Tradeoff Evaluation

Design Choice	Algorithms to Compare
Invalid Access Prevention	C2PL (Detection) vs. CB-A (Avoidance)
Write Intention Declaration	CB-R (Synchronous) vs. O2PL-I (Deferred)
Write Permission Duration	CB-R (Single Transaction) vs. CB-A (Until Revoked or Dropped)
Remote Update Action	O2PL-I (Invalidation) vs. O2PL-P (Propagation)

Comparison between C2PL and CB-A, as both:

- Allow intertransaction caching
- Don't use propagation
- Synchronously activate consistency actions

Tradeoff Evaluation

Design Choice	Algorithms to Compare
Invalid Access Prevention	C2PL (Detection) vs. CB-A (Avoidance)
Write Intention Declaration	CB-R (Synchronous) vs. O2PL-I (Deferred)
Write Permission Duration	CB-R (Single Transaction) vs. CB-A (Until Revoked or Dropped)
Remote Update Action	O2PL-I (Invalidation) vs. O2PL-P (Propagation)

Comparison between CB-R (“pessimistic”) and O2PL-I (“optimistic”), as both:

- Are avoidance-based
- Are invalidation-based
- Retain write permissions only until transaction commit

Tradeoff Evaluation

Design Choice	Algorithms to Compare
Invalid Access Prevention	C2PL (Detection) vs. CB-A (Avoidance)
Write Intention Declaration	CB-R (Synchronous) vs. O2PL-I (Deferred)
Write Permission Duration	CB-R (Single Transaction) vs. CB-A (Until Revoked or Dropped)
Remote Update Action	O2PL-I (Invalidation) vs. O2PL-P (Propagation)

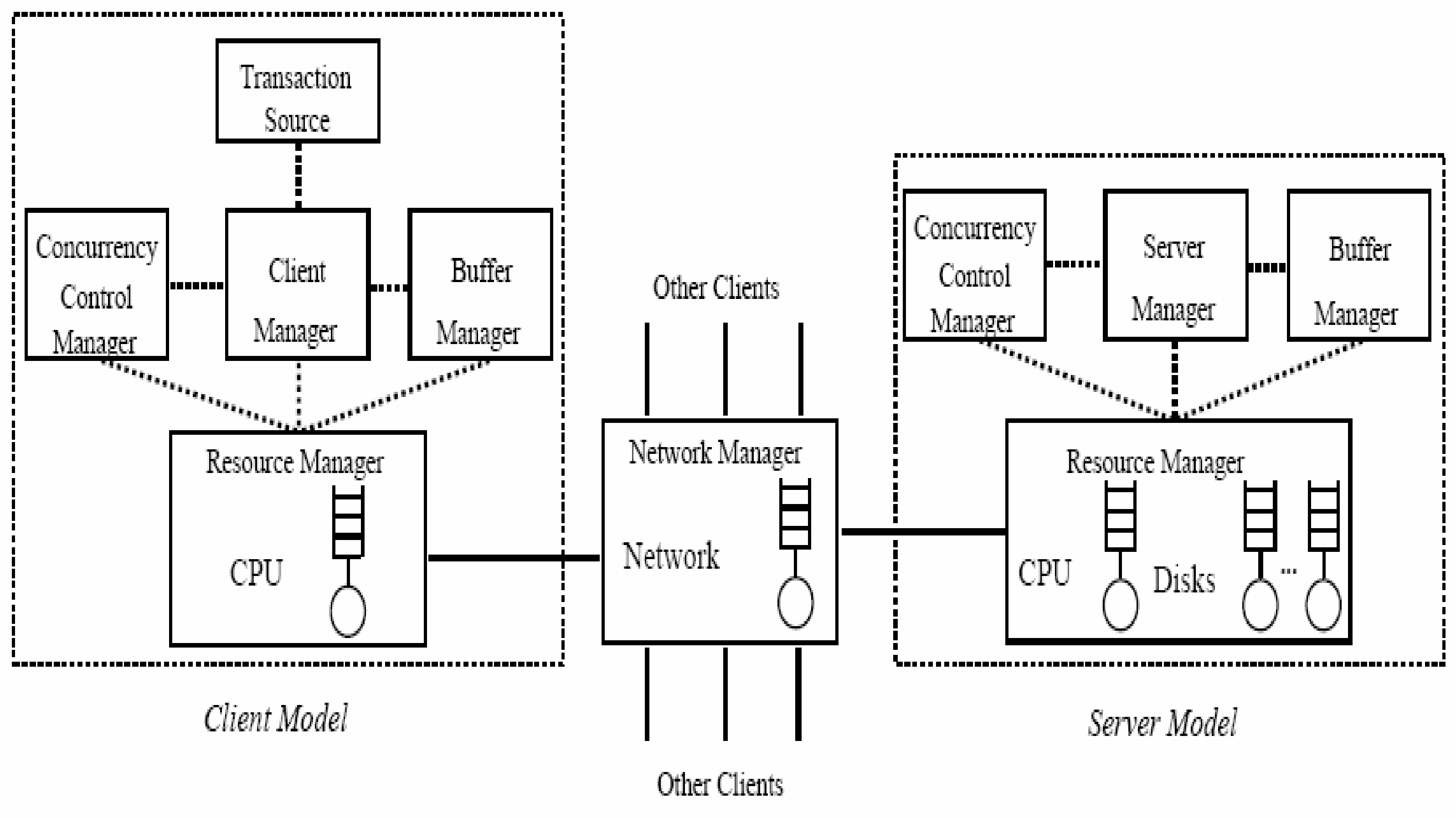
Comparison between CB-R and CB-A as they only differ for this aspect.

