Scalable In-Memory Transaction Processing with HTM

Yingjun Wu and Kian-Lee Tan

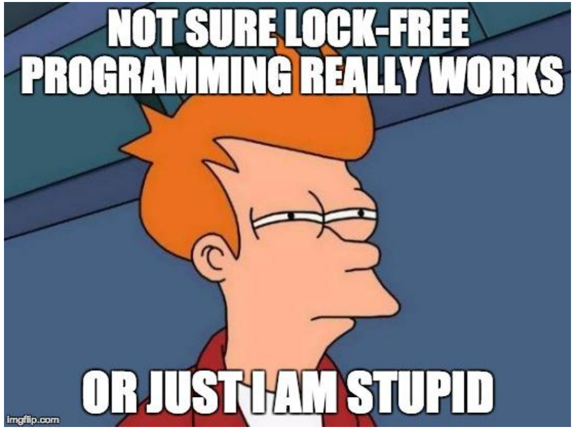
National University of Singapore

HTM simplifies implementing concurrent programs

Lock-free programming

Atomic Buildins

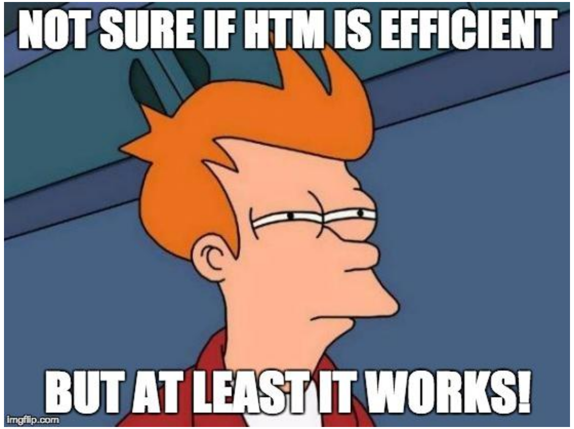
__sync_bool_compare_and_swap(…)
__sync_fetch_and_add(…)
__sync_synchronize(…)
…



