A Database System Performance Study with Micro Benchmarks on a Many-core System

Fang XI † Takeshi MISHIMA ‡ and Haruo YOKOTA †

† Department of Computer Science, Graduate School of Information Science and Engineering,
Tokyo Institute of Technology 2-12-1 Ookayama, Meguro, Tokyo, 152-8552 Japan
‡ NTT Information Sharing Platform Laboratories, 3-9-11, Midori-cho, Musashino-shi, Tokyo, 180-8585 Japan
E-mail: † xifang@de.cs.titech.ac.jp, ‡ mishima.takeshi@lab.ntt.co.jp, † yokota@cs.titech.ac.jp

Abstract The upcoming generation of computer hardware brought several new challenges for software engineers. As in the CPU part, now the multi-core processing is main stream, while the future is massively parallel computing performed on many-core processors. This hardware trend motivates a reconsideration of data-management software architecture, because the highly paralleled computing ability may be very challenging for database management system. In order to understand whether the database system can take full advantage of on-chip parallelism, we provide a performance study on a 48-core platform with a set of simple micro benchmarks, as each test stresses the database system in different ways. With an analysis on the performance gain we observed a non scalable problem and examined the bottlenecks in the database system.

Keyword DBMS, Many-core processors, Performance evaluation

1. Introduction

The next generation of computer hardware poses several new challenges for underling software. In the recent years, the main trend of new hardware is the increasingly widespread use of multi-core processors and Solid State Drive (SSD). Unfortunately, much software has had a long way to catch up before it could take advantage of so much new hardware. How to efficiently use the new hardware resources for general software especially database management systems (DBMSs) becomes into hot topic.

Driven by the Moore's Low, computer architecture has entered a new era of multi-core structures. A traditional approach of getting higher performance of processors is to increase the clock speed of them, since a faster CPU can finish one task quickly then switch to the next. However, recently microprocessor manufacturers find it has become increasingly difficult to make CPUs go faster due to size, complexity, clock skews and heat issues. So they continue the performance curve by another route of developing dual core and multi-core processors. That is, putting multi CPUs on a single chip and relying on the parallelism ability to get higher performance gain which brings the computing world into so-called "multi-core era". The multi-core processor is the mainstream now and there will be many-core processors with more cores on one die in the near future.

The increasingly powerful concurrent processing ability of modern CPUs is stressing the scalability of the DBMSs. With the clock speed increasing, one query can be finished within a shorter time. More queries can be finished within a specific length of time which leads to high throughput. In the multi-core area, we can't benefit from the increasing of clock speed anymore, but we have to efficiently utilize the parallelism ability brought by increasing of core number instead. Thus higher emphasis is placed on parallelism ability of DBMSs than ever before. Most DBMSs are designed back to the 1980's, when processors have only single core. These DBMS approaches may result in faster query execution on single core CPUs, but not on multi-core CPUs, because the current parallelism methods are insufficient and of bounded utility [1],[2]. Several prior studies show running hundreds of queries in parallel will result in different contention problems which are called non scalable problems, namely, we can't get higher throughput by increasing the number of concurrent queries [3],[4],[5]. To have a deep understanding of the existing DBMSs' scalability on many-core platforms is crucial for DBMS performance improvement. But most of the existing studies are based on the multi-core CPUs with less than 20 cores, or simulation-based studies [6],[7],[8],[9].

On the other hand large capacity flash memory SSD has gained momentum to boost the I/O performance of the whole system. New flash-memory storage devices offer durable storage with much faster random access speed and some other outstanding features such as lightweight, noise free, shock resistance and no mechanical delay. The utilization of many-core processors will bring dramatic performance improvement to the DBMS, which also bring high performance requirement to the I/O part of the whole

system. SSD is providing more potential for better system performance in the case of intense I/O competition brought by hundreds of concurrently running queries.

