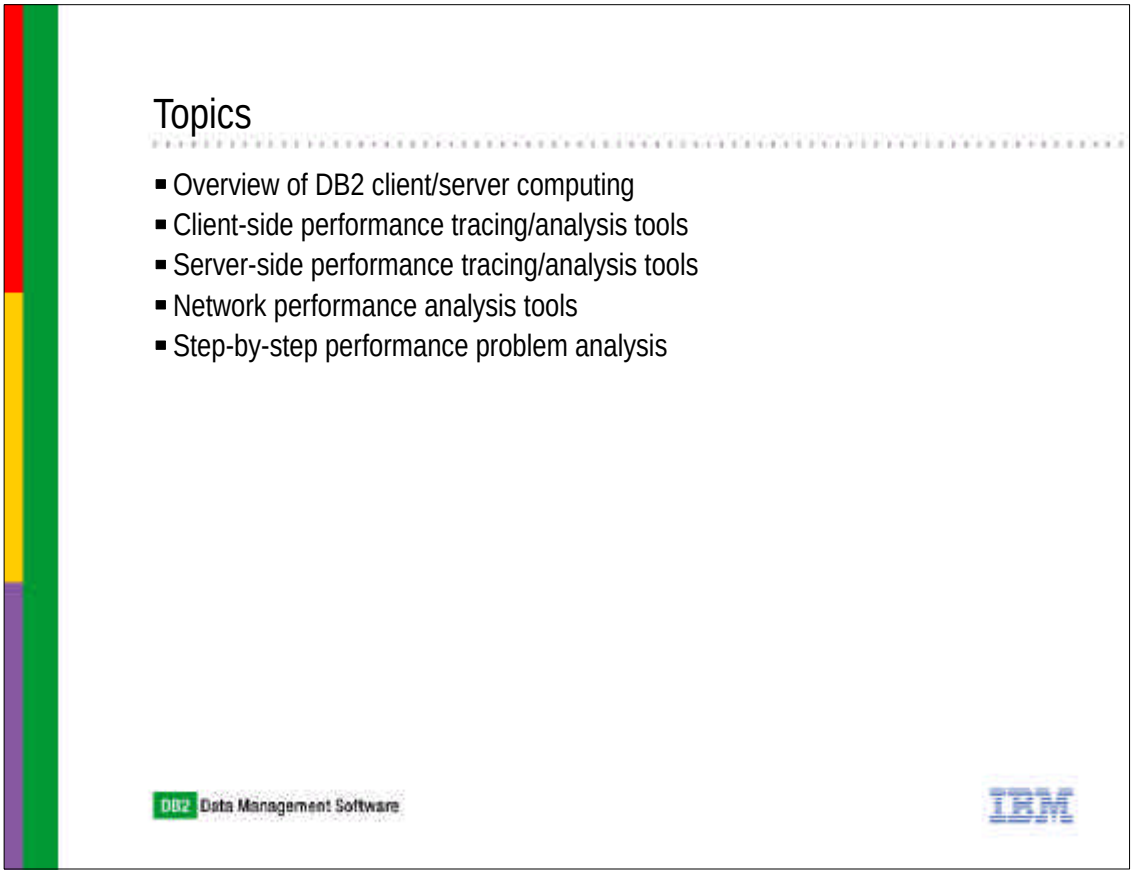
Abstract

Trying to solve any database performance problem can sometimes be a challenge. Now add a network, and that challenge becomes more daunting. The presenter will cover the tools and techniques he uses to determine where the bulk of time occurred so that a person's time is more wisely spent diagnosing the largest time consumer. As a prelude, he will also briefly discuss what should happen in a DB2 UDB for z/OS and OS/390 client/server environment.

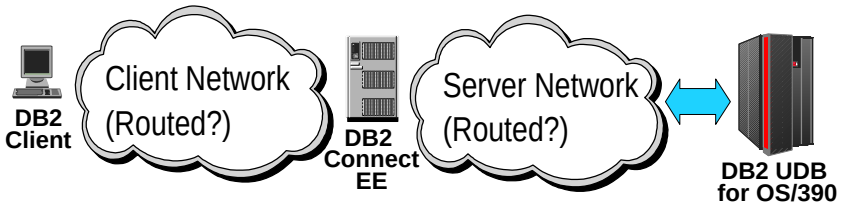


Initially, a brief overview of client/server computing will be discussed so that one can get an understanding of what to expect in this environment. Then, intermixed with each other, a discussion of the various tools that can be used to understand the performance of a client/server application environment from a client, server, and network point-of-view. Also, a discussion of where the various tools perform their function will be given. Finally, a discussion of the step-by-step approach to analyzing these kind of performance problems will be given and if time permits a run-through of an sample problem will be discussed.

Is There a Problem?

Response time is slow!!!!

What happened to the throughput?!?!



What Do I Do?

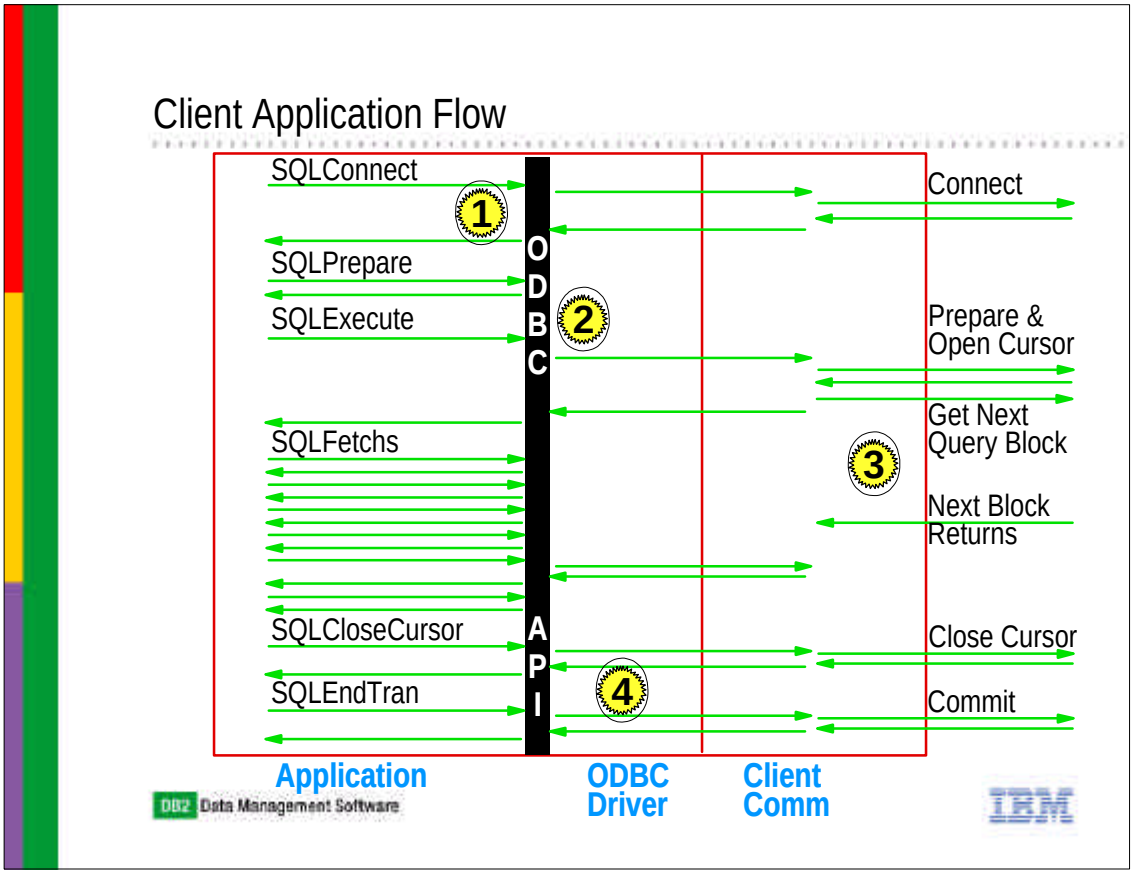
Where Do I Start?

DB2 Data Management Software

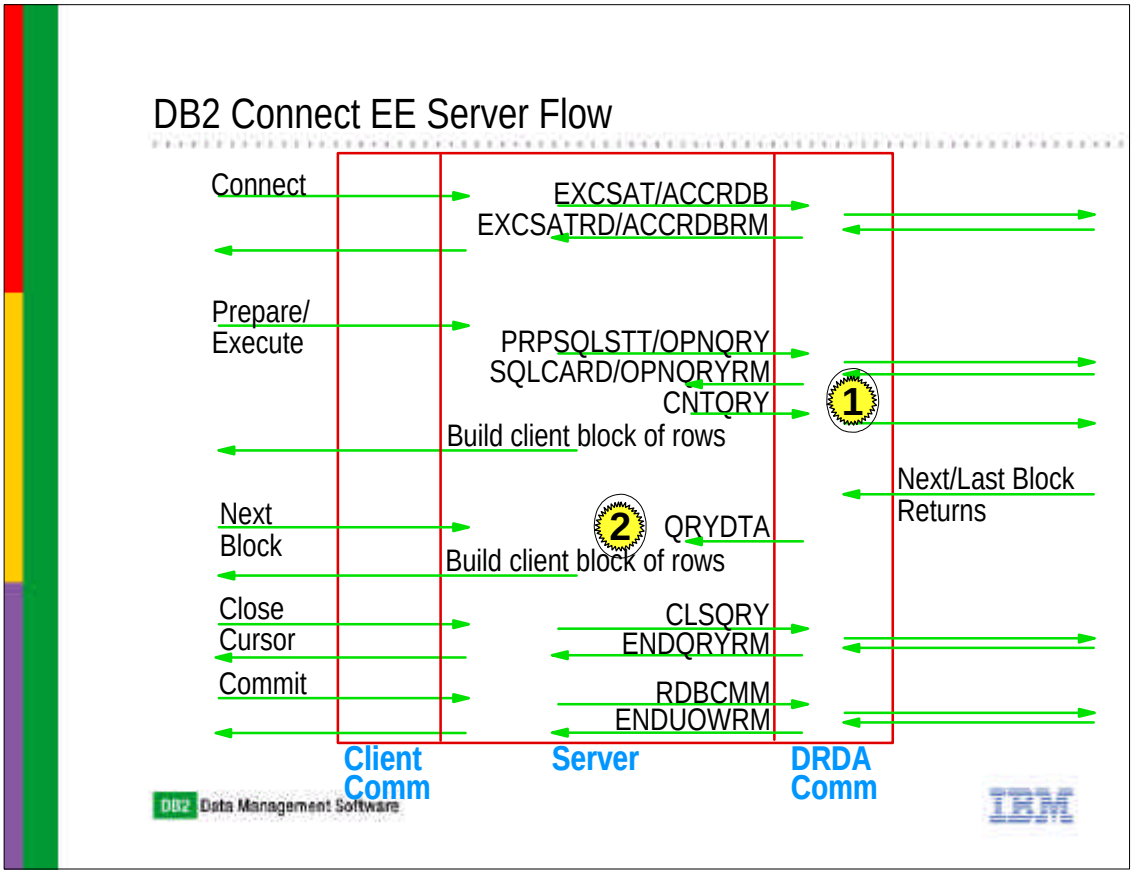
IBM

One reason why I am giving this presentation to, hopefully, an audience of DBAs is that normally when users complain about a peformance problem they come to the DB2 person. Also, usually the other people who should be involved, systems programmers, network specialists, application developers, etc., are so specialized and always find that their part of the environment is perfect. I feel that DBAs end up taking the lead on solving these kinds of problems. So, I am hoping to provide you all with some insight in tools and techniques on handle to get a handle on trying to solve distributed performance problems.

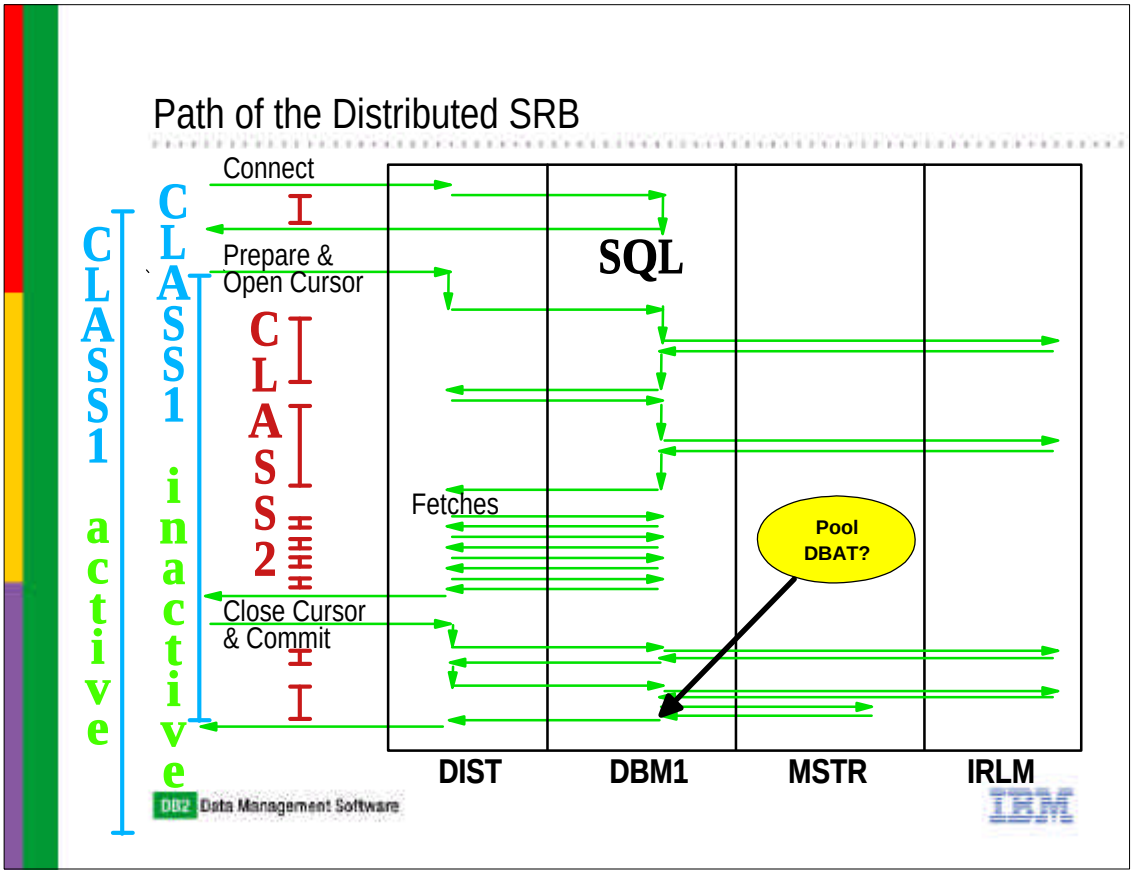
Please do not hesitate to email me about anything in this presentation for further clarification.



