SYBASE®

Application Tracing in Adaptive Server® Enterprise (ASE) 15

Mehul Wagle and Nitin Verma
Server Performance Engineering and Development Group
Sybase, Inc.

technology

WHITE PAPER

SYBASE

Table of contents

1. Motivation **3**

2. Feature Overview **3**

3. Command Interface **4**

3.1 How to enable tracing *4*

3.2 How to disable tracing *4*

4. Scope of the feature **4**

4.1 How to trace SQL Text *6*

5. Use Cases **7**

5.1 How to trace a specific client session *7*

5.2 How to trace sessions of a specific login *8*

5.3 How to find which sessions are being traced *10*

5.4 How to rebind to an existing trace *10*

5.5 How to create trace output file at various paths *11*

6. Restrictions/Limitations **11**

7. Conclusion **12**

1. Motivation

Sybase ASE 15.0.1 already has several tracing options that customers can enable to collect useful diagnostic information. This information includes query-level details like execution plan, i/o cost, etc. For example, the "set showplan on" command will display the plan for a SQL query just before executing it. However customers can only enable diagnostics if they have access to their SQL client sessions connected to the server. Hence tracing a running client application will require stopping the application, modifying its code to include the relevant diagnostic SQL commands at appropriate places, and finally restarting it. Restarting a client application is not feasible in production environments, and modifying application code may not even be possible if it comes pre-packaged. Often customers face situations where a specific application is running significantly slow or not as per expectations. The cause is unknown since database applications behave like a black box. Even someone as powerful and equipped as a database administrator has little access to server-side tasks initiated by an application during execution. Keeping this in mind, it would be worthwhile if database users could trace their server-side tasks to monitor the health of their application or if database administrators could do the same for all running user tasks so as to monitor the health of the server.

With this in hindsight, the upcoming flagship release of Sybase Adaptive Server® Enterprise (ASE) 15.0.2, introduces the "Application Tracing" feature. The primary motivation behind constructing this server-side framework is to capture execution-level traces of the server for any given client session in order to help debug performance bottlenecks. This white paper essentially highlights the significant elements of this new feature set.

2. Feature Overview

The "Application Tracing" feature provides database users, having special privileges, the capability to turn on the commonly used server diagnostic options (refer section 4 for list of supported options) for a running client session, and capture the trace output into a text file. Just like the server log file helps administrators to debug the server-wide execution, an application trace file helps individual users to monitor session-specific execution. All that a user needs to know for using this feature is the SPID (Server Process ID) of the session to be traced. The SPID is a unique number assigned by the ASE database server to any task running on it, just like PID of a process in UNIX. There exists a stored procedure in ASE named "sp_who", which reports all tasks running on the server at that moment. Once the SPID for the session to be traced is determined, the user just needs to issue the command:

```
set tracefile "<file-path>" for <spid>
```

Thereafter the traceable diagnostic options, when explicitly enabled by the user (e.g. `set statistics io on`), will take effect only on the target session and the corresponding output will be redirected to the specified file. The diagnostic output will continue to be logged into the trace file as long as tracing is active for that session.

In addition to being able to trace sessions by their SPID, one can also trace sessions of a particular login. For this the database administrator must create a "login trigger" (simply a SQL stored procedure executed at login time) for the target user login and provide the necessary permission to allow that login to trace itself (refer section 5.2 to understand how this is done). Once a login trigger is installed for

the desired login, it will be executed every time a user logs in to the server using that login. Subsequently that entire session will be traced until the user logs out or explicitly disables tracing.

3. Command Interface

3.1 How to enable tracing

- for own session:

```
Syntax: set tracefile "<file-path>"
```

Note: Double quotes are compulsory for file path; files with relative path are created under $SYBASE.

- for another session:

```
Syntax: set tracefile "<file-path>" for <spid>
```

Note: Double quotes are compulsory for file path; files with relative path are created under $SYBASE.

- for rebinding with tracing already in progress on another session (used in case the tracer session quits):

```
Syntax: set tracefile for <spid>
```

- for target session's lifetime irrespective of its state of execution:

```
Syntax:
   set tracefile "<file-path>" [for <spid>]
   set export_options on
   set <option-name> on/off
```

Note: This interface is needed in ASE because 'set' options, enabled for a session while it is executing a stored procedure, are reset back once the procedure quits.

