Distributed Database Systems (ECS - 265)

Staring into the Abyss : An Evaluation of Concurrency Control with One Thousand Cores

Presented By

Sanjat Mishra

10.09.2018

Road Map

- What this paper is about?
- What problems does it address?
- What methods does this paper use to draw its conclusions?
- What criteria does this paper consider while drawing the conclusion?

What’s this paper about?

States the problems that todays Database Management System will face when paired with with a ‘many-core’ system.

Why are we talking about a thousand core system?

Right now, Multi Core systems are the only way of increasing the computing power required to carry out large scale operations!

What’s a Concurrency Control Problem?

- It is the coordination of the simultaneous executions of transactions in a multi user database.

- Problems that emerge without concurrency control:
  - Lost Update
  - Uncommitted Data
  - Inconsistent Retrieval

Methodology Adopted in the paper

CHOOSES WORKLOADS OR TEST DATABASES. (OLTP IN THIS CASE)

PERFORMS AN EVALUATION OF 7 CONCURRENCY CONTROL SCHEMES.

USES A SIMULATOR TO BENCHMARK PERFORMANCES ON A ‘MANY-CORE’ MACHINE AND THEN SCALES IT TO A THOUSAND CORE MACHINE.

Online Transaction Processing (OLTP)

The OLTP system supports that part of an application that interacts with the end users.

Features of OLTP Transactions :

1. They are short lived
2. They touch only a small subset of data during index look ups
3. They are repetitive

ACID Properties

Atomicity – Either the entire transaction takes place at once or doesn’t happen at all.

Consistency – The integrity constraints of a DB must be me so that the DB is consistent before and after a transaction.

Isolation – Ensures multiple transactions can occur concurrently without leading to inconsistency.

Durability – Ensures that once transaction is done, the updates are stored and written to the disk and persist even when system fails.

Concurrency Control Schemes

Two Phase Locking (2PL)

- DL_DETECT
- NO_WAIT
- WAIT_DIE

Timestamp Ordering (T/O)

- TIMESTAMP
- MVCC
- OCC
- H-STORE

Two Phase Locking (2PL)

Transactions have to acquire locks for an element in the DB before they are allowed to execute a read or write on that element.

The Database maintains the lock for each tuple or a higher logical level.

Ownership of locks is governed by the following rules;

1. Different transactions can’t simultaneously hold conflicting locks.
2. Once a transaction surrenders ownership of a lock, it can never obtain new locks.

Phases of 2PL

Growing

Growing Phase

- The Transaction can acquire as many locks as it wants to without releasing locks.

Shrinking

Shrinking Phase

- The Transaction enters the shrinking phase after it releases locks. Here, it is prohibited from obtaining more locks.

Types of Two Phase Locking

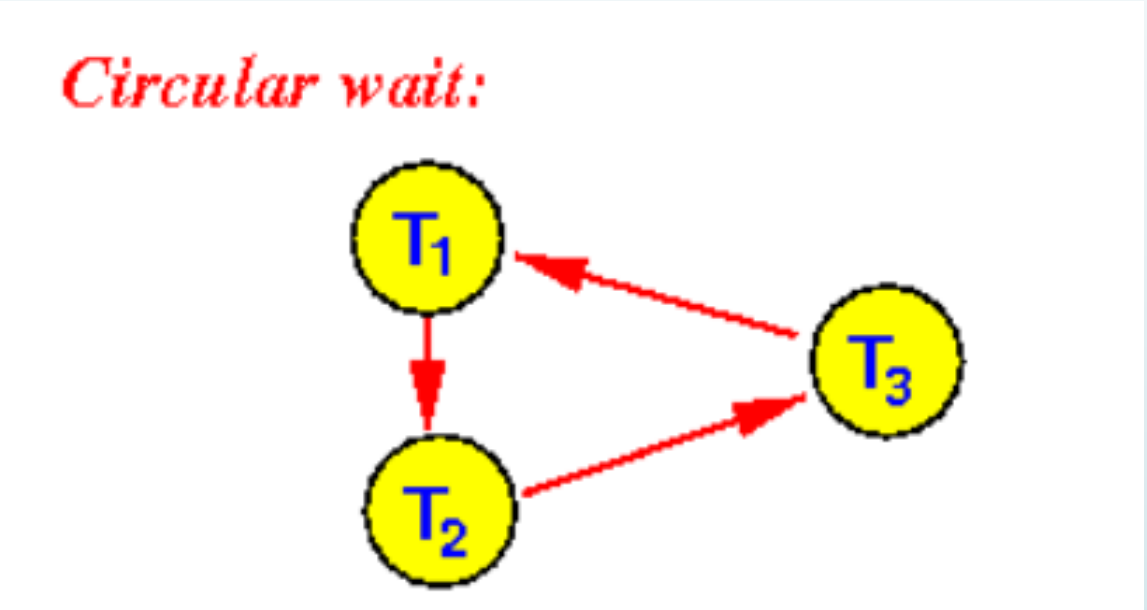
1. 2PL with Deadlock Detection (DL_DETECT)

   The DBMS monitors a waits-for graph for cycles.

   If a cycle is detected, this means there's a deadlock between those processes.

   When a deadlock is found, the system must choose which transaction to abort.

   Usually a transaction with lesser number of resources is aborted first.



Types of Two Phase Locking

2. 2PL with Non-Waiting Deadlock Prevention (NO_WAIT)

This scheme aborts a transaction if a deadlock is suspected.

When a lock request is denied, the scheduler automatically aborts the transaction requesting the lock.

Types of Two Phase Locking

3. 2PL with Waiting Deadlock Prevention (WAIT_DIE)

This is a non pre-emptive variation of the NO_WAIT scheme.

Here, each transaction needs to acquire a timestamp before execution.

The execution is based on timestamp ordering and helps prevent deadlocks.

In case of a deadlock, the younger of the transactions is aborted.

Timestamp Ordering (T/O)

Assigns a time stamp to every transaction and generates a serialization order a priori . The DBMS then enforces this order.

DBMS solves conflicts in the proper order of timestamp.

Broad way of categorizing the various schemes under T/O :

1. How the DBMS checks for conflicts?
2. When the DBMS checks for conflicts?

Every time a transaction updates a tuple in the database, it checks the timestamp of the previous operation on the same tuple.

If the timestamp of the new operation is lower than the timestamp of the previous operation on the same tuple, then the new operation has to be aborted.

In this method, the read operation always creates a copy of the tuple before it reads and only reads the copy.

16

Basic T/O (TIMESTAMP)

Multi version Concurrency Control (MVCC)

In this scheme, every write operation creates a new version of the tuple in the database.

Each version of the tuple is tagged with the timestamp and transaction id of the transaction that created it.

The DBMS maintains an internal list of the versions of an element.

For a Read operation, the DBMS determines which version of the element is to be accessed by checking the timestamp.

