PL/SQL & SQL – was nicht messbar ist, kann man nicht lenken

Jonas Gassenmeyer, syntegris information solutions GmbH

Beschwerden über langsame Applikationen sind in der Regel undankbar, wenn unklar ist, ob eher das Netzwerk oder die Datenbank selbst die Ursache für Performance-Probleme darstellen. Um solche Unklarheiten zu vermeiden, bietet die Oracle-Datenbank hervorragende Bordmittel, um Laufzeiten und Kenngrößen in PL/SQL und SQL zu beziffern. Der Artikel zeigt grundlegende Funktionsweisen des Hierarchical Profiler und des (Session-)Tracing auf, um den Einstieg ins SQL- und PL/SQL-Performance-Tuning zu finden.

Geschäftsanwendungen, die auf einer Oracle-Datenbank beruhen, sind in der Regel dadurch gekennzeichnet, dass mehr als nur ein simpler SQL*Plus-Client mit einfachen SQL-Statements zum Lesen und Schreiben von Daten involviert ist. PL/SQL-Packages können den Zugriff auf Daten kapseln (APIs); üblicherweise läuft die eigentliche Anwendung auf einem Application Server und darin laufende andere Programmiersprachen (PHP, Python etc.) können weitere rechenintensive Schritte enthalten. Schlussendlich muss eine Oberfläche an den Benutzer ausgeliefert werden, etwa über den Browser (HTTPS).

Unabhängig von der System-Architektur kann sich die Suche nach Ursachen für Performance-Probleme schwierig gestalten. Netzwerk-, Applikations- und Datenbank-Spezialisten sollten grundsätzlich eng miteinander arbeiten, um die genauen Ursachen möglichst schnell einzugrenzen. Datenbank-Entwicklern und -Administratoren stehen hervorragende Möglichkeiten zur Verfügung, um PL/SQL und SQL messbar zu machen und somit festzustellen, ob die Ursache in der Datenbank zu finden ist. Die Mess-Ergebnisse lassen sich dann objektiv zur Bewertung des Problems heranziehen.

Es gibt Unmengen von Werkzeugen, die mehr oder weniger ähnliche Messungen ermöglichen und Ergebnisse aufbereiten. Einige sind kostenpflichtig, andere wiederum können komplett quelloffen (Open Source) und kostenlos bezogen werden. Es kann allerdings sein, dass vor der Nutzung eine Installation erforderlich ist, was voraussetzt, dass Performance-Messungen bereits vor einem eigentlichen Problem antizipiert wurden.

Das Gute am Hierarchical Profiler (HPROF) und am Tracing ist, dass diese Tools mit der Installation einer jeden Oracle-Datenbank (lizenzunabhängig) bereitstehen. Angenommen, es gibt zwei Muster von Performance-Beschwerden:

- Immer, wenn ich xy mache, wird es langsam (häufig alle Sessions)
- Warum war xy langsam (häufig spezifische Session)?

Dazu sei verdeutlicht, dass nur wiederholbare Performance-Probleme (der erste Typ) mittels HPROF und Tracing analysiert werden können, weil eine Wiederholung der

Aktion notwendig ist. Das ist wahrscheinlich auch der Grund, weshalb „v$active_session_history" (beziehungsweise „dba_hist_active_sess_history") (ASH) so gerne zur Analyse herangezogen werden. Wie das „history" im Namen schon vermuten lässt, ermöglichen diese Objekte in einer Enterprise Edition (plus Diagnostics und Tuning Pack) eine lizenzkostenpflichtige Möglichkeit, auch vergangenheitsbezogene Analysen (der zweite Typ) durchzuführen.

Was aber kann man sich nun unter dem Hierarchical Profiler und Tracing vorstellen? In beiden Fällen wird bei der Durchführung einer Performance-Messung eine Datei mit der Dateiendung „.trc" (Trace File) auf dem Datenbank-Server erzeugt. Außerdem erfolgt die Messung über einen Start- und Stoppbefehl: Ist der involvierte Code (PL/SQL oder Session) identifiziert, so wird zu Beginn ein Marker gesetzt, dass die Messung beginnen soll. Dann wird der Code ausgeführt und abschließend ein weiteres Kommando zum Stoppen abgesetzt. Fragen, wie zum Beispiel vom Client-Rechner auf die Datei auf dem Server zugegriffen werden kann, wie dessen Inhalt zu verstehen beziehungsweise besser aufzubereiten ist und welche Voraussetzungen überhaupt erfüllt sein müssen, damit das Trace File auch tatsächlich geschrieben werden kann, sind im zweiten Teil des Artikels erklärt.