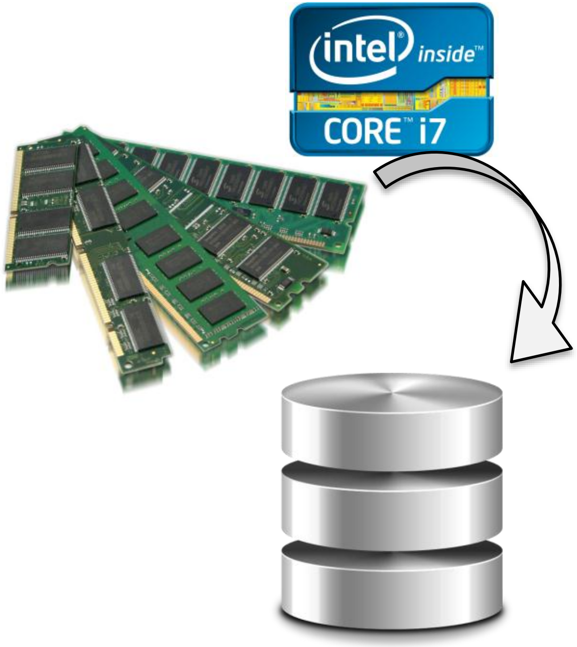
Hardware transactional memory

TSX Instructions

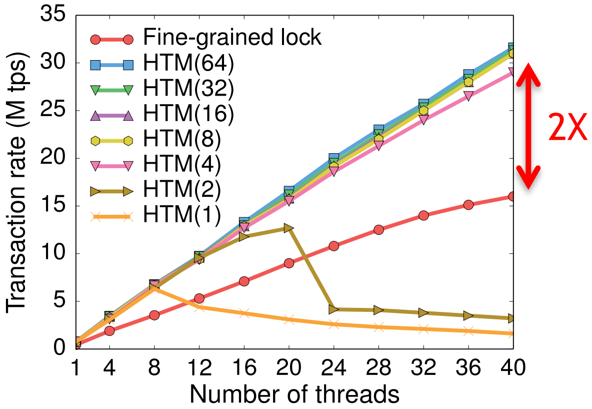
_xbegin()
_xend()
…



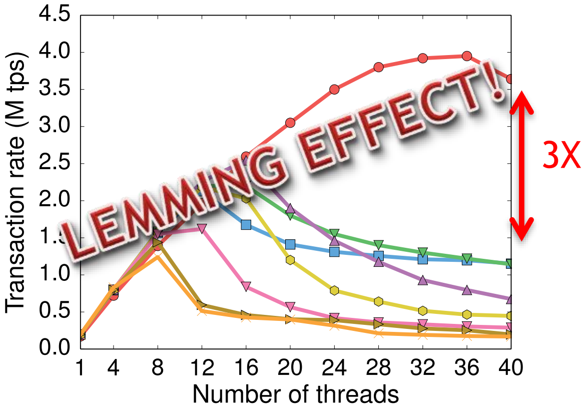
HTM is not a silver bullet for transaction processing



Processing multi-key transactions on a standard hash map.



Low-contention workload.
(theta = 0)

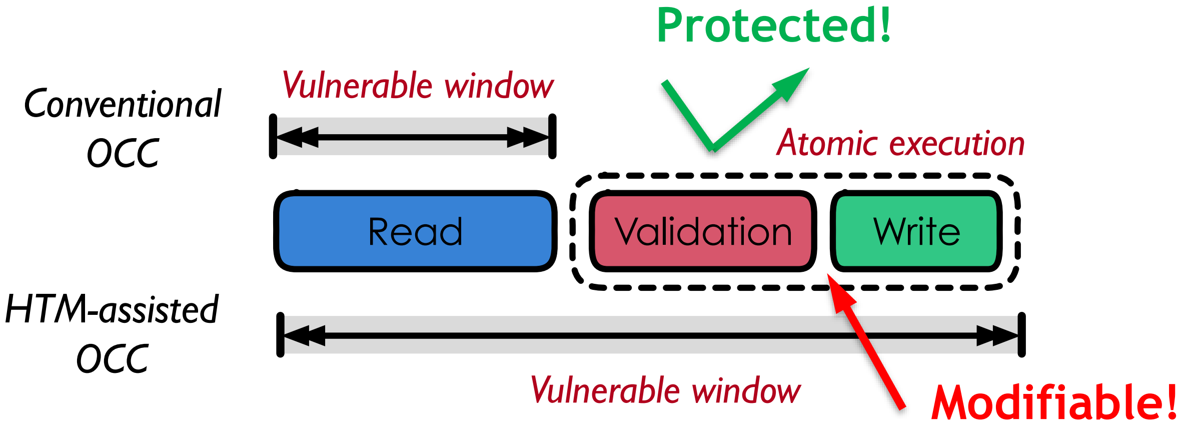


High-contention workload.
(theta = 0.8)

HTM-assisted main-memory database

HTM is not a silver bullet for transaction processing

- Existing works apply HTM to OCC protocol.
  - High database transaction abort rate;
  - High database transaction restart overhead.

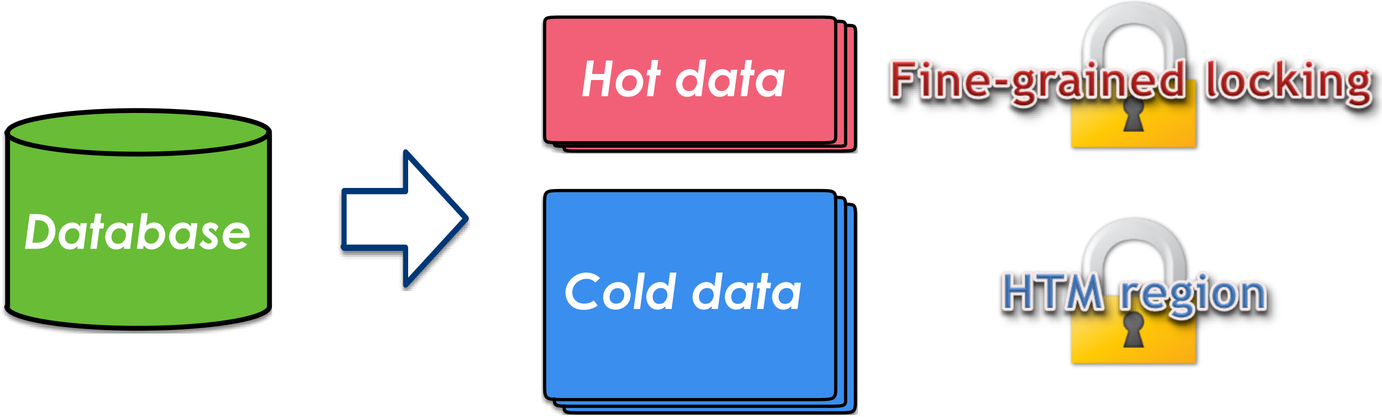


Our proposal: HTCC

- A new HTM-assisted concurrency control protocol that targets at supporting scalable and robust transaction processing even under highly contended workload.
    - Reduce transaction abort rate using a hybrid protocol;
    - Minimize transaction restart overhead using delta restoration.

