Reading and Storing Data Directly From Oracle SGA using Pro*C/C code

Miladin Modrakovic, May 2004
rankoni@hotmail.com

There are tools on the market like Precise, Quest or even Oracle that are able to poll information directly from sga without impacting the Oracle kernel. This paper is an attempt to give you a general idea how such tools could be realized. This little program has it all: it polls information straight from SGA and stores it into Oracle tables with a very high sampling frequency (easily more than hundred times per second). Gathering data without impacting Oracle kernel and storing inside tables is the core of expensive tools. The rest I will call it cosmetic (charts, GUI, reports etc.). This article is the sequel to the presentation called Direct Oracle SGA Memory Access written by Kyle Hailey .He is one of the few people, if not the only one who described how to map the Oracle SGA area and published it on his website(**http://oraperf.sourceforge.net/**).

What I did here is an extension to his work and an attempt to give you more lights on this topic .The Pro*C code I wrote could be rewritten in a more efficient way (e.g. using multithreaded programming) but for now it serves it's main purpose.

SGA Direct Access and Pro*C

- High frequency sampling rate.
- Low overhead while gathering data.
- Accessing SGA when querying Oracle is not possible (hangs etc…).
- Storing historical data into Oracle tables for performance analysis.

High Frequency Sampling

The most of the Oracle fixed views disclose aggregated information. Also Oracle kernel statistics is not updated frequently enough. Some of the columns are updated every couple of seconds. In order to get valuable diagnostic data you need to a query database hundreds of times per second. The test shows that using SQL queries is very difficult to grab data more than ten times per second. Even then, you will burn CPU and encounter measurement intrusion effect.

Using SGA direct access method I was easily able to poll and store data more than hundred times per second. The average polling rate on my testing machine was between 130 and 150 times per second. Good enough.

Low Overhead While Gathering Data

Queering Oracle fixed views to gather system statistics imposes measurement intrusion effect upon the Oracle kernel. The method described here samples from the Oracle SGA

and operating system memory and it does not impose any overhead on Oracle kernel. There is a low overhead on the machine itself.

Storing Historical Data

Because Pro*C stores data into Oracle tables it is easy to extract trend information. Also trend data can be plotted and forecasts can be made of future values based on the historical data. You can store Oracle tables inside monitoring or some other Oracle instance. In the case that monitoring instance is not accessible you will still be able to poll and store valuable collecting data.

Pro*C

You need SQL to access and manipulate Oracle data. If the job requires the procedural processing power of C we use embedded SQL The Oracle Pro*C precompiler enables the user to embed SQL statements in a high-level source program.

There are many different ways to compile Pro*C code. I have used one below:

make -f demo_proc.mk filename

If you receive error ' is up to date' during compilation use option below

make -f demo_proc.mk build EXE=sample1 OBJS=sample1.o

where **demo_proc.mk** is a template on which to base your makefile. It's located in $ORACLE_HOME/precomp/demo/proc directory.

Shared Memory

Oracle Shared Memory is a piece of memory sharable by all processes within instance. A process creates a shared memory segment using shmget(). Each process attach to a fix address**.** The original owner of a shared memory segment can assign ownership to another user with shmctl(). It can also revoke this assignment. Other processes with proper permission can perform various control functions on the shared memory segment using shmctl(). Once created, a shared segment can be attached to a process address space using shmat().A shared memory segment is described by a control structure with a unique ID that points to an area of physical memory. The segment identifier is called the shmid.

Accessing Shared Memory Segments

A shared memory segment can be addressed by its key: a 32 bit integer or its shared memory id: an integer assigned by system.

There are four primitive operations on shared memory segments:

1 . int shmget (long key, int nbytes, int flags)

gets nbytes bytes of shared memory and returns a shared memory id:

int shmid;
shmid = shmget(key, NB, 0666 | IPC_CREAT);

The flag IPC_CREAT requests the creation of the segment if it did not exist already.

2 . char *shmat (int shmid, int address, int flags)

attaches the shared memory segment to an address space:

*char *pmem;*
pmem = shmat(shmid, 0, 0);

3. int shmdt(char *pmem)

Detaches the shared memory segment before destroying it.

4. int shmctl (int shmid, int cmd, int arg)

To destroy a shared memory segment use:

shmctl(shmid, 0, IPC_RMID);

The shared memory id could be found using oradebug or sesresv utility.

For detail description of oradebug utility see white paper Oradebug –Undocumented Oracle Utility located at http://www.evdbt.com/Oradebug_Modrakovic.pdf

Finding shared memory id using **sysresv** utility:

set LD_LIBRARY_PATH (e.g. export LD_LIBRARY_PATH=$ORACLE_HOME/lib)

sysresv

```
IPC Resources for ORACLE_SID "demo9i" :
Shared Memory:
ID              KEY
30002           0x21078f90
Semaphores:
ID              KEY
583246          0x0c9c9f50
Oracle Instance alive for sid "demo9i"
```

The Session_wait.pc

The session_wait program is fully functional and for educational and testing purposes only. This program read x$ksuse directly from the SGA and runs the equivalent to

```
select sid,seq#,event,p1,p2,p3,seconds_in_wait
 from v$session_wait
 order by sid;
```

The Code headers: **event.h** is for translating the event# into events names.

