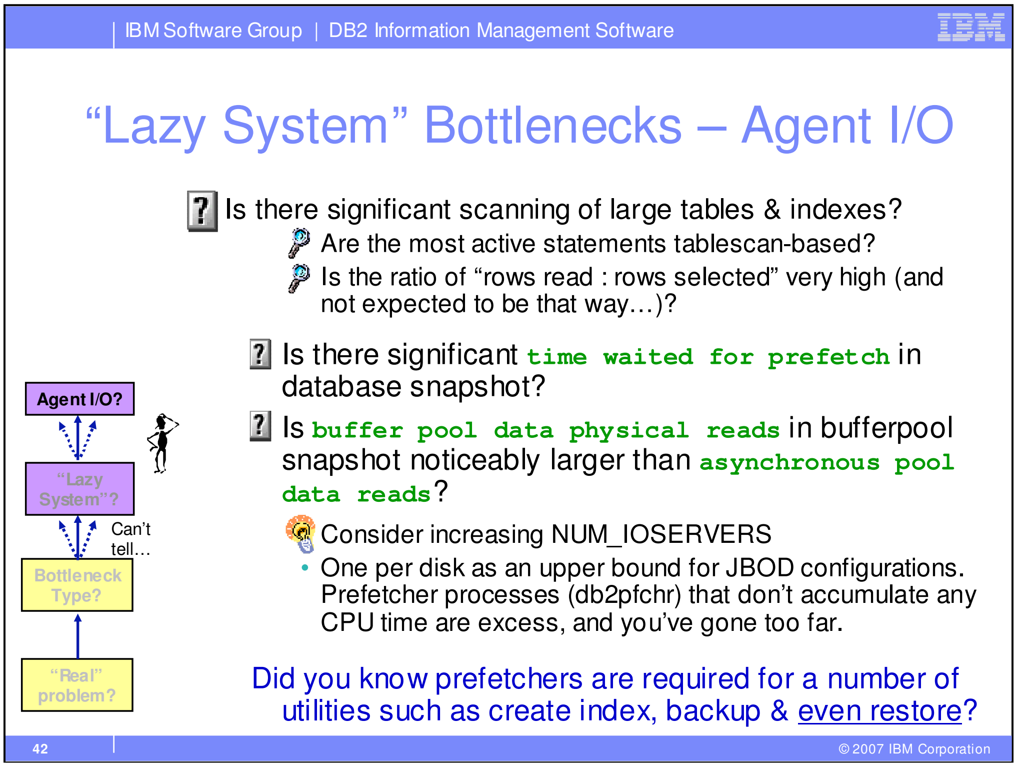
Are you getting dirty page steal triggers? more than a trivial number of synchronous writes?

Cleaners keep steady supply of clean pages for agents - else agents have to stop to write out modified data. Cleaners with async I/O can write out much more efficiently than one page at a time – may transfer bottleneck to disk.

Possibly increase number of cleaners – up to 1 per CPU (increasing number of cleaners past what you need has low impact - you can back them off when you see there are ones not getting any CPU time.)

Possibly decrease SOFTMAX or CHNGPGS_THRESH if they are particularly high, in order to keep cleaners working steadily.

(Interesting tidbit - cleaners can also be needed in a "query only" workload to flush large temps to disk efficiently)



"Lazy System" Bottlenecks – Agent I/O

- Is there significant scanning of large tables & indexes?
  - Are the most active statements tablescan-based?
  - Is the ratio of "rows read : rows selected" very high (and not expected to be that way…)?
- Is there significant `time waited for prefetch` in database snapshot?
- Is `buffer pool data physical reads` in bufferpool snapshot noticeably larger than `asynchronous pool data reads`?
  - Consider increasing NUM_IOSERVERS
  - One per disk as an upper bound for JBOD configurations. Prefetcher processes (db2pfchr) that don't accumulate any CPU time are excess, and you've gone too far.

Did you know prefetchers are required for a number of utilities such as create index, backup & even restore?