Messungen mit dem Hierarchical Profiler (PL/SQL)

Ist der zu messende PL/SQL-Code identifiziert, lässt sich der Hierarchical Profiler (HPROF) durch ein „SYS.DBMS"-Package („dbms_hprof") starten und stoppen. Dabei ist keine Re-Kompilierung des eigenen Codes notwendig. Dennoch sind einige vorbereitende Maßnahmen unabdinglich. *Listing 1* zeigt, dass zunächst über ein Directory-Objekt ein Zugriff auf ein Verzeichnis auf dem Datenbank-Server erfolgen muss. Außerdem benötigt das Schema natürlich entsprechende Rechte, das „dbms_hprof"-Package auszuführen. Dann kann es aber schon mit der Messung losgehen.

Listing 2 zeigt, dass das eben angelegte Directory als Parameter in der „start_profiling"-Prozedur mitgegeben wird. Zusätzlich vergeben wir einen Namen, den das spätere Trace File trägt. Vorsicht: Wenn ein File mit dem gleichen Namen existiert, wird Oracle diese Datei ohne Vorwarnung überschreiben. Alle darauffolgenden PL/SQL-Aufrufe werden gemessen, bis das Profiling durch die Prozedur „stop_profiling" beendet ist. Im Beispiel wird die Prozedur „lauf_forrest_lauf" aufgerufen, es wären jedoch auch mehrere Aufrufe möglich.

Obwohl die Doku *(siehe „https://goo.gl/imjkkM")* gute Erklärungen liefert, ist die rohe „.trc"-Datei gewöhnungsbedürftig. Man kann hier bereits von oben nach un-

```
create directory hp_dir as '/tmp/plshprof/results';
grant read, write on directory hp_dir to demo_user;
grant execute on dbms_hprof to demo_user;
```

Listing 1

```
dbms_hprof.start_profiling ('HP_DIR','forrest.trc);
lauf_forrest_lauf;
dbms_hprof.stop_profiling;
```

Listing 2

```
dbms_hprof.analyze(
   location    => 'HP_DIR'
, filename     => 'forrest.trc'
, run_comment  => 'Demo'
);
```

Listing 3

```
with prep as(
   select
       fi.module || '.' || fi.function n
    , pci.subtree_elapsed_time pci_subtree_elapsed_time
   , fi.subtree_elapsed_time fi_subtree_elapsed_time
   , pci.function_elapsed_time pci_function_elapsed_time
   , fi.function_elapsed_time fi_function_elapsed_time
   , pci.calls --wie viele Aufrufe der Routine
   , fi.sql_text --neu in 12.2
   from dbmshp_function_info fi
   left join dbmshp_parent_child_info pci on fi.symbolid = pci.childsymid
and fi.runid = pci.runid
   )
select
   --sorgt fuer Einrueckungen bei Subroutinenaufrufen
       rpad(' ',level, '.') ||level||') '|| n aufruf
   , calls
   --Microsekunden -> Sekunden
   , pci_subtree_elapsed_time/1000000 sek_inkl_subprog
   , fi_subtree_elapsed_time/1000000 sek_inkl_subprog_summiert
   , pci_function_elapsed_time/1000000 sek_atomar
   , fi_function_elapsed_time/1000000 sek_atomar_summiert
   , sql_text
from prep
--hier wird die Hierarchie aufbereitet
connect by parentsymid = prior symbolid
start with parentsymid is null
```

Listing 4

```
select sql_id , sql_text from dbmshp_function_info;
```

Listing 5

ten gehen und Zeiten („P#X“) ermitteln, die einzelne Subroutinen benötigt haben. Die eigentliche Stärke wird allerdings erst erkennbar, wenn die Rohdaten aufbereitet sind. Der Name „hierarchical“ kommt nicht von ungefähr, denn in den Messdaten ist auch festgehalten, wie häufig und vor allem von welchen darüber liegenden Routinen eine Subroutine aufgerufen wurde. Eine solche hierarchische Darstellung vereinfacht die Suche nach dem „dicken Fisch“ – also jenem Unterprogramm, das einen Zeitfresser darstellt.