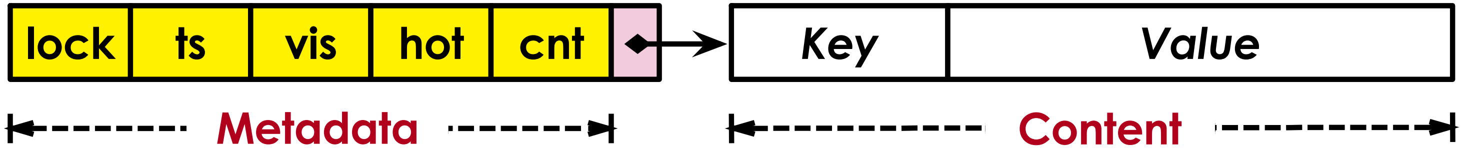
Data Classification

- Split the data into hot and cold records and process them differently.



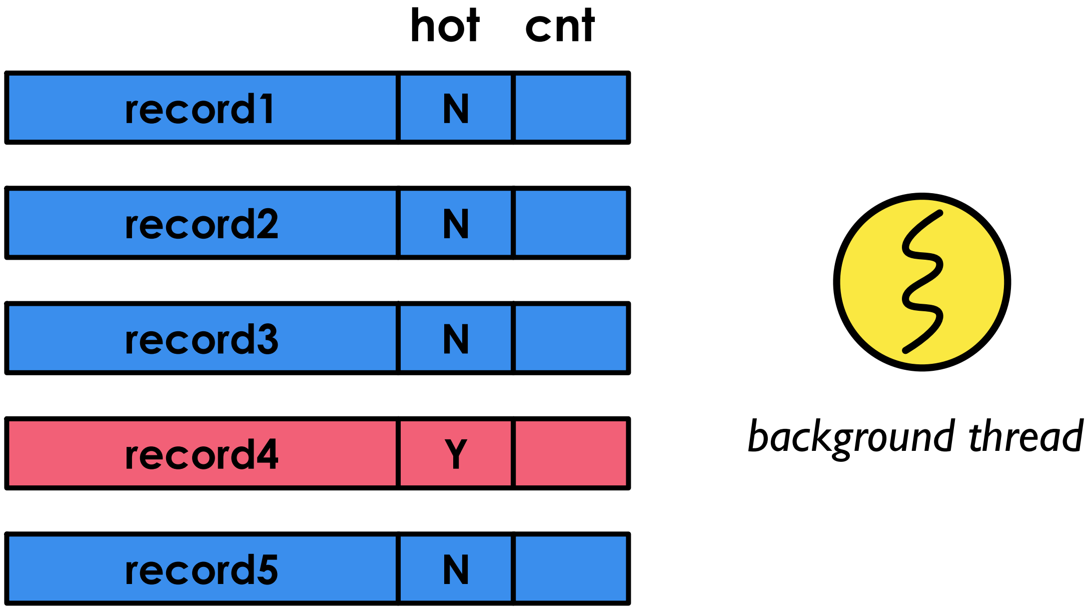
Data Classification

- Data structure.



Data Classification

- Data structure.