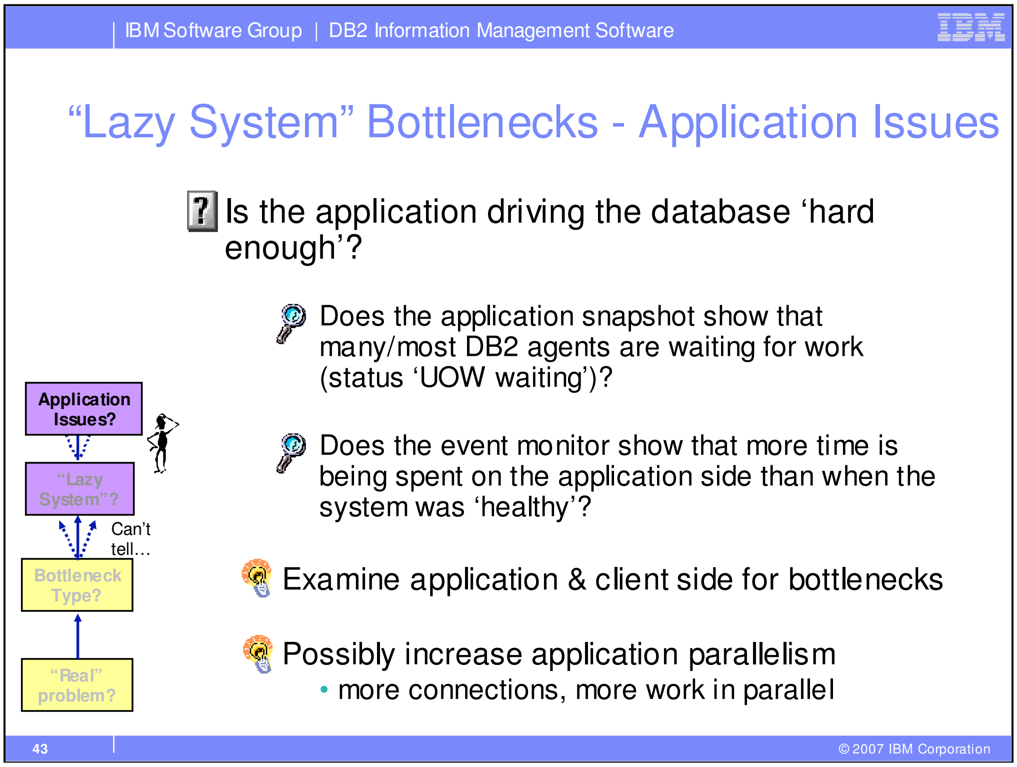
Case 2 - prefetchers

Table & index scans & utilities running w/o sufficient prefetchers configured can do smaller-scale I/Os via the agent, or wait on prefetchers to become free

Are there tablescans in the workload, and are a substantial fraction of I/Os synchronous? Time spent waiting on prefetch? By-statement synchronous I/O in scans?

If this is the case, and there aren't already a reasonable supply of prefetchers ("One prefetcher per disk" as an upper bound, but not to ridiculous numbers. One or a few per RAID stripe depending on stripe size) then consider increasing NUM_IOSERVERS.

Need these in odd places (restore for example), good to have a few around.



IBM Software Group | DB2 Information Management Software

"Lazy System" Bottlenecks - Application Issues

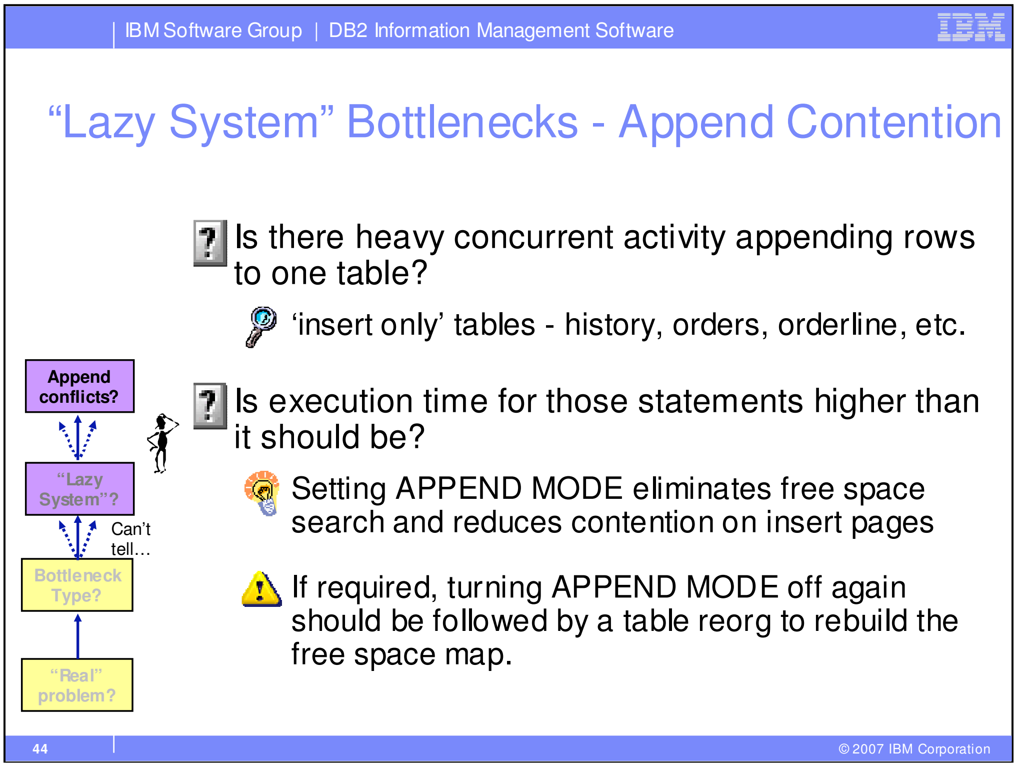
- Is the application driving the database 'hard enough'?
  - Does the application snapshot show that many/most DB2 agents are waiting for work (status 'UOW waiting')?
  - Does the event monitor show that more time is being spent on the application side than when the system was 'healthy'?
- Examine application & client side for bottlenecks
- Possibly increase application parallelism
  - more connections, more work in parallel

