A Method for High-speed Transaction Processing on Many-core CPU

Sho Nakazono and Hiroyuki Uchiyama

Abstract

New services have been proposed in fields such as Internet of Things and Fintech (finance & technology). Many more services have been developed by automatically calling the application programming interface of the services among machines or services. Thus, the amount of database processing such as read, update, and delete with a guarantee of correctness in a database is increasing yearly. This trend will probably continue. In this article, we introduce a method for high-speed transaction processing on a many-core CPU (central processing unit) to process these database operations.

Keywords: database, transaction processing, scale-up

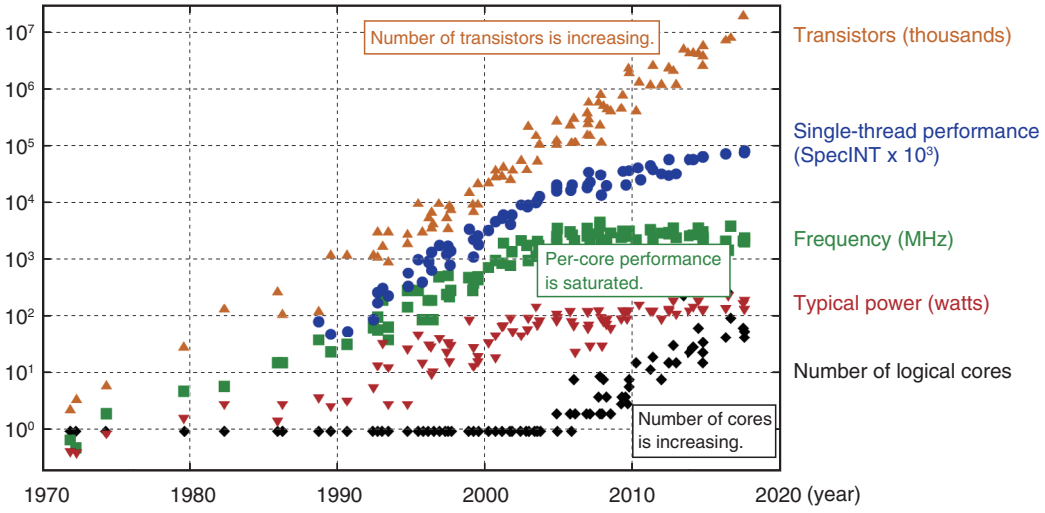
1. A huge amount of database processing

Machine-to-machine communication is featured in services with Internet of Things (IoT) and various web services. For example, IoT devices automatically connect to each other or web services by automatically calling one another's web application programming interface (API) to generate efficient and attractive new services. Thus, massive amounts of database processing we have never experienced are generated every day. The number of transistors has increased under Moore's Law by increasing the number of central processing unit (CPU) cores (**Fig. 1**). However, the current database design does not take into account many-core CPU machines. It is well-known that the processing speed of a database decreases under many-core CPU environments [1]. To obtain sufficient processing throughput of a database, Tu et al. proposed Silo for read-mostly workloads in which Silo scales up the processing speed on many-core CPU environments [2]. However, Silo does not scale for write-heavy workloads.

We need to update a database with huge amounts of sensor data, such as placement, temperature, and status, for hundreds of thousands of items in supply chain management. In database processing, such as cashless payment, micropayment, and small remittance, the amount of updating data will dramatically increase. These processes must be executed at a certain isolation[*1] level of the transaction. When each CPU core processes its tasks in parallel, current methods, such as Silo, guarantee a strong isolation level by processing update operations one by one for the same data items. However, this decreases processing speed because each CPU core waits for the others then continues to process its own tasks.

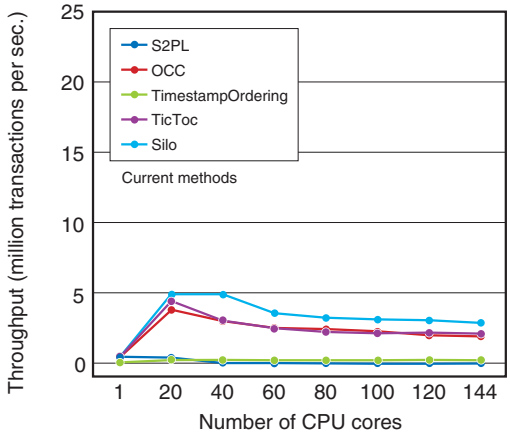
Figure 2 plots the total processing throughput of current methods for increasing the number of CPU cores. After the upper limit of processing speed for 38 cores, as shown on the x-axis, total throughput decreases as the number of CPU cores increases. Therefore, if the processing speed is not sufficient in terms of service requirements, database administrators generally accelerate processing speed by selecting a weak isolation level[*2]. However, this approach involves risks. For example, in a bitcoin exchange, engineers adopted a weak isolation level on their database to increase speed. A cracker group attacked

*1 Isolation: Transaction isolation means that data processed by a transaction are protected or isolated from other concurrent transactions. There are levels of transaction isolation. Serializable is one of the transaction isolation levels and the strictest. Any concurrent execution of a set of serializable transactions are guaranteed to produce the same effect as running them one at a time in a certain order. With our method, one can execute transactions based on Serializable.