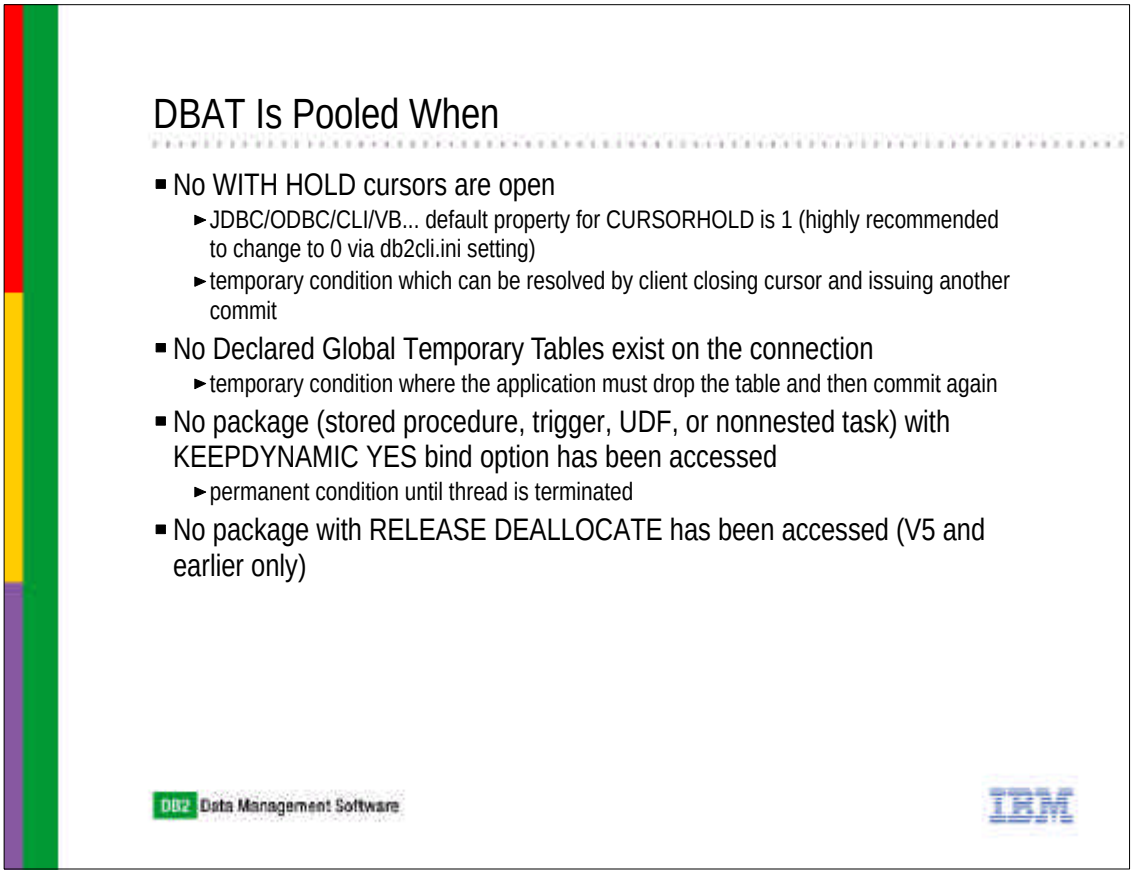
- The most common kind of application client is one that utilizes the ODBC/CLI/JDBC driver of the DB2 UDB for Linux/Unix/Windows product of which DB2 Connect provides as well. This page depicts what would happen if the DB2 client is in fact a DB2 Connect PE client (only available on Windows).
- There are a lot of ODBC/CLI/JDBC application calls which do not directly map to network exchanges with a DB2 server. However, at point 1, a connect API does require an exchange with the server. For the most part, a prepare of an SQL statement does not necessarily cause a network exchange but rather the API request to execute it does. Thus, at point 2, if the application prepares and then executes a dynamic statement, a chained request is sent up to the DB2 server. However, if the application requests any descriptive information about the columns of the query, input parameter markers, etc., then the prepare request chained with a possible describe request will be sent ahead of the request to execute the statement such as an INSERT, UPDATE, DELETE, and cursor SELECT.
- Point 3 is where a regular DB2 client differs from a DB2 Connect PE client and will be explained more on the next chart.
- Finally, there are default settings in the ODBC/CLI/JDBC driver which can cause extra network messages. One of them is the default for CURSORHOLD which is 1 or always on. With this setting, the DB2 UDB for z/OS and OS/390 server cannot early close a WITH HOLD cursor even when the server has fetched to the end of the cursor. This forces a network exchange on the Close Cursor request. Now, with the other default setting for AUTOCOMMIT which is 1 or on, generally the Close Cursor request could be chained with a COMMIT request. However, if the application has turned off AUTOCOMMIT so that a complete unit-of-work which spans multiple SQL can be processed at the server, then the COMMIT request will be a separate flow.



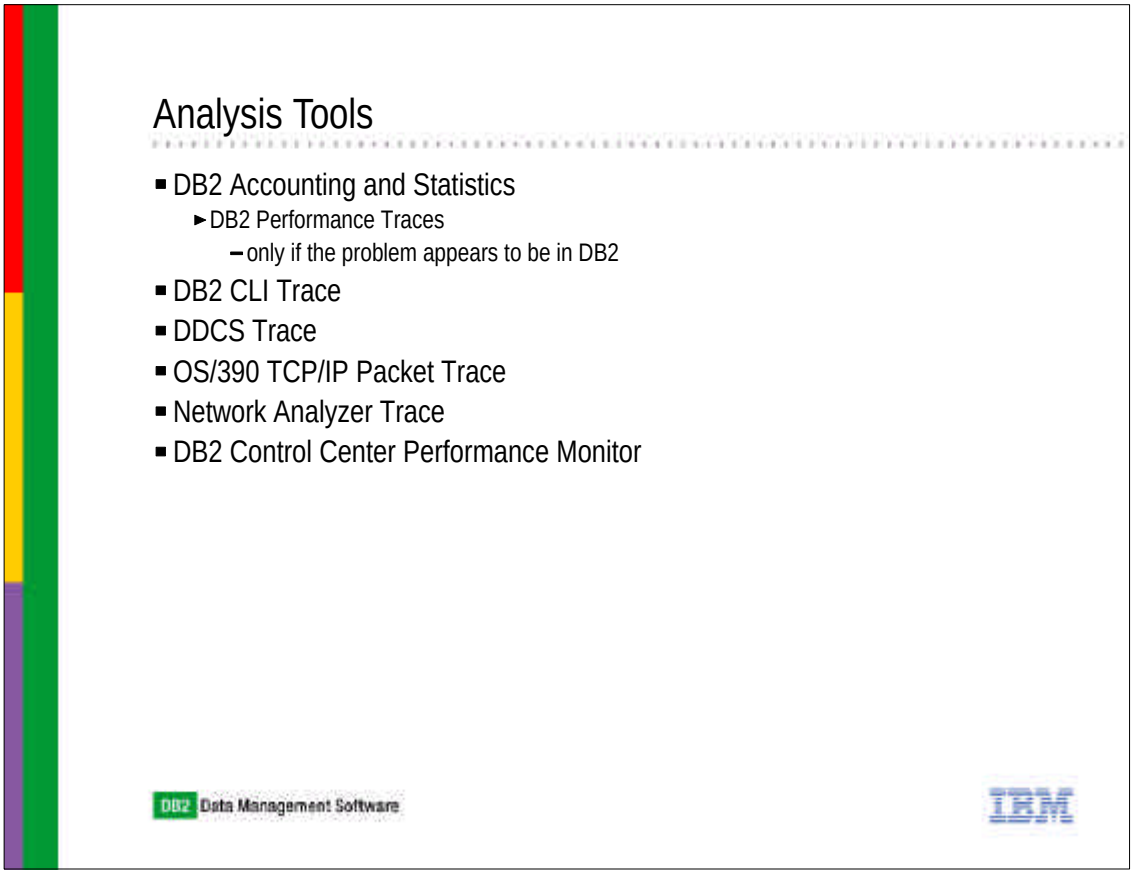
- On the DB2 Connect EE server, we have client messages come in on one side as requests native to the DB2 UDB for Linux/Unix/Windows (LUW) client/server protocol. On the other side, one has Distributed Relational Database Architecture requests going up to the DB2 UDB for z/OS and OS/390 server. Thus, all DB2 Connect EE server requires on its processor platform is CPU (the more the merrier), memory, and great LAN connections. Also, it is on the DB2 Connect EE platform where data transformations such as character codepage conversion and numeric datatype conversions take place. Of anything that requires close attention, it is the character codepage conversions which, if a lot of character data is being down sent from the host, might cause the most CPU consumption.
- However, based on quite a few studies, once one knows the relative cost in CPU consumed to process a transaction and the memory required for a client connection while working on a transaction, then scaling up the resource requirements of CPU and memory based on transaction rate and number of concurrent connections that are processing transactions is very linear.
- Point 1 depicts processing that is unique to DB2 Connect code. When an open query reply is processed, DB2 Connect checks to see if the cursor block that it has just received from the DB2 server is the last block of the query. If it is not, then DB2 Connect asynchronously requests the server to continue the query (i.e. send the next block). While this is happening, DB2 Connect then builds the client a block of rows and sends it. While the client is processing the rows, the server may have already come back with the next block. However, that block is not read in by the DB2 Connect EE server until a request comes back from the client for its next block (point 2).



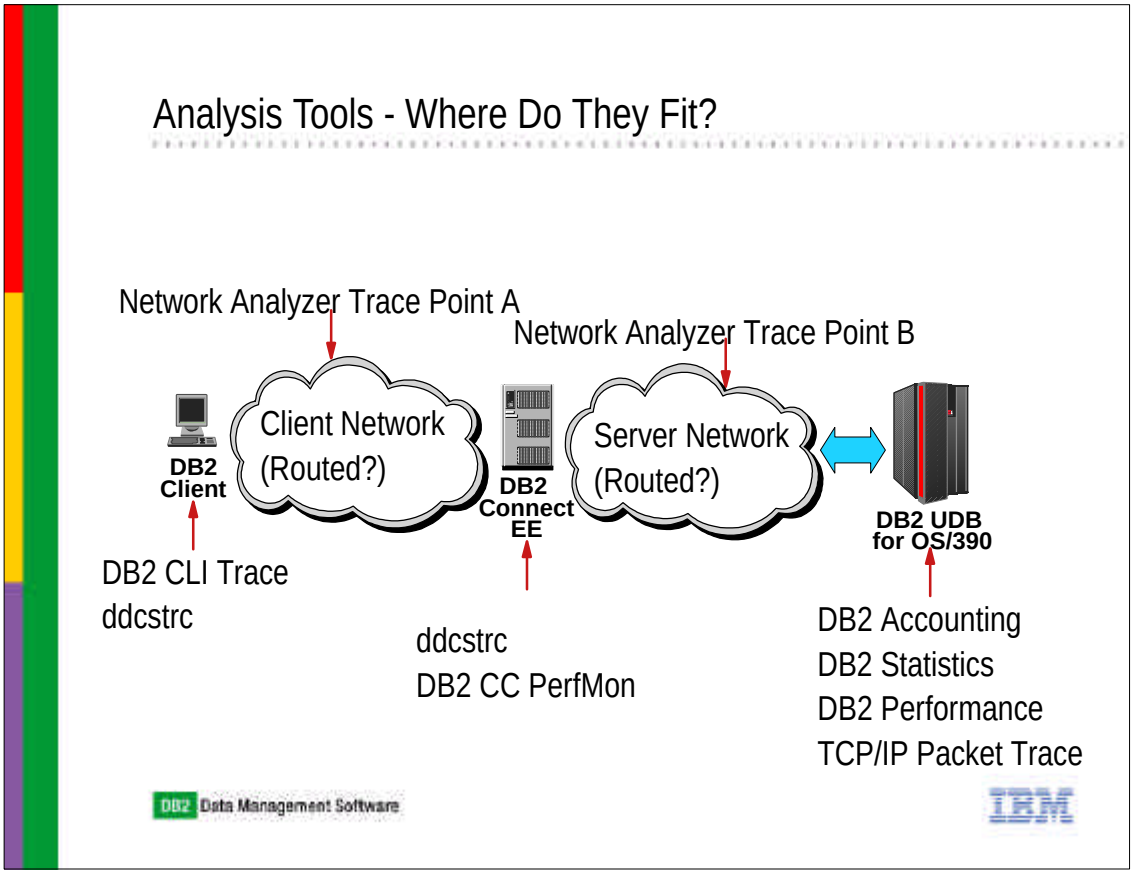
- With DB2 for z/OS and OS/390, there are two modes of running distributed threads: active, in which every connection is a DataBase Access Thread (DBAT) even when waiting for new client transactions up until it is disconnected (maximum is 1999 if MAXDBAT was set to that value), and inactive, in which DBATs are pooled and handed out to connections as needed. The ZPARM setting of CMTSTAT (ACTIVE or INACTIVE) controls this. Within INACTIVE, one can have up to 150000 connections to anyone DB2 subsystem (CONDBAT).
- The processing of a connection request into DB2 starts out with a Service Request Block (SRB) running at the priority or goal of the distributed region. DB2 decodes the request and, if running inactive, attempts to find a "pooled" DBAT on which to process the request. If a DBAT does not exist, and if MAXDBAT has not been reached, DB2 creates a DBAT. Otherwise, if running ACTIVE, DB2 must create a DBAT. The processing to create a DBAT requires a lot of CPU cycles and it is somewhat serialized because DB2 must keep track of its DBATs. Once both modes have a DBAT on which to process the connection request, DB2 performs most of this work under a normal SRB as well as also does some Task Control Block (TCB) processing for performing security verification.
- For Active processing, once the connection has been successfully processed and before the reply issued back to the requester, DB2 issues a WorkLoad Manager (WLM) call to create an enclave. This enclave will be around until the thread deallocates and its service class goal will be set at this time. The service class will never change, thus ACTIVE-always DBATs can have only a service class with a single-period velocity goal. At this time the class 1 timers are started.
- For INACTIVE processing, the connection request response is sent to the client. When the next message comes into DB2, DB2 then issues a create of an enclave with the information to classify it. An inactive-style DBAT can have a service class with periods and response time goals in addition to velocity goals.
- For an ACTIVE thread, the class 1 elapsed timer is on from thread creation to thread termination. At thread termination, an accounting record is cut.