43

© 2007 IBM Corporation

It's not always DB2's fault. ☺

Sometimes a 'lazy system' can arise from the application just not driving the database hard enough. We can tell this from application stat in application snapshot (waiting for work from the application = UOW waiting). We can also do some cool event monitor queries (see section 2) to see how much time is being spent 'between statements'.



“Lazy System” Bottlenecks - Append Contention

Append conflicts?

“Lazy System”?

Can’t tell…

Bottleneck Type?

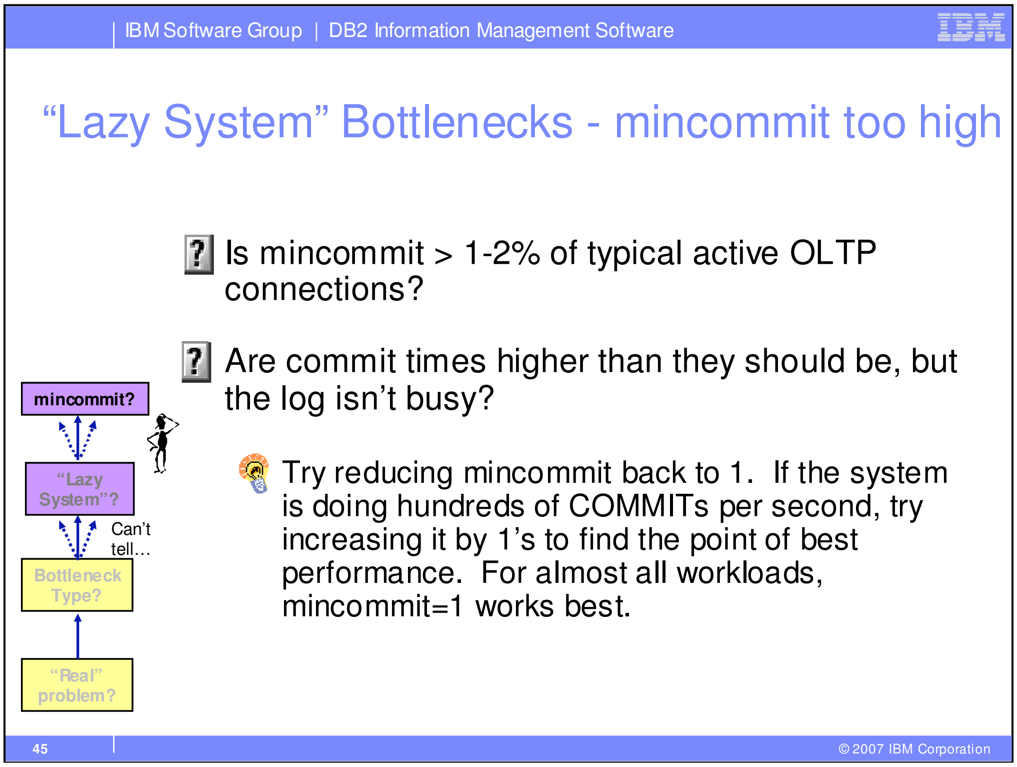
“Real” problem?

- ? Is there heavy concurrent activity appending rows to one table?
  - ‘insert only’ tables - history, orders, orderline, etc.
- ? Is execution time for those statements higher than it should be?
  - Setting APPEND MODE eliminates free space search and reduces contention on insert pages
  - ⚠ If required, turning APPEND MODE off again should be followed by a table reorg to rebuild the free space map.

On some types of workloads which generate a substantial amount of insert activity, searching for free space, and contention for allocation of new space, in the table can cause a bottleneck to develop.

For ‘insert only’ tables – logs, history tables, etc. – setting APPEND MODE avoids the free space search algorithm, and can help lower insert times.

Note that when APPEND MODE is turned off, to enable DB2 to reuse any free space in the table (arising from deletes, etc.), a REORG needs to be done to rebuild the free space map.