Optimistic Concurrency Control (OCC)

In this scheme, the DBMS tracks the read/write sets of each transaction and stores all of the “write” operations in a separate workspace.

When a transaction commits, the system checks and determines whether the transactions read set overlaps with any operation in the write set.

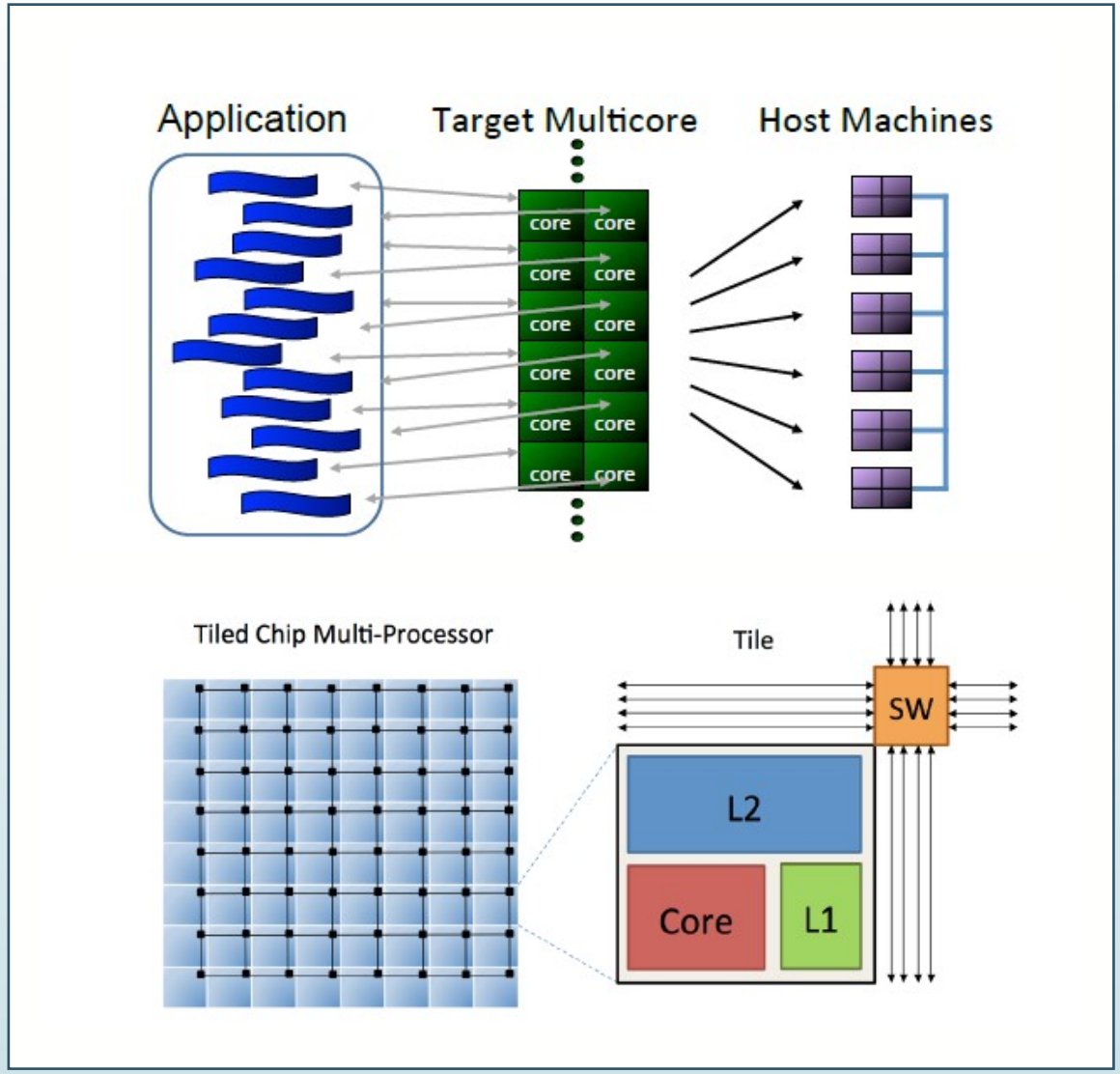
T/O with Partition Level Locking (H-STORE)

- In this scheme, the database is divided into disjoint sets of memory called partitions.
- Each partition is protected by a lock and is assigned a single threaded execution engine that has exclusive access to the partition.
- A transaction needs to have all the locks of all the partitions that it needs to access before it is allowed to start running.
- Hence, the DBMS needs to know before hand about which transactions access which partitions.

Test Set up

1. Graphite Simulator
   - Simulator for large scale multi core systems.
   - Can scale to 1024 cores.
   - The target architecture is a tiled chip multi processor where each tile contains a low power in order processing core.

2. Custom DBMS
   - Custom lightweight DB.
   - Number of worker threads = Number of cores , where each thread is mapped to a separate core.



Some Useful Terms

- **USEFUL WORK** : The time that the transaction is actually executing application logic and operating on tuples.
- **ABORT** : Overhead incurred when DBMS rolls back all of the changes made by a transaction.
- **TS ALLOCATION** : Time taken to allocate the timestamp from centralized allocator.
- **INDEX** : The time that the transaction spends in hash index for tables.
- **WAIT** : The total amount of time the transaction has to wait (either for a lock or for a value that’s not ready yet)
- **MANAGER** : The time that the transaction spends in lock manager or the timestamp. (Excludes wait time)

Workloads

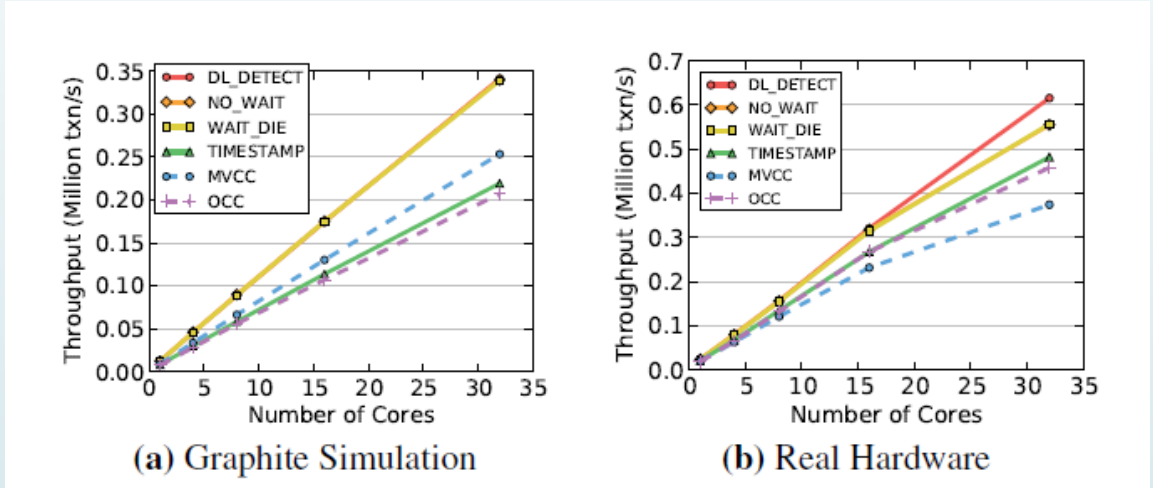
1. Yahoo Cloud Serving Benchmark (YCSB)
   - Collection of workloads that are representative of large scale services
   - 20GB YCSB database containing one table and 20 million records.
   - Single primary key column and DBMS creates a single hash index for the primary key.
   - Each transaction by default access 16 records at a time. (Read or Write)
   - Uses a term theta to determine level of contention
     - When Theta = 0, all tuples are accessed with same frequency.
     - When Theta = 0.6, a hotspot of 10% of tuples are accessed by 40% of the transactions.
     - When Theta = 0.8, a hotspot of 10% of tuples are accessed by 60% of the transactions.