- The preferred way to run distributed processing into DB2 is with CMTSTAT=INACTIVE. Besides the term inactive thread processing, this mode is also called thread pooling. Starting in V6, DBATs are pooled amongst a larger number of client connections. Normally, connections go inactive when a commit on the thread is processed. Also, at this time, an accounting record is cut. However, there are conditions which can prevent this from happening.
- If there are any open WITH HOLD cursors, the thread will not go inactive. However, this is a termporary condition which can be relieved via the client closing the cursor and then issuing another commit. By the way, the default cursor property for any JDBC/ODBC/CLI/VB... client is all cursors are requested WITH HOLD. This can be changed by setting CURSORHOLD=0 in the db2cli.ini file.
- If any declared global temporary tables (DGTT) exist on the connection or "session", then the DBAT will not go inactive. Again, this is a temporary condition which can be alleviated by the client dropping the DGTT and issuing another commit.
- If the thread has used any package which was bound with KEEPDYNAMIC YES, then the thread will not go inactive until it is terminated (a permanent condition).
- Finally, for V5, if the thread had used a RELEASE DEALLOCATE package, then it would no longer go inactive. This is not a condition for V6 and beyond.



Now, we are going to discuss the various tools from the server, client, and network perspective and how they can be used together to solve or understand the answer to the question, "where did the time go?"

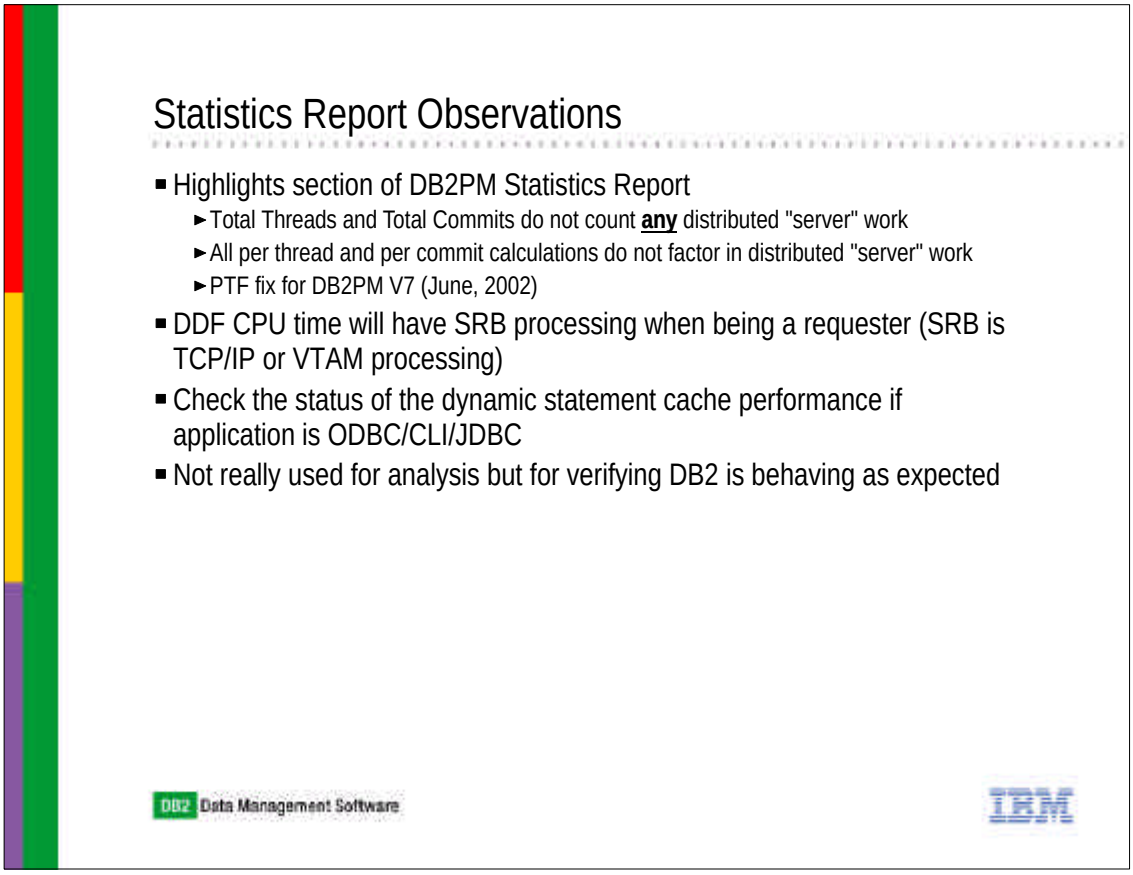


On the DB2 server, the accounting trace is the most useful. Next comes the DB2 statistics trace whose information is only used to quickly determine the overall status of DB2. The DB2 performance trace should only be used after a determination has been made that the problem is in the DB2 server. Finally, the TCP/IP packet trace which runs on the mainframe, can also be used to provide another measurement point to better understand where the time was spent.

At the client, for ODBC/CLI/VB/JDBC applications, the DB2 CLI trace function provides an invaluable look at the performance of the application. If this client is also going directly to the mainframe, then the ddcstrc function can be used, although it is quite verbose and hard to summarize its information.

If the client goes through a DB2 Connect EE server, then the only thing today to understand its performance is ddcstrc. Thus, the only thing worth looking at during performance problem determination is the CPU utilization on the server platform and whether memory is being paged.

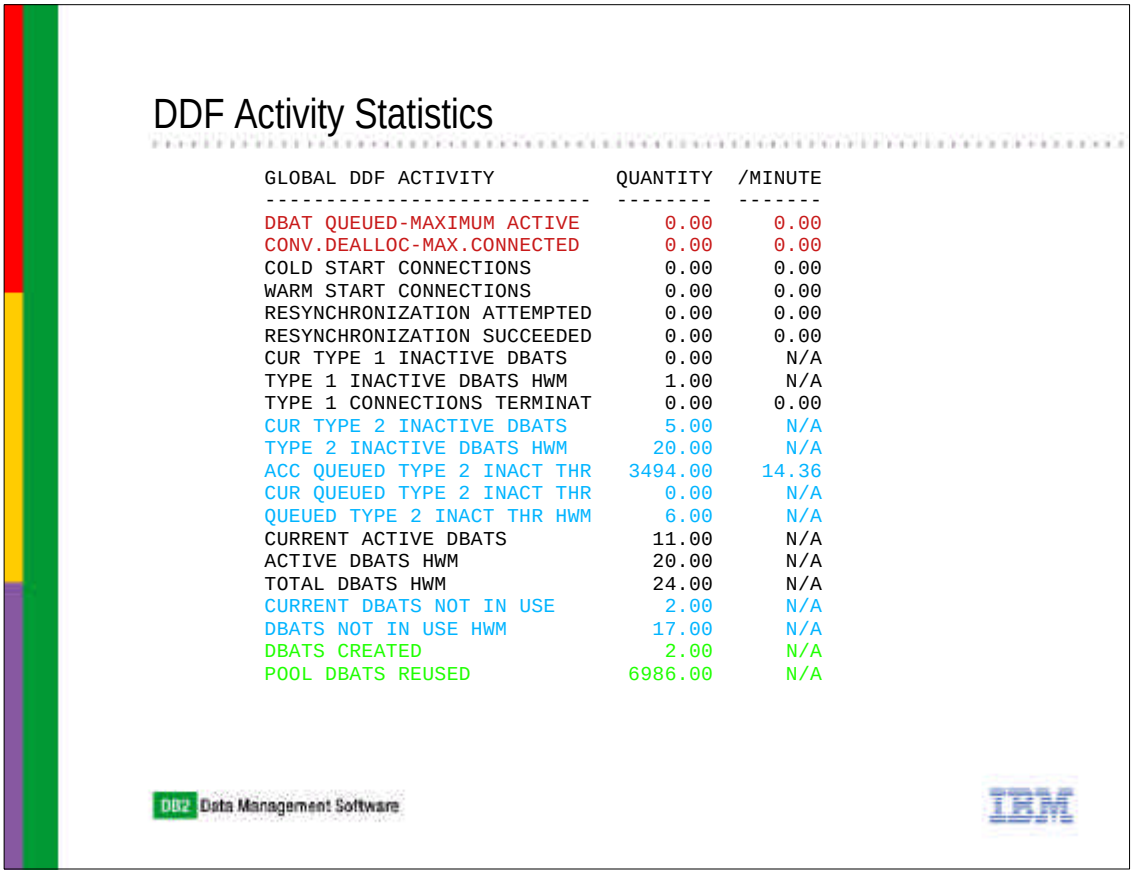
Finally, as a last resort sometimes, network analyzers or sniffers can be placed at various points to monitor an application's performance from a network perspective.



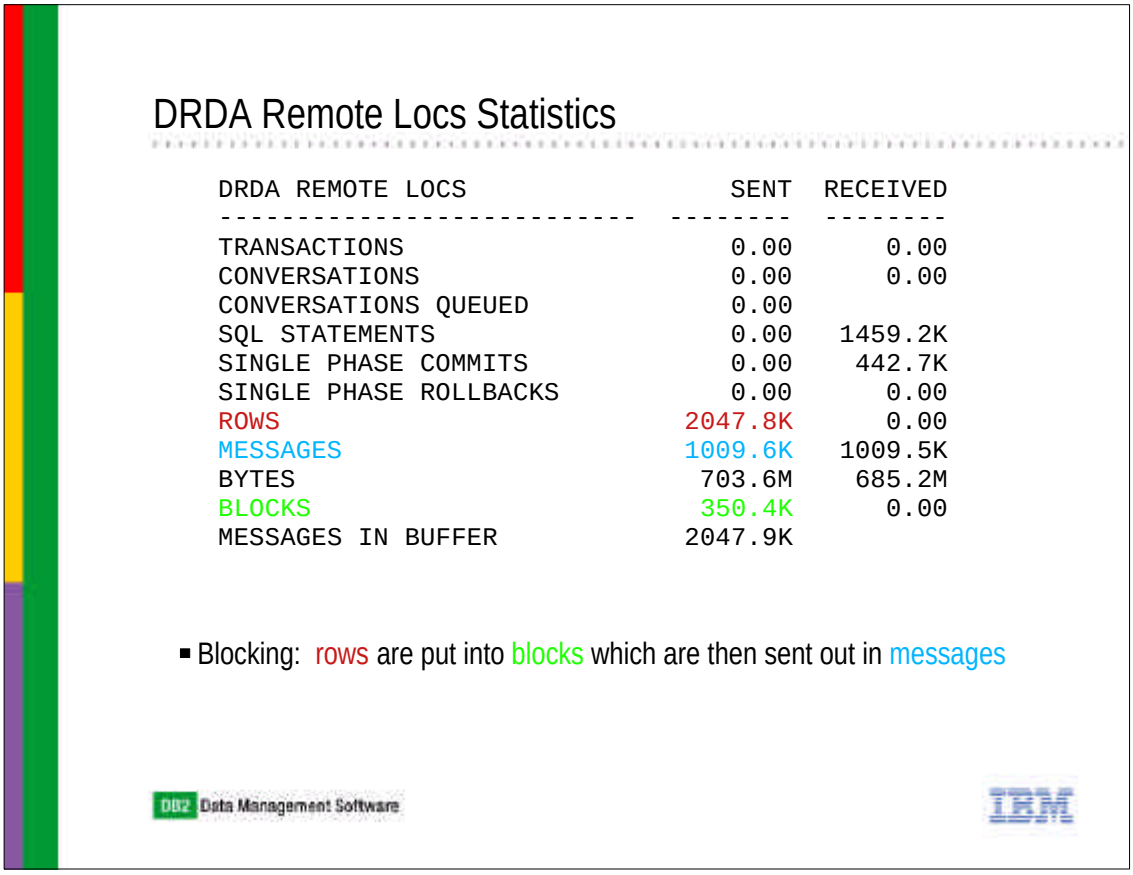
If using DB2PM, the statistics report currently does not factor into its total commits or total threads any distributed commits.

If using DB2 as a requester, normally all the processing gets charged to the allied thread address space. However, there is still some SRB processing which is charged to the distributed region due to the TCP/IP and VTAM message processing costs which cannot be charged to the allied region.

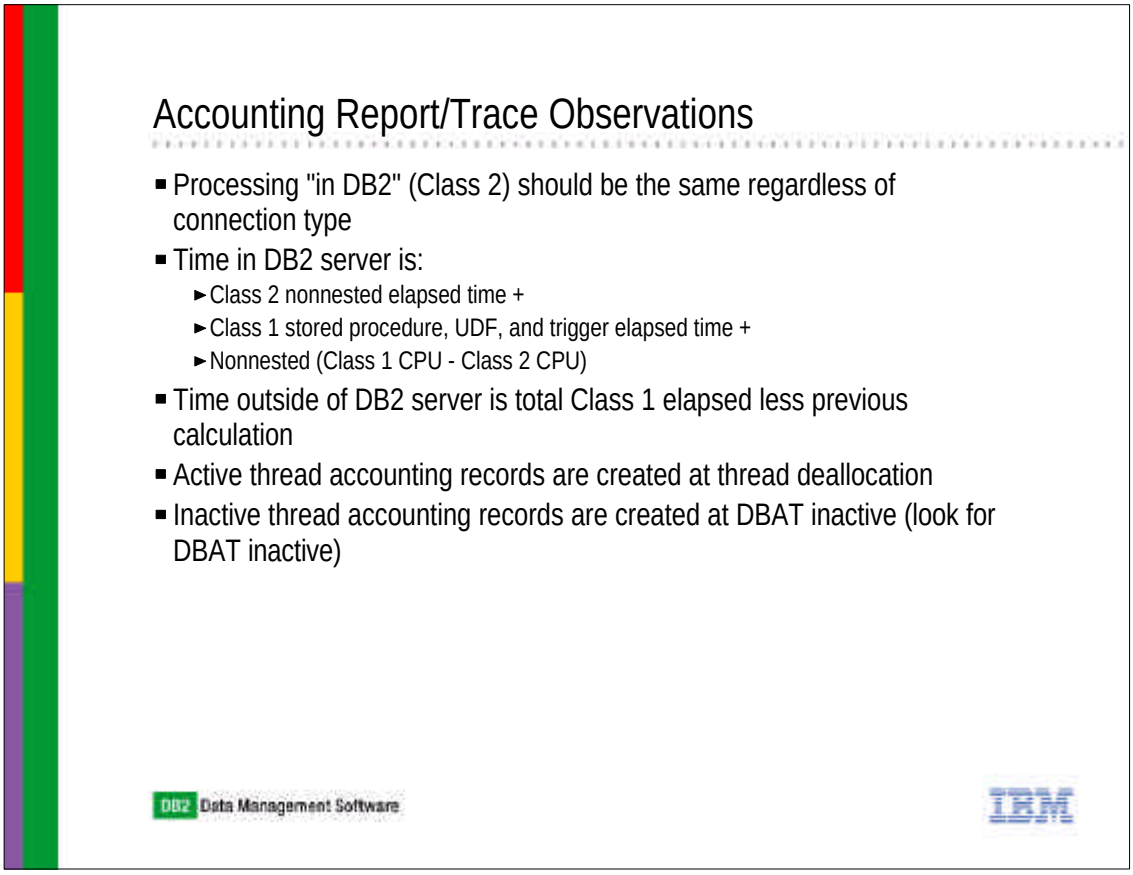
If used ODBC/CLI/JDBC/VB... applications, check to make sure that the dynamic statement cache of DB2 is running at least greater than 80% hit rate.