Messdaten des HPROF auswerten

Um eine solche Voraggregierung und Hierarchisierung der Messdaten zu vereinfachen, kann der Nutzer das Kommandozeilen-Tool „plshprof” verwenden. Dieses ist in jedem Full-Client oder im Datenbank-Server (Datenbank-Installation) vorhanden. Damit wird eine HTML-Seite erzeugt, die in einem vordefinierten Bericht Auskunft über die Laufzeiten gibt *(weitere Infos siehe „https://goo.gl/7k69Bx“)*.

Das setzt nicht nur voraus, dass man mit den vordefinierten Berichten zufrieden ist, sondern auch, dass man direkten Zugriff auf die „.trc“-Datei hat, die ja auf dem Datenbank-Server abgelegt wurde. Ist das nicht der Fall, bleibt man am besten in der Session und ruft eine weitere Prozedur im bereits bekannten Package auf *(siehe Listing 3)*. Das bekannte Directory und die in „start_profiling” definierte „.trc“-Datei sind erneut als Parameter für den Aufruf erforderlich.

Auch dieser Aufruf erfordert eine Vorbereitung, denn es muss ein Skript ausgeführt werden („$ORACLE_HOME/rdbms/admin/dbmshptab.sql“). Im Hintergrund passiert nicht wirklich etwas Sensationelles: Es werden drei Tabellen (und eine Sequenz) im Schema erstellt, die eine Analyse der Messwerte auf SQL-Basis ermöglichen. Hier gab es eine nennenswerte Neuerung in 18c. Zukünftig bleiben dem Entwickler lästige Suchen in Installationsordnern nach dem Skript erspart: Das PL/SQL-Package stellt einen Prozeduraufruf („dbms_hprof.create_tables“) bereit, mit dem das Setup erzeugt werden kann. Vor allem sollte man einen genauen Blick auf die Tabellen „dbmshp_function_info” und „dbmshp_parent_child_info” sowie die Spalten „subtree_elapsed_time” und „function_elapsed_time” werfen. Hier sind die Laufzeiten für eine Routine in Millisekunden aufgeführt.

Wie der Name „Parent Child“ schon erwarten lässt, enthält „dbmshp_parent_child_info” die Aufrufhierarchie. Mit einer geschickten „connect by“-Klausel in SQL *(siehe Listing 4)* lässt sich dann eine Darstellung ermitteln, die zum einen eine isolierte Betrachtung erlaubt, wie viele Zeit in einer Subroutine verbracht wurde („function_elapsed_time”), und zum anderen zeigt, wie viel Zeit inklusive aller bis dahin ausgeführten Routinen („subtree_elapsed_time”) benötigt wurde.

Es soll noch betont sein, dass ab der Datenbank-Version 12.2 auch die „sql_id” eines innerhalb der PL/SQL-Routine ausgeführten SQL-Statements mit angezeigt werden kann *(siehe Listing 5)*. Das leitet dann auch wunderbar zum Session-Tracing über. Denn was bringt dem Oracle-Spezialisten die Info, dass ein SQL langsam läuft, ohne weitere Details darüber zu erhalten, wofür sich der Optimizer bei der Statement-Ausführung entschieden hat? Solche im HPROF fehlenden Details lassen sich durch Tracing ermitteln.

Messungen mittels Session-Tracing (SQL)

Genau wie HPROF wird das Tracing (auch als „SQL Trace“ oder „event 10046“ bekannt) über ein „SYS.DBMS“-Package („dbms_monitor”) ein- und ausgeschaltet und auch hier wird eine „.trc“-Datei auf dem Server abgelegt. Das waren aber auch schon die Gemeinsamkeiten. Nicht nur, dass bereits ein Standard-Directory („select value from v$diag_info where name= ‘Default Trace File’”) existiert – im Gegensatz zum HPROF wird beim Tracing eine ganze Session (nicht nur PL/SQL) gemessen.