Create the file events.h

```
sqlplus "/as sysdba" @events.sql

/* events.sql */
spool events.h
select 'char event[][100]={' from dual;
select '"'||name||'",' from v$event_name;
select ' "" };' from dual;
spool off

/*Find the start ADDR of KSUSECST (V$SESSION_WAIT) */

select addr from x$ksusecst where rownum < 2;

/* Find the Number of Records in the structure */

select count(*) from x$ksusecst;

/* create function to_dec */

create or replace function to_dec
( p_str in varchar2,
  p_from_base in number default 16 ) return number
is
        l_num   number default 0;
        l_hex   varchar2(16) default '0123456789ABCDEF';
begin
        for i in 1 .. length(p_str) loop
                l_num := l_num * p_from_base + instr(l_hex,upper(substr(p_str,i,1)))-1;
        end loop;
        return l_num;
end to_dec;

/* Find size  in bytes of a row  in KSUSECST */

select ((to_dec(e.addr)-to_dec(s.addr))) row_size
from   (select addr from x$ksusecst where rownum < 2) s,
(select max(addr) addr from x$ksusecst where rownum < 3) e;

/*Get offset of all fields */

select c.kqfconam field_name,
c.kqfcooff offset,
c.kqfcosiz sz
from x$kqfco c,x$kqfta t
```

```
where t.indx = c.kqfcotab
and t.kqftanam='X$KSUSECST'
order by
offset;
```

Set Oracle environment:

Set ORACLE_HOME and ORACLE_SID – for monitoring database.

```
CREATE TABLE sga$.session_wait
(
  sid                   number  (6),
  seq                   number          (8),
  event                 varchar2        (40),
  p1                    varchar2        (10),
  p2                    varchar2        (10),
  p3                    varchar2        (20),
  seconds_in_wait       number (14)
)
tablespace &&tablespace_name;
```

Compile code

make -f demo_proc.mk build EXE=session_wait OBJS=session_wait.o

Runtime Oracle environment:

Set ORACLE_HOME and ORACLE_SID - database where sga$session_wait table resides. Note: Data could be stored in monitoring instance itself or inside some other instance. If storing data in different instance than monitoring you need to create sga$.session_wait table inside that instance.