"Lazy System" Bottlenecks - mincommit too high

- Is mincommit > 1-2% of typical active OLTP connections?
- Are commit times higher than they should be, but the log isn't busy?
  - Try reducing mincommit back to 1. If the system is doing hundreds of COMMITs per second, try increasing it by 1's to find the point of best performance. For almost all workloads, mincommit=1 works best.

Mincommit can be very helpful in some situations, but it will reduce performance if used in the wrong situations. It causes COMMITs to be 'batched' together – so that when one application commits, it blocks until N-1 other applications also commit, and then the flush to the log happens together, and they are all unblocked. If there are less than N applcations committing within 1s, they will be allowed to go ahead without additional wait time – but 1s is a long time in an OLTP system! It is really only suited to high-volume OLTP with a high number of connections. Note that even if MINCOMMIT helps during periods of high load, it may start to cause problems when the number of active connections drops.

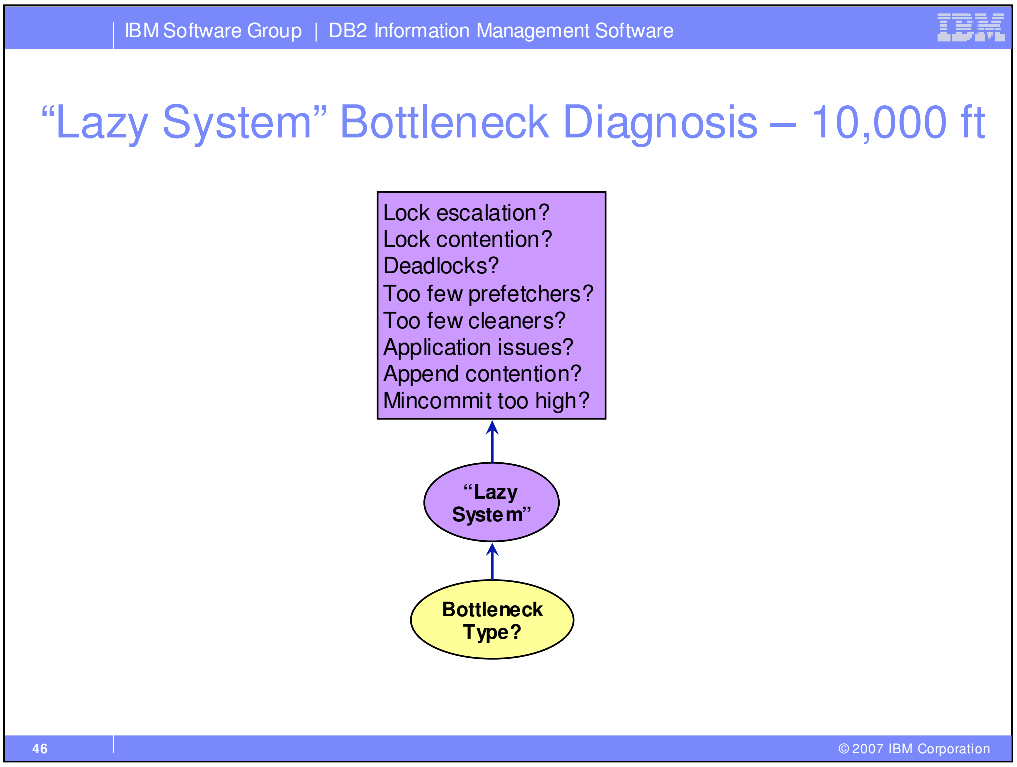
If mincommit is set > 1, it is often useful to set it back down to 1, and increase it very slowly, observing the performance of the system as you go.

symptoms of being too high?

- neither CPU nor disk over utilized
- "long" commit response times, as high as 1s (statement event monitor)
  - out of proportion to work done in transaction & to other statements
- can cause odd 'inverted' behavior at very low numbers of users
  - ok at < MINCOMMIT users
  - drop in throughput in [MINCOMMIT .. 10xMINCOMMIT] users