In this paper, several experiments are conducted on a 48-core machine to clarify how the DBMS performs on the many-core platform. With a micro benchmark including simple insert operations, a non scalable problem is detected on the SSD equipped system. The reasons of the non scalable problem are analyzed and two potential bottlenecks of the system are picked out. With multi DBMS instances and multi-SSD setting, the two bottlenecks are evicted off the critical section. Achieved better performance and scalability confirms our assumptions about the bottlenecks of the system. Furthermore other bottlenecks inside the DBMS are observed and clarified by a no WAL (Write Ahead Logs) experiment.

2. System Configuration

The DBMSs performance experiments are conducted on a many-core server. In order to have accurate evaluation result, we put the client part on a separated client server. Clients generate transactions and communicate with the storage manager on the server part through network.

2.1 Testing Platform Hardware

All experiments are conducted on a 48 core AMD Opteron system. It has four processor sockets and one 12-core AMD Opteron6174 processor per socket. Each core runs at a clock of 2.2GHZ with 128KB L1 cache (64KB data cache + 64KB instruction cache 2-way associative), a 512KB L2 cache (16-way associative). 12 cores of one processor share a 12MB L3 cache (2*6MB 32-way associative). Each core has a 40-entried TLB. The server that we used to evaluate the benchmark has a 32GB off-chip memory, and four SSD each 100GB, two HDD each 500GB.

Considering putting the client part together with the server part may affect the performance of the server platform, we introduce the client and server separated architecture. Client part is a 16core 2.4GHZ Intel Xeon E5620 CPU, 24G memory. This is powerful enough for our client workload.

2.2 Testing Platform Software

Both the server and client run on the Ubuntu10.10 Linux operating system which is a multi-core supporting OS version and a DBMS of PostgreSQL 9.0.3 [10], which is an open source database management system providing a powerful optimizer and many advanced features.

A basic tuning of the SHMMAX setting is needed in Linux kernel in order to support PostgreSQL. PostgreSQL has a default shared_buffers value of 32MB, but it is said that this parameter should be set at 25% of the system's RAM which allows the system to keep a good performance in parallel with the database server. A very big value of shared_buffers is needed in our experiments, and this setting needs the support of OS. So the SHMMAX value of the Linux kernel on the server part is changed into 30GB in order to support some of the later database settings of shared_buffers.

We evaluate the storage managers with a small suite of micro benchmark. In this paper two aspects of the PostgreSQL are of our interest: the overall throughput value which is calculated by transactions per second (tps) and the scalability of the system which explains how throughput varies with the number of active clients. Ideally the DBMS would be both fast and scalable, but actually different bottlenecks will make storage manager tending to be neither fast nor scalable.

3. Experiments

The record insertion workload of micro benchmarks is introduced in this paper. This record insertion workload repeatedly inserts records into a database table. A very small transaction of only one record insertion operation is used. Each client uses a private table and there is no logical contention between clients. Complex workloads like TPC-H, TPC-C ,which have both insert and select operations, make the analysis of the bottlenecks of the whole system very complex. It's basic to have an understanding about the performance of different parts of the whole system. The select and insert operations separately stress the whole system in totally different way. The insert intensive workload mainly stresses the free space manager, buffer pool, and log manager part inside the storage engine. By using the simple insert related micro benchmark we can simple the analysis and mainly find out bottlenecks inside the DBMS without introducing other affects like memory access latency and so on.

With gradually increasing the number of concurrent client threads, we calculate the overall throughput of all the concurrent clients.

3.1 SSD and HDD based Experiments

One PostgreSQL instance is started on the many-core

server machine, and all the databases and tables are created inside the same PosrgreSQL instance. The tables initially are empty and data is gradually inserted to the tables by executing our micro benchmark. Because the main memory is big enough, the data always fits in the memory, that is, there is not any I/O access of data files to a HDD or an SSD. We set the checkpoint_segments to 256 and check_point_timeout to 1 hour. We also changes the settings of the background writer part which controls the data exchange between memory and disk, which is setting bgwriter_delay to 1000ms, bgwriter_lru-maxpages and bgwriter_lru_multiplier to 0. Therefore, I/O accesses are caused only by writing write ahead logs (WALs).