Original data up to 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten.
New plot and data collected for 2010-2017 by K. Rupp.
This chart is provided under the permissive 'Creative Commons Attribution 4.0 International Public License'.
Adjusting points are adding comments.
Original data: https://github.com/karlrupp/microprocessor-trend-data

Fig. 1. 42 years of trends in microprocessor data.



OCC: optimistic concurrency control
S2PL: strict two-phase locking

Fig. 2. Write-heavy benchmark results for current methods.

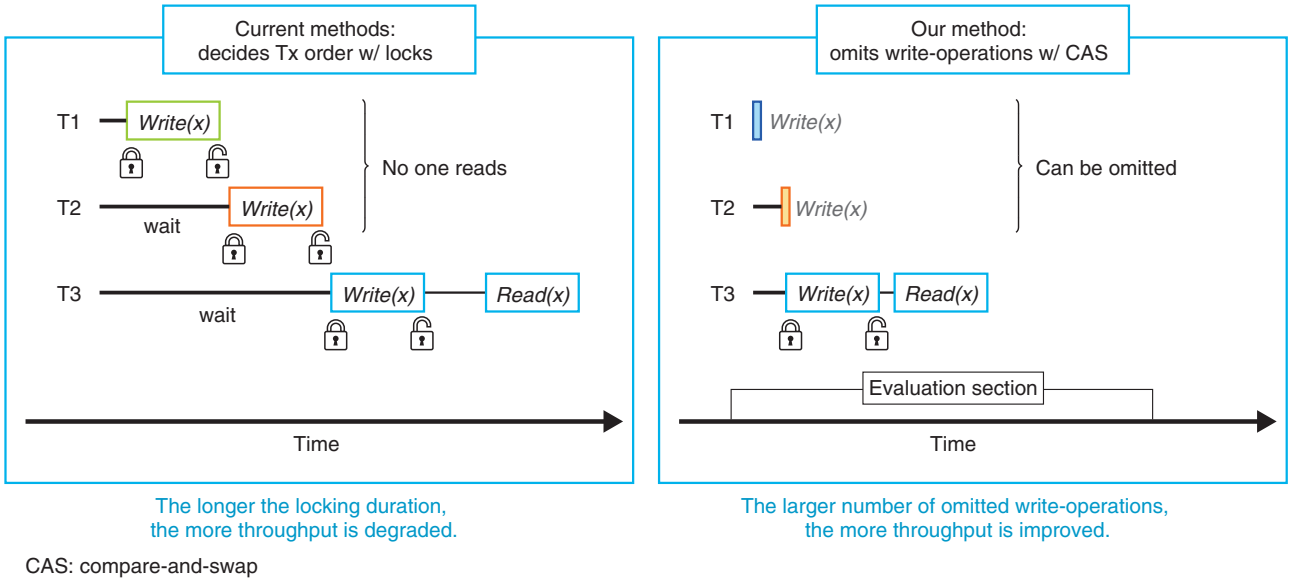
*2 Isolation level: Transaction isolation levels refer to the degree of transaction isolation. The SQL (Structured Query Language) standard defines four levels of transaction isolation. For example, when a database uses the Read Committed isolation level, a transaction can detect different data for the same query, even though they are within a single transaction, if other transactions commit changes after the first read-operations start and before the second read-operations start. However, in the Read Committed isolation level, a database processes transactions faster than in the Serializable isolation level.

the exchange system and withdrew all coins in the exchange illegally. Consequently, the exchange closed. As shown in this example, if database administrators set an incorrect isolation level on their database, they may negatively affect their business and users. Therefore, high-speed processing (especially update operations) of a database under a correct isolation level is an important technical issue to provide services safely and at low cost.

2. Our method

As mentioned above, there are methods of accelerating the processing speed for read-mostly workloads. However, a method for write-heavy workloads has not been proposed. This is because each CPU core must wait if another core accesses the same data item. To address this issue, NTT Software Innovation Center developed a method for drastically accelerating the processing of update operations. Our method is based on the principle that if no one reads an updated data item, the update operation is omittable. With this principle, our method reorders update operations and generates those that no one reads under the safest isolation level, i.e., the strict Serializable level.

