ECE7995
(5) System Buffer III: Lock Contention in the Caching
Demand for High Scalability on Buffer Management

- Effective management of buffer cache is critical for I/O-intensive applications.
  - Memory is several orders of magnitude faster than hard disks.
  - Both temporal and spatial localities can be exploited for high cache hit ratio.
- I/O concurrency is increasingly high.
  - Hundreds of thousands of threads/transactions;
  - Multi-processors and multi-cores systems.
- High concurrency demands high scalability on buffer management.
  - This is in addition to the requirement of high hit ratio.
  - The scalability can be severely limited by lock contention.
Lock Operation for Replacement algorithms

Lock Synchronization

Replacement Algorithm \((\text{modify data structures, etc.})\)

Buffer Pool

Hard Disk

Page Accesses

Looking-up Hash Table

Buffer Page

Hits: bottleneck

Misses: bottleneck
Replacement Algorithm: Optimized for Hit Ratio or Scalability?

for high hit ratio

2Q, LIRS, MQ, ARC, SEQ, ....

Can we have both?

for high scalability

CLOCK, CLOCK-Pro, CAR, ....

- Clock-based approximations exhibits sub-optimal performance;
- The transformation can be difficult and demand great efforts;
- Some algorithms don’t have clock-based approximations.
Use of Locks: Optimized for Lock Contention or Hit Ratio?

Centralized Lock

Replace the Replacement Algorithm (modify data structures, etc.) in the Buffer Pool.

Distributed Lock

- The lock contention may not be evenly reduced;
- Localized history access information can harm hit ratio;
- Some replacement algorithms do not work without global information.
A Framework that Makes Any Replacement Algorithms *(Almost)* Lock Contention Free

The objectives of our solutions:

- **Do not require changes** of existing replacement algorithms
- **Retain the advantages of high hit ratios**
- **Replacement algorithms scale** as well as the clock-based approximations.

Our solutions:

- Amortizing local acquisition cost by batching requests;
- Reducing lock holding time by prefetching
Reducing Lock Contention by Batching Requests

Replacement Algorithm (*modify data structures, etc.*)

Buffer Pool

One queue per thread

Replacement Algorithm (*modify data structures, etc.*)

Buffer Pool
Amortized Lock Acquisition Cost

- Hardware: SGI Altix 350 SMP with 16 Itanium 2 processors;
- Software: PostgreSQL 8.2.3
- Workload: DBT-1 test kit (simulating TPC-W)

(Lock acquisition cost + lock holding time) normalized over requests in a batch

![Graph showing the relationship between batch size and time (microsecond).]
Reducing Lock Contention by Batching Requests

When to commit accesses in a queue:
1. TryLock()
2. If the queue is full, Lock()

Replacement Algorithm (*modify data structures, etc.*)

Buffer Pool
Reducing Lock Contention by Prefetching

Thread 2
Cache Miss
Stalk

Thread 1

Pre-read data that will be accessed in the critical section

Time

Thread 2

Thread 1
Performance Evaluation

- **Hardware Systems:**
  - SGI Altix 350 SMP with 16 Itanium 2 processors
  - Dell PowerEdge 1900 Server (two quad-core Xeon X5355 processors)

- **Software:**
  - PostgreSQL 8.2.3 + Linux Red Hat Enterprise Linux AS;

- **Workloads**
  - DBT-1 (simulating TPC-W) from OSDL database test suite;
  - DBT-2 (simulating TPC-C) from OSDL database test suite;
  - TableScan: each transaction sequentially scan a table of 800,000 rows (128-byte)

- **Tested policies**

<table>
<thead>
<tr>
<th>Name</th>
<th>Replacement</th>
<th>Enhancement</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgClock</td>
<td>Clock</td>
<td>None</td>
</tr>
<tr>
<td>pg2Q</td>
<td>2Q</td>
<td>None</td>
</tr>
<tr>
<td>pgBatching</td>
<td>2Q</td>
<td>Batching</td>
</tr>
<tr>
<td>pgPref</td>
<td>2Q</td>
<td>Prefetching</td>
</tr>
<tr>
<td>pgBat-Pre</td>
<td>2Q</td>
<td>Batching and Prefetching</td>
</tr>
</tbody>
</table>
Reduction of Lock Contention
(SGI Altix 350, DBT-1, no misses)

Lock contention: a lock cannot be obtained without blocking;
Number of lock contentions per million page accesses.

Reduced by over 7000 times!
Improvement of System Throughput
(DBT-1, no misses)

SGI Altix 350
(16 Itaniums)

Poweredge 1900
(two quad-core Xeon)

Throughput is doubled!
Benefits from both High Hit Ratio and Low Lock Contention
(SGI Altix 350, DBT-1, #processors = 8)
Summary

- A scalable replacement algorithm is critical with rapidly increasing number of cores/processors.
- It is challenging to design a replacement algorithm of both high hit ratio and high scalability.
- Our framework can make any replacement algorithm (of high hit ratio) scalable;
- The framework enables relatively high system performance with different buffer cache sizes.