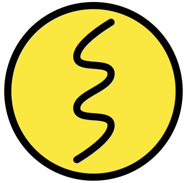
	hot	cnt
record1	N	
record2	N	
record3	N	
record4	Y	
record5	N	

Data Classification

- Data structure.

	hot	cnt
record1	N	97
record2	N	5
record3	N	9
record4	Y	23
record5	N	17

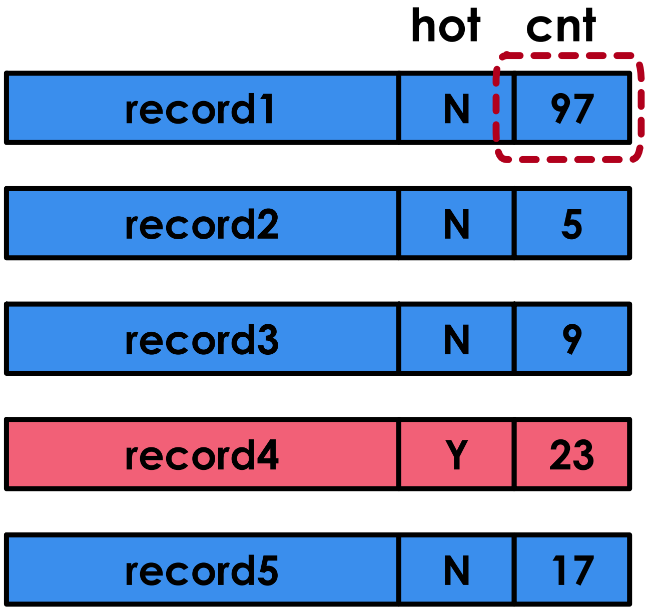
Periodically check abort count.



background thread

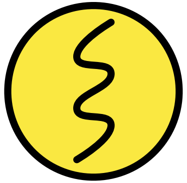
Data Classification

- Data structure.



	hot	cnt
record1	N	97
record2	N	5
record3	N	9
record4	Y	23
record5	N	17

Detect top K hot records.



background thread

Data Classification

- Data structure.

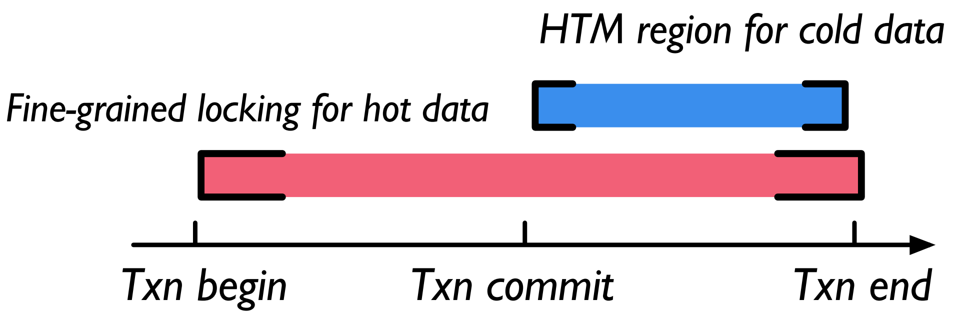
	hot	cnt
record1	Y	97
record2	N	5
record3	N	9
record4	N	23
record5	N	17

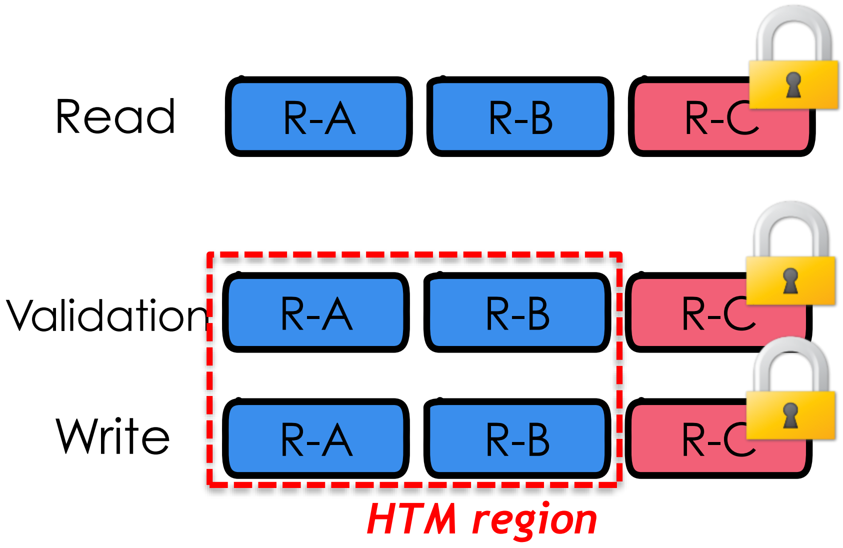
Set the hot flag transactionally!

background thread

Hybrid Protocol

- Transaction phases.