Da unter Umständen sehr viele Aktivitäten/Befehlsanweisungen in einer Session ablaufen können, ist es ratsam, vor der Aktivierung einen genauen Plan zu skizzieren, was und wie lange etwas zu sehen sein soll, um dem Performance-Problem auf den Zahn zu fühlen. Was hier in einem schnellen Satz abgehandelt ist, stellt sich in der Praxis häufig als sehr viel

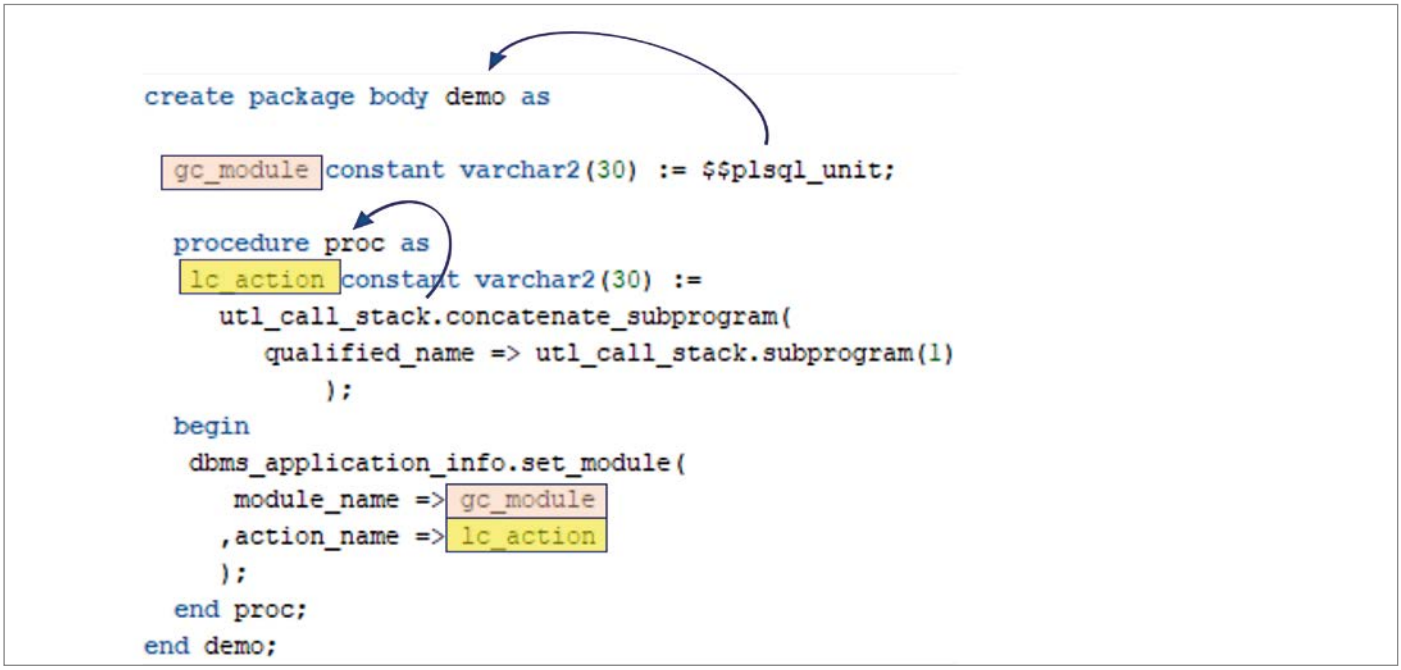

Abbildung 1: So könnte eine Code-Instrumentalisierung aussehen

anspruchsvoller dar. Irgendwie muss erreicht werden, dass die Session identifiziert werden kann, sobald sie auf einer Instanz aktiv wird. Deshalb passt für das Einschalten des Tracing der Spruch „Viele Wege führen nach Rom" wahrscheinlich am besten; denn „dbms_monitor" erlaubt es zum einen, in der selbst aktivierten Session (eine einzige Verbindung zur Datenbank involviert) das Tracing jederzeit zu aktivieren oder zu deaktivieren. Das setzt allerdings voraus, dass der Code irgendwie abgewandelt und eventuell neu kompiliert werden kann, was nicht immer der Fall sein wird. Deswegen können zum anderen verschiedene Kriterien (Modul und/oder Action und/oder Client Identifier oder die Session-ID selbst) zur Identifikation der Session herangezogen werden. Das ist dann gut, wenn man in einer zweiten, unabhängigen Session ein Tracing für die eigentliche Applikationssession steuern möchte. Am besten klappt das, wenn die Anwendung vernünftig instrumentalisiert ist. *Abbildung 1* zeigt, was darunter zu verstehen ist.