Workloads

1. TPC-C
   - Current industry standard for evaluating performance of OLTP systems
   - Consists of 9 tables that simulate a warehouse centric order processing application.
   - Has 5 different types of transactions (only New Order and Payment are modeled in this paper)

Simulator vs Real Hardware

- The graph shows that the simulator generates results that are comparable to the Real Hardware.
- The trends of MVCC , TIMESTAMP and OCC are a bit different.
- After 32 cores, the both T/O based and WAIT_DIE schemes drop due to cross-core communication and timestamp allocation overhead.



General Optimizations

1. Memory Allocation

   While scaling DBMS to large core counts, DBMS spends most of the time in waiting for memory allocation.

   Hence a new malloc function was developed which assigns each thread its own memory pool and then resizes the pool according to the workload.

2. Lock Table

   This is a key contention point in DBMS. Instead of having a centralized lock table or timestamp manager, each transaction latches on to the tuple it needs.

3. Mutexes

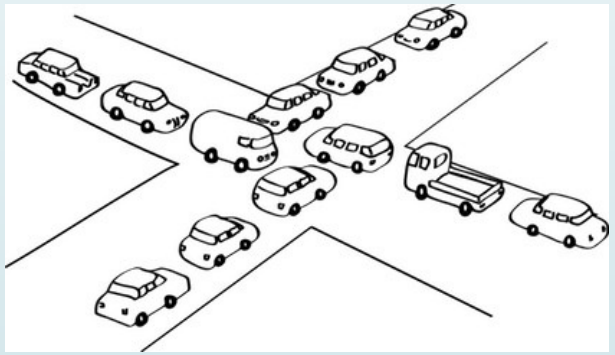
   Accessing a mutex lock is expensive and requires several messages to be sent across the chip. Reduces scalability.

Scalable Two Phase Locking

Deadlock Detection

The main bottle neck occurs when multiple threads compete to understand their waits-for graph and detect cycles.

By partitioning the data structures across cores and making the deadlock detector lock free , each core has its own local copy and doesn’t need to wait.



Lock Thrashing

Even with improved detection, the DL_DETECT doesn’t scale due to thrashing. This occurs when a transaction holds its lock until it commits, blocking all other concurrent transactions that need the same lock.

This becomes a bottleneck in most 2PL schemes.

Solution to Lock Thrashing

- Lock thrashing can be solved by aborting some transaction that are waiting to acquire locks.
- This can reduce the number of active transaction at a particular time.
- Ideally, setting a timeout helps the system run at optimal throughput. The timeout threshold varies cases to case.
- Restarting a transaction is relatively faster than rolling back and performing the changes again.
- Trade off between performance and transaction abort rate.

Scalable Timestamp Ordering

Timestamp Allocation

Using mutexes for timestamp allocation increases the duration and decreases scalability.

One solution is to use *atomic addition* operation to advance a global timestamp. This requires fewer instructions and is faster since the critical sector is locked down for a smaller period.

But this is still insufficient for a 1000-core CPU.

Other methods that can work:

- Atomic Addition with batching.
- CPU Clocks
- Hardware Counters

Comparing Timestamp Allocation Methods

- Mutex performs the worst.
- Throughput of atomic addition reduces with increasing number of cores.
- Batching suffers from contention after a point.
- CPU Clock is the ideal candidate as its decentralized.

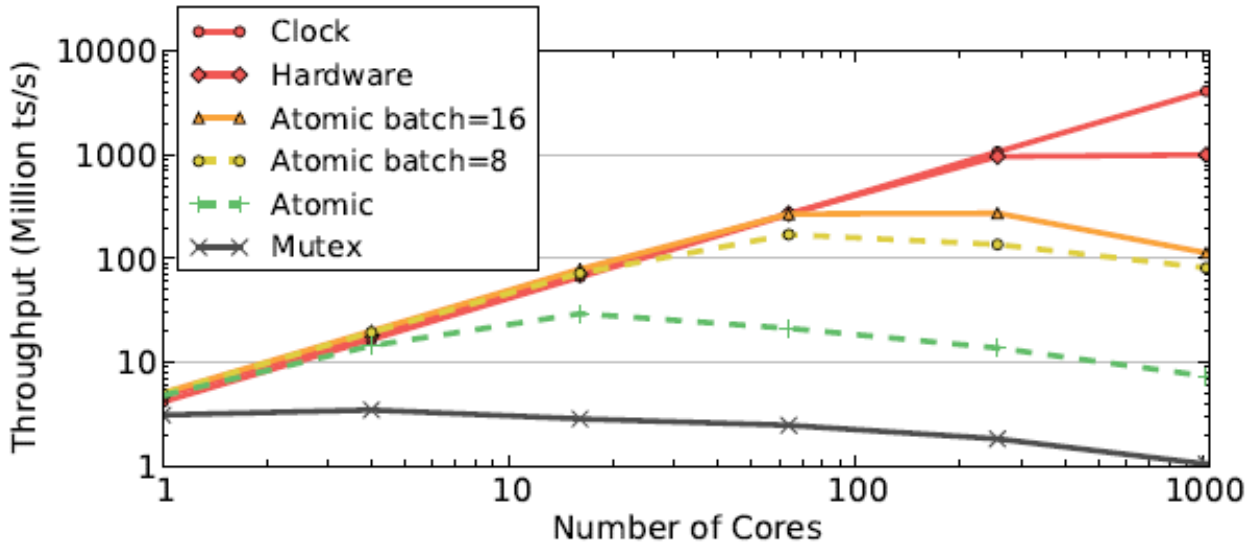


Figure 6: Timestamp Allocation Micro-benchmark – Throughput measurements for different timestamp allocation methods.

Comparing Timestamp Allocation Methods on Workload

- When there's no contention, the results are almost similar.
- When there's contention, transaction have to restart and hence performance depreciates.