3.2 How to disable tracing

- enabled by the current session (either for itself or for another session):

```
Syntax: set tracefile off
```

- enabled by another session (used in case that session quits without turning off the tracing):

```
Syntax: set tracefile off for <spid>
```

4. Scope of the feature

The 'set' commands in ASE 15.0.2 whose output can be traced under application tracing context are:

1. `set show_sqltext <on/off>` (refer section 4.1)
2. `set showplan <on/off>`
3. `set statistics io <on/off>`
4. `set statistics time <on/off>`
5. `set statistics plancost <on/off>`

6. All 'set option' subcommands. To enumerate a few:
   - `set option show <normal/brief/long/on/off>`
     - show diagnostics from all modules; each module will thus be set in the off/brief/normal/long level that is coherent both between modules and with the desired global brief/normal/long level of information
   - `set option show_lop <normal/brief/long/on/off>`
     - show logical operators used
   - `set option show_managers <normal/brief/long/on/off>`
     - show data structure managers used
   - `set option show_log_props <normal/brief/long/on/off>`
     - show logical properties used
   - `set option show_parallel <normal/brief/long/on/off>`
     - show parallel query optimization
   - `set option show_histograms <normal/brief/long/on/off>`
     - show histogram processing
   - `set option show_abstract_plan <normal/brief/long/on/off>`
     - show abstract plan details
   - `set option show_search_engine <normal/brief/long/on/off>`
     - show search_engine details
   - `set option show_counters <normal/brief/long/on/off>`
     - show optimization counters
   - `set option show_best_plan <normal/brief/long/on/off>`
     - show best plan details
   - `set option show_code_gen <normal/brief/long/on/off>`
     - show code generation details
   - `set option show_pio_costing <normal/brief/long/on/off>`
     - show physical io estimates
   - `set option show_lio_costing <normal/brief/long/on/off>`
     - show logical io estimates
   - `set option show_elimination <normal/brief/long/on/off>`
     - show partition elimination

Below are some 'set' options whose output is not captured by application tracing. These commands impact query results or query plan.

1. `set rowcount <value>`
2. `set forceplan <on/off>`
3. `set plan optgoal <allrows_mix/allrows_dss>`

The output of all **dbcc** traces (`dbcc traceon(<flag>)`) goes to the trace file, when enabled under application tracing context.
e.g. `dbcc traceon(100)` will display a parse tree for each command.


NOTE: The traced output will always go to the specified trace file and does not require traceflags 3604 or 3605 to be ON. In case these traceflags are turned on, they do not affect the trace output format in any way.

4.1 How to trace SQL Text

Sybase ASE 15.0.2 introduces a new 'set' command that can be used to print SQL Text for *ad-hoc queries, stored procedures, cursors* and *dynamic prepared statements*. This command is very helpful in debugging and tracing applications and should always be used in conjunction with 'set tracefile' command.

```
Syntax: set show_sqltext on/off
```

In application tracing context, the presence of SQL Text in the beginning of each tracing output is much needed, mainly when we trace some other session. Without SQL Text, it would be quite hard to correlate the trace output (showplan, statistics io, etc.) with its corresponding SQL or command.

Additionally this command also prints a timestamp before each SQL Text that it outputs. It prints the current date and time up to a granularity of 10 milliseconds. This fingerprint-like information is quite useful when digging into the command sequence issued by a client application. It especially helps in computing delays between two points of interest during application execution and also allows mapping of application-level events from trace file with server-level events from server log.

 NOTE: The "set show_sqltext" command can also be used outside the 'set tracefile' context. In that case, this command will only work for the current session.

Once enabled, this interface prints the following:

- ✓ In case of an *ad-hoc query,* prints SQL statement text preceded by timestamp.
  - For Example:

  "**`select count(*) from sysobjects`**" prints:

```
2007/06/20 22:51:35.46
SQL Text: select count(*) from sysobjects
```

- ✓ In case of a *stored procedure,* prints the enclosing procedure name and line number for SQL statements and the SQL text itself for exec-immediate statements. The timestamp is printed only once at the beginning.
  - For Example:

  "**`sp_who`**" prints:

```
2007/06/20 22:44:47.67
SQL Text: sp_who
Sproc: sp_who, Line: 0
Sproc: sp_who, Line: 20
...
Sproc: sp_autoformat, Line: 163
...
Sproc: sp_autoformat, Line: 211
...
SQL Text: UPDATE #colinfo_af SET maxlength=(SELECT
isnull(max(char_length(status)),1) FROM #who1result ), autoformat = 1,
mbyte = case when usertype in (24, 25, 34, 35) then 1 else 0 end WHERE
colname='status'
...
Sproc: sp_who, Line: 67
Sproc: sp_who, Line: 69
```

