Operating System Integrated Tracer for Database Management System in Big Data Processing

KEIKO TANIGAWA[†1] KAZUYA HISAKI[†1]
KEISUKE HATASAKI[†1]

Abstract: Big data processing has become more important recently. Various data I/O processing acceleration technologies have been developed for it, for example, many-core CPUs, large capacity memory, flash storage, and parallel processing. However, the current OS is designed to run various applications on commodity hardware, and the OS can be a bottleneck in big data processing. Therefore, in big data processing on high performance hardware, an SQL bottleneck analysis needs to drill down into the OS layer to confirm which is the cause of bottleneck, the SQL or the OS. This paper proposes an OS integrated trace tool which analyzes OS functions related to the behavior of SQL data processing. The results of the experiments show that it can extract the kernel functions from SQL and the OS trace within a practical time.

Keywords: operating system, database management system, trace, bottleneck analysis

1. Introduction

Recent improvements in computing hardware have made it possible to handle a large amount of data. A server has over a hundred of cores because the frequency of its single processor has peaked. The capacity of the memory can be increased by lamination. Storage has a performance of more than a million inputs/outputs per second (IOPS), such as the non-volatile memory express (NVMe) solid state drive (SSD). The I/O interface has been changed from serial attached small computer system interface (SAS) / serial advanced technology attachment (SATA) to peripheral component interconnect-express (PCIe) for SSDs that process with low latency.

Middleware (M/W) has also been changed along with hardware improvement. Many-core servers can handle parallel processing and reduce the processing time. Capacity enlargement and cost reduction of memory enable in-memory DBs that do not depend on disk I/O. For example, Oracle and SAP have multiple types of DB processing such as relational DB (RDB), columnar DB, and in-memory DB. They also support multi-tenant features and handle mixed workloads on a DB server with one OS.

In such hardware advances, big data processing becomes more important. It makes it possible to use new resources that are created by human activity and the Internet of Things in addition to a conventional enterprise system, and it is expected that it will be able to analyze massive data and create new value for social and /or enterprises.

In such a background, the OS can be bottleneck in order to maximize hardware performance. The OS is designed to run various applications; therefore, it cannot always process all applications optimally. Moreover, usually, the OS has been updated after new hardware has been developed, and it is not used up to its maximum performance. For example, a block layer of Linux is designed so that it handles a few thousand IOPS, but recent storage devices have high I/O performances of over a million IOPS. The OS as the execution platform needs to maintain the required performance by eliminating its I/O bottleneck. To that end, we need to understand the behavior of applications, M/W and OSs. We propose an OS integrated tracer that analyzes the OS bottleneck point rapidly from application behavior to improve transaction-optimized OS for high I/O processing applications. It organizes the SQL trace log and OS trace log considering their correlation and extracts their bottleneck points. We experiment with extraction of OS kernel functions related to the bottleneck SQL of TPC-C and TPC-H by using our tracer and determine whether this can be done in a practical amount of time.

2. Issues

The purpose of this paper is to optimize OS behavior for big data processing on high performance hardware. With regards to this point, the following researches are referred to.

2.1 Analysis using Trace

Trace technologies are widely used to grasp the system situation. Linux provides various tracing systems [1, 2] for performance analysis and fault analysis of the OS kernel. They record processes, events, functions, and process times in the kernel land. Kobayashi [3] analyzes scheduling latency using ftrace [2] as an example of kernel trace. M/W also provides tracing features [4, 5] for hardware configuration or application improvement. They record snapshots of transaction data, for example, DBMS records processes of SQL and hardware usage such as CPU process times, number of disk I/O, and disk I/O process times per SQL statement. They are used to configure hardware for DBMS, for example, to grasp reinforcing resources [6].

2.2 Issues

Each component of the IT system such as OS, DBMS, and the application has tracing features. However, bottleneck analysis of

†1 Research & Development Group, Hitachi, Ltd.

the system requires many hands because they are closed within each layer. It is hard to link their trace data, which are huge and in different formats. For example, SQL trace data logged by DBMS contains several thousands of files. Bottleneck SQL data provided by DBMS does not have time data because it is statistic data; therefore, we do not know which SQL is really a bottleneck. The OS kernel function trace also has a huge amount of data.

Conventional bottleneck analysis is as follows.

(a) Calculate statistical information per kernel function from OS trace log while the application is running.
(b) Select some candidate tuning functions.
(c) Tune one of the candidates and re-execute the application.
(d) Examine bottleneck SQLs from DBMS trace log.
(e) When it does not work, repeat (b) and (c).