One part of the statistics report to check on the health of DB2, is the Global DDF Activity section. If one observe's a high rate of "DBAT QUEUED-MAXIMUM ACTIVE", this suggests that the system has a too low a value for MAXDBAT. It should be increased, and then performance remonitored. If one observe's a high rate of "CONV.DEALLOC-MAX.CONNECTED", this suggests that CONDBAT is too low and, everytime it happens, connection requests are refused. The value for CONDBAT should be very high anyway (at least 25000). However, if running CMTSTAT=ACTIVE, then CONDBAT is ignored and MAXDBAT is the only value used to control both concurrent connections as well as concurrent active DBATs. When running with CMTSTAT=INACTIVE, the last two counters give a clue as to how well thread pooling is working. The higher the ratio of "POOL DBATS REUSED" to "DBATS CREATED" the better.



The DRDA Remote Locs statistics report can be used to deduce if DB2 is being asked to block cursors. Thus, attention should be paid to the number of rows sent versus the number of blocks sent. If these values are equal, then every cursor request has resulted in a single row being returned. Not very efficient. The higher the ratio the better but remember this is an average within the statistics reporting interval. Blocks are sent out on messages. Thus the number of messages sent will usually be higher than the number of blocks sent because other SQL processing is performed other than cursor SELECTs.



An accounting trace is by far the most important trace as far as trying to determine where the time went while processing was in DB2. By the way, just because the request is coming from a distributed client does not mean that its processing when in the DB2 DBM1 (database manager) region will be any different than an equivalent request from a local thread.
Time in the DB2 server can be calculated via the shown equation. The addition of the difference between the nonnested task class 1 cpu and class 2 cpu is the amount of cpu consumed in the distributed region only. This equation cannot begin to factor in whether or not the task while in the distributed region waited for processing resources or not. You can use the TCP/IP packet trace to do this.

Using Accounting Trace Information

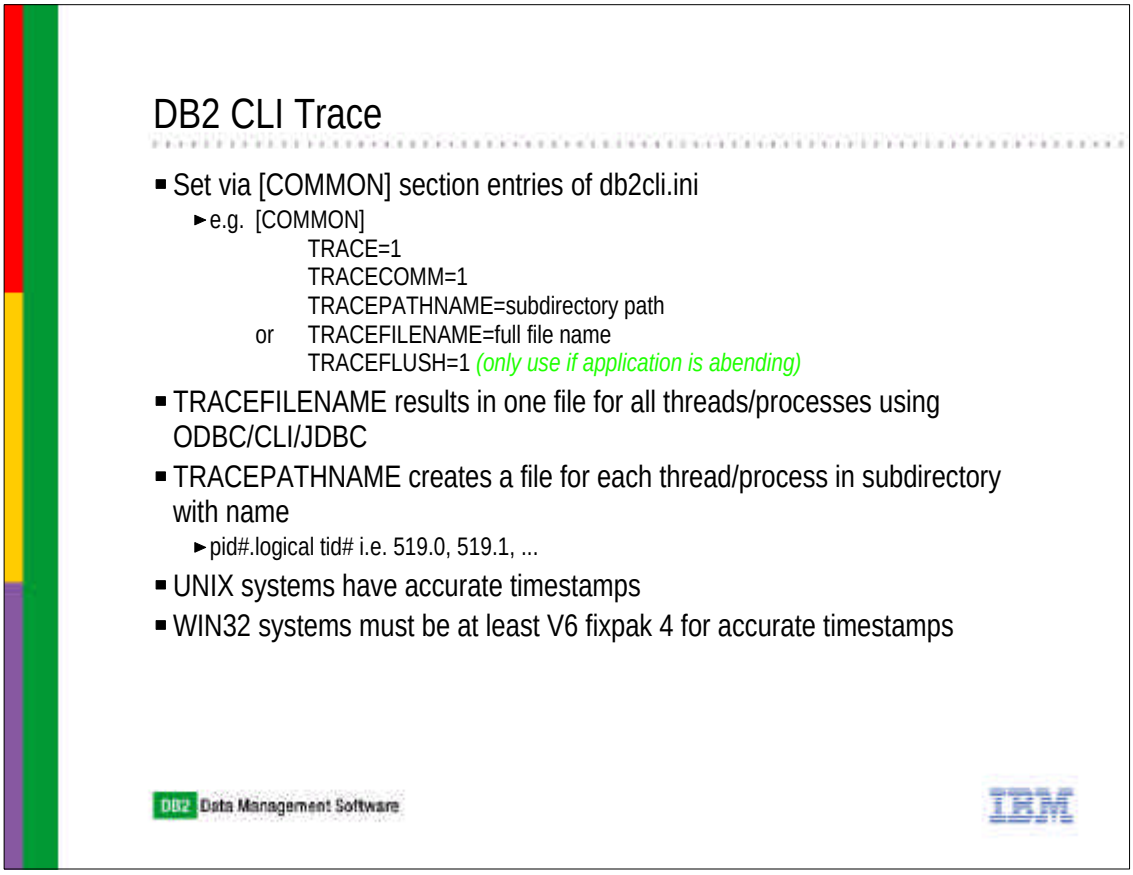
```
AVERAGE        APPL(CL.1)   DB2 (CL.2)     SQL DML     TOTAL
-------------  -----------  -----------    --------  --------
ELAPSED TIME   24:59.7823   3:50.24315     SELECT       50974
 NONNESTED     24:59.7823   3:50.24315     INSERT       30949
 STORED PROC     0.000000     0.000000     UPDATE       13029
 UDF             0.000000     0.000000     DELETE        1292
 TRIGGER         0.000000     0.000000
                                           DESCRIBE         0
CPU TIME       1:05.93530    40.218570     DESC.TBL         0
 AGENT         1:05.93530    40.218570     PREPARE          0
  NONNESTED    1:05.93530    40.218570     OPEN          5670
  STORED PRC     0.000000     0.000000     FETCH        15100
  UDF            0.000000     0.000000     CLOSE         5670
  TRIGGER        0.000000     0.000000
 PAR.TASKS       0.000000     0.000000     DML-ALL     122684

SUSPEND TIME          N/A   1:18.03509
 AGENT                N/A   1:18.03509
 PAR.TASKS            N/A     0.000000     --- DISTRIBUTED ACTIVITY -------
                                           REQUESTER           : 10.10.10.10
NOT ACCOUNT.          N/A   1:51.98949     COMMITS(1) RECEIVED: 4738
DB2 ENT/EXIT          N/A       213306     SQL RECEIVED        : 101914
EN/EX-STPROC          N/A         0.00     MESSAGES SENT       : 106653
EN/EX-UDF             N/A         0.00     MESSAGES RECEIVED   : 106653
                                           BYTES SENT          : 12943505
                                           BYTES RECEIVED      : 21614937
                                           MESSAGES IN BUFFER  : 9430
                                           ROWS SENT           : 13326
                                           BLOCKS SENT         : 5670
```

Time in DB2 = 3:50.24315+(1:05.93530-40.218570) = 4:15.95988
Time outside of DB2 = 24:59.7823-4:15.95988 = 20:43.82242

DB2 Data Management Software

IBM

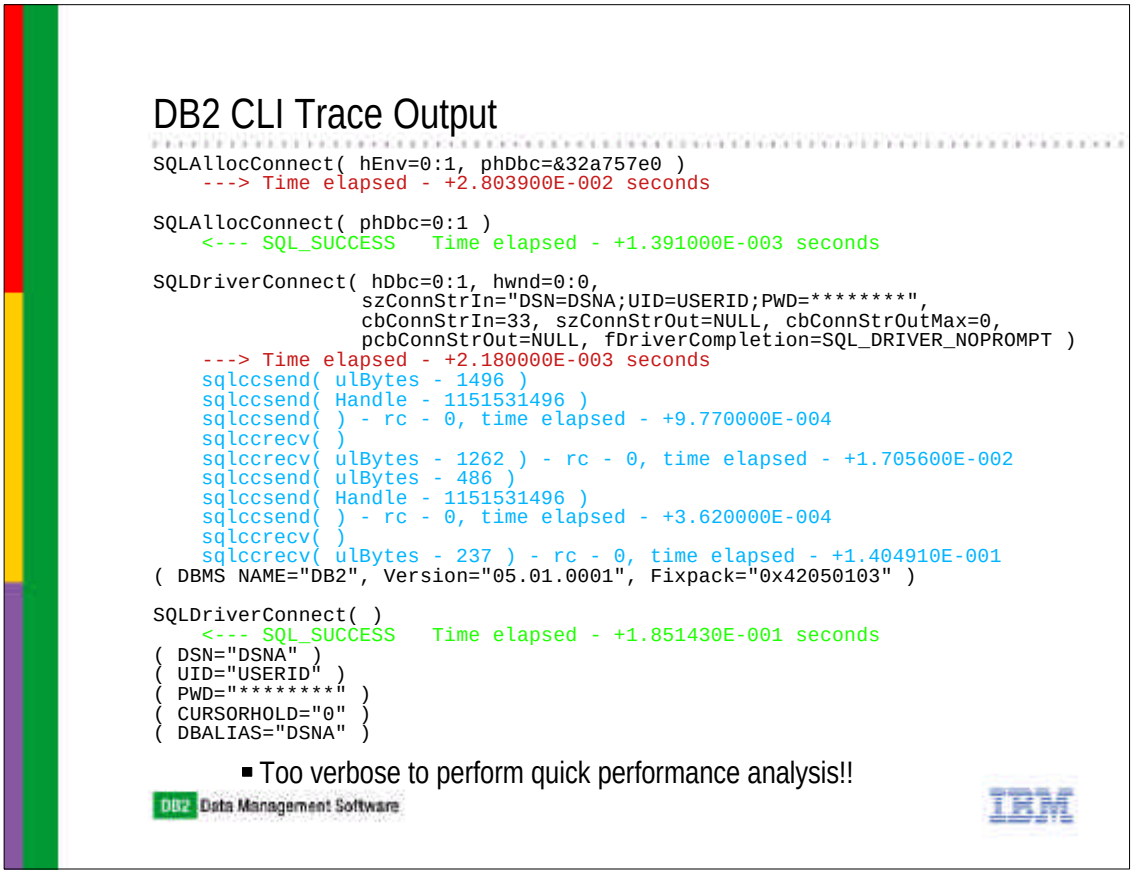
- This is an accounting trace of a recent performance problem. Even though the time in the DB2 server is about 14%, there is an indication in this information that the entire platform has a problem. Can you tell what it is? Look at the "NOT ACCOUNT." time in DB2. It is almost half the total time in DB2. This indicates a serious delay waiting for processing resources which can equate to either there are other more important tasks in the same LPAR or, with floating processor resources, there are other more heavily weighted LPARs taking precedence on the overall system. The latter was the case here.
- The number of messages sent in this case is the total of the SELECT (singleton), INSERT, UPDATE, DELETE, and OPEN DMLs processed plus the number of commits. This is an active thread accounting trace report.
- Finally, if the application was dynamic, then PREPAREs would be part of the inbound message usually chained with a DESCRIBE INPUT (JDBC), and INSERT/UPDATE/DELETE/OPEN. If a DESCRIBE was chained with a PREPARE, then the other kinds of DML would be separate messages.