Das Package „dbms_application_info" in geschickter Kombination mit „utl_call_stack" (Version 12) und dem Compilerflag „$$plsql_unit" kann wahre Wunder bewirken und lässt sich fast universell in ihren Code kopieren. Unter der Voraussetzung, dass die PL/SQL-Routinen sprechend benannt sind und wiedergeben, was der Code gerade tut, enthält ein unabhängiger Betrachter diese Details ebenfalls. *Listing 6* zeigt, dass „v$session" fast immer ein guter Startpunkt ist, um sich diese Informationen zusammenzureimen; das impliziert auch, dass mehrfache Testausführungen notwendig sein können, bis mit der eigentlichen Messung begonnen wird.

Wer Vollzugriff auf seinen Code hat, sollte am besten schnell mal nachschauen, ob dort schon Session-Informationen an die Oracle-Datenbank herangetragen werden (Code-Instrumentierung). Wenn nicht, sollte man das am besten schleunigst nachpflegen. Wer allerdings nicht die Möglichkeit hat, den problematischen Code zu ändern und so zu instrumentalisieren, dass eine Session fürs Tracing einwandfrei identifiziert werden kann, für den ist Kreativität gefragt. David Kurtz zeigt zum Beispiel unter *„https://goo.gl/*

```
select module, action, client_info from v$session;

MODULE                ACTION          CLIENT_INFO
------------------    ------------    --------------
JDBC Thin Client      do_something    sqlclient_v1
```

Listing 6

```
dbms_monitor.serv_mod_act_trace_enable(
  service_name=>'orcl',
  module_name=>'api',
, action_name=>'running');
```

Listing 7

```
  select payload
  from v$diag_trace_file_contents
  where trace_filename = 'X'
```

Listing 8

```
tkprof session.trc out.txt pdbtrace=demo/pwd@orcl
```

Listing 9

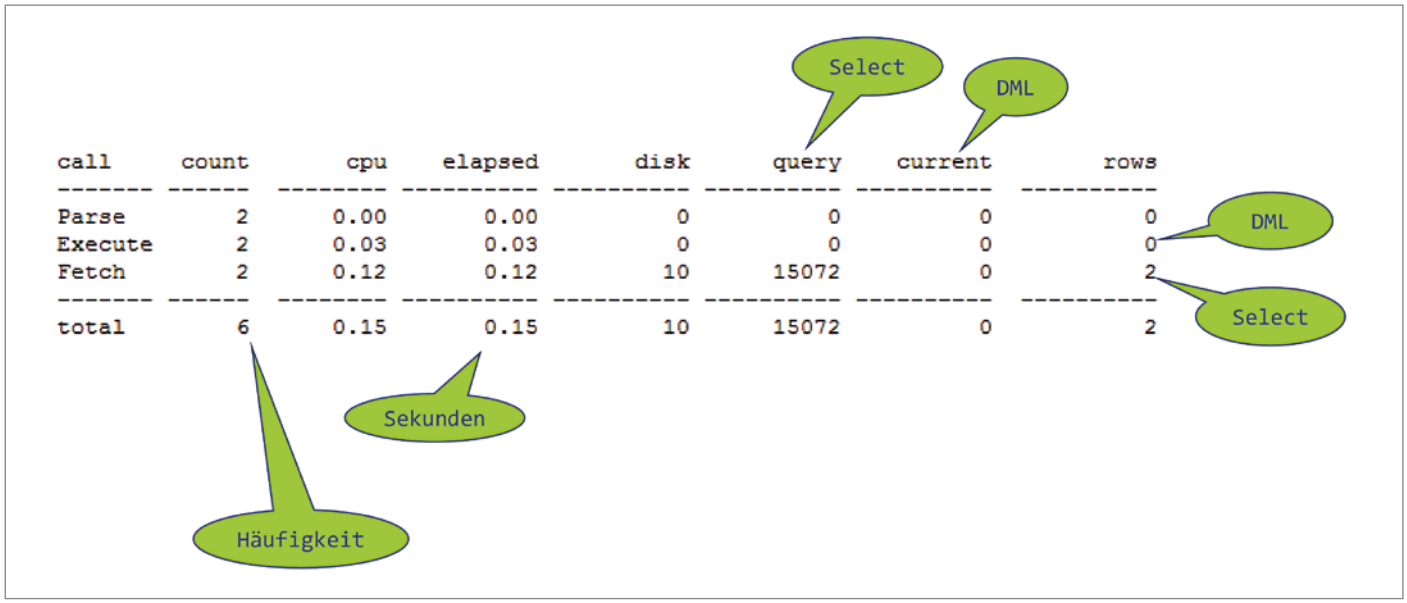

Abbildung 2: Tabellen-Darstellung im „tkprof"-Output

QAW47D" (Folie 27) einen Weg mittels Trigger, um PeopleSoft zu „markieren".