Figure 3 shows the difference between current methods and our method. With current methods,



CAS: compare-and-swap

Fig. 3. Difference between current methods and ours.

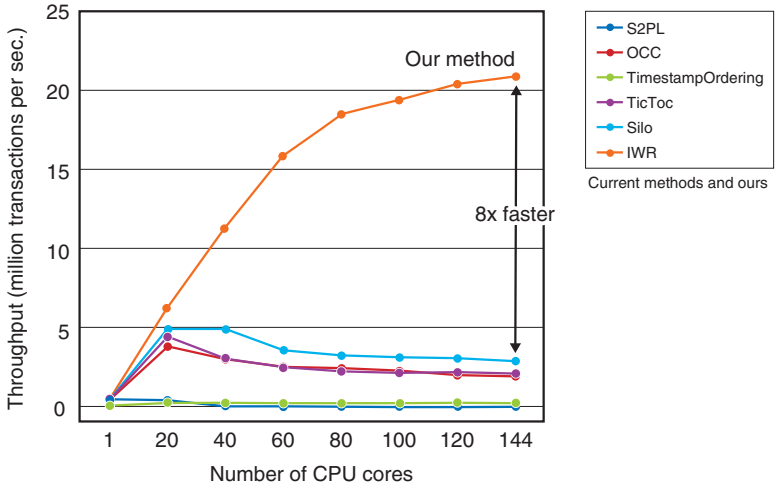


Fig. 4. Write-heavy benchmark results for current methods and ours.

transactions T1, T2, and T3 process update (write) operations for data item x in parallel. We denote Write(x) as a transaction that writes a certain value to x. The x-axis shows the passing of time. With current methods, when each write-operation is processed, the related transaction acquires a lock for a data item. Therefore, T2 can start processing only after T1 processing is finished. In the same way, before T2 finishes processing, T3 cannot start processing. Because no one reads the values updated by T1 and T2, we can omit the write-operations related to T1 and T2.

Our method specifies write-operations, the results of which are not read by anyone, and omits them. Thus, our method can accelerate the processing of update operations and generate omittable write-operations by reordering the read/write-operations of transactions based on the database theory "multi-version view serializability." Our method can process update operations that current methods cannot do efficiently. In **Fig. 4**, we add the results of our method

in Fig. 2. In 144 CPU cores, we can see that our method sufficiently scales up as the number of CPU cores increases. Our method is about 8x faster than Silo, which is the current fastest method, and processes 20 million operations per second. This throughput is the same as 1.7 trillion operations per day [3]. Therefore, our method has sufficient power to process a possible 1 trillion callings of APIs.

3. Future development

We are developing a built-in database based on our method and will prepare its interface as a key-value store. In such a database, users will be able to request read/write-operations simultaneously. By embedding this method into databases for various services, we believe users will be able to develop applications that fulfill their functions on modern many-core hardware. We also plan to develop other interfaces, such as SQL and O/R (object-relational) Mapper, for many users to use.

References

[1] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker, "Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores," Proc. of the VLDB Endowment, Vol. 8, No. 3, pp. 209–220, 2014.

[2] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden, "Speedy Transactions in Multicore In-memory Databases," Proc. of the 24th ACM Symposium on Operating Systems Principles (SOSP'13), pp. 18–32, Farmington, PA, USA, Nov. 2013.

[3] C. Huys, "The API Billionaires Club is about to welcome trillionaire members. But how should you deal with it?", AE Stories, 2016. https://www.ae.be/blog-en/api-billionaires-club-about-to-welcome-trillionaires-members

Sho Nakazono
Research Engineer, Distributed Computing Technology Project, NTT Software Innovation Center.

He received a B.E. in environment and information studies and an M.E. in media and governance from Keio University, Kanagawa in 2014 and 2016. He joined NTT Software Innovation Center in 2016 and is studying concurrent programming and transaction processing.

Hiroyuki Uchiyama
Senior Research Engineer, Distributed Computing Technology Project, NTT Software Innovation Center.

He received a B.E. and M.E. in systems science and applied informatics from Osaka University in 2000 and 2002. He joined NTT Cyber Space Laboratory in 2002 and studied the XML filter engine and distributed stream processing. From 2008 to 2014, he joined the commercial development project of the distributed key-value store and distributed SQL query engine. He is currently studying a high-speed transaction engine, LASOLV™ Computing System, and optimization of hybrid online analytical processing and machine learning.