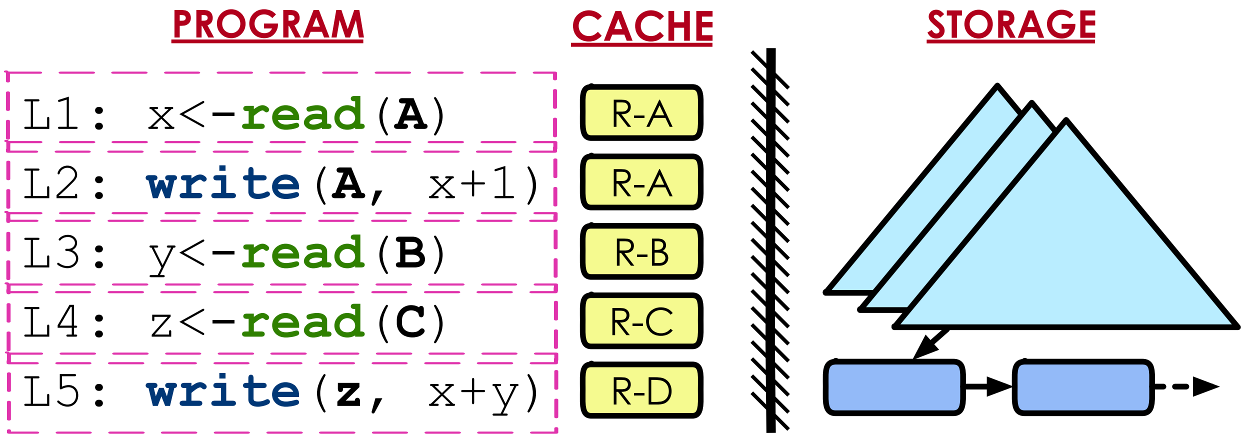


COMMIT!

Fine-grained locking performs well for high-contention workload;
HTM performs well for low-contention workload.

Delta Restoration

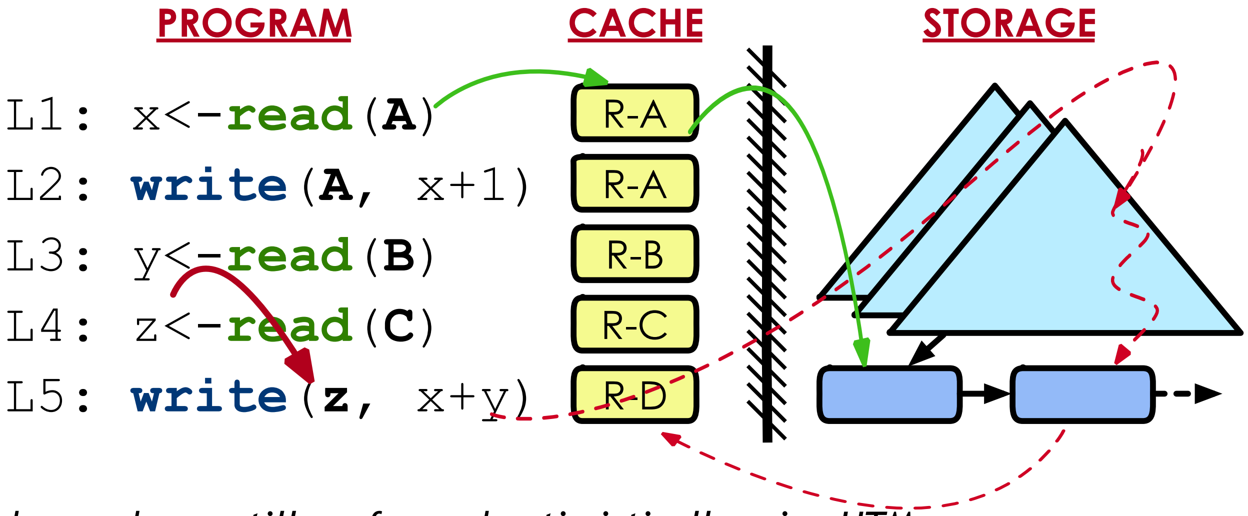
- Workset caching during the *read phase*.



```
L1:  x<-read(A)
L2:  write(A, x+1)
L3:  y<-read(B)
L4:  z<-read(C)
L5:  write(z, x+y)
```

Delta Restoration

- Operation restoration during the *validation phase*.