- ✓ In case of *dynamic prepared statements*, prints the statement which creates a light weight stored procedure during *prepare* phase and prints the prepared statement name with statement text during execution. Timestamp is preceded before the SQL text in both cases.
    - For Example:

"EXEC SQL PREPARE select_statement FROM :m_sqlstring;"

prints:

2007/01/10 05:21:12.39
SQL Text: create proc select_statement as SELECT title FROM titles where title_id = 'TC4203'

"**EXEC SQL EXECUTE select_statement INTO :m_title;**"

prints:

2007/01/10 05:21:22.33
Prepared Statement: select_statement, SQL Text: SELECT title FROM titles where title_id = 'TC4203'

- ✓ In case of *cursors*, prints timestamp followed by cursor name, operation (and statement text only for DECLARE and OPEN operations).
    - For Example:

2007/01/10 22:44:54.78
Cursor: typelist, OPEN, SQL Text: SELECT DISTINCT type FROM titles
2007/01/10 22:45:52.96
Cursor: typelist, FETCH
2007/01/10 22:46:27.12
Cursor: typelist, CLOSE

5. Use Cases

Here are some of the use cases showcasing the usability of the feature.

5.1 How to trace a specific client session

This is useful for tracing already running applications connected to the server. All that one needs to know is the SPID of the client session to trace.

spid # 10 (tracer session)	spid # 11 (traced session)	Action taken
`set tracefile "C:\Trace\trace11" for 11 go`	A TPC-E driver tool starting client sessions executing SQL stored procedures and queries.	Enable tracing of session #11.
`set export_options on go`		Any settings made hereon will be alive for entire lifetime of the traced session.
`set show_sqltext on go`		Trace SQL text.
`set showplan on go`		Trace Query Plan.

	TPC-E 'trade_result' transaction in execution.	The Query Text and Plan will go into the file "C:\Trace\trace11".
set tracefile off go		Disable tracing of session #11.
	TPC-E 'trade_result' transaction in execution.	The o/p will no longer be logged.

5.2 How to trace sessions of a specific login

This might be helpful in the following situations:

1. A client application connects to the server via several SQL sessions, identical in processing, and user is interested in debugging any one session. Tracing by login would help in characterizing the behavior of the application in general, since the sessions initiated would have more or less similar workloads (e.g. TPC-C client).
2. In an enterprise-wide ASE server installation, the database administrator may notice that certain logins are causing either stalls or hangs during server execution. In such situations, the administrator may want to enable tracing for the problematic logins to check for unexpected or suspect activity.
3. A user may want to trace his client session in its entirety (from start till end).

spid # 10	Action taken
Login as 'SA/SSO'.	Connect to the server as administrator.

create procedure ltrigg as
  declare @filepath varchar(100)
  select @filepath="trace_for_spid" +
convert(varchar(10),@@spid) + "_at_" +
convert(varchar(10),getdate(),5) + "_" +
convert(varchar(10),getdate(),8) + ".txt"
 	
  set tracefile @filepath
  set export_options on
  set show_sqltext on
  set showplan on
go
	Define a stored proc to enable tracing whenever a session executes it.
	
	<= Generate unique trace file path from current spid, date.

create role trace_role grant set tracing to trace_role grant execute on ltrigg to trace_role go	Define a special role with perms to do tracing and to execute the login trigger.
declare @login varchar(10) select @login="mwagle" exec sp_role "grant", trace_role, @login exec sp_modifylogin @login, "add default role", "trace_role" exec sp_modifylogin @login, "login script", ltrigg go	Grant the special role to the login of interest to be traced and enable the role by default. Finally install the defined stored proc as login trigger for the target login.
quit	Quit the admin session.

A TPC-C driver tool starting 50 client sessions of TPC-C workload with username '@login' above.	Multiple client sessions created with same login.
Each of the 50 clients will fire the typical set and mix of stored procedures and queries in conformance with the TPC Benchmark™C (TPC-C) specification.	The Query Text and Plan for the 50 clients automatically goes into separate trace files under $SYBASE.