guidelines

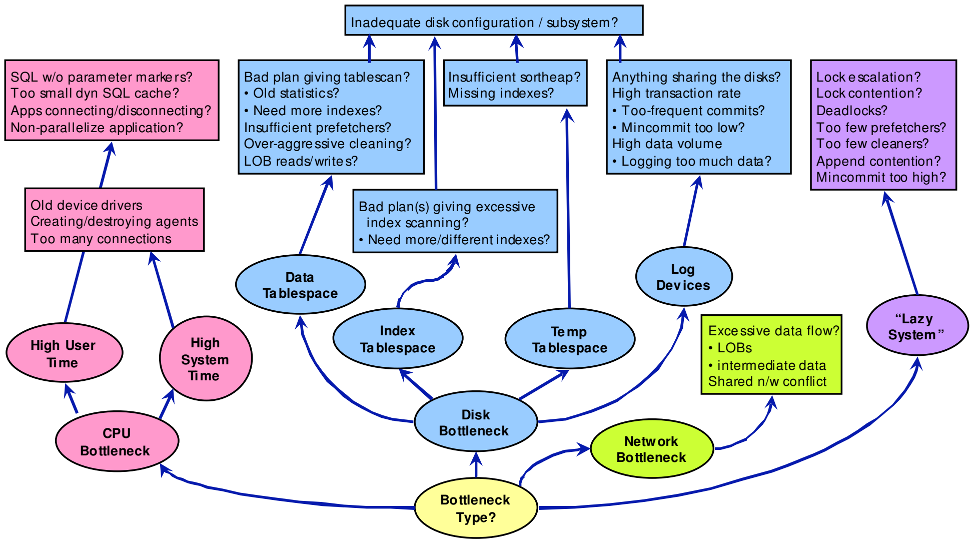
- 1 is usually OK; at 50 or more high-throughput connections, try increasing to 2
- build slowly after that if necessary
- should be done online to monitor behavior



“Lazy System” Bottleneck Diagnosis – 10,000 ft

Lock escalation?
Lock contention?
Deadlocks?
Too few prefetchers?
Too few cleaners?
Application issues?
Append contention?
Mincommit too high?

And the Whole Thing …



Inadequate disk configuration / subsystem?

SQL w/o parameter markers?
Too small dyn SQL cache?
Apps connecting/disconnecting?
Non-parallelize application?

Bad plan giving tablescan?
- Old statistics?
- Need more indexes?

Insufficient prefetchers?
Over-aggressive cleaning?
LOB reads/writes?

Insufficient sortheap?
Missing indexes?

Anything sharing the disks?
High transaction rate
- Too-frequent commits?
- Mincommit too low?

High data volume
- Logging too much data?

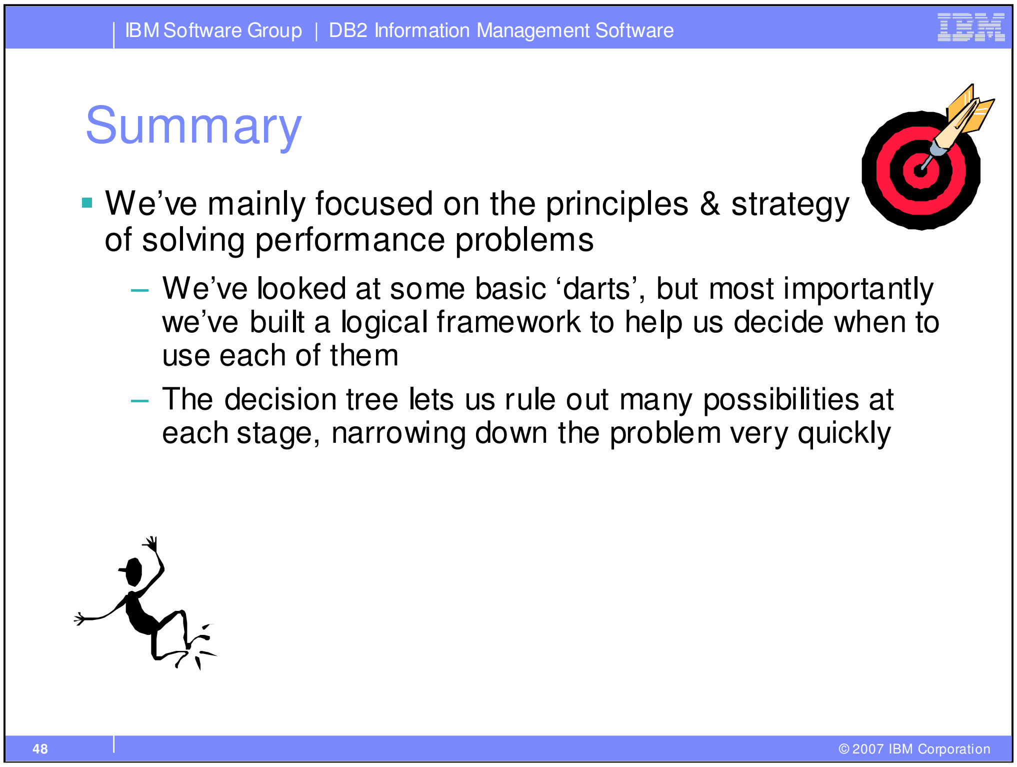
Lock escalation?
Lock contention?
Deadlocks?
Too few prefetchers?
Too few cleaners?
Append contention?
Mincommit too high?