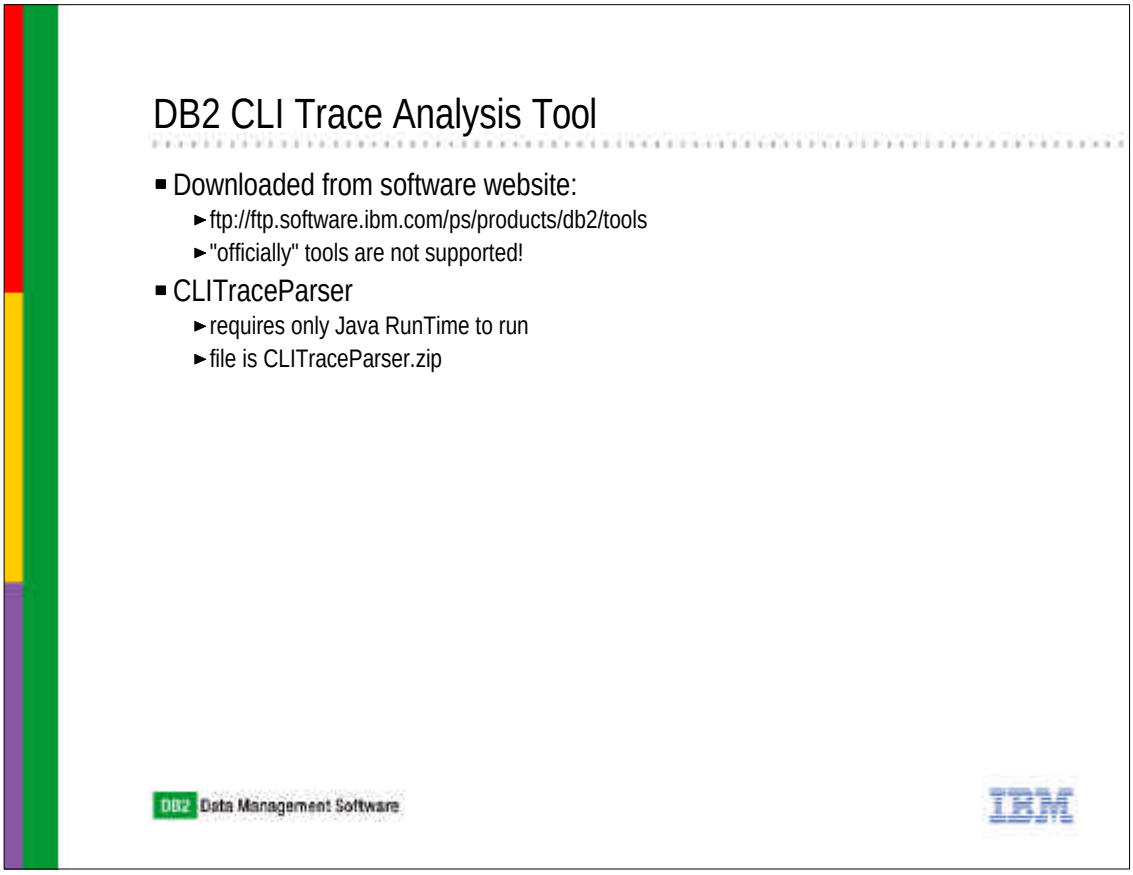
- Now, we come to a trace on the client when using ODBC/CLI/VB/JDBC... kinds of applications. Next to an accounting trace in DB2 on the mainframe, this trace is a very important tool in understanding performance problems.
- Add the information into your db2cli.ini file (which can be found in the installed DB2 client/Connect directory on Windows platforms and in the sqllib/cfg subdirectory of your instance on UNIX/LINUX platforms). By the way, if all you have done up to now is just install the DB2 client/Connect products, the db2cli.ini file contains example information. Remove all lines in it and leave the first which says that all comments start with a semi-colon. However, if you have used the Client Configuration Assistant on Windows plaftorms and have clicked to configure the alias as a datasource, then there may be some valid lines at the end of the file saying:

  [dbalias]
  DBALIAS=dbalias

  These are OK to leave in the file.
- I recommend using the TRACEPATHNAME feature because one gets a separate trace file for each thread/process and not one big intermixed file.



- This is a sample DB2 CLI trace output. It is really too verbose to actually perform any performance analysis. But it is full of lots of information. It also can be used to solve functional problems.
- The lines which start with "---> Time elapsed" have the amount of time spent in the APPLICATION since the last API request.
- The lines which start with "<---- Time elapsed" have the amount of time spent processing the particular ODBC/CLI... request.
- The lines wutg sqlccsend and sqlccrevr in them represent the time to process the communication with a server and are generated via TRACECOMM=1.



There is a tool which one can download from the specified web site called the CLITraceParser. With it and a real Java RunTime on your workstation (even the mainframe within Unix Systems Services), you can summarize the information from one of the trace files.

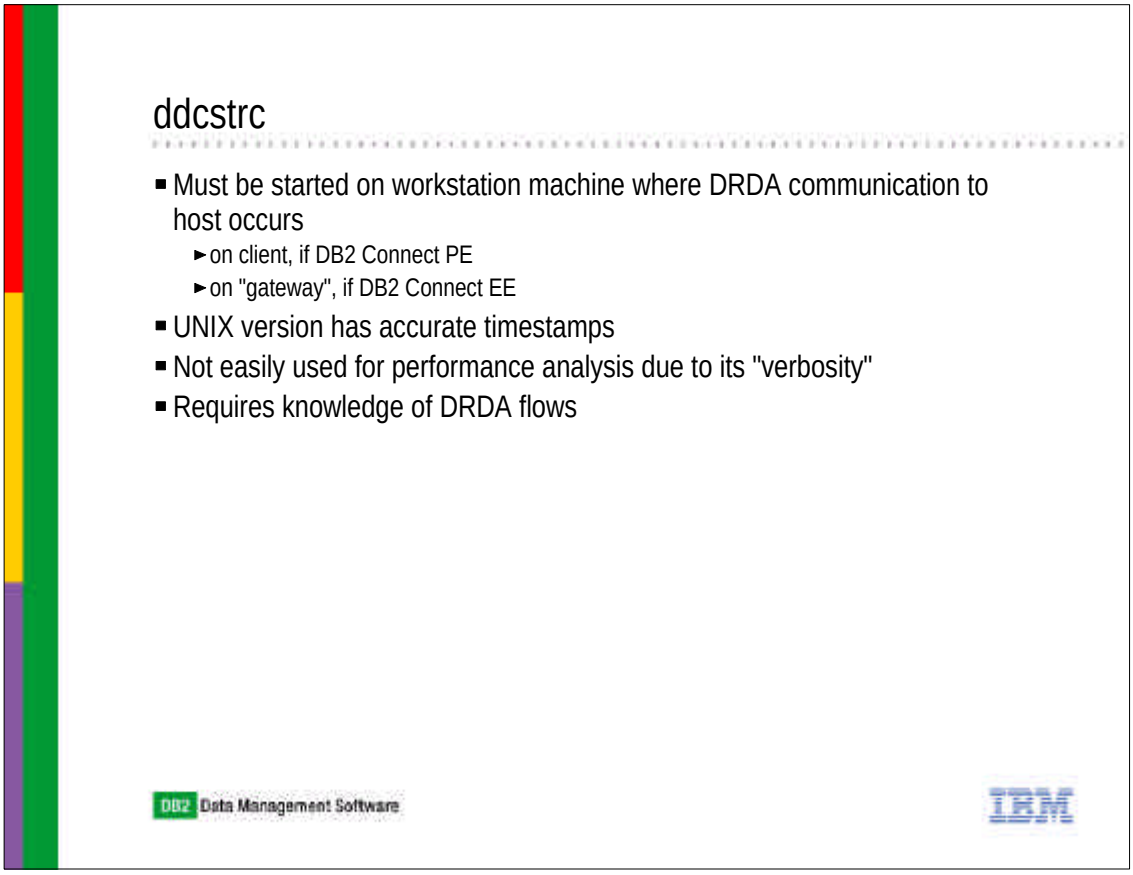
CLITraceParser Output

```
Overall Trace statistics
=========================================================
           309 statements in trace.
        23.674 seconds total trace time.
        20.373 seconds spent for application processing.
         3.301 seconds spent for CLI processing.
Network Specific CLI processing time statistics
=========================================================
            20 network flows sent to transmit
          9470 bytes, requiring a total of
         0.009 seconds.
            20 network flows received, transmitting
         14062 bytes, requiring a total of
         2.924 seconds.
End of overall trace statistics report
*********************************************************************************************************
Function specific statistics
=========================================================
                                Timing                          Network Send          Network Receive
Function Name       Total  Application CLI          Flows Bytes       Time   Flows Bytes       Time
---------------------------------------------------------------------------------------------------------
SQLTransact         8      0.022       0.941        8     1536        0.003  8     216         0.927
SQLGetFunctions     1      0.006       0.001        0     0           0.000  0     0           0.000
SQLSetConnectOption 2      0.009       0.001        0     0           0.000  0     0           0.000
SQLColAttribute     61     0.027       0.060        0     0           0.000  0     0           0.000
SQLBindCol          32     0.007       0.026        0     0           0.000  0     0           0.000
SQLAllocEnv         1      0.000       0.001        0     0           0.000  0     0           0.000
SQLPrepare          4      0.006       0.006        0     0           0.000  0     0           0.000
SQLRowCount         4      0.006       0.002        0     0           0.000  0     0           0.000
SQLNumParams        4      0.001       0.002        0     0           0.000  0     0           0.000
SQLFetch            39     0.078       0.023        0     0           0.000  0     0           0.000
SQLDriverConnect    1      0.002       0.185        2     1982        0.001  2     1499        0.158
SQLFreeStmt         4      0.006       0.004        0     0           0.000  0     0           0.000
SQLExecute          4      0.006       1.891        10    5952        0.004  10    12347       1.840
SQLBindParameter    25     0.019       0.032        0     0           0.000  0     0           0.000
SQLNumResultCols    8      0.006       0.004        0     0           0.000  0     0           0.000
SQLGetData          104    0.308       0.112        0     0           0.000  0     0           0.000
SQLAllocConnect     1      0.028       0.001        0     0           0.000  0     0           0.000
SQLGetEnvAttr       1      0.000       0.001        0     0           0.000  0     0           0.000
SQLSetEnvAttr       1      0.000       0.001        0     0           0.000  0     0           0.000
SQLAllocStmt        4      19.837      0.007        0     0           0.000  0     0           0.000
End of function specific statistics report
```

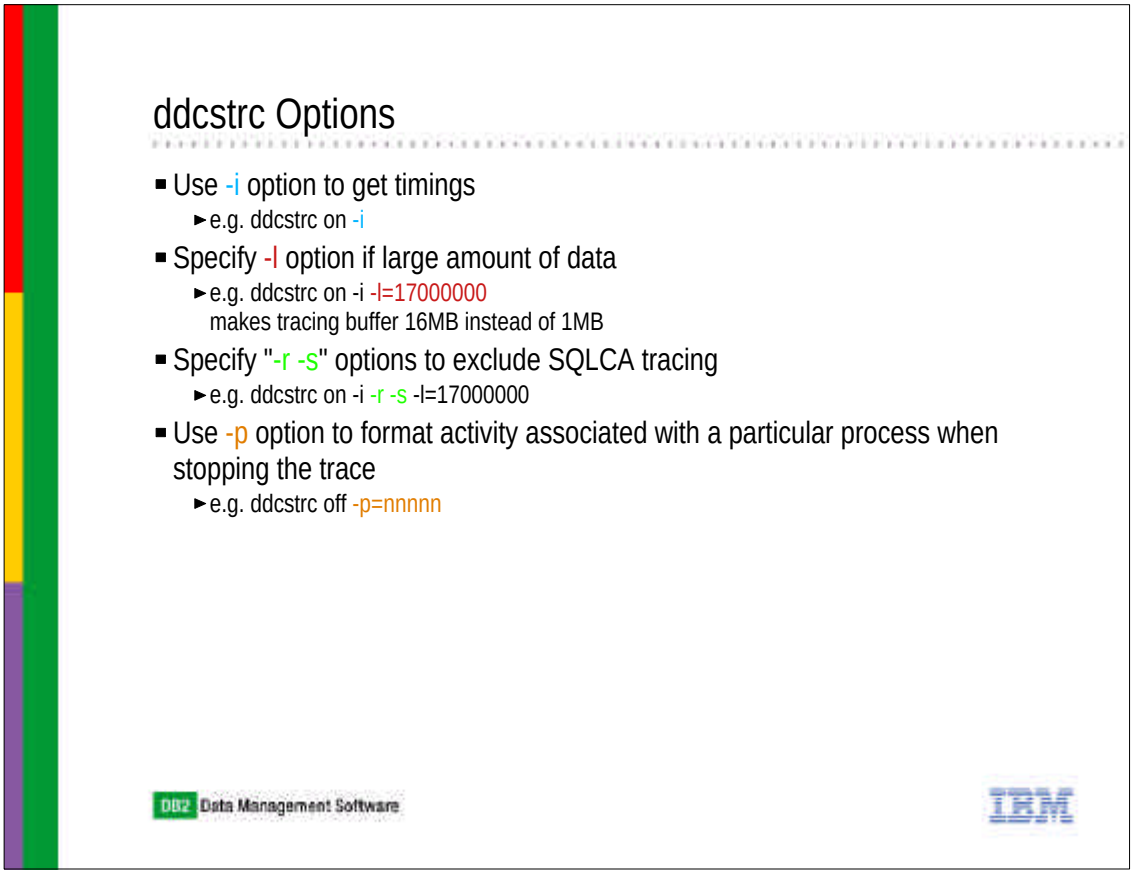
DB2 Data Management Software

IBM

This is an example of CLITraceParserOutput. It breaks down the amount of time in the file between application and the CLI driver. Within the CLI driver, further time is summarized between each of the API calls issued by the client and whether those calls had to spend time waiting for server responses. This file has an example of a client which does connection pooling. Note the high amount of application time before the SQLAllocStmt call. This API would be one of the first to be issued by an application which reused a connection after waiting a period of time for the next application transaction to begin.

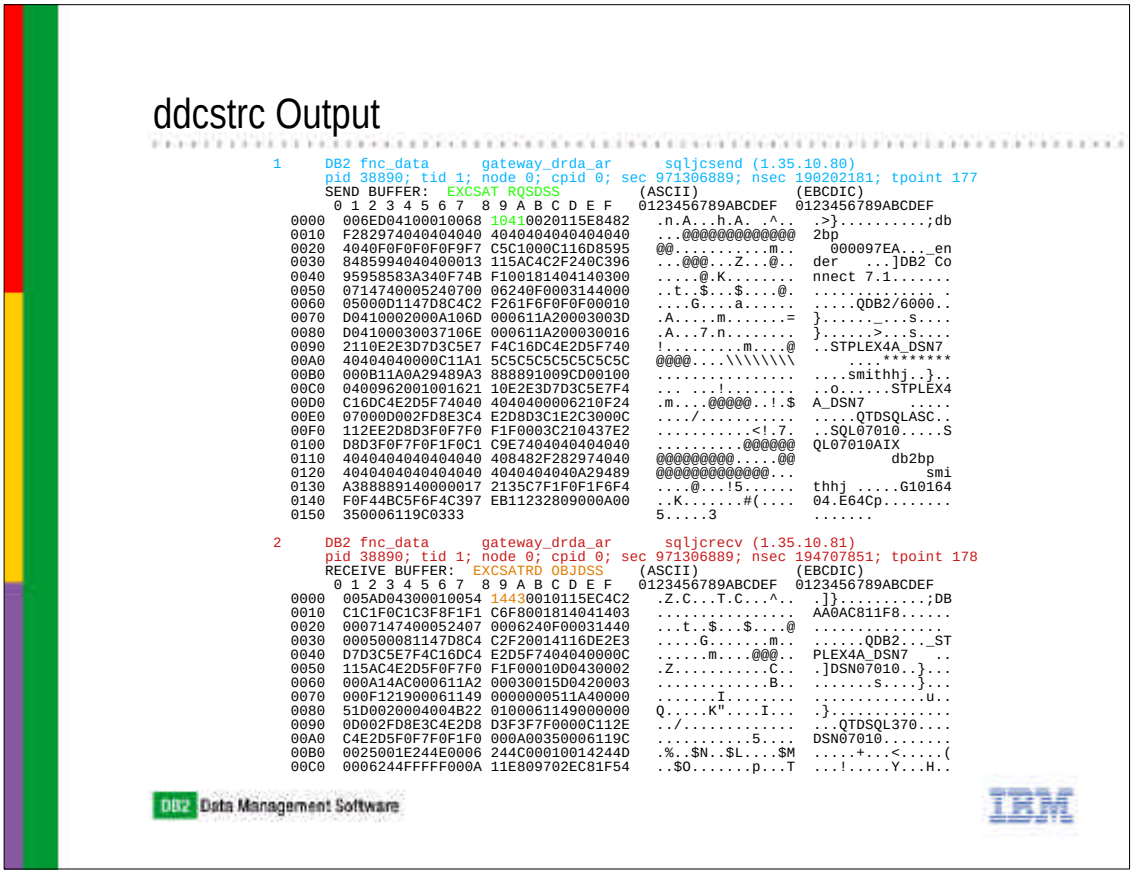


The ddcstrc function runs only on a workstation which is perfoming DRDA communication to a host. It is unfortunately too verbose to be very useful in solving performance problems. There are currently no tools to sumarize its information. However, when doing application problem determination, it provides invaluable information about what was actually sent to and returned from the host.
To read it, one requires an extensive knowledge of DRDA flows.