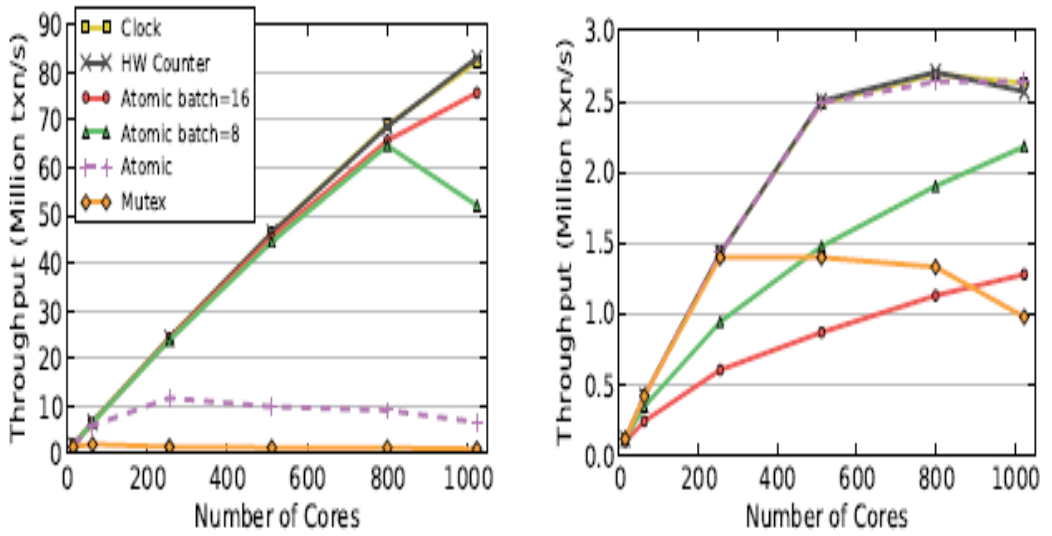


Figure 7: Timestamp Allocation – Throughput of the YCSB workload using TIMESTAMP with different timestamp allocation methods.

Distributed Validation

- This is specifically meant for OCC where there is a critical section after the read phase.
- Normally, mutexes are used to protect the critical section but this decreases scalability.
- Instead, using per tuple validation that breaks the operation into smaller fragments is faster.

Local Partitions

This scheme is meant for H-STORE . By enhancing H-STORE to use the shared memory effectively, scalability is achievable.

By giving direct data access to transactions for remote partitions, overhead decreases .

The read only tables don’t create additional copies and hence reduces memory footprint.

Experimental Analysis

The experiment done can be grouped into 2 categories:

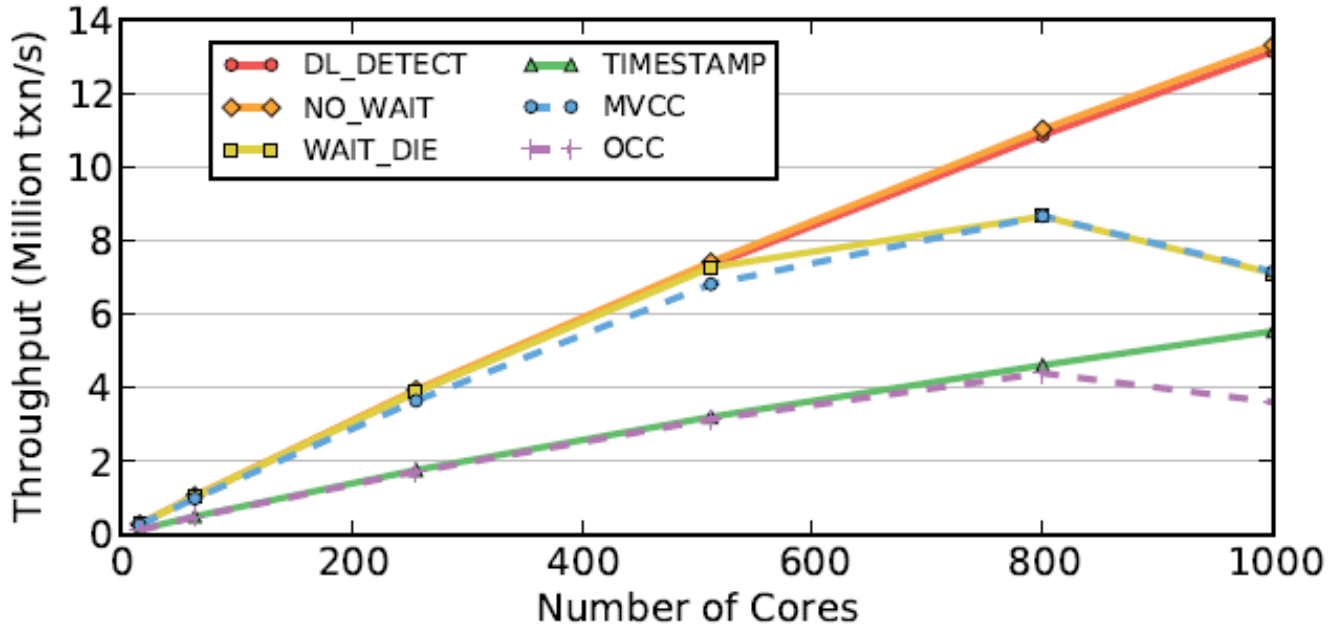
- Based on Scalability
- Based on Sensitivity to Data changes

Scalability experiment tells us how well the model performs when the number of cores increases.

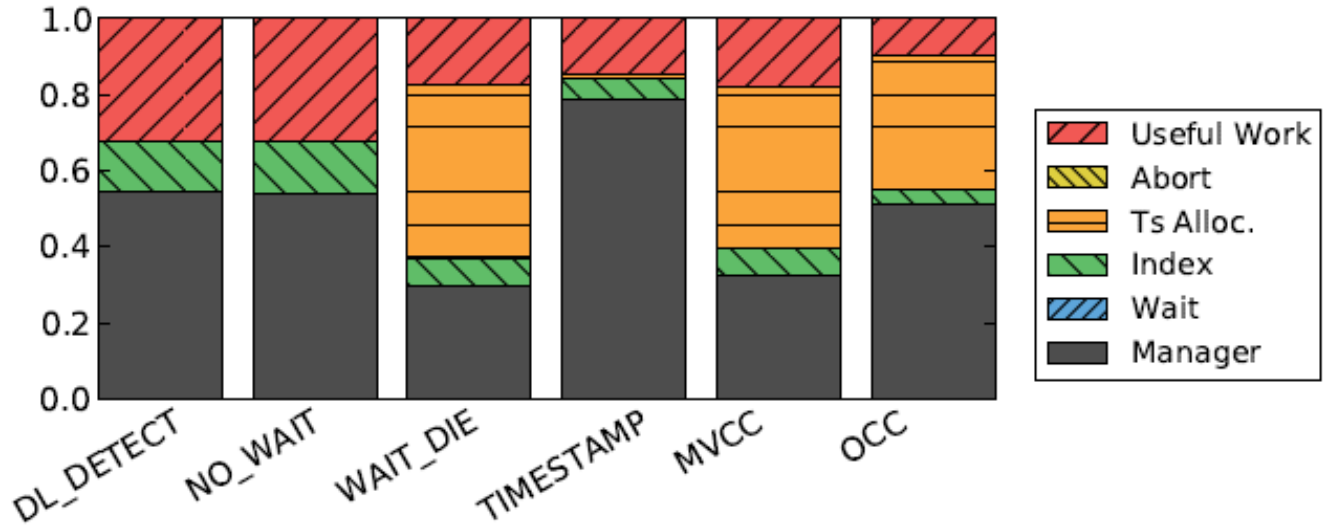
The Sensitivity experiment tells us how well the model handles changes to data or more complicated transaction scenarios.