Old device drivers
Creating/destroying agents
Too many connections

Bad plan(s) giving excessive index scanning?
- Need more/different indexes?

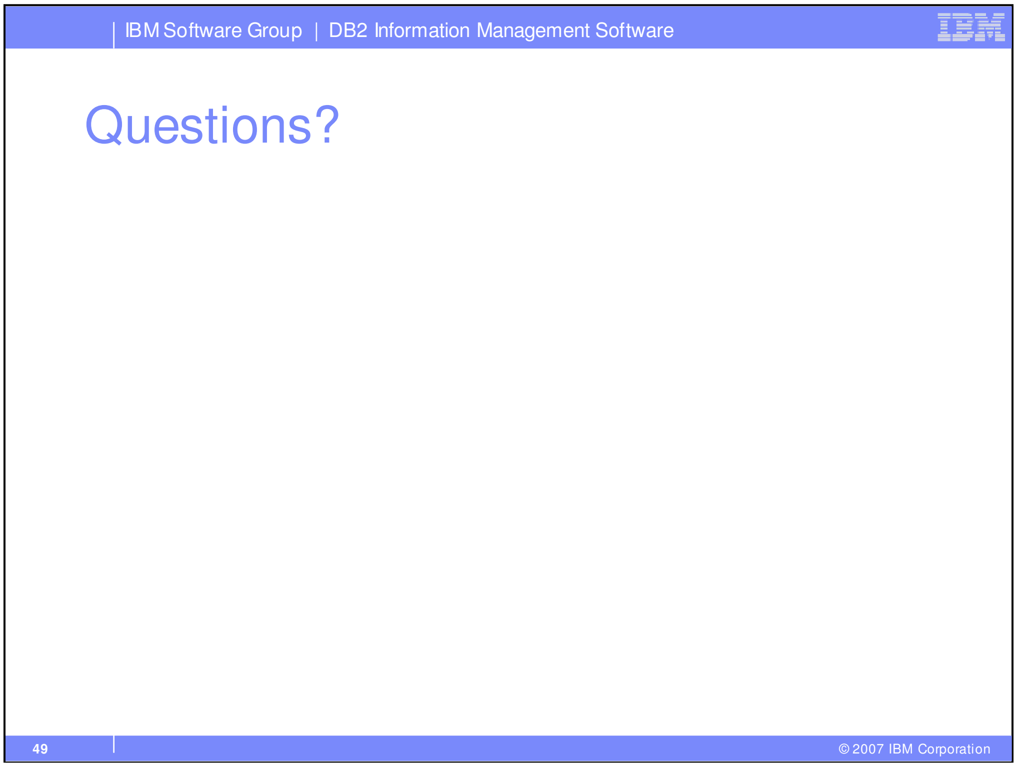
Excessive data flow?
- LOBs
- intermediate data