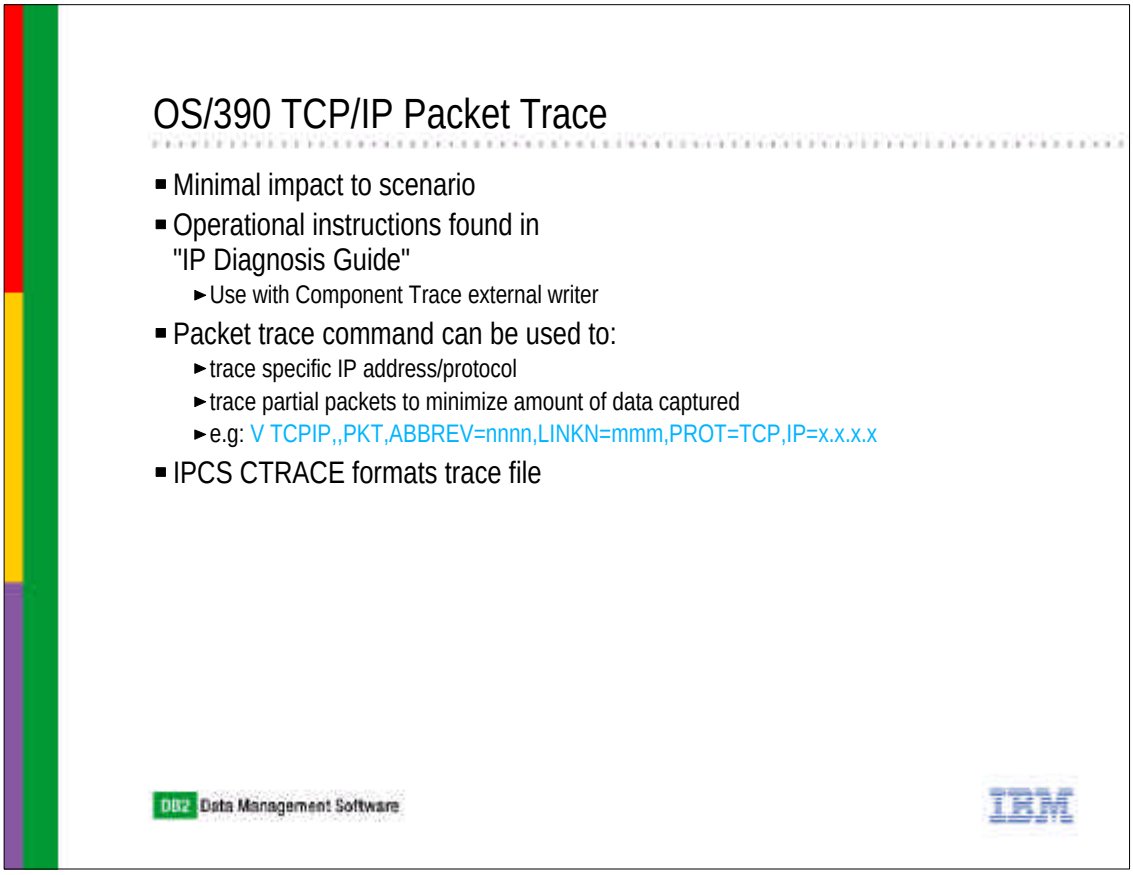


Here are the command options.
I recommend starting the trace if to be used as follows:

ddcstrc on -i -r -s -l=17000000



This is a sample of a ddcstrc just showing a connect. EXCSAT stands for exchange server attributes and the EXCSATRD stands for exchange server attributes reply data.
In these connect flows, there are four chained DRDA messages which make up a connect. EXCSAT followed by ACCSEC (access security), SECCHK (security check), and ACCRDB (access relational database).



- Sometimes you want to see what the TCPIP stack says at a point close to the network device driver.
- Use the mentioned manual to guide you on starting up the component trace external writer which is where a packet trace is written.
- The packet trace is then started via a TCPIP command and you can limit what information is put into the trace.
- It has minimal impact to the whole scenario because initially the packets are written to a OS/390 dataspace and then subsequently stored on disk.
- You must then use the IPCS CTRACE command to format the trace.

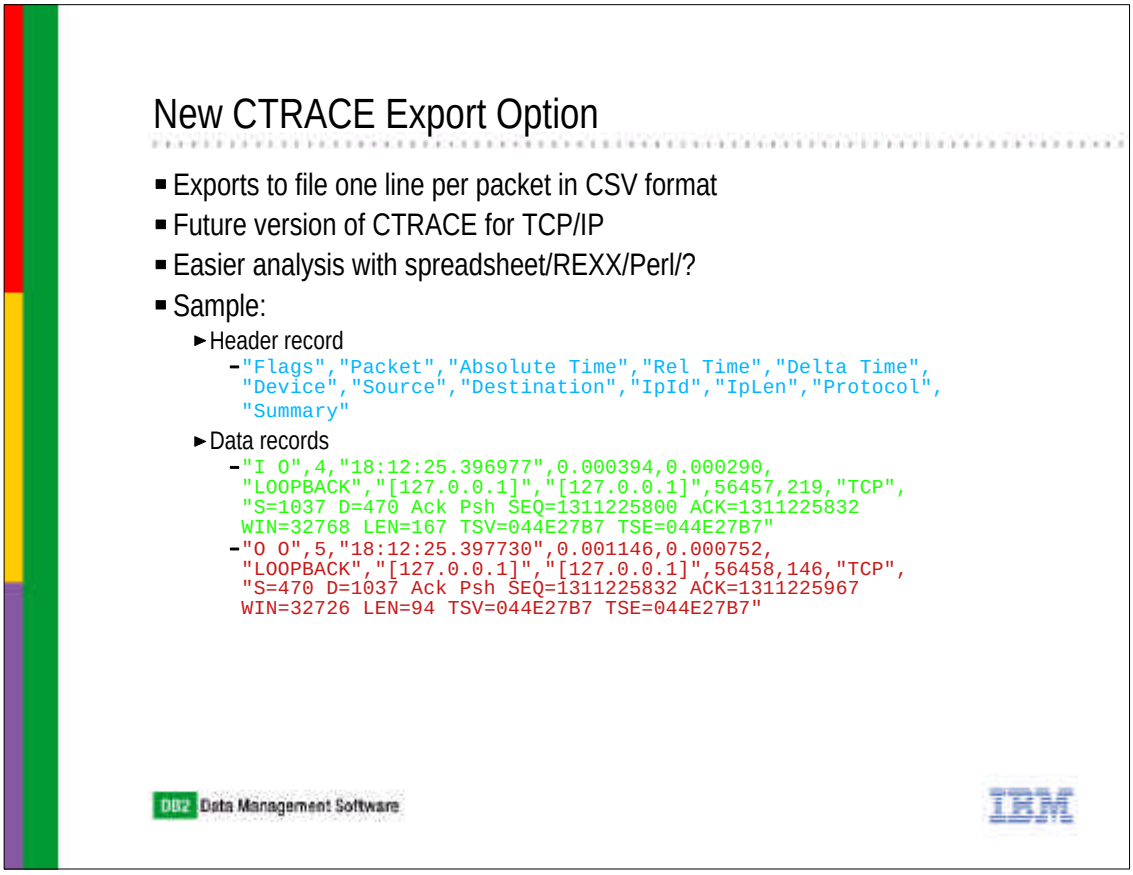
TCP/IP Packet Trace CTRACE Output

```
STLCEL1   PACKET    00000001  18:12:25.396977  Packet Trace
FROM LINK =  LOOPBACK          DEV = Loopback              FULL
 TOD CLOCK = XB4811389  EDEF0187  TIME ZONE = XFFFFA21F
 PKT 4          LOST RECORDS = 0       HDR SEQUENCE NUM = 0
IP SRC = 127.0.0.1                IP DST = 127.0.0.1
 HDLEN = 5    TOS = X00  TOTLEN = 219    ID = 56457  FLAGS = (none)
 FRAGOFF = 0      TTL = 64   PROTOCOL = TCP    CHECKSUM = X9F91  FFFF
TCP SRC PORT = 1037               TCP DST PORT = 470
 SEQ NUM = 1311225800  ACK NUM = 1311225832  FLAGS = ACK PSH
 HDLEN = 8    WINDOW = 32768  CHECKSUM = X3601  FFFF  URGENT PTR = 0
TCP OPTION = NOP
TCP OPTION = NOP
TCP OPTION = TIME STAMP  LEN = 10   VALUE = X044E27B7  ECHO = X044E27B7
 HEADER LENGTH = X0034
0000 450000DB DC890000 40069F91 7F000001 *.....i.. ..j"...|E.......@.......*
0010 7F000001 040D01D6 4E27B7C8 4E27B7E8 *".....O+..H+..Y|........N'..N'..*
0020 80188000 36010000 0101080A 044E27B7 *.............+..|....6........N'.*
0030 044E27B7                            *.+..            |.N'.            *
  DATA LENGTH = X00A7
0000 00A7D001 000100A1 10410059 115EE2D4 *.x}..........SM|.........A.Y....*
0010 C9E3C8C8 D1404040 40404BC2 C1E3C3C8 *ITHHJ     .BATCH|.....@@@@@K.....*
0020 4040404B E2D4C9E3 C8C8D140 4BC4E2D5 *   .SMITHHJ .DSN|@@@K.......@K...*
0030 C5E2D7C3 E24B0000 00000000 00000000 *ESPCS...........|.....K..........*
0040 00000000 00000000 00000000 4BE3E2D6 *............TSO|............K...*
0050 40404040 40404040 40404040 40404040 *                |@@@@@@@@@@@@@@@@*
0060 40404000 14116DC1 D7D7D3C4 C2F0C140 *    ..._APPLDB0A |@@@...m........@*
0070 40404040 40404000 0C115AC4 E2D5F0F6 *         ...!DSN06|@@@@@@@...Z.....*
0080 F0F1F100 081147D8 C4C2F200 1C140414 *011....QDB2.....|......G.........*
0090 03000724 07000624 0F000314 74000514 *................|...$...$....t...*
00A0 C0000714 400006                     *{... ..         |....@..         *
```