Read Only Workload

- The Read only arrangement provides a benchmark before moving to more complex arrangements.
- In a perfectly scalable case, linear increase should be present.
- Timestamp allocation bottle necks the related schemes.
- OCC and TIMESTAMP waste cycles while making copies of data to be read.



(a) Total Throughput

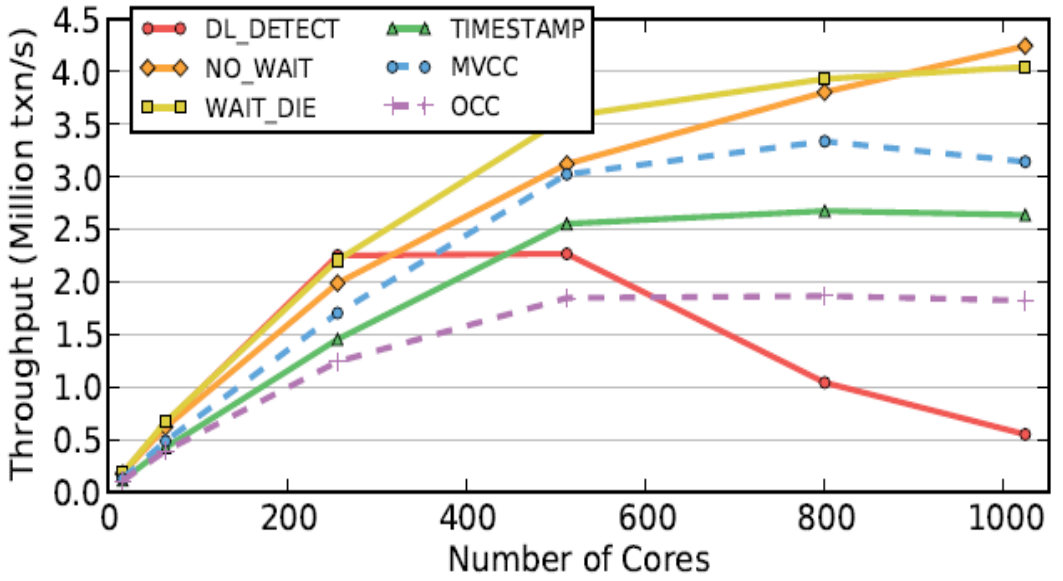


(b) Runtime Breakdown (1024 cores)

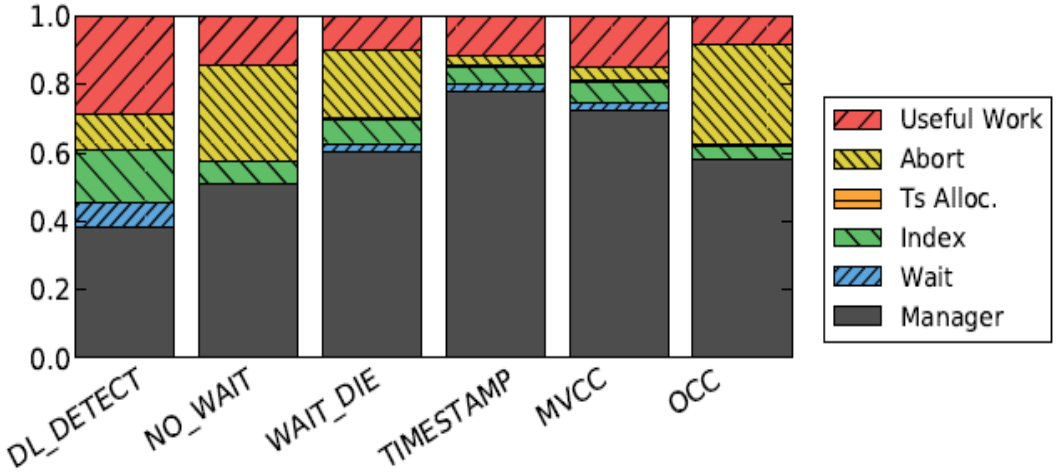
Figure 8: Read-only Workload – Results for a read-only YCSB workload.

Write Intensive Workload (Medium Contention)

- Large size of the workload means contention can vary and may be less.
- Hence, we introduce the “theta” factor to reflect real world data which has high contention chances.
- NO_WAIT and WAIT_DIE alone scale past 512 cores.
- DL_DETECT spends most time in waiting.
- OCC spends large portion in aborting.
- MVCC and TIMESTAMP perform good as they overlap operations and reduce waiting time.



(a) Total Throughput

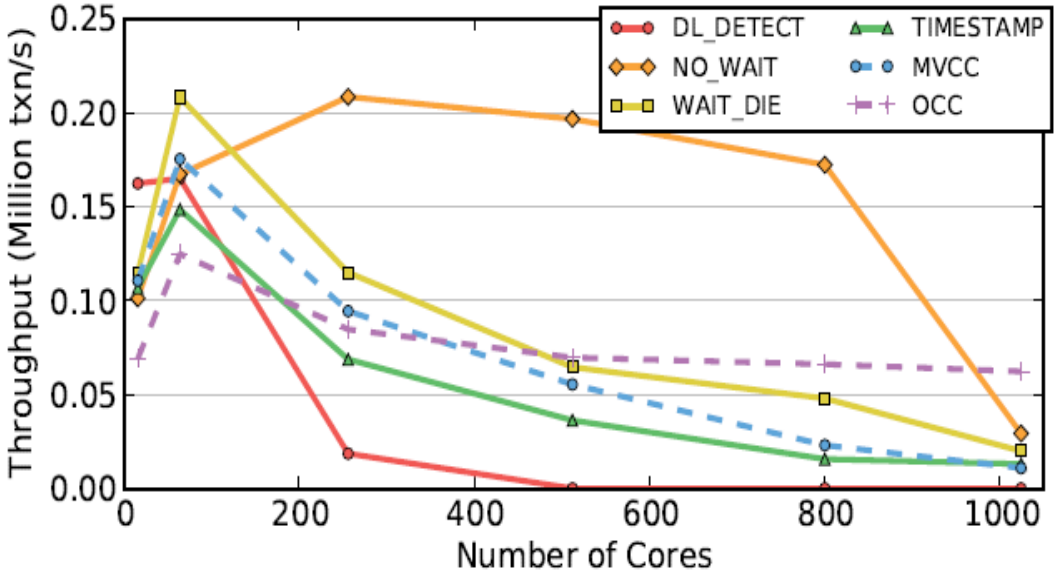


(b) Runtime Breakdown (512 cores)