Shared n/w conflict

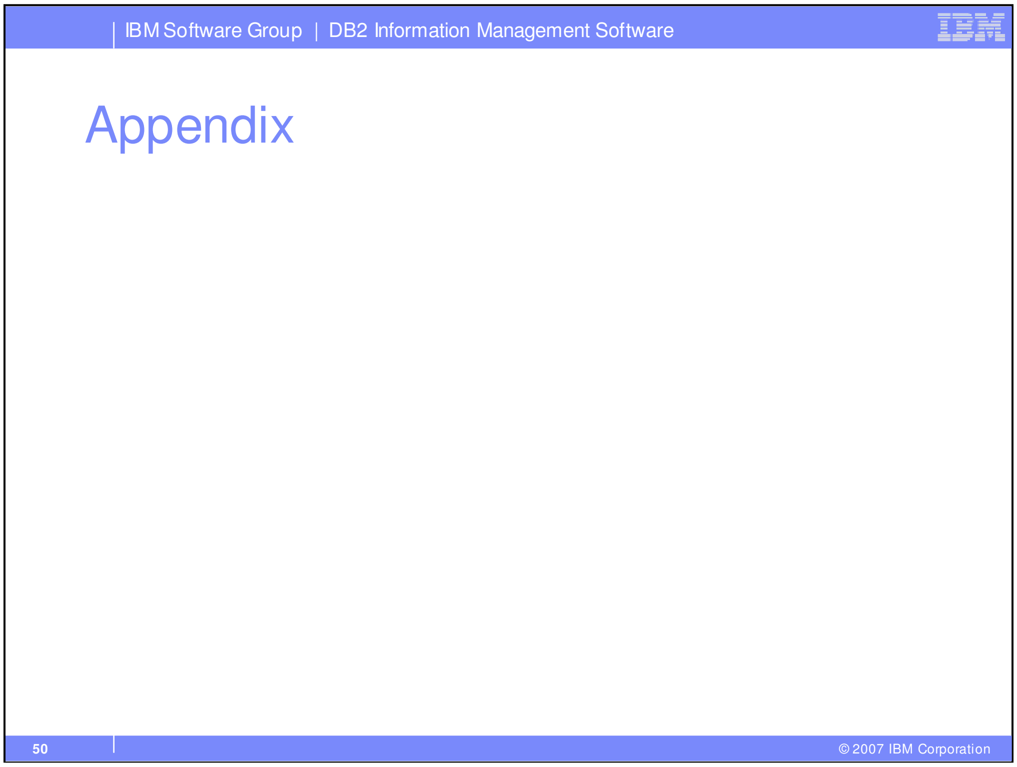


Summary

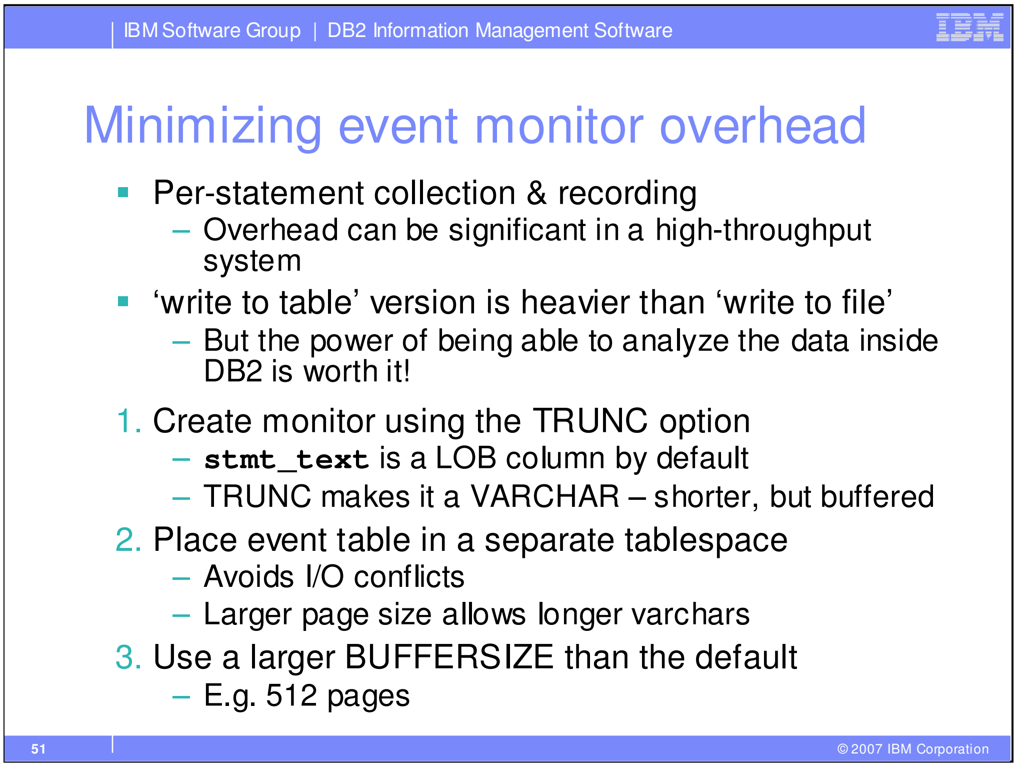
- We've mainly focused on the principles & strategy of solving performance problems
  - We've looked at some basic 'darts', but most importantly we've built a logical framework to help us decide when to use each of them
  - The decision tree lets us rule out many possibilities at each stage, narrowing down the problem very quickly



Questions?



Appendix



IBM Software Group | DB2 Information Management Software

Minimizing event monitor overhead

- Per-statement collection & recording
  - Overhead can be significant in a high-throughput system
- ‘write to table’ version is heavier than ‘write to file’
  - But the power of being able to analyze the data inside DB2 is worth it!

1. Create monitor using the TRUNC option
   - **`stmt_text`** is a LOB column by default
   - TRUNC makes it a VARCHAR – shorter, but buffered
2. Place event table in a separate tablespace
   - Avoids I/O conflicts
   - Larger page size allows longer varchars
3. Use a larger BUFFERSIZE than the default
   - E.g. 512 pages