If it does not work at one analysis, (c) and (d) are repeated and it takes more than a few hours. Therefore, measures that can extract a bottleneck easily and reduce the analysis cycle time are needed.

3. Proposal of OS Integrated Tracer

3.1 Overview of OS Integrated Tracer

With the goal of bottleneck analysis facilitation, our proposed method extracts OS kernel functions related to bottleneck SQL on DB applications by associating SQL trace logs with the kernel trace log. The procedure is as follows (as shown in Fig. 1):

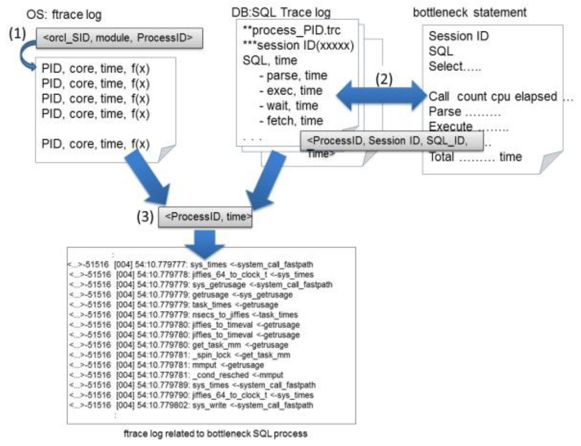
1. OS kernel trace
   Monitor and collect kernel functions related to DBMS processes only in accordance with the naming rules of DBMS, and adjust the time of trace data.
2. DBMS trace
   Create a set of tables, <process_id, session_id, SQL_id, processing_time> by each SQL statement from all of the SQL trace files provided by DBMS, get the SQL statement that takes the longest process time. Then, calculate <process_id, adjusted process time> of the extracted SQL.
3. OS kernel function extraction
   Extract kernel functions from kernel trace log on the basis of <process_id, adjusted process time>.

3.2 Flow of OS Integrated Tracer

We have implemented a prototype of the proposed method as shown in Fig. 2. It consists of two phases: trace data collection and bottleneck point extraction.

In the trace data collection phase, it uses the ftrace provided by OS, which is shown as (4) in Fig. 1, and the SQL trace tool provided by DBMS, which is shown as (6) in Fig. 1. During application runs, the ftrace and SQL trace tool collect data on each behavior and write this into log files. They start with user operation (1) and (6). In the OS trace, data written into the ftrace log file is only regarding DB processes by filtering because all the data of ftrace is very huge and will be a burden. Therefore, it gets the current DB processes periodically and writes them into the DB process list, which contains the process id and process name (2). Then, it creates the DB process filter, which has the process id from the list (3), and decides which data is written into the log file (4) by the filter. In the SQL trace, the SQL trace tool collects SQL trace data into multiple SQL trace files per DB process (6).

In the bottleneck point extraction phase, it runs automatically by one user operation. At first, it shapes SQL trace logs to get SQL. This takes the maximum process time (7) and sorts SQLs by process time per SQL trace file. Next, it extracts bottleneck SQL for the maximum process time of all of the SQLs that are on top of each file (8). Then, it corrects the process time of the ftrace log (5) and bottleneck SQL data (9) and collates them by process id and process time.



```
<...>-51516  [004] 54:10.779777: sys_times <-system_call_fastpath
<...>-51516  [004] 54:10.779778: jiffies_64_to_clock_t <-sys_times
<...>-51516  [004] 54:10.779779: sys_getrusage <-system_call_fastpath
<...>-51516  [004] 54:10.779779: getrusage <-sys_getrusage
<...>-51516  [004] 54:10.779779: task_times <-getrusage
<...>-51516  [004] 54:10.779779: nsecs_to_jiffies <-task_times
<...>-51516  [004] 54:10.779780: jiffies_to_timeval <-getrusage
<...>-51516  [004] 54:10.779780: jiffies_to_timeval <-getrusage
<...>-51516  [004] 54:10.779780: get_task_mm <-getrusage
<...>-51516  [004] 54:10.779781: _spin_lock <-get_task_mm
<...>-51516  [004] 54:10.779781: mmput <-getrusage
<...>-51516  [004] 54:10.779781: _cond_resched <-mmput
<...>-51516  [004] 54:10.779789: sys_times <-system_call_fastpath
<...>-51516  [004] 54:10.779790: jiffies_64_to_clock_t <-sys_times
<...>-51516  [004] 54:10.779802: sys_write <-system_call_fastpath
```