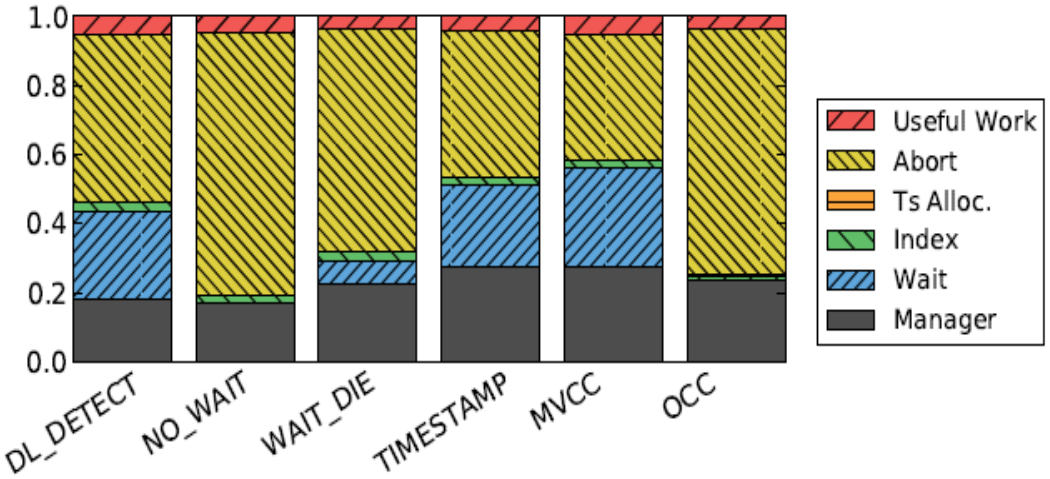
Figure 9: Write-Intensive Workload (Medium Contention) – Results for YCSB workload with medium contention (*theta*=0.6).

Write Intensive Workload (High Contention)

- When high contention, all of the schemes fail to scale.
- Due to higher number of conflicts, most of the time is spent in aborting transactions or waiting for lock release.



(a) Total Throughput



(b) Runtime Breakdown (64 cores)

Figure 10: Write-Intensive Workload (High Contention) – Results for YCSB workload with high contention (*theta*=0.8).

Sensitivity to Contention

- With increase in theta value, the schemes virtually become non-scalable.
- Increase in the number of cores stops to matter.

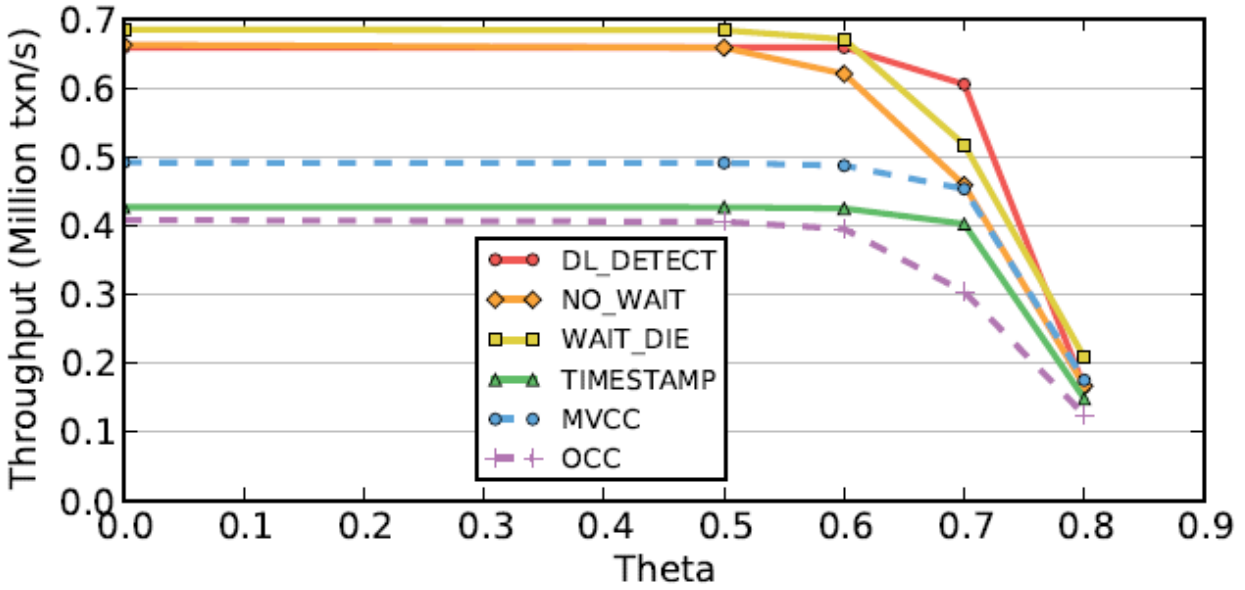
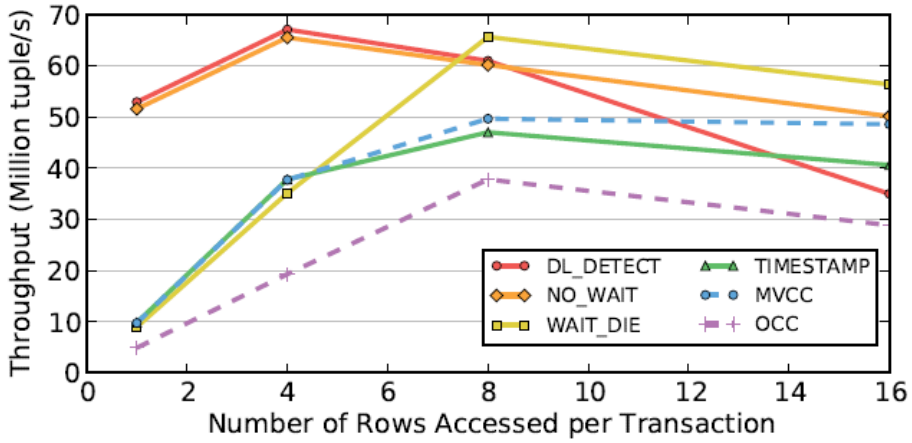