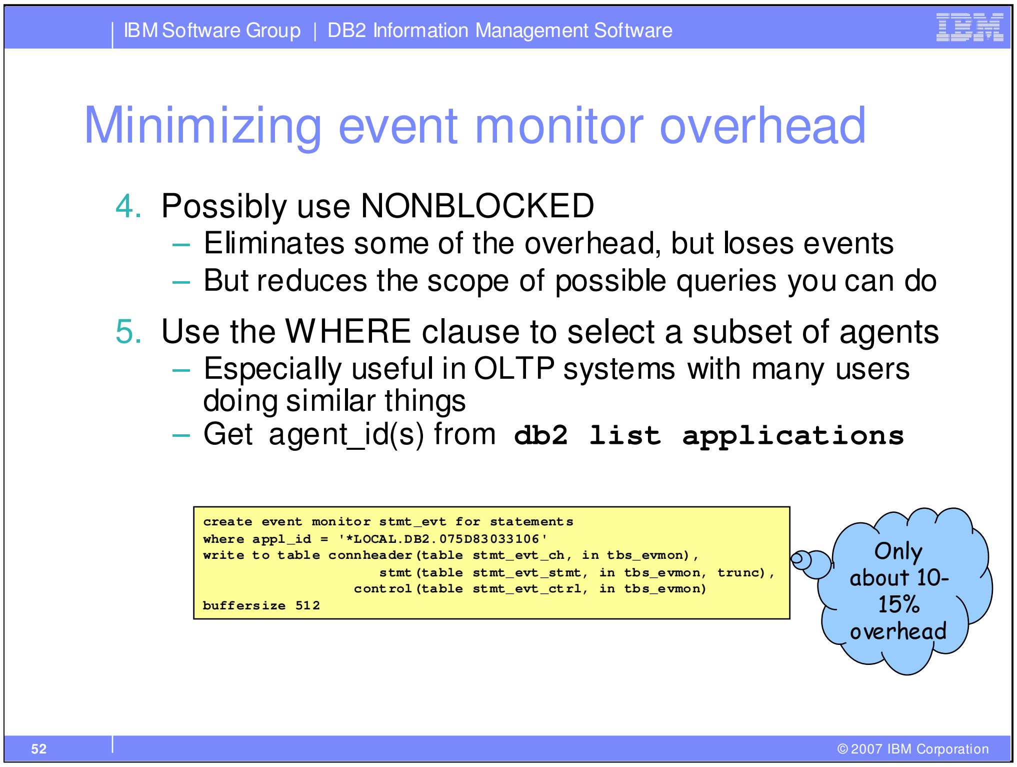
51

Statement event monitors data is powerful, but collecting the data can be very expensive. The cost is directly proportional to the rate of execution of statements, so in a BI application, it probably doesn’t matter, but in OLTP, it does. Here are some steps to minimize statement event monitor overhead.

Recording the statement text is expensive, and because statements can be very long, the default mode is to use a LOB column. This is expensive, and possibly unnecessary if the statements in your system are of “normal” size. The TRUNC option on CREATE EVENT MONITOR tells DB2 to use a VARCHAR column for the statement text. Much smaller maximum capacity than a LOB, but much faster since the data can be buffered in the bufferpool.

Unless otherwise specified, the event monitor table goes into the default tablespace. This may cause conflicts with your regular data, potentially at the container/disk level. By specifying tablespaces in CREATE EVENT MONITOR, you can (1) avoid conflicts with other data, and (2) specify a larger page size, so that up to a 32k long VARCHAR could be used to hold statement text (reducing the possibility of the TRUNC option actually causing truncation.

The default buffer size is very small (something like two 4k pages?), so specifying even a modest buffer size will cost very little to the system, and help a great deal.



IBM Software Group | DB2 Information Management Software

Minimizing event monitor overhead

4. Possibly use NONBLOCKED
   - Eliminates some of the overhead, but loses events
   - But reduces the scope of possible queries you can do
5. Use the WHERE clause to select a subset of agents
   - Especially useful in OLTP systems with many users doing similar things
   - Get agent_id(s) from **`db2 list applications`**

```
create event monitor stmt_evt for statements
where appl_id = '*LOCAL.DB2.075D83033106'
write to table connheader(table stmt_evt_ch, in tbs_evmon),
                     stmt(table stmt_evt_stmt, in tbs_evmon, trunc),
                  control(table stmt_evt_ctrl, in tbs_evmon)
buffersize 512
```

Only about 10-15% overhead

52

The NONBLOCKED option tells DB2 that if the buffer is full, not to block, but to drop the event record on the floor & continue. This does make it run faster in a busy system, but since you've dropped some records, it limits what operations you can do on the trace. For example, you can't do reliable math on timestamp deltas like we did in Exmaple 4. In general I don't recommend NONBLOCKED.

The WHERE clause is helpful. It would be great to be able to filter efficiently on a number of different qualifiers – application name, operation, type, etc., but WHERE isn't that powerful (yet). The overhead of event monitors is greatest in OLTP systems, and the WHERE clause can be used to focus your data collection on a single connection, instead of picking up all connections.

If you roll all these techniques together, you can keep the overhead down to around 10-20%.