Performance Report

Environment

All experiments are tested on a NUMA machine Lenovo x3950, with Intel(R) Xeon(R) CPU at 2.3GHz and 1.5 TB SSD, running Ubuntu 14.04, Kernel 3.19. There are 8 sockets and each has 15 cores and 128GB physical DRAM. The sequential write bandwidths of the SSD is more than 700 MB/s.

I only used one socket, with 15 cores to run PG server. Because the cross-socket memory access is very slow in this machine. And clients are executed on the other sockets.

Benchmarks:

The benchmark code is in:
https://github.com/liumx10/pg-bench
I provided three benchmarks. TPCB is a standard OLTP benchmark. ssibench was used in a research paper which studied serailizable transactions. Simple ssibench is transformed from ssibench. It has more conflicts but is not easy to abort transactions. So the conflict list can grow much longer.

Methods

I tried three methods to improve the speed of conflict tracking:

1) Replace linked list with hash table
2) Add an additional linked list to “skip”, just like a two-level skip list;
3) Reduce the contention on the lock “SerializableFinishedListLock” by moving some operations out the protection of the lock. I explained why this operation is right in the email
https://www.postgresql.org/message-id/5b6b452.16851.15cf1ec010e.Coremail.liu-mx15@mails.tsinghua.edu.cn

Performance

Every test is executed 5 times. The row data is in the google sheet
https://docs.google.com/spreadsheets/d/1-N1cnOsq5fYOxyhQw90x9TblkitCEwEB-jy0s2MJxuo/edit?usp=sharing

Here is the summary: (the number means KTPS)

	original code	hash table	skip list	finer lock
simple ssibench	5.572	5.002	5.492	5.962
ssibench	3.784	3.386	3.784	3.966
tpcb	6.19	6.136	6.276	6.316

Comments

1) Skip list is good at searching, **but bad at inserting.** Therefore the function "RWConflictExists" is faster, but the function "SetRWConflict" is slower. In the end the result shows that it has the same performance of the original code.
2) Hash table **also has the inserting problems**. What's more, it is slower when **removing conflicts** from hash table, which made more contentions on the lock "SerializableFinishedListLock" . Therefore it's obviously slower than the original code.
3) "fine-grained lock" has a better performance. But it has a worse performance when I tested it in another machine two weeks ago. Honestly, I don't know why "reducing lock contention" drops the performance sometimes. My guess is in the email : https://www.postgresql.org/message-id/50422608.2673.15d4c5c99c8.Coremail.liu-mx15@mails.tsinghua.edu.cn