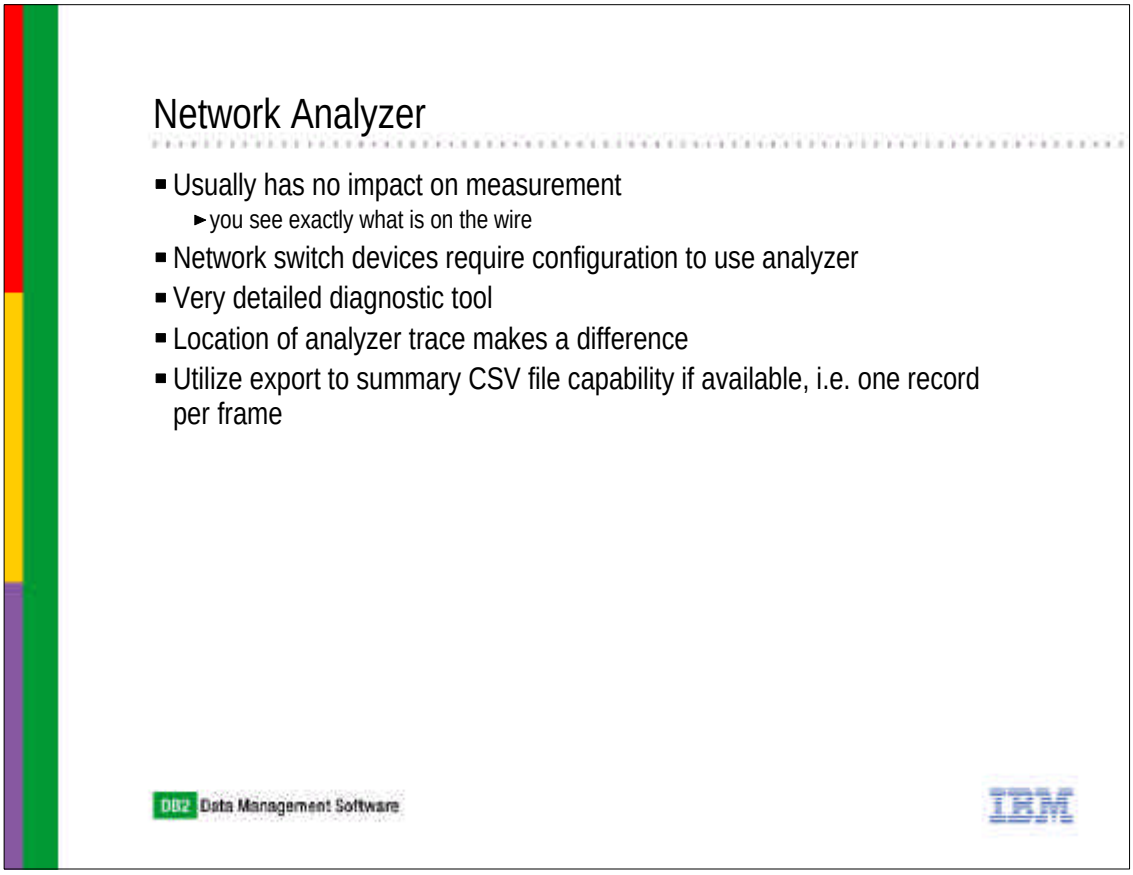
DB2 Data Management Software

IBM

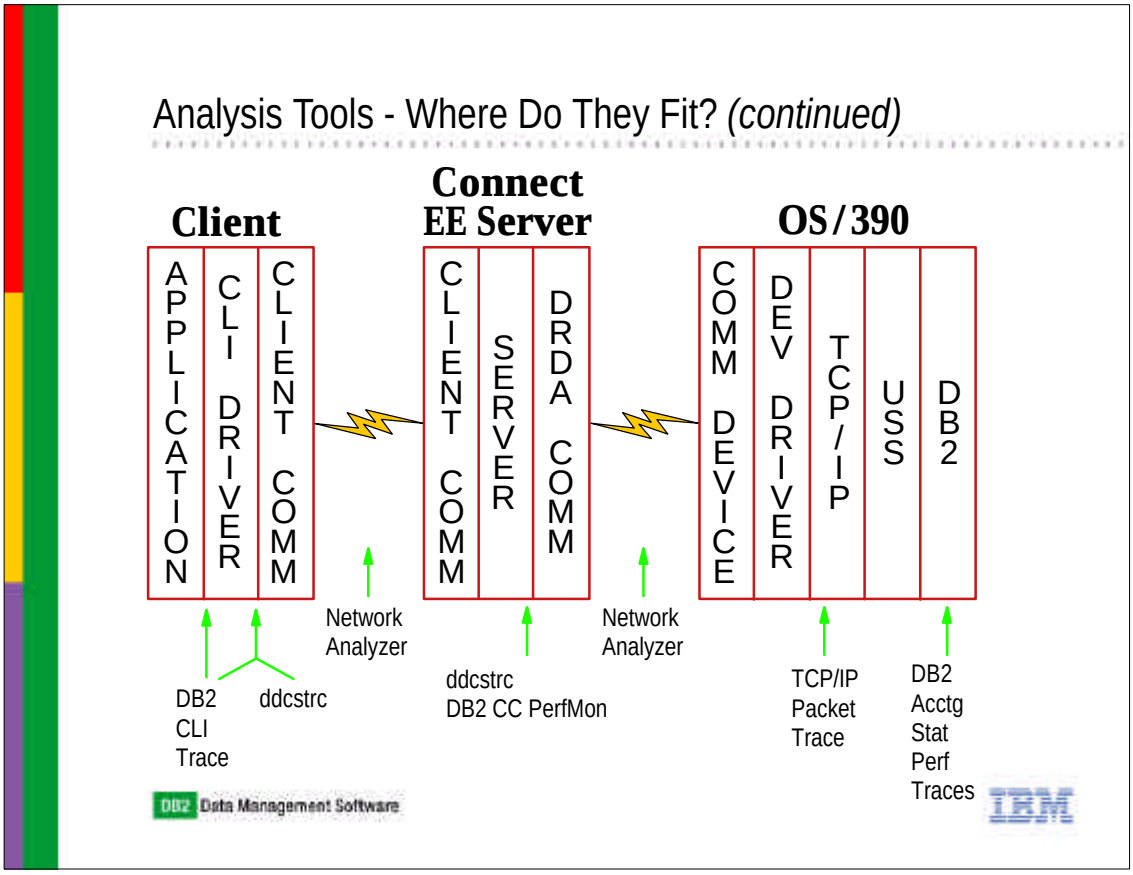
Here is an example of the packet trace output. Again very verbose. So I worked with the TCPIP service developmer to get them to change the CTRACE tool to put out a special form of this trace. I suggested that a file of single lines which represented the direction of the trace, source|destination ip addresses, and the actual, relative, and delta times of the trace entry would help people analyze their performance problems faster.
And so....



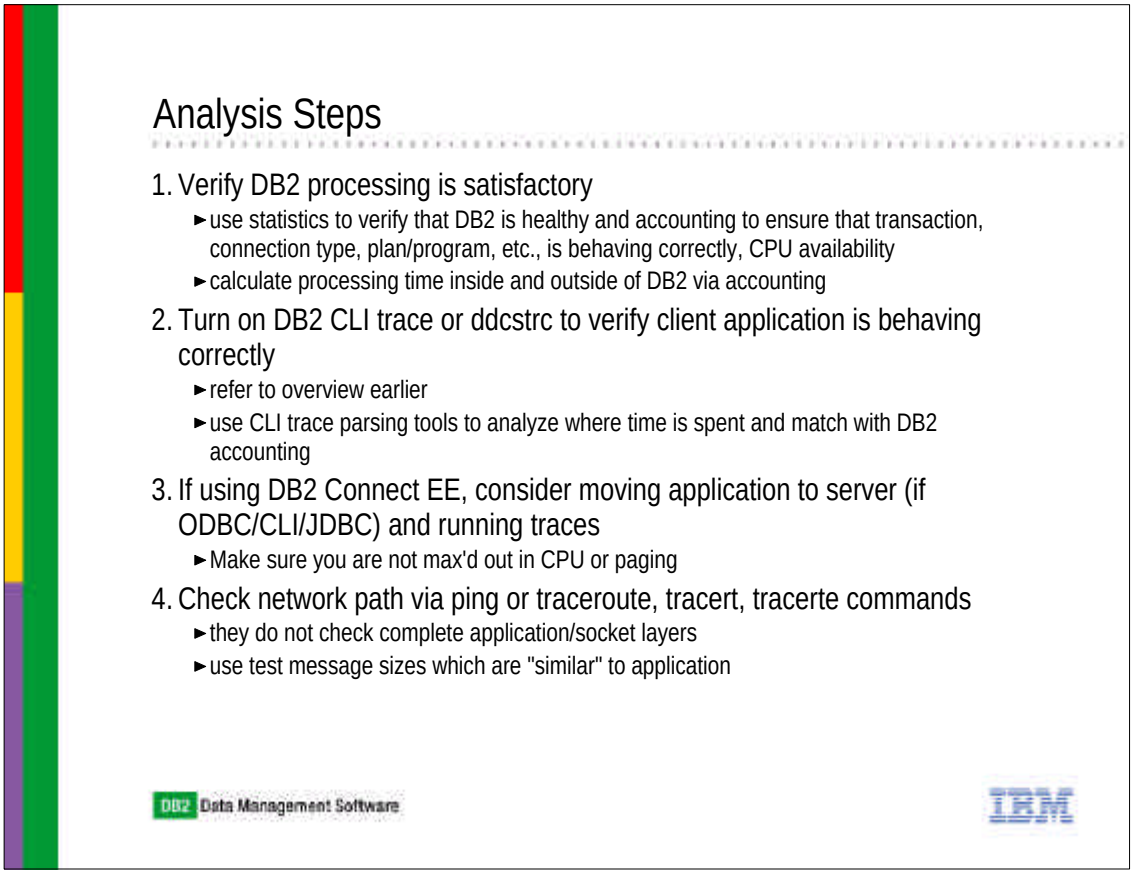
A new packet trace analysis tool has been written which outputs a comma, separated variables file which can then be input to a spreadsheet or a perl/REXX program to analyze where the time went. It should be delivered with z/OS 1.2.



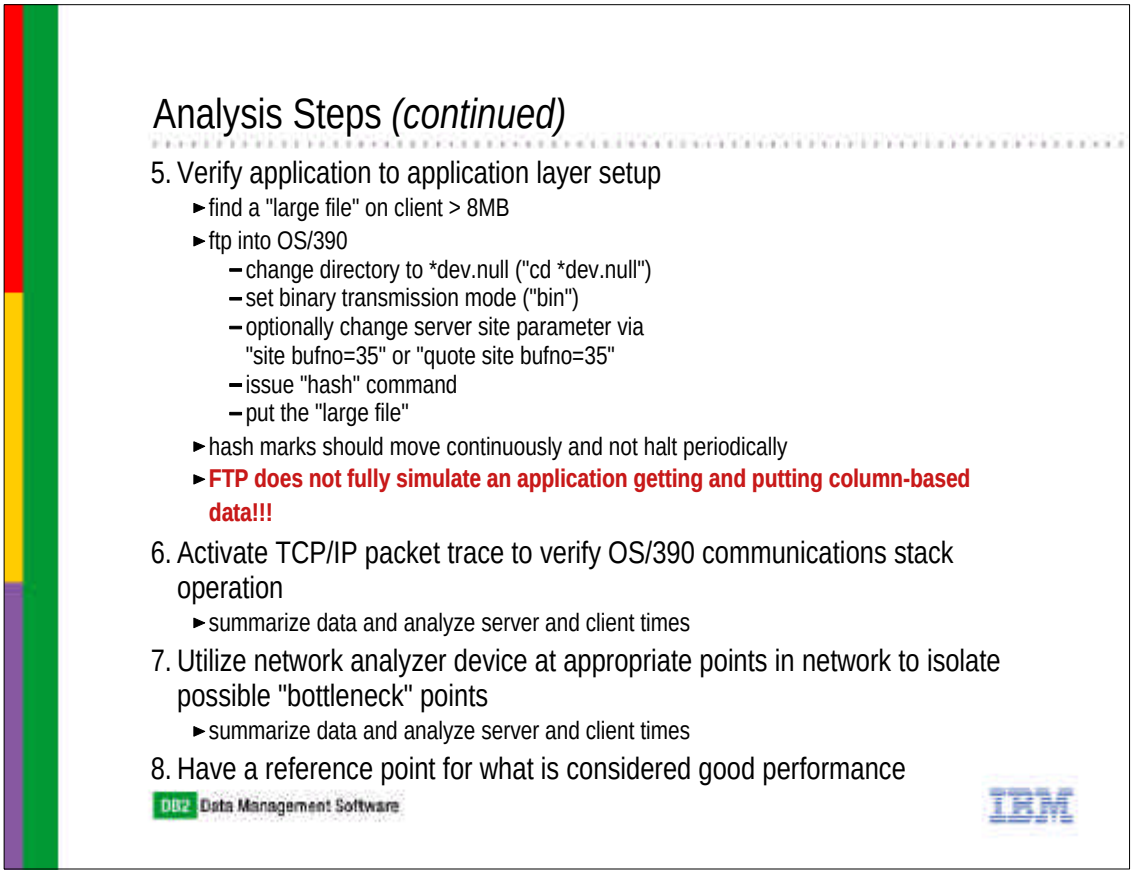
These kind of devices provide the absolute best time recording of what happened going across a network. They are also called sniffers. However, network analyzers have a limited memory so that their buffers will wrap after a short period of time.
Most analyzer supplied software will permit you to generate a CSV file which you can then use a spreadsheet or a REXX/perl program to analyze.



Now we have gone through the tools that I use to try and answer the question, "where did the time go?" This chart depicts where the various tools probe.



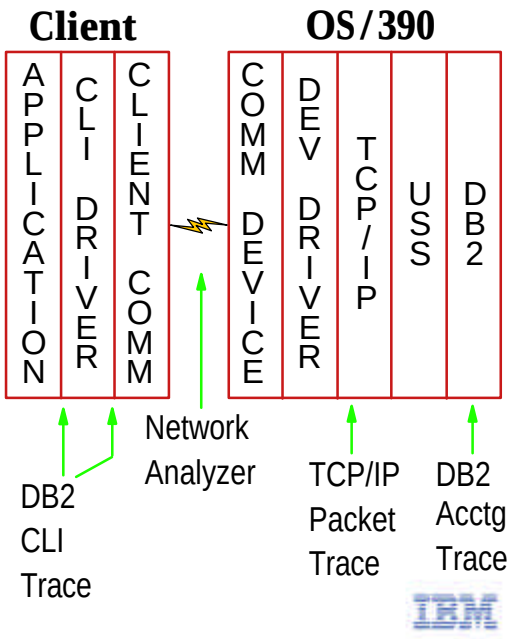
- Start off by verifying via accounting and statistics traces that DB2 itself is behaving satisfactorily. Also the CPU platform itself. Once you have determined that DB2 is not the prime time consumer, calculate with an accounting trace when all the other traces are on the time inside and outside of the DB2 server.
- Next go to the client, and get the DB2 CLI trace if possible and use the CLITraceParser to get the time in application and outside of the application.
- If using a DB2 Connect EE server, try and rerun the application from it getting the appropriate traces.
- Check to see the kind of network route between the client and the DB2 Connect EE server and the DB2 Connect EE server and the DB2 mainframe and/or the client and the DB2 mainframe. The best command is a form of traceroute (AIX) which lists each router hop and the elapsed time between it and the source point. Try and run the ping or traceroute tools with message sizes similar to that used by the application (one important point to remember is that ping and traceroute do not measure the complete application end-to-end response time; in fact, a ping response on most systems can be returned for the server's network access point or adapter).



- Now, let's see if the network path between any two of the major points is configured correctly. Step 5 usually has to be done on a FastEthernet network connection because these connections are usually auto-configured and can be configured incorrectly. The most important thing to watch out for is that the network connection is not half-duplex.
- Use the TCPIP packet trace and summarize the data to get server and client times.
- Sometimes, you will have to work with a network support person to get a sniffer to be used. Become friends with these people, you will need their help.
- Finally, I can't stress the last point enough. When you finally put an application into production, take all of the traces, etc., and save them as an example of good performance. Even when you change various parts of the environment, rerun the traces and analyses. It is very hard to proof to someone that there is no problem when you do not have any proof of what was considered good before.

Sample Application Environment and Analysis Steps

- Sample application
  - connect to db2
  - prepare with parameter markers an insert to table of two integer columns
  - insert 10000 rows
  - commit
  - disconnect



DB2 Data Management Software

IBM

Now we are going to go through an analysis of a sample application.