Fig. 1. Overview of OS integrated tracer

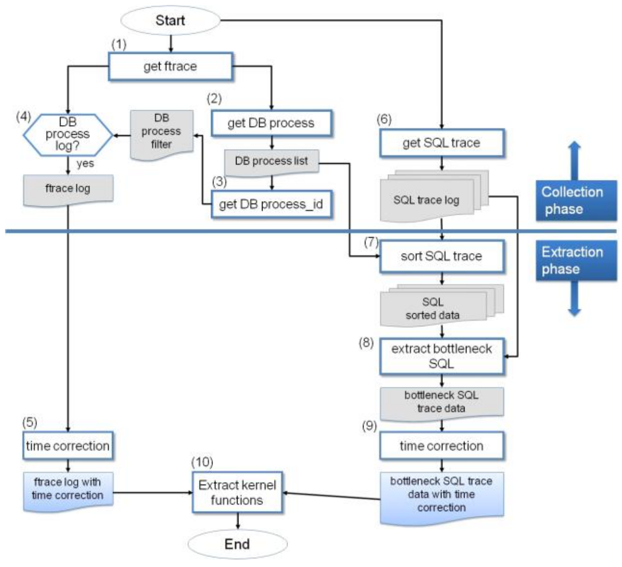


Fig. 2. Flow of OS integrated tracer

4. Experimental Evaluation

4.1 Experimental Evaluation

To evaluate our tracer, we experiment with bottleneck analysis of TPC-C and TPC-H. Table 1 and Table 2 show the experimental environment. A DBMS server assigns 60 virtual CPU cores with NVMe SSD. Clients require TPC-C transactions or TPC-H analysis using a benchmark tool [7].

Table 1. Evaluation platform specifications

Specifications	DBMS server	Client
CPU	Intel Xeon E7-4890, 2.80 GHz	AtomD525, 1.8 GHz
Memory	256 GB	32 GB
Storage	NVMe PCIe SSD 100GB	HDD
OS	CentOS 7, kernel version 3.10.0-123	CentOS 6.5
Software	Oracle DB 12c	TPC-C, TPC-H

Table 2. Environmental parameter

	TPC-C	TPC-H
DB size	30GB	45 GB
CPU cores as DB	60 virtual cores	60 virtual cores
Number of users	600 useres from 200 users per client	60 users from one client

4.2 Experimental Results

1. TPC-C

There are a total of 600 users from three clients requiring TPC-C transactions for 15 minutes. The benchmark tool issues five types of transactions: customer registration, browse products, order products, process orders, and browse orders. It has a similar profile to the TPC-C standard transactions: new order, payment, order status, delivery, and stock level. It can manage their transaction's ratio. In this experiment, the ratio is 1 : 2 : 2 : 1 : 1.

In this case, less than 50% CPU usage is an I/O wait, and the system load is higher than the user one because various lock wait events occur by DBMS, such as DB buffer cache wait, lock contention, index block contention, etc. The system load is higher than the user one; therefore, it could be effective to tune kernel functions or parameters.

The extracted SQL that took the longest processing time is the INSERT statement of order products. In the kernel mode, the following processes are frequently done.

(1) find_busiest_group() is called 120 times, which is the number of cores, and its processing time is 2309 us. It is about 30% of all of the kernel processing time of the SQL. It checks the load of cores to decide whether core migration is necessary.

(2) After the core migration, the NUMA page migration processes are called. They are repeated page assignments and faults, and when enough memory cannot be allocated, core migration occurs. In this experiment, there are four NUMA page migration processes, which takes 711 us.

2. TPC-H

60 users require TPC-H analysis for 15 minutes. The benchmark tool requires five types of transactions: sales rollup by month and channel, sales cube by month and channel, sales moving average, top sales by quarter, and sales per quarter by country. As well as the TPC-C, it has a similar profile to the TPC-H standard transactions, and the transaction's ratio is 1 : 1 : 1 : 1 : 1 in this experiment.

Unlike TPC-C, I/O wait hardly ever occurs except when the benchmark starts and the system load is one third or less of the user one because the memory is enough to load all of the data, and its workload type is read-intensive; therefore, it becomes CPU intensive.

The extracted SQL that took the longest processing time is to get customer information ranked within a quarter, and in the kernel mode, the following processes are frequently done.