In the experiment with a HDD, the x_log directory is created on a HDD. On the other hand, in the experiment with an SSD, the x_log directory is assigned to an SSD. This means that WALs of the former experiment are written to a HDD and those of the latter experiment are written to an SSD. Hence, comparing the throughputs of the two experiments, we can identify the difference between a HDD and an SSD. Furthermore the I/O part of the system is evaluated in this experiment.

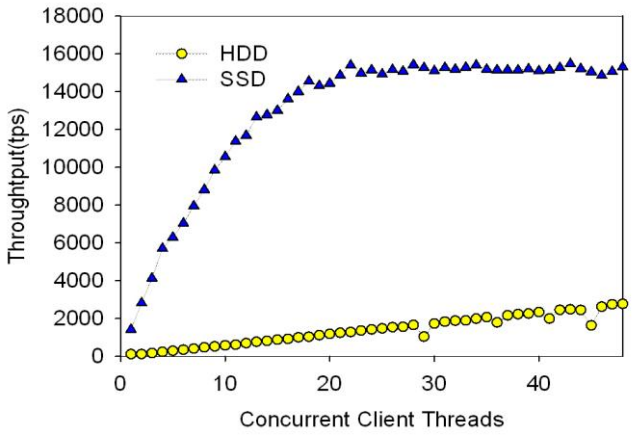

Fig. 1 Throughput of SSD and HDD

Fig. 1 shows the throughputs of the experiments. The concurrent client threads vary along the x-axis with the corresponding throughput for two experiments with different storage devices on the y-axis. The throughput with an SSD is higher than that with a HDD because an SSD offers higher throughput. In other words, the performance of HDD based system is greatly restricted by I/O throughput. Therefore, switching a HDD to an SSD for logging WALs introduces higher throughput.

However, although the throughput with a HDD increases linearly, with the growing number of clients the throughput with an SSD saturates and reaches plateau. This means the experiment with an SSD suffers from some bottlenecks.

In this experiment, there is no logical contention because a different process inserts to a different table. However, all processes share some internal locks for the transaction management information, WAL buffers and so on. On the aspect of hardware, with the throughput increasing there will be more data needed to be flushed to SSD. So we consider the non scalable problem of the SSD based setting is caused by the following two bottlenecks:
(1) the limitation of the I/O bandwidth of an SSD;
(2) the contention during concurrently writing the WAL.
In the following sections, we conducted additional experiments to clarify the two bottlenecks.

3.2 Analysis of the Non Scalable Problem on SSD

In this section, to confirm the two bottlenecks under an SSD based setting as stated in Sect.3.1, we conducted two experiments: an experiment running multi-instance with one SSD and an experiment running multi-instance with multiple SSDs.

3.2.1 SSD based Multi-Instance Experiment

To confirm how the contention of WAL writing locks affects the performance, comparing the throughput with one instance and that with multiple instances is useful. Each PostgreSQL instance has its own WAL management part. In the case of a fixed number of concurrent clients, the more the number of instances increases the more the contention of WAL writing locks in each instance decreases.

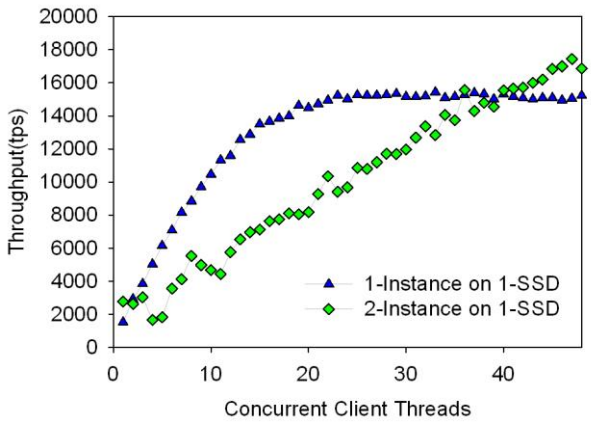

Fig. 2 Throughput of 2-instances on 1-SSD and 1-instance on 1-SSD

A 2-instance test is introduced in this experiment, we

start two PostgreSQL instances with different ports on the server and we evenly assign the client connections onto two different instances.