Sample Accounting Trace Information

TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)
ELAPSED TIME	12.448745	1.740990
NONNESTED	12.448745	1.740990
STORED PROC	0.000000	0.000000
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
CPU TIME	3.206548	1.663116
AGENT	3.206548	1.663116
NONNESTED	3.206548	1.663116
STORED PRC	0.000000	0.000000
UDF	0.000000	0.000000
TRIGGER	0.000000	0.000000
PAR.TASKS	0.000000	0.000000
SUSPEND TIME	N/A	0.006908
AGENT	N/A	0.006908
PAR.TASKS	N/A	0.000000
NOT ACCOUNT.	N/A	0.070966
DB2 ENT/EXIT	N/A	20005
EN/EX-STPROC	N/A	0
EN/EX-UDF	N/A	0

SQL DML	TOTAL
SELECT	0
INSERT	10000
UPDATE	0
DELETE	0
DESCRIBE	0
DESC.TBL	0
PREPARE	1
OPEN	0
FETCH	0
CLOSE	0
DML-ALL	10001

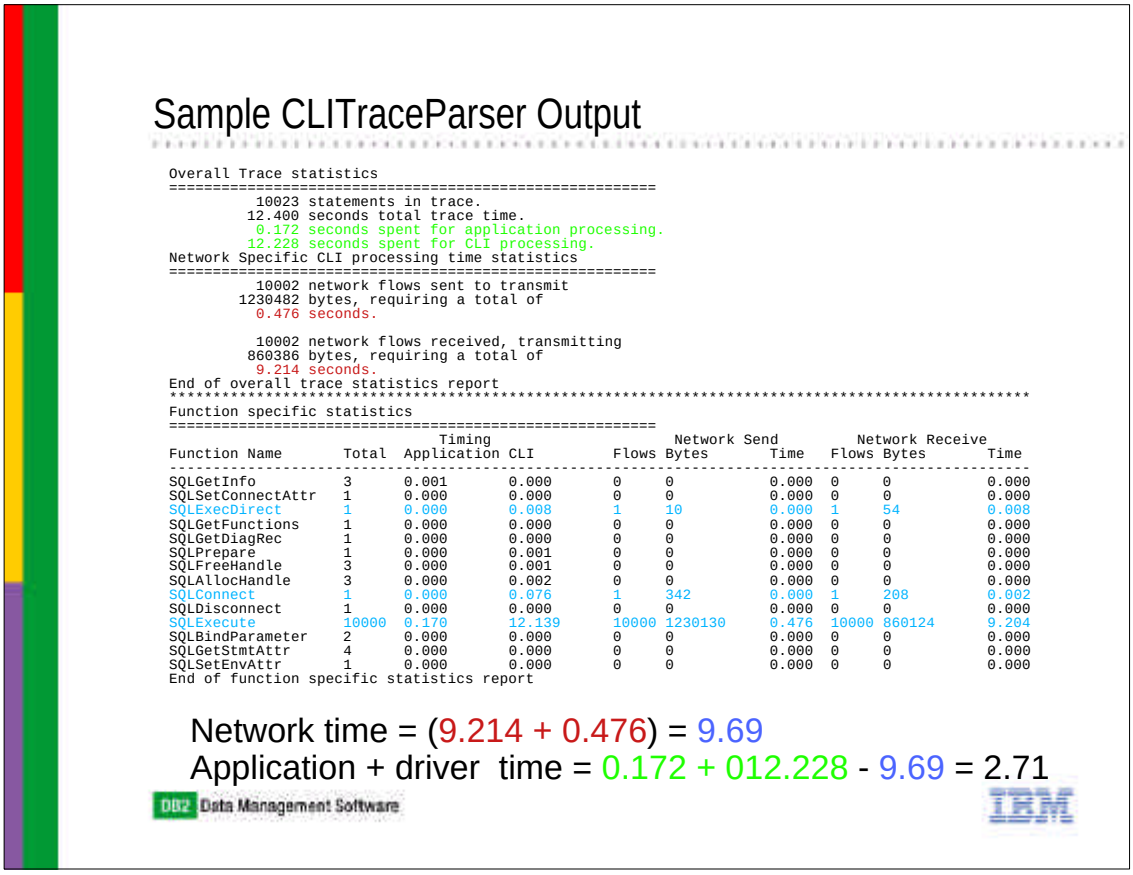
```
--- DISTRIBUTED ACTIVITY -------
REQUESTER           : 10.10.10.10
SQL RECEIVED        : 10001
MESSAGES SENT       : 10002
MESSAGES RECEIVED   : 10002
BYTES SENT          : 860386
BYTES RECEIVED      : 1230482
MESSAGES IN BUFFER  : 0
ROWS SENT           : 0
BLOCKS SENT         : 0
```

Time in DB2 = 1.740990 + (3.206548 - 1.663116) = 3.284422
Time outside of DB2 = 12.448745 - 3.284422 = 9.164323

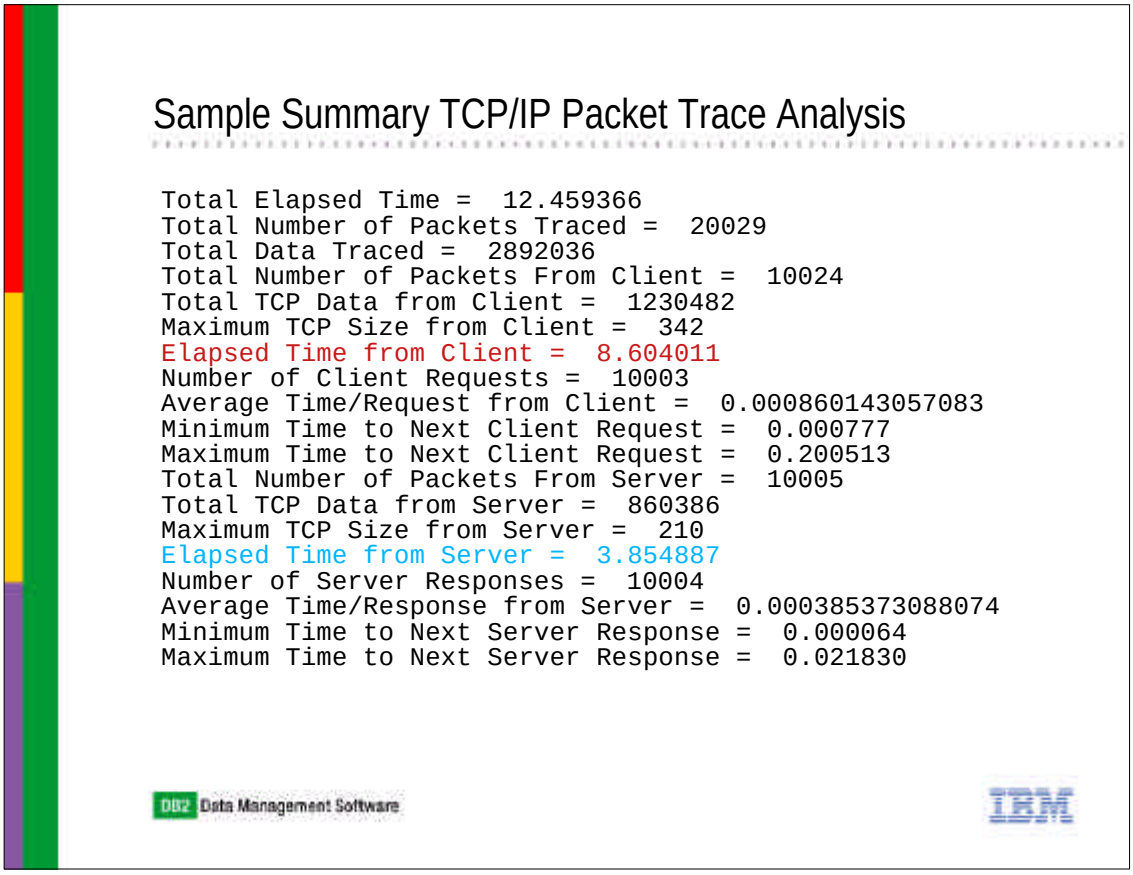
DB2 Data Management Software

IBM

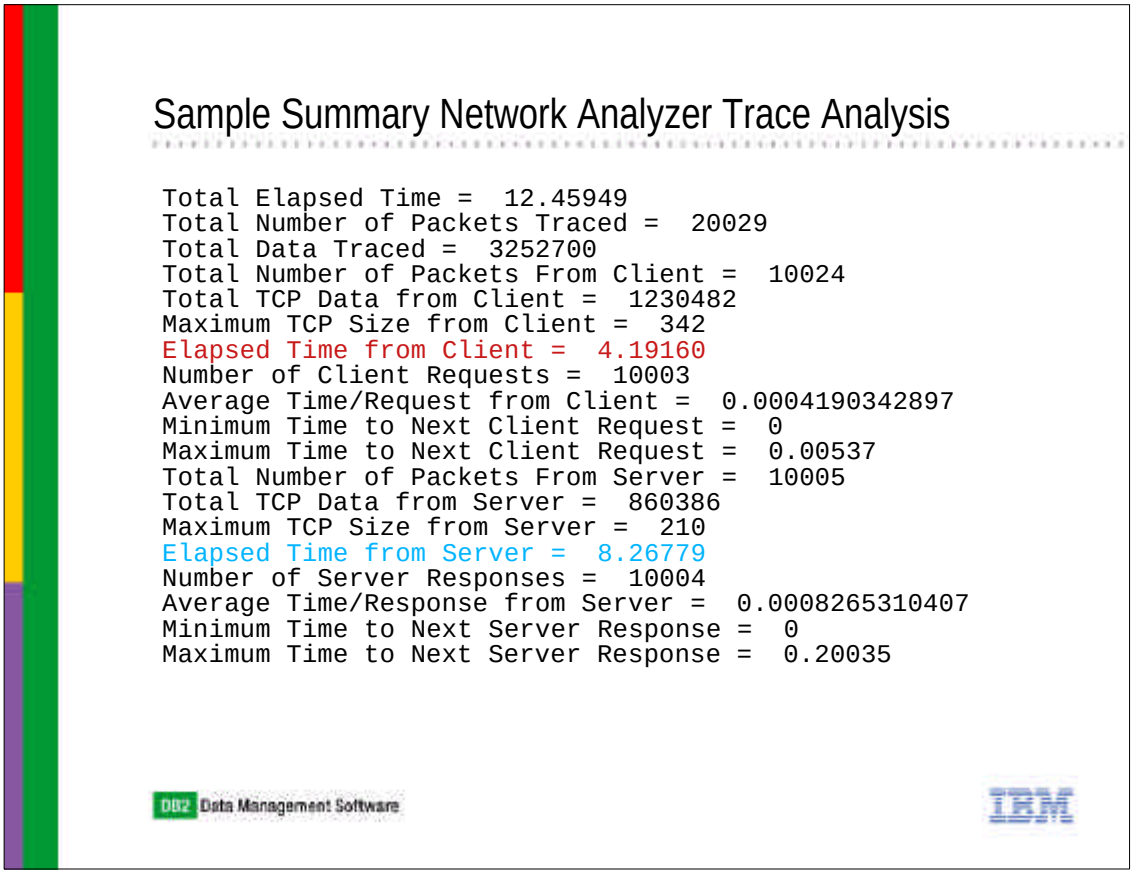
Here are the calculations for the time in DB2 and outside of the DB2 server.



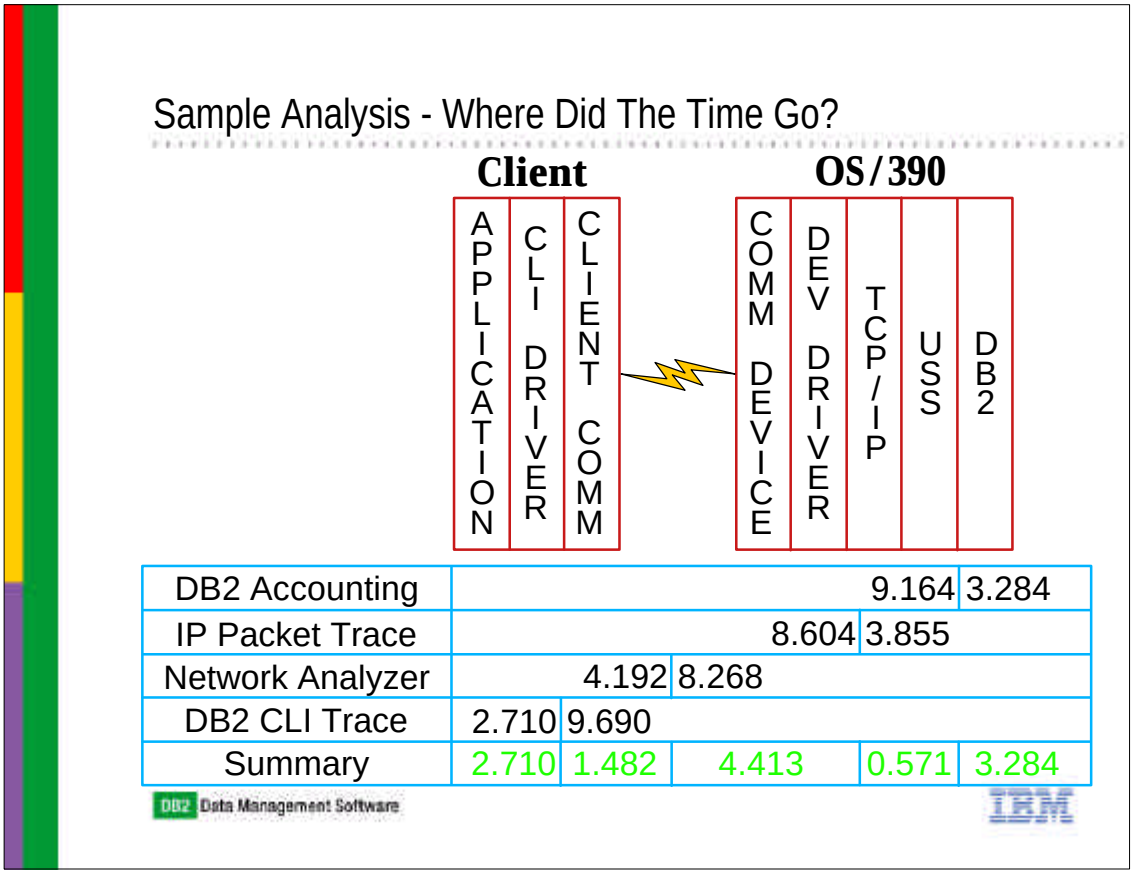
Here is the CLITraceParser output of the application.
Note how both the server traces and the client traces suggest the problem is not in each other.



I have a mainframe REXX program which can analyze the CSV file from the packet trace tool. This is its output. I will make it available to anyone who emails me for it.



This is my REXX program's output of a sniffer trace.
Again, note how the server packet trace suggested that most of the time was outside of the server and the network analyzer said most of the time was the server.
Hmmm!



Now, the best way to see where the time went is to pictorially display it like I did here.
Using the base set of information obtained, you can get a further breakdown of where the time went and then you can spend your time trying to reduce the biggest time consumers.