(1) NUMA page migration processes are called with context switching. They are repeated page assignments and faults, and when it cannot allocate memory, core migration occurs. In this experiment, NUMA page migration processes are always repeated 33 times, which takes 1872 us.

(2) TLB flush kernel function change_protection_range() is called with context switching. Its processing time is short, but the cost increase caused by increasing TLB misses can be a problem rather than the processing time.

5. Discussions

5.1 Effect of our tracer

Our tracer runs automatically after collecting trace logs, therefore it can reduce man-hours.

As described in Section 2.2, the conventional method takes time. In our experiment, the estimated time is 1 hour 22 minutes for one execution. If it does not work in one execution, e.g. if there are 5 candidates, it takes more than 5 hours. By contrast, our tracer takes 17 minutes 12 seconds including application execution time because it can process b), c), and d) at once. It not only reduces bottleneck extraction time, but also extracts the bottleneck point associated with SQL with kernel processing.

Table 3. Generated data of TPC-C

Application execution time	15 min.
OS ftrace file size	700 MB
Number of SQL trace files	13289
Totaling SQL trace file size	29 GB
Number of SQLs	362266

Table 4. Reduction in analysis time

#	Procedure	Processing time	
		Conventional trace	OS integrated trace
a	Execute application	15 min.	15 min.
b	Select bottlenek kernel functions	10 sec.[1]	-
c	Re-execute application	15 min.	-
d	Search bottleneck SQL from all of SQLs	1sec.[2] × 3108SQLs = 51 min. 48 sec.	2 min. 12 sec.

5.2 Improvement of bottleneck point

In the TPC-C, lots of transactions are issued and each transaction is done in a relatively short time; therefore, many I/O operations wait to be executed because there is a high probability that the OS will judge that CPUs are idle, and as a result, the core load balancing process occurs frequently. It seems that core migration results in much kernel processing time being required.

In the TPC-H, as shown in 4.2, the load in the user land is higher than in the system land; therefore, modification of SQLs seems to be more effective than kernel tuning. However, OS trace log shows some kernel functions are called frequently and repeatedly. Especially, NUMA page migration processes take about 80% of all of the kernel processing time. These processes also occur in the TPC-C, and we think that it is useful to configure parameters of these memory management processes with migration.

In both of TPC-C and TPC-H, migration process result in taking much time. Examples of improvement in the OS behavior by kernel parameter tuning related to migration processes are as follows.

1. Core migration
   The trigger function is find_busiest_group(), and it is issued with the number of cores when the kernel decides the CPU is idle. In TPC-C, the tuning parameter sched_migration_cost_ns reduces the checking of the CPU load frequency by 20%.
2. NUMA page migration
   One of the trigger functions is task_numa_work() and memory management functions such as getting an accessible virtual address, checking the page assigned, getting page allocation, TLB flush, etc. are called. In TPC-H, the tuning parameter sched_latency_ns reduces the context switching frequency by 10% and the NUMA page migration processing time related to NUMA page migration by 20%.

6. Related Work

There are many proposals about improvement of OS bottleneck. They can be categorized into four approaches.

1. Memory management
   Oracle and SAP provide customized Linux with memory management for their DBMS[5, 8, 9]. They control memory parameters for high performance devices.
2. Lock control
   For low latency I/O devices such as next-generation NVMs, polling for completion method takes less overhead than interrupt driven [10]. In some researches [11, 12, 13, 14, 15], on a large-scale NUMA system, spinlock is the cause of cache line contention, especially whenever the number of sockets or cores increases, and the performance decreases remarkably.
3. Context switch control
   [16] reports an improvement of load-balancing feature on multi-core embedded systems. It proposes a task migration method with a load-balancing operation zone by CPU usage and decides whether tasks on each zone can be migrated.
   [17, 18, 19] 's methods are that I/O processing on kernel land is offloaded to user land. They reduce kernel processing time by skipping I/O scheduling and context switching because applications can access virtual I/O devices directly.
   [20] proposes that user programs run on kernel land. It forks child processes on the kernel land, therefore context switching does not occur.
4. Multi-queue block layer improvement
   [21] reports on design of an I/O mechanism to allocate a multi-stage queue in the block layer for improved scalability of the layer. The multi-stage queue is assigned to every CPU core or every socket, and it consists of a software staging queue that processes staging and scheduling as the usual OS and a hardware dispatch queue that processes the ordering guarantee between the software staging queue and multi-queued driver.