This setting can reduce the contention in each PostgreSQL instance when writing to WAL. Hence, comparing the throughputs of the 2-instance test with the previous 1-instance test, we can find out whether the contention of writing the WAL has some affect on the system performance and scalability.

The Fig. 2 is the 2-instance test result compared with previous 1-instance SSD based test result. A highest throughput is achieved in this new 2-instance test. This test result confirmed our assumption that contention of writing the WAL is a bottleneck of the system. By introducing multi-instance of PostgreSQL, the WAL writing contention can be removed and a better performance can be achieved.

In the 2-instance test, the overall throughout is lower when the number of concurrent client threads is less than 38. The multi-instance setting makes the I/O write more randomly, and leads to a longer I/O write latency. The throughput is affected by the longer I/O write latency. Because the random write throughput is lower than the sequential write throughput on SSD [11].

3.2.2 Two SSD based Experiment

To confirm how the contention of the I/O bandwidth affects the performance, the experiment with multi SSD setting is needed. By attaching different x_log files of the different PostgreSQL instances to different SSDs, the bandwidth needs for each SSD is reduced and all random writes are eliminated. By reducing the bandwidth need for each SSD, the I/O contention is lightened.

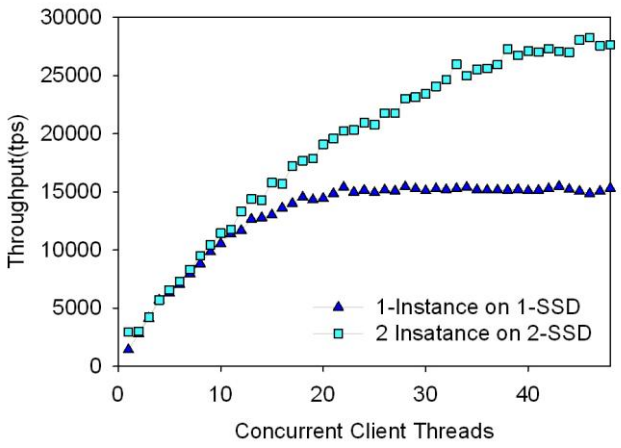

Fig. 3 Throughput of 2-instance on 2-SSD and 1-instance on 1-SSD

Fig. 3 is the test result compared with the 1-instance 1-SSD result. A much higher throughput is achieved with the 2-instance and 2-SSD setting, because this multi-SSD and multi-instance setting can reduce both the two bottlenecks of I/O bandwidth contention and WAL writing contention.

The better throughput and scalability performance support our assumption that the two bottlenecks of I/O contention and WAL writing contention dominate the non scalable problem.

3.3 No WAL based Experiments

In this section, we conduct no WAL setting based experiments and want to detect whether there are other bottlenecks besides the contentions of I/O bandwidth and the WAL writing.

In the previous section we observed two sources of system bottlenecks, and both of them actually are due to the WAL. One bottleneck is caused by contention of writing the WAL buffer, the other one of I/O bandwidth contention is caused by contention of flushing the WAL to SSDs. With giving up the WAL part of the DBMS, both the two bottlenecks mentioned above are removed completely. Therefore with the no WAL setting we can further evaluate the other parts of the system.

3.3.1 Performance without WAL

In order to ignore the WAL part, we need the following settings to PostgreSQL. We set the fsync and synchronous_commit to off, which makes the commit state return and the updated data write into the disk do not to wait the finish of the WAL writing. These settings make the WAL writing and WAL flushing off the critical section.