PROGRAM **CACHE** **STORAGE**

```
L1:  x<-read(A)
L2:  write(A, x+1)
L3:  y<-read(B)
L4:  z<-read(C)
L5:  write(z, x+y)
```

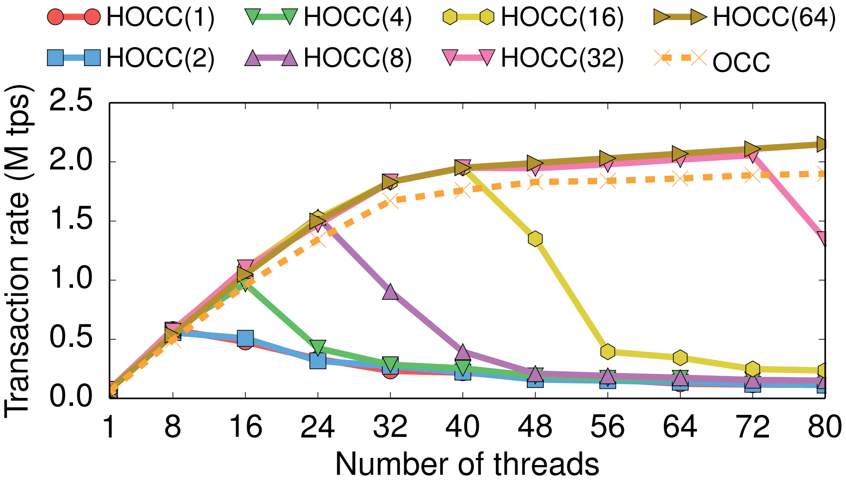
Accesses to cold records are still performed optimistically using HTM;
Deadlock never happens because of HTM's guarantee of atomicity and isolation.

Experiments

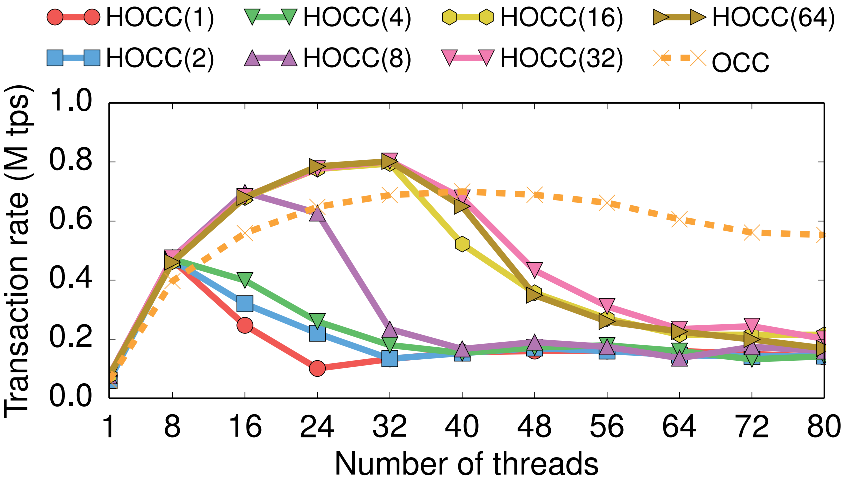
- Intel Xeon Processor E7-4820, 4 sockets, 40 cores.
- We compare with the following protocols:
  - 2PL: classic two-phase locking.
  - OCC: classic optimistic concurrency control.
  - SOCC: Silo’s OCC implementation.
  - HOCC: Existing HTM-assisted OCC.
  - HTO: Existing HTM-assisted timestamp ordering.

Experiments: Bottlenecks

- Database transaction rate with different restart threshold.



