Improving OLTP scalability using speculative lock inheritance

Ryan Johnson, Ippokratis Pandis, Anastasia Ailamaki
OLTP scalability limitations

- Application: scalable
- Hardware: scalable
- Storage manager?
  - Good at interleaving threads
  - True concurrency is harder
  => Increased latch contention

Contention within storage manager limits scalability
Contestation in the lock manager

- Centralized service
  - Locks managed globally
- Fine-grained parallelism
  - Each lock has its own latch
- Skewed access
  - Some hotter than others

Culprit: shared high-level locks
Contributions

• Identify lock manager bottleneck
  – Latch contention during lock state updates

• Eliminate most requests for hot locks
  – Pass hot locks directly between transactions
  – No changes at application level

• Improve throughput by 20-50%
In this talk...

- Introduction
- Inside the lock manager
  - Locking vs. latching
  - Lock acquire and release
  - Approaches for reducing overheads
- Speculative lock inheritance
- Conclusions
Overview of latching vs. locking

<table>
<thead>
<tr>
<th></th>
<th>Latch</th>
<th>Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serializes</td>
<td>Code segments</td>
<td>Transactions</td>
</tr>
<tr>
<td>Protects</td>
<td>Data structures</td>
<td>Database records</td>
</tr>
<tr>
<td>Duration</td>
<td>Short</td>
<td>Long</td>
</tr>
<tr>
<td># Held at once</td>
<td>0-2</td>
<td>Unbounded</td>
</tr>
<tr>
<td>Implementation</td>
<td>Straightforward</td>
<td>Complex</td>
</tr>
</tbody>
</table>

Latches outnumber lock operations 5:1 or more
Inside the lock manager - acquire

Requirements
⇒ Find/create many locks in parallel
⇒ Each lock tracks many requests
⇒ Each transaction tracks many locks
Inside the lock manager - release

A) Compute new lock mode (supremum)

B) Process upgrades

C) Grant new requests

Lock strengths

IS < IX < S

Intent locks => long request chains
In this talk...

• Introduction
• Inside the lock manager
• Speculative lock inheritance
  – Approach and implementation
  – Experimental results
• Conclusions
Speculative lock inheritance

Agent thread execution

 trx1

 trx2

 trx3

 Hot lock
 Cold lock

Lock Manager

Hot locks bounce between agents and LM
Speculative lock inheritance

Agent thread keeps hot locks, bypasses lock manager

Hot lock
Cold lock
Criteria for lock inheritance

- Hot (latch contention)
- Held in shared mode
- No other trx waiting (fairness)
- Not row-level (too many of those)
- Parent (if any) also eligible
Avoiding starvation

Speculation is lightweight, non-intrusive

Held

eligible && !waiting

Inactive

discard

inherit

Invalid

waiter arrives

Update lock requests using atomic ops
⇒ sidestep the lock manager
⇒ clean up invalid requests lazily
Breakdown of lock behavior

- **row-level**: 34%
- **used**: 4%
- **exclusive**: 3%
- **invalidated**: 6%
- **missed**: 10%
- **cold**: 27%

A few locks cause 97% of contention
Experimental setup

• Hardware
  – Sun T5220 “Niagara II” (sparcv9-solaris10)
  – 16 pipelines, 64 contexts, 64GB RAM

• Software
  – Shore-MT
  – Extended to include lock inheritance

• Benchmarks (mem-resident)
  – Telecom: Nokia Benchmark (NDBB)
  – Banking: TPC-B
  – Online Sales: TPC-C
Scalability comparison

Eliminate contention with low overhead

Throughput (ktps)

Hardware Contexts Utilized

NDBB (SLI)

TPC-B (SLI)

NDBB

TPC-B

234% (53%)

73% (19%)
Impact of lock inheritance

Benchmark (Baseline vs. SLI)

CPU time breakdown

- NDBB
- TPC-B
- TPC-C

SLI Overhead
LM Contention
Other Contention
LM Overhead
Computation

Lock bottleneck gone (but others may arise)
Reducing overheads of locking

• Rdb/VMS
  – Distributed DBMS
  – Lock “carry-over” reduces network traffic

• DB2 “keep table lock” setting
  – Connection holds all table locks until close
  – Leads to “poor concurrency”

• H-Store
  – Single-threaded, non-interleaved execution model
  – No locking or latching at all
Conclusions

• All latches are potential bottlenecks
  – Even “fine-grained” ones if there’s skew

• Best solutions may be indirect
  – Sidestep hard problems
  – Look to distributed databases

• Lock inheritance eliminates one bottleneck
  – Distribute accesses to hot locks
  – Improve throughput 20-50% today
  – Benefit increases with more cores