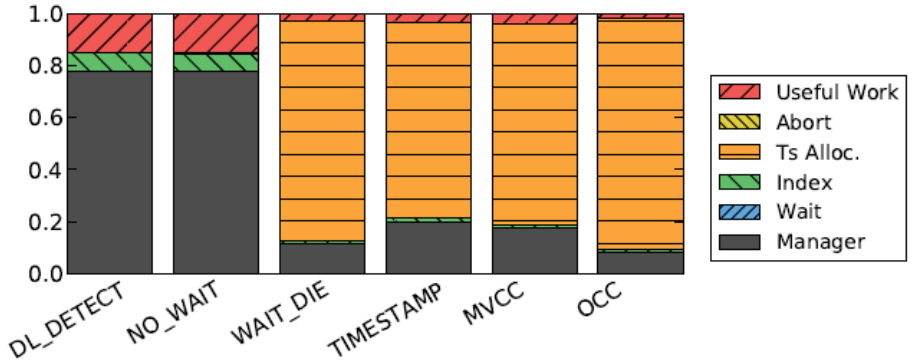
Figure 11: Write-Intensive Workload (Variable Contention) – Results for YCSB workload with varying level of contention on 64 cores.

Working Set Size

- Working set is the number of records the transactions need to access.
- When the working set size increases, the chances of contention also increase.
- Shorter Transactions lead to higher through put as contention chances decrease.
- When short transactions, DL_DETECT and NO_WAIT have best throughputs.
- With increase in size, thrashing also increases.
- When transactions are small, T/O schemes suffer because cost of timestamp is high.
- This later gets amortized and they scale better.



(a) Total Throughput



(b) Runtime Breakdown (transaction length = 1)

Figure 12: Working Set Size – The number of tuples accessed per core on 512 cores for transactions with a varying number of queries (*theta*=0.6).

Read/Write Mixture

- MVCC performs best consistently.
- TIMESTAMP suffers due to copy overhead.

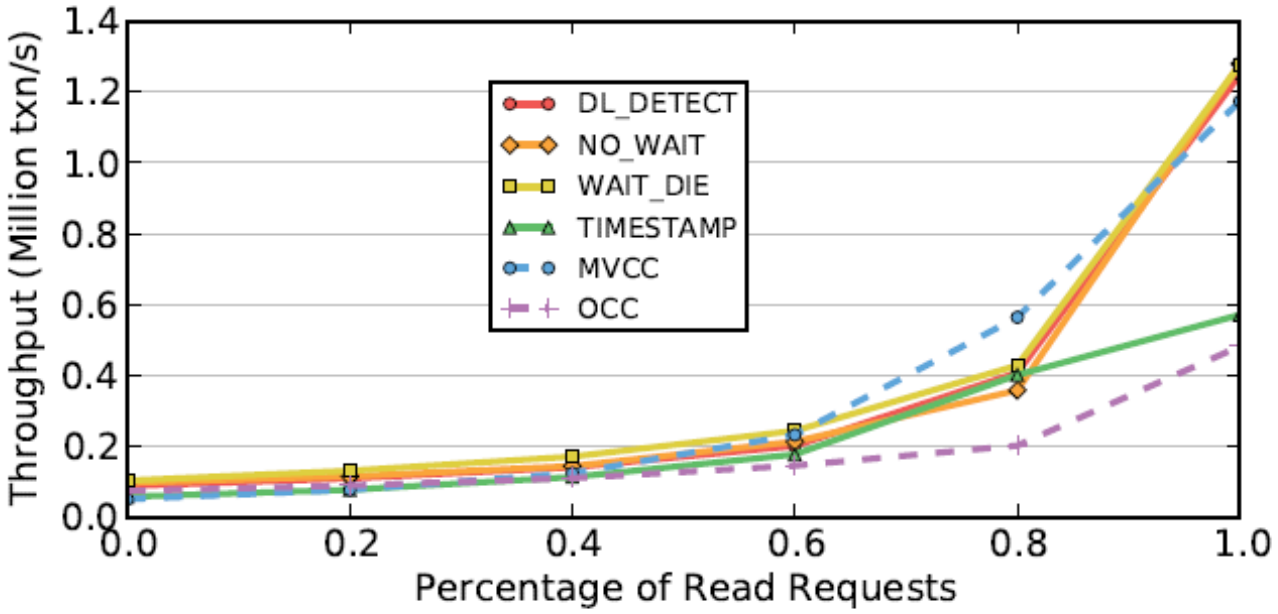


Figure 13: Read/Write Mixture – Results for YCSB with a varying percentage of read-only transactions with high contention (*theta*=0.8).

Database Partitioning

- When the database is partitioned and cores are assigned, H-STORE initially performs the best.
- This approach is best when the data to be accessed is split across less number of partitions.
- With increase in number of partitions, every scheme suffers.

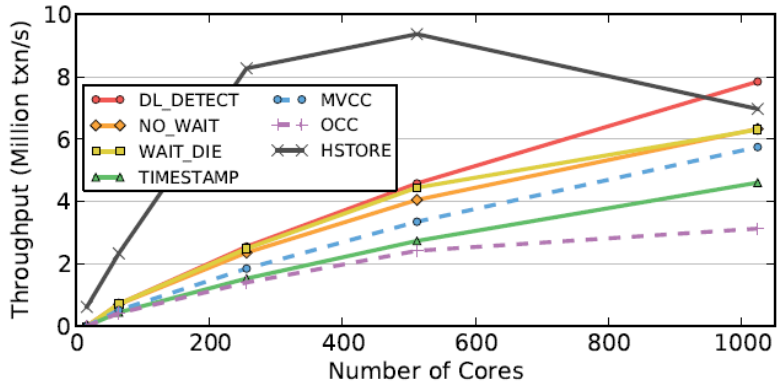
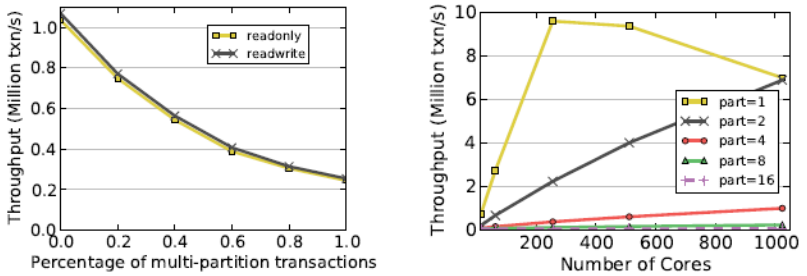


Figure 14: Database Partitioning – Results for a read-only workload on a partitioned YCSB database. The transactions access the database based on a uniform distribution (*theta*=0.0).



(a) Multi-Partition Percentage **(b)** Partitions per Transaction

Figure 15: Multi-Partition Transactions – Sensitivity analysis of the H-STORE scheme for YCSB workloads with multi-partition transactions.