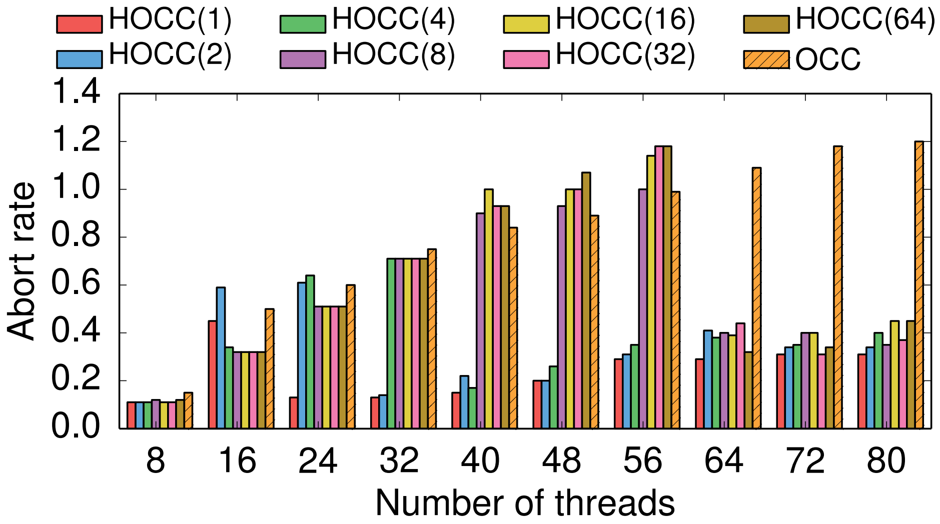
TPC-C: 40 warehouse (low contention).



TPC-C: 4 warehouse (high contention).

Experiments: Bottlenecks

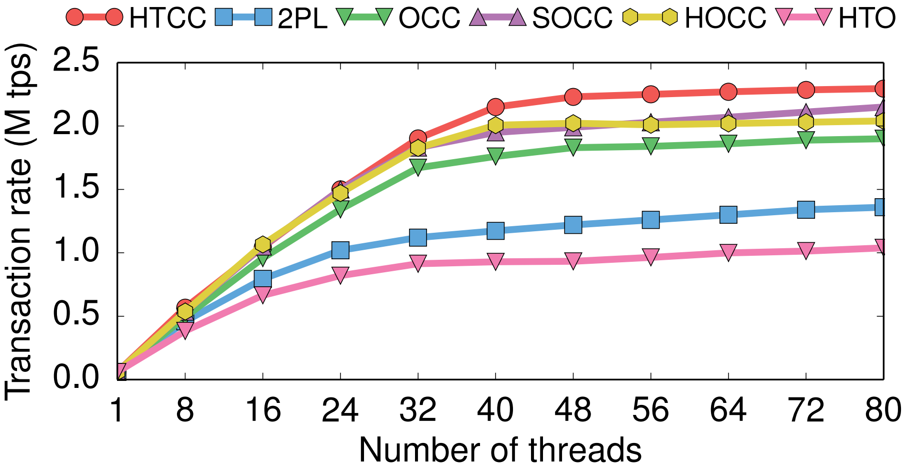
- Database transaction abort rate with different restart threshold.



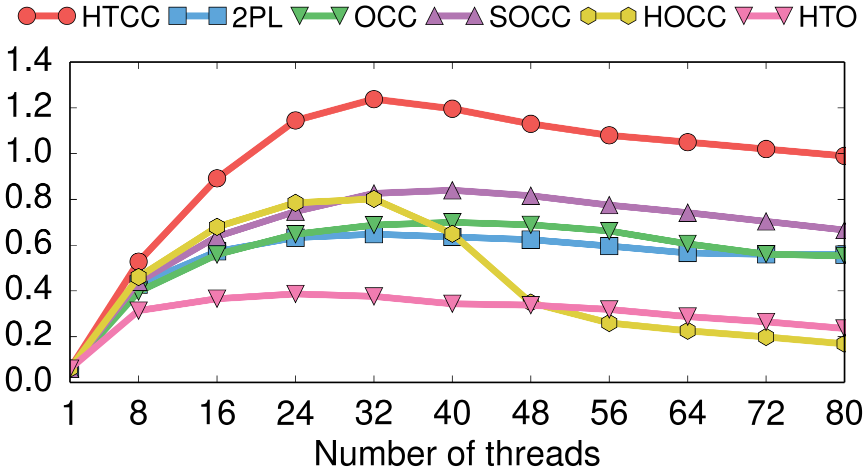
TPC-C: 4 warehouse (high contention).

Experiments: Scalability

- Database transaction rate under different workloads.



TPC-C: 40 warehouse (low contention).



TPC-C: 4 warehouse (high contention).

Conclusion

- We proposed HTCC, an HTM-assisted concurrency control protocol that achieves scalable and robust in-memory transaction processing on multicores.
  - Hybrid synchronization mechanism for reducing transaction abort rate;
  - Workset caching for minimizing transaction restart overhead.

Thanks!