7. Conclusions

We developed an OS integrated tracer for application bottleneck analysis to widen the range of the OS layer. The tracer extracts kernel functions related to bottleneck SQL by organizing the correlation between them easily. The tracer reduces OS kernel function extraction time, and thus we expect that users can optimize OS behavior related to application workloads within a practical time. It can help to make system tuning easy and speedy. The tracer will be enhanced for various DBMS and various types of hardware, for example, by plug-in configuration data such as process naming rules, trace file creating methods, hardware architecture, and requesting data to be collected. It can also be a plug in for trace tools and analysis engines. Moreover, it is useful to create the tuning list of the OS parameters and that of the DBMS for bottleneck points.

Reference

[1]Linux Kernel Trace Systems, http://elinux.org/Kernel_Trace_Systems
[2]Linux Ftrace,
http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/trace/ftrace.txt
[3]Yoshitake Kobayashi, "Ineffective and effective way to find out

1 estimate time by search tools
2 number of bottleneck SQLs, "INSERT", by TPC-C experimentation

latency bottlenecks by Ftrace," unpublished (February 2012).
Adrian M. Caulfield, Arup De, Joel Coburn, Todor I. Mollov, Rajesh K. Gupta, and Steven Swanson, "Moneta: A High-performance Storage Array Architecture for Next-generation, Non-volatile Memories," Proceedings of the 43rd International Symposium of Microarchitecture(MICRO) (December 2010).

[4]Oracle AWR, http://docs.oracle.com/cd/E11882_01/server.112/e41573/autostat.htm#PFGRF027

[5]SAP, htttp://help.sap.com/saphelp_nw70/helpdata/en/1f/83113f4bc511d189750000e8322d00/content.htm?frameset=/en/ae/ed0ad513d0074e944879f05ef318d5/frameset.htm&current_toc=/en/7f/6b1538e665c93ee10000009b38f842/plain.htm&node_id=32&show_children=false

[6]Sai Peck Lee and Dzemal Zildzic, "Oracle Database Workload Performance Measurement and Tuning Toolkit," InSITE2006, vol.3 (2006).

[7]Swingbench, http://dominicqiles.com/swingbench.html

[8]Oracle Linux, http://www.oracle.com/technetwork/server-storage/linux/technologies/uek-overview-2043074.html

[9]SUSE Linux, "SUSE Linux Enterprise Server for SAP applications," http://www.suse.com/products/sles-for-sap/features

[10]Jisoo Yang, Dave B. Minturn, and Frank Hady, "When Poll is Better than Interrupt," 10th USENIX Conference on File and Storage Technologies (February 2012).

[11]Silas Boyd-Wickizer, M. Frans Kaashoek, Robert Morris, Nickolai Zeldovich, "Non-scalable locks are dangerous," Ottawa Linux Symposium (July 2012).

[12]Samy Al Bahra, "Nonblocking Algorithms and Scalable Multicore Programming," ACM Queue, Vol. 11, issue 5 (June 2013).

[13]Andi Kleen, "Scaling Existing Lock-based Applications with Lock Elision," ACM Queue, Vol. 57, issue 3 (March 2014).

[14]Davidlohr Bueso, Scott Norton, "An Overview of Kernel Lock Improvements,"- LinuxCon North America (August 2014).

[15]Davidlohr Bueso, "Scalability Techniques for Practical Synchronization Primitives," ACM Queue, Vol. 12, issue 11 (December 2014).

[16]Geunsik Lim, Changwoo Min and Rounglk Eom, "Load-Balancing for Improving User Responsiveness on Multicore Embedded Systems," Ottawa Linux Symposium (July 2012).

[17]Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schupbach, and Akhilesh Singhania, "The multikernel: a new OS architecture for scalable multicore systems," SOSP '09 Proceedings of the ACM SIGOPS 22nd Symposium on OS (October 2009).

[18]Gerd Zellweger, Simon Gerber, Kornilios Kourtis, and Timothy Roscoe, "Decoupling Cores, Kernels and Operating Systems," 11th USENIX Symposium on OSDI (October 2014).

[19]Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe, "Arrakis: The Operating System is the Control Plane," 11th USENIX Symposium on OSDI (October 2014).

[20]Takashi Sato and Yoshikatsu Tada, "Execution System for User Programs in Kernel Mode," Journal of Communication and Computer 10 (October 2013).

[21]Matias Bjorling, Jens Axboe, David Nellans, and Philippe Bonent, "Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems," ACM SYSTOR (June 2013).