Sample trace output:

(trace_for_spid32_at_25-06-07_22:38:04.txt)

```
   2007/06/25 22:38:04.22
   SQL Text: use tpcc

   Sproc: tc_startup, Line: 5

   ...
   Sproc: neworder_local, Line: 0

   QUERY PLAN FOR STATEMENT 1 (at line 0).

       STEP 1
           The type of query is DECLARE.

   ...
   Sproc: neworder_local, Line: 55

   QUERY PLAN FOR STATEMENT 3 (at line 55).

       STEP 1
           The type of query is UPDATE.

           2 operator(s) under root

          |ROOT:EMIT Operator
          |
          |   |UPDATE Operator
          |   |  The update mode is direct.
          |   |
          |   |   |SCAN Operator
          |   |   |  FROM TABLE
          |   |   |  district
          |   |   |  Using Clustered Index.
          |   |   |  Index : d_clu
          |   |   |  Forward Scan.
          |   |   |  Positioning by key.
          |   |   |  Keys are:
          |   |   |    d_w_id ASC
          |   |   |    d_id ASC
          |   |   |  Using I/O Size 4 Kbytes for data pages.
          |   |   |  With LRU Buffer Replacement Strategy for data
pages.
          |   |
          |   |  TO TABLE
          |   |  district
          |   |  Using I/O Size 4 Kbytes for data pages.
```

5.3 How to find which sessions are being traced

The stored procedure sp_helpapptrace displays server-wide information about all open tracing sessions. This procedure prints the SPIDs of all sessions being traced, the SPIDs of sessions tracing them and also name of the trace file.

Syntax: `sp_helpapptrace`

Columns in the output:

- `traced_spid`: SPID of the session being traced.
- `tracer_spid`: SPID of the session tracing the `<traced_spid>`. In case the tracer session has exited, it prints 'exited' in place of the SPID.
- `trace_file`: The full path of the file capturing the trace output for the `<traced_spid>`.

Only SA/SSO is allowed to run this stored proc. This procedure is quite useful at the time of trace rebinding method, which is discussed in the next section.

5.4 How to rebind to an existing trace

When a session starts tracing another session and quits without disabling the tracing, then a new session is allowed to rebind with the earlier tracing context using the rebind facility. This allows users to enable tracing and quit the tracer session. Later they can rebind to the traced session and enable/disable 'set' options. Only SA/SSO is allowed to use this facility.

spid # 10	spid # 11	spid # 12	Action taken
`set tracefile "/tmp/trace11" for 11 go`			Enable tracing of session #11.
`set show_sqltext on go`			Trace SQL text.
	`sp_myproc go`		The sproc text will go into the file "`/tmp/trace11`".
`quit`			Quit gracefully or abnormally.
		`sp_helpapptrace go` **`traced_spid tracer_spid trace_file`** **`----------- ------- ---- ----------`** **`11 exited /tmp/trace11`**	Session #12 can query which sessions are being traced.

		`set tracefile for 11` `go`	Rebind to tracing of session # 11.
		`set show_sqltext off` `go`	Stop printing the SQL text.
		`set tracefile off` `go`	Disable tracing of session #11.
	`sp_myproc` `go`		The o/p will no longer be logged.

5.5 How to create trace output file at various paths

The location of the trace output depends on the path specified as parameter to the "set tracefile" command.

Command	Action taken
`set tracefile "myfile"` `go`	The file 'myfile' will be created in $SYBASE/
`set tracefile "mydir/myfile"` `go`	The file 'myfile' will be created in $SYBASE/mydir/
`set tracefile "/tmp/myfile"` `go`	The file 'myfile' will be created as /tmp/myfile

6. Restrictions/Limitations

1. Only 'SA/SSO' (a user with administrative privileges) or users with "set tracing" permission granted will be allowed to use application tracing. Users with "set tracing" permission will only be allowed to trace their own sessions. They will therefore not be allowed to even query tracing information (using sp_helpapptrace) unlike 'SA/SSO'.
2. SET TRACEFILE interface doesn't open an existing file. This protects a malicious user from corrupting the existing critical files like database device files or $SYBASE files present on public file systems like the NFS.
3. Application tracing is allowed only for user tasks and not for system tasks. So an attempt to attach with some system process (e.g. housekeeper) via "set tracefile <file-path> for <system-spid>" will yield an error.
4. One cannot trace more than one session at a time from a given session.
5. One cannot trace the same session from multiple sessions.
6. The file storing the trace output will be closed either when the session being traced quits, or when tracing itself is disabled.
7. While writing the trace output, in case ASE runs out of file space it will close the trace file and turn off the tracing.

7. Conclusion

As part of Sybase's commitment to provide features that add real value to its customers in mission critical environments, Sybase ASE 15.0.2 provides the "Application Tracing" feature. The new feature enables customers to debug their pre-packaged applications and client sessions connected to the ASE database server. The primary goal of the feature is to provide, both users and administrators alike, the capability to x-ray the server execution without interrupting any client-side activities. Users can now attempt to inspect slow or unexpected behavior experienced during application execution. DBAs can do the same across all executing applications. Upgrading from one version to another becomes much easier for DBAs as they can now easily pinpoint any issues with the help of these tracing options.