TPC-C Workload (4- Warehouses)

- More worker threads than warehouses.
- Cross core communication takes place.
- All schemes fail to scale when there are few warehouses than cores.
- H-STORE isn't optimal as data is scattered across multiple partitions.
- 2PL schemes suffer from thrashing.
- T/O experiences high abort rates but outperforms others as Reads aren't blocked by Writes.

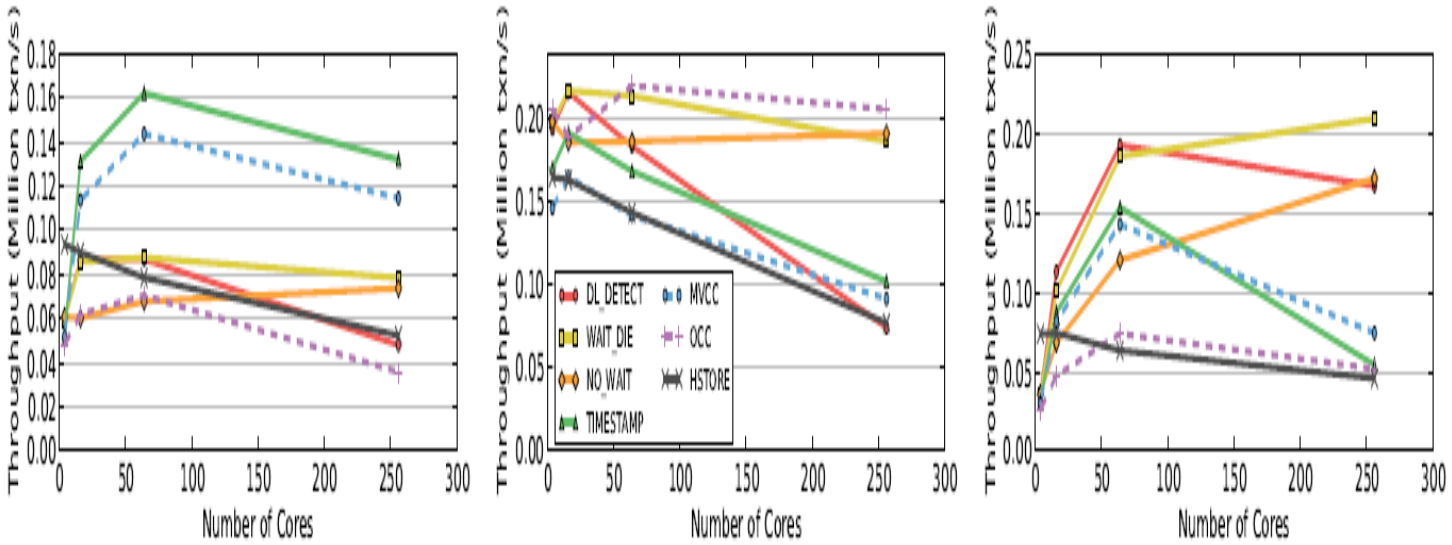


Figure 16: TPC-C (4 warehouses) – Results for the TPC-C workload running up to 256 cores.

TPC-C Workload (1024 Warehouses)

- Here, number of warehouses = number of cores.
- Even if there is no contention, bottleneck is maintaining and assigning locks and Timestamp Allocation.
- MVCC suffers from write overheads.
- OCC suffers from acquiring latches.
- Performance only better in Payment as bottle neck is eliminated.

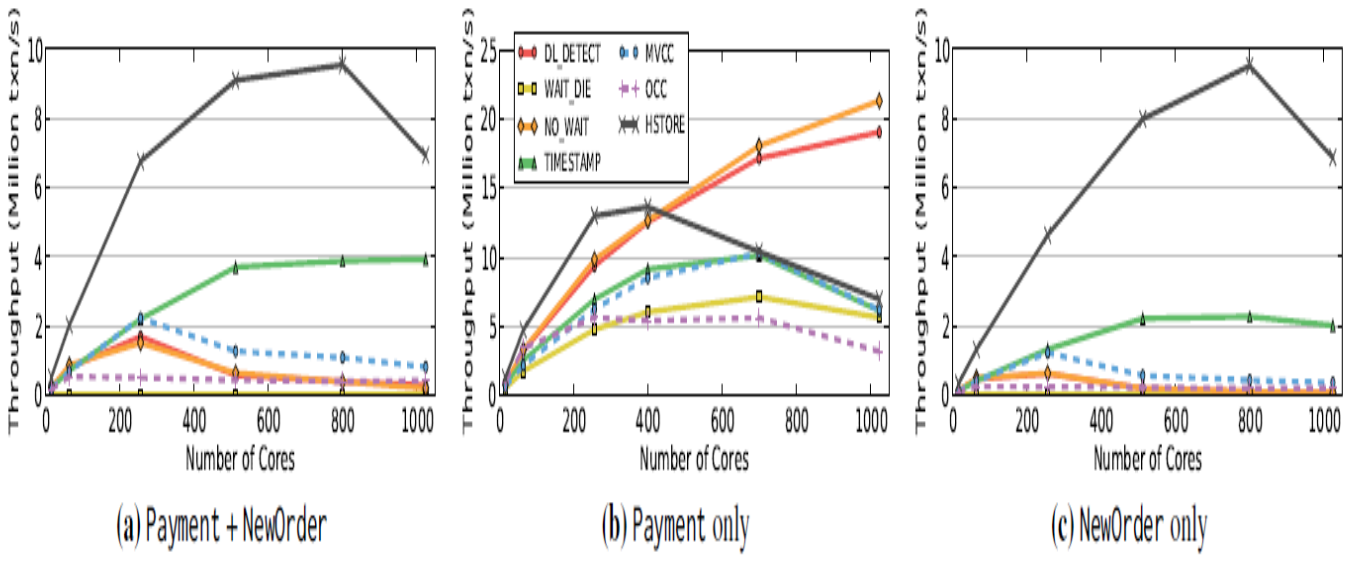


Figure 17: TPC-C (1024 warehouses) – Results for the TPC-C workload running up to 1024 cores.

Conclusion

- Every scheme suffers from bottle necks under different scenarios.
- No scheme is ideal for real world application when number of cores are high.
- Extra cores are never utilized to their full potential.

2PL	**DL_DETECT**	Scales under low-contention. Suffers from lock thrashing.
	NO_WAIT	Has no centralized point of contention. Highly scalable. Very high abort rate.
	WAIT_DIE	Suffers from lock thrashing and timestamp bottleneck.
T/O	**TIMESTAMP**	High overhead from copying data locally. Non-blocking writes. Suffers from timestamp bottleneck.
	MVCC	Performs well w/ read-intensive workload. Non-blocking reads and writes. Suffers from timestamp bottleneck.
	OCC	High overhead for copying data locally. High abort cost. Suffers from timestamp bottleneck.
	H-STORE	The best algorithm for partitioned workloads. Suffers from multi-partition transactions and timestamp bottleneck.