Run program

session_wait sgid
where sgid is shared memory id and could be found using oradebug or sesresv utility.

```
/*
    Script:             session_wait.pc
    Author:             Miladin Modrakovic
    Dated:              May 2004            */

#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sqlca.h>
#include <errno.h>
#include "events.h"
#include <string.h>
#define ID_len 20
```

```
/* SGA BASE ADDRESS */
#define SGA_BASE                0x80000000

/* START ADDR of KSUSECST(V$SESSION_WAIT) */
#define KSUSECST_ADDR       0x63D4BED7

/* NUMBER of ROWS/RECORDS in  KSUSECST */
#define SESSIONS                218

/* SIZE in BYTES of a ROW in KSUSECST */
#define RECORD_SZ               2142

#define KSUSSSEQ 2196     /* sequence # */
#define KSUSSOPC 2198     /* event # */
#define KSUSSP1R 2200     /* p1 */
#define KSUSSP2R 2204     /* p2 */
#define KSUSSP3R 2208     /* p3 */
#define KSUSELTM 2924

void *sga_attach (void *addr, int shmid)
{

    if ( addr != 0 ) addr=(void *)shmdt(addr);
    addr=(void *)shmat(shmid,(void *)SGA_BASE,SHM_RDONLY);
    if (addr  == (void *)-1) {
       printf("shmat: error attatching to SGA\n");
       exit();
    }
    return addr;
}

EXEC SQL BEGIN DECLARE SECTION;

varchar myID[ID_len], mypasswd[ID_len];  /* Database Logon */
varchar evn                     [10];
varchar p1r                     [10];
varchar p2r                     [10];
varchar p3r                     [10];
varchar name                    [30];
int  sid                         [4];
int  cpu                        [12];
int  uflg                        [8];
int  stm                         [8];
int  wtm                        [12];
int  frequency;

EXEC SQL END DECLARE SECTION;

/* Declare functions */

void get_username();
void sql_error();
void clear_kb(void);


/*SetUp variables */
```

```
int main(int argc, char **argv)
{
  void  *addr;
  int   shmid[100];
  void  *sga_address;
  int   shmaddr;
  void  *current_addr;
  long  p1r, p2r, p3r;
  unsigned int   cpu,i, tim, sid, uflg, flg, evn, psqlh, sqlh, wtm, ctm, stm, ltm ;
  unsigned int   cur_time = 0;
  int   seq;
  int   seqs[SESSIONS];
  int   cmit_time=0;

            puts("Enter sample rate:");
            scanf("%d", &frequency);
            /* Clear stdin of any extra characters. */
            clear_kb();

            /* Register sql_error() as the error handler. */
            EXEC SQL WHENEVER SQLERROR DO sql_error("Oracle error \n");

            /* get values for username and password */
            get_username(&myID, &mypasswd);

            /* Connect to Oracle. */
            EXEC SQL CONNECT :myID IDENTIFIED BY :mypasswd;

    for (i=0;i<SESSIONS;i++) { seqs[i]=0; }
    if (argc != 2) {
       fprintf(stderr, "Usage: %s shmid \n", *argv);
       exit(1);
    }
    shmid[0]=atoi(argv[1]);
    addr=0;
    addr=sga_attach(addr,shmid[0]);

/* LOOP OVER ALL SESSIONS until CANCEL */
   while (1) {
              /* set current address to beginning of Table */
         current_addr=(void *)KSUSECST_ADDR;
         sleep(frequency);
for ( i=1; i < SESSIONS ; i++ ) {
         seq=*(unsigned short *)((int)current_addr+KSUSSSEQ);
         evn=*(short *)              ((int)current_addr+KSUSSOPC);
         p1r=*(long *)               ((int)current_addr+KSUSSP1R);
         p2r=*(long *)               ((int)current_addr+KSUSSP2R);
         p3r=*(long *)               ((int)current_addr+KSUSSP3R);
         wtm=*(int *)                ((int)current_addr+KSUSELTM-4);
         if ( wtm > cur_time )  cur_time=wtm;
         if (  evn != 0 )    {
            EXEC SQL INSERT INTO sga$.session_wait
                              (sid, seq, event, p1, p2, p3,seconds_in_wait)
                     VALUES
                              (:i, :seq, :event[evn], :p1r, :p2r, :p3r, :cur_time - :wtm);
```

```
          if ((cmit_time++ %100)==0) EXEC SQL COMMIT;
                        }
          current_addr=(void *)((int)current_addr+RECORD_SZ);
        }
       }
      }
void get_username(varchar *username, varchar *userpasswd)
{
char name[ID_len];
char *password;
char myprompt[11]="Password: ";
           printf("Enter username: ");
           scanf("%s",&name);
           strcpy((char*)(username->arr), name);
           username->len = strlen((char*)(username->arr));
           password = (char*)getpass(myprompt);
           strcpy((char*)(userpasswd->arr), password);
           userpasswd->len = strlen((char*)(userpasswd->arr));
}

void sql_error(char *msg)
{
           char err_msg[512];
           int buff_len, msg_len;

           /* Avoid infinitely loop due to the ROLLBACK error*/
           EXEC SQL WHENEVER SQLERROR CONTINUE;

           printf("\n%s\n",msg);
           /* Call sqlglm() to get the complete text of the error message */
           buff_len = sizeof(err_msg);
           sqlglm(err_msg, &buff_len, &msg_len);
           printf("%.*s\n", msg_len, err_msg);

           /* Rollback the stuff */
           EXEC SQL ROLLBACK RELEASE;
           exit(1);
}

void clear_kb(void)
/* Clears stdin of any waiting characters. */
{
   char junk[80];
   gets(junk);
}
```

Viewing collected statistics

Collected information is stored into sga$.session_wait table. The actual load depends of sampling rate, number of sessions etc .You can write your own queries to poll useful information or use some graphical tools.

The simple output look like this :

SID	SEQ	EVENT	P1	P2	P3	SECONDS_IN_WAIT
1	5464	pmon timer	300	0	0	0
2	53891	rdbms ipc message	300	0	0	0
3	62545	rdbms ipc message	149	0	0	0
4	33246	rdbms ipc message	300	0	0	4
5	5220	smon timer	300	0	0	536532
6	859	rdbms ipc message	180000	0	0	1573378
7	4280	SQL*Net message from client	1413697536	1	0	17
12	25	SQL*Net message from client	1650815232	1	0	0

Conclusion

Hopefully this work will inspire and shed some lights on this unknown topic.

No liability for the contents of this document can be accepted. Use the concepts, examples and other content at your own risk. As this is a first version, there may be errors and inaccuracies that may of course be damaging to your system. Proceed with caution, and although this is highly unlikely, the author does not take any responsibility for that.

References:

Direct Oracle SGA Memory Access Kyle Hailey http://oraperf.sourceforge.net/

Programming in C UNIX System Calls and Subroutines using C, IPC:Shared Memory **http://www.cs.cf.ac.uk/Dave/C/**

JEHAN-FRANÇOIS PÂRIS **http://www2.cs.uh.edu/~paris/4330/shared.htm**

Oracle Corporation - Metal Link Note: 1083994.6 Compiling Pro*C Program