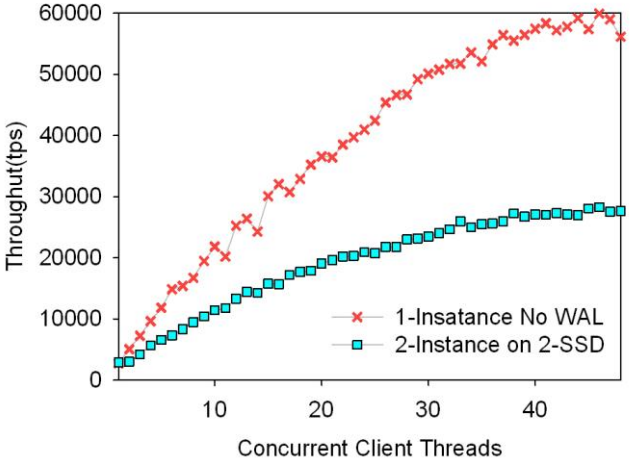

Fig. 4 Throughput of 2-instance on 2-SSD and 1-instance no WAL setting

Repeat the client thread increasing test. The test result compared with WAL log on 2-SSD based test is shown in Fig. 4. The system has a dramatic overall performance improvement with the no WAL setting. But this performance gain is got by sacrificing the guarantee of ACID.

A non scalable problem is observed in this test, when the number of concurrent client threads comes to 38. This conforms there are some other bottlenecks in the system in addition to the two ones mentioned earlier in Sect.3.1. We consider the new bottlenecks come from the contention of spin operations used inside the PostgreSQL. As all the postgres processes share the same memory space which is managed by the buffer pool manager part of PostgreSQL. The DBMS has to use some operations such as spin to manage the utilization of the shard memory space.

In the next sub section we use the multi-instance experiment to clarify the bottlenecks of the spin operation contention.

3.3.2 No WAL based Multi-Instance Experiments

To clarify the affect of the spin operation contention, the multi- instance no WAL experiments are introduced. Different PostgreSQL instances have different buffer pool manager parts. By increasing the concurrent PostgreSQL instances, we can reduce the spin operation contention in each instance.

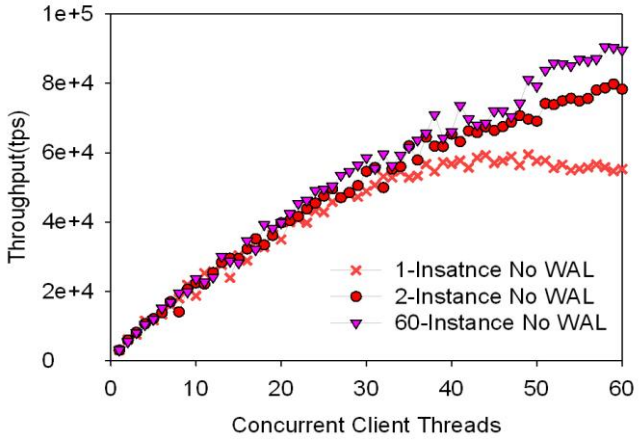

Fig. 5 Multi-instance test result with no WAL setting

The 2-instance and the 60-instance tests are conducted separately. In the 2-instance setting, all the concurrent clients are evenly assigned to 2 instances. As the number of concurrent clients' increases, there will be several clients connected to the same instance. Therefore the 2-insance setting can only reduce the spin operation contention in each instance. In order to totally remove the spin operation contention, we have to make sure different clients connect to different instances. There are at most 60 concurrent clients, so 60-instance is needed correspondingly. Test results compared with the 1-instance no WAL setting result is shown in Fig. 5.

Multi-instance tests achieve higher throughput and a more scalable performance. This result confirmed our assumption about the bottleneck of spin operation contention.