Nachdem einige Möglichkeiten aufgezeigt wurden, wie man Sessions identifizieren kann, zum Abschluss noch, wie die eigentliche Messung durchgeführt wird. *Listing 7* zeigt, wie die Messung in einer von der Applikation unabhängigen Session gestartet werden kann.

Sobald die eigentliche Session der Applikation mit dem entsprechenden Modul „api" und der Action „running" aktiv wird, tut Oracle sein Übriges. Hat man genug Messdaten geschrieben, wird die Prozedur „serv_mod_act_trace_disable" aufgerufen, um die Messung zu stoppen. Oracle hat dann ein „.trc"-File im vordefinierten Trace-Verzeichnis (siehe oben) erstellt. Da Oracle standardmäßig einen Namen für die Datei vergibt, kann es bei dessen Suche Schwierigkeiten geben. Wenn ein „alter session"-Befehl möglich ist, kann man mit „set tracefile_identifier" ein Suffix vergeben, um das Auffinden zu erleichtern. Ab Oracle Version 12.2 wird auch kein Zugriff mehr auf den Server benötigt. Stattdessen lässt sich der Inhalt der „.trc"-Dateien über den Client abfragen *(siehe Listing 8)*.

Messdaten des Tracing auswerten

Für eine Aggregation der Messdaten stellt Oracle ebenfalls ein Kommandozeilen-Tool bereit: „tkprof" sollte definitiv genutzt werden, denn ein Trace File im Rohformat kann zum Zeitfresser werden. Vor 18c benötigt „tkprof" eine lokale „.trc"-Datei; in zukünftigen Versionen kann man mit dem „pdbtrace"-Parameter dafür sorgen, dass sich der Aggregator die Rohdaten selbst über eine angegebene Datenbank-Verbindung ermittelt *(siehe Listing 9)*. Voraussetzung ist wieder, dass man den Namen (im Beispiel „session.trc") der Trace-Datei kennt.

Weitere Parameter für „tkprof" erläutert die Dokumentation genauer *(siehe „https://goo.gl/3QXe5s")*. Typischerweise fällt in der Output-Datei auf, dass dort Tabellen enthalten sind, die für ein ausgeführtes Statement die Randbedingungen zusammenfassen; *Abbildung 2* zeigt eine solche Tabelle.

Die Laufzeit ist in drei Schritte unterteilt: Es wird dargestellt, welche Ressourcen (I/O, CPU etc.) für den semantischen und syntaktischen Check („Parse"), das Ausführen („Execute") und die letztendliche Datenermittlung („Fetch") verwendet wurden. Nennenswert ist noch, dass beim Blick in die Spalten zwischen DML- und Select-Ausführungen unterschieden werden muss. Eine vollständige Analyse würde den Rahmen des Artikels sprengen, abschließend sei jedoch noch erwähnt, dass sich die Untersuchung nicht ausschließlich auf die Zeit der Ausführung konzentrieren sollte. Möglich ist auch, dass ein Statement auf die Ausführung warten musste. Solche Wait-Events lassen sich im Tracing ebenfalls ermitteln, da die „dbms_monitor"-Prozeduren und „tkprof" Parameter bereitstellen. Es empfiehlt sich, zu Beginn einfach ein wenig mit diesen und anderen Parametern herumzuspielen und die Unterschiede in den jeweils erstellten Output-Files zu vergleichen. Das vermittelt eine genaue Idee davon, was aus den Messdaten alles herausgelesen werden kann – es ist eine ganze Menge.

Fazit

Der Artikel vermittelt einen Eindruck, wie PL/SQL und SQL mit Hierarchical Profilers und Session-Tracing messbar gemacht werden können. Die Code-Beispiele haben gezeigt, wie man Messungen durchführt, und es wurden Ansätze der Interpretation erklärt. Jederzeit lässt sich das Thema vertiefen. Dazu kann man Anfragen, Anregungen oder Feedback jederzeit per Mail an den Autor schicken.

Jonas Gassenmeyer
jonas.gassenmeyer@syntegris.de