In order to avoid these bottlenecks of spin operation contention, the design and implementation of existing PostgreSQL need to be changed. Shorter critical sections in the log manager and buffer pool manager part are the respected designs. The other possible solution to this non scalable problem is to use several PostgreSQL instances as we did in this experiment.

4. Summary and Future Work

The new hardware of a many-core processor with SSDs transfers the attention of the DBMS researchers from the throughput into the scalability of the system. DBMS must provide scalability in order to achieve full utilization of the parallelism ability of many-core processors. With several performance evaluation experiments of the PostgreSQL on a 48core CPU platform, a non scalable problem is discovered. Several experiments with multi-instance and multi-SSD settings confirm the causes of the non scalable and suggest the possibility of both better scalability and higher throughput. Even though the micro benchmark is used, the test results are very meaningful for other complex workloads like TPC-C which is also insert operation intensive workload.

When DBMS running on new hardware such as many-core processor, the following two kinds of contentions may come to system bottlenecks: shared resource contention of I/O bandwidth and shared data contentions inside DBMS such as log manager and buffer pool manager parts. It is natural that we can get a dramatic performance and scalability increase with giving up WAL part of the DBMS. However, simple omission of logging is not practical because it cannot guarantee ACID. Therefore, we must consider how to reduce the I/O limitation with guarantee of ACID. One conceivable method is using replication to guarantee ACID. The other method verification in our experiment to overcome the bottlenecks of performance and scalability is increasing the DBMS instance number. However, simply increasing instance

number is not practical because we must guarantee the consistency between instances. These challenges are left to our future work.

References

[1] M. Stonebraker, S. Madden, D. Abadi, S. Harizopoulos, N. Hachem and P. Helland, "The End of an Architectural Era (It's Time for a Complete Rewrite)", Proc. of VLDB, pp. 1150-1160, 2007.

[2] J. Cieslewicz and K. A. Ross, "Database Optimizations for Modern Hardware", Proc. of IEEE, pp. 863-878, 2008.

[3] R. Johnson, I. Pandis, N. Hardavellas, A. Ailamaki, and B. Falsafi, "Shore-MT: A Scalable Storage Manager for the Multicore Era", Proc. of EDBT'09, pp. 24-35, 2009.

[4] R. Johnson, I. Pandis and A. Ailamaki, "Critical Sections: Reemerging scalability concerns for database storage engines", Proc. DaMoN, pp. 35-40, 2008.

[5] N. Hardavellas, I. Pandis, R. Johoson, N. G. Mancheril, A. Ailamaki, and B. Falsafi, "Database Servers on Chip Multiprocessors: Limitations and Opportunities", Proc. of CIDR, pp. 79-87, 2007.

[6] Y. Ye, K. A. Ross, N. Vesdapunt, "Scalable Aggregation on Multicore Processors", Proc. of DaMoN, pp. 1-9, 2011.

[7] N. Hardavellas, I. Pandis, R. Johoson, N. G. Mancheril, A. Ailamaki and B. Falsafi, "Database Servers on Chip Multiprocessors: Limitations and Opportunities", Proc. of CIDR, pp. 79-87, 2007.

[8] R. Lee, X. Ding, F. Chen, Q. Lu and X. Zhang, "MCC-DB: Minimizing Cache Conflicts in Multi-core Processors for Databases", In Proceddings of the 35th International Conference of Very Large Data Bases, Lyon, France, August 2009.

[9] J. H. Tseng, H. Yu, S. Nagar, N. Dubey, H. Franke, P. Pattnaik, H. Inoue and T. Nakatani, "Performance Studies of Commercial Workloads on a Multi-core System", Proc. of IEEE Workload Characterization Symposium, pp. 57-65, 2007.

[10] M. Stonebraker and L. A. Rowe, "The Design of Postgres", Proc. of ACM SIGMOD, pp. 340-355, 1986.

[11] H. Kim and S. Ahn, "BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage", Proc. of FAST'08, pp. 239-252, 2008.