Tradeoff Evaluation

Design Choice	Algorithms to Compare
Invalid Access Prevention	C2PL (Detection) vs. CB-A (Avoidance)
Write Intention Declaration	CB-R (Synchronous) vs. O2PL-I (Deferred)
Write Permission Duration	CB-R (Single Transaction) vs. CB-A (Until Revoked or Dropped)
Remote Update Action	O2PL-I (Invalidation) vs. O2PL-P (Propagation)

Comparison between O2PL-I and O2PL-P as they only differ for this aspect.

Performance model (i)



Reference System Model

Performance model (ii)

Parameter	PRIVATE	HOTCOLD	UNIFORM	FEED
TransSize	16 pages	20 pages	20 pages	5 pages
HotBounds	p to p+24, $p = 25(n\text{-}1)+1$	p to p+49, $p = 50(n\text{-}1)+1$	-	1 to 50
ColdBounds	626 to 1,250	rest of DB	all of DB	rest of DB
HotAccProb	0.8	0.8	-	0.8
ColdAccProb	0.2	0.2	1.0	0.2
HotWrtProb	0.2	0.2	-	1.0/0.0
ColdWrtProb	0.0	0.2	0.2	0.0/0.0
PerPageInst	30,000	30,000	30,000	30,000
ThinkTime	0	0	0	0
	Low data contention	**Moderate data contention**	**High data contention**	**One producer n-consumers**

Workload parameter settings for n clients

Private Model

Large Client Cache, (relatively) slow local area network.
Emphasis is *mainly* on message exchange cost, rather than server I/O

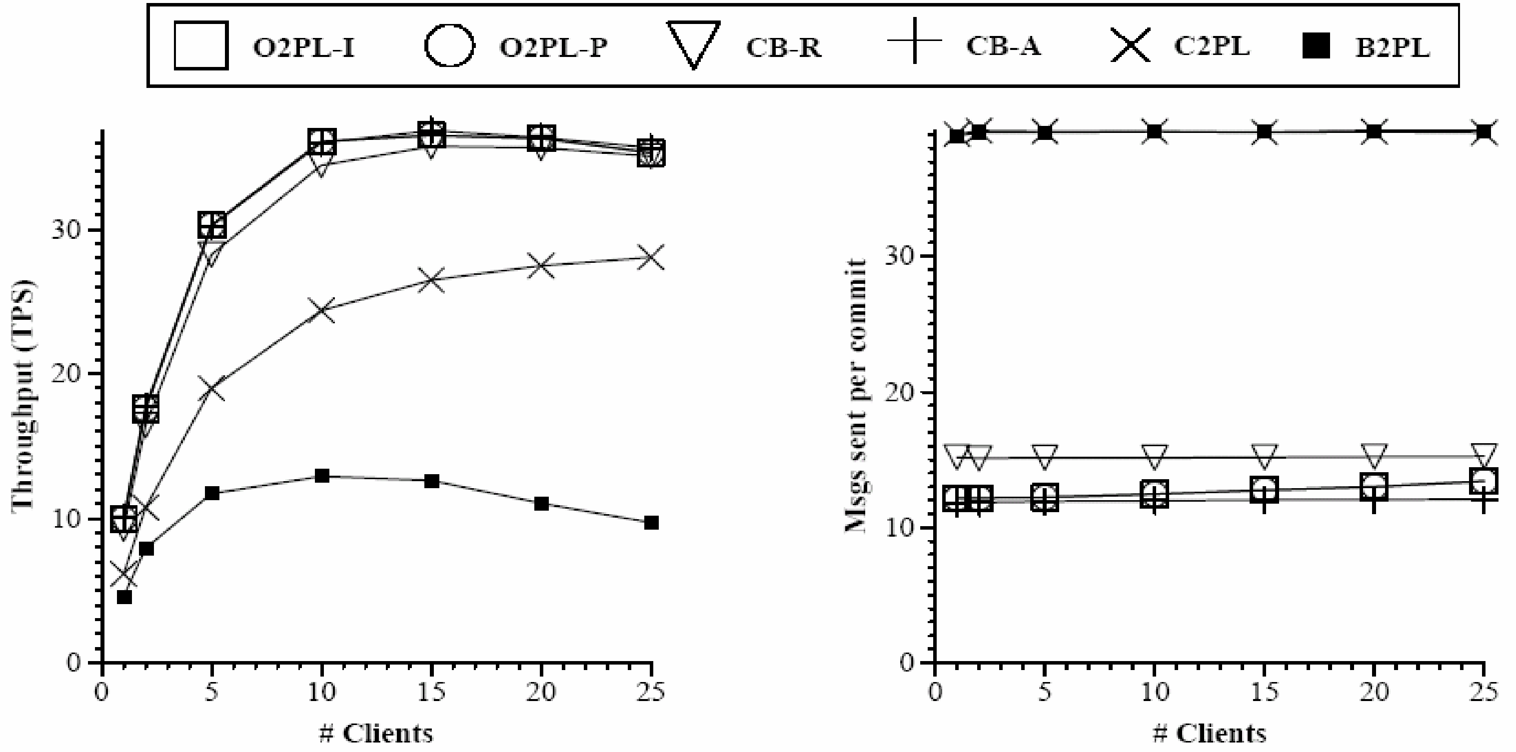


Figure 5: Throughput
(PRIVATE, 25% Client Cache, Slow Net)

Figure 6: Messages Sent/Commit
(PRIVATE, 25% Client Cache, Slow Net)

Private Model

LOOSER: Detection

- Again, due to high message overhead:
  - one req. per accessed item
  - replies are always images in B2PL
- Detection based approaches require more optimism!

Tradeoff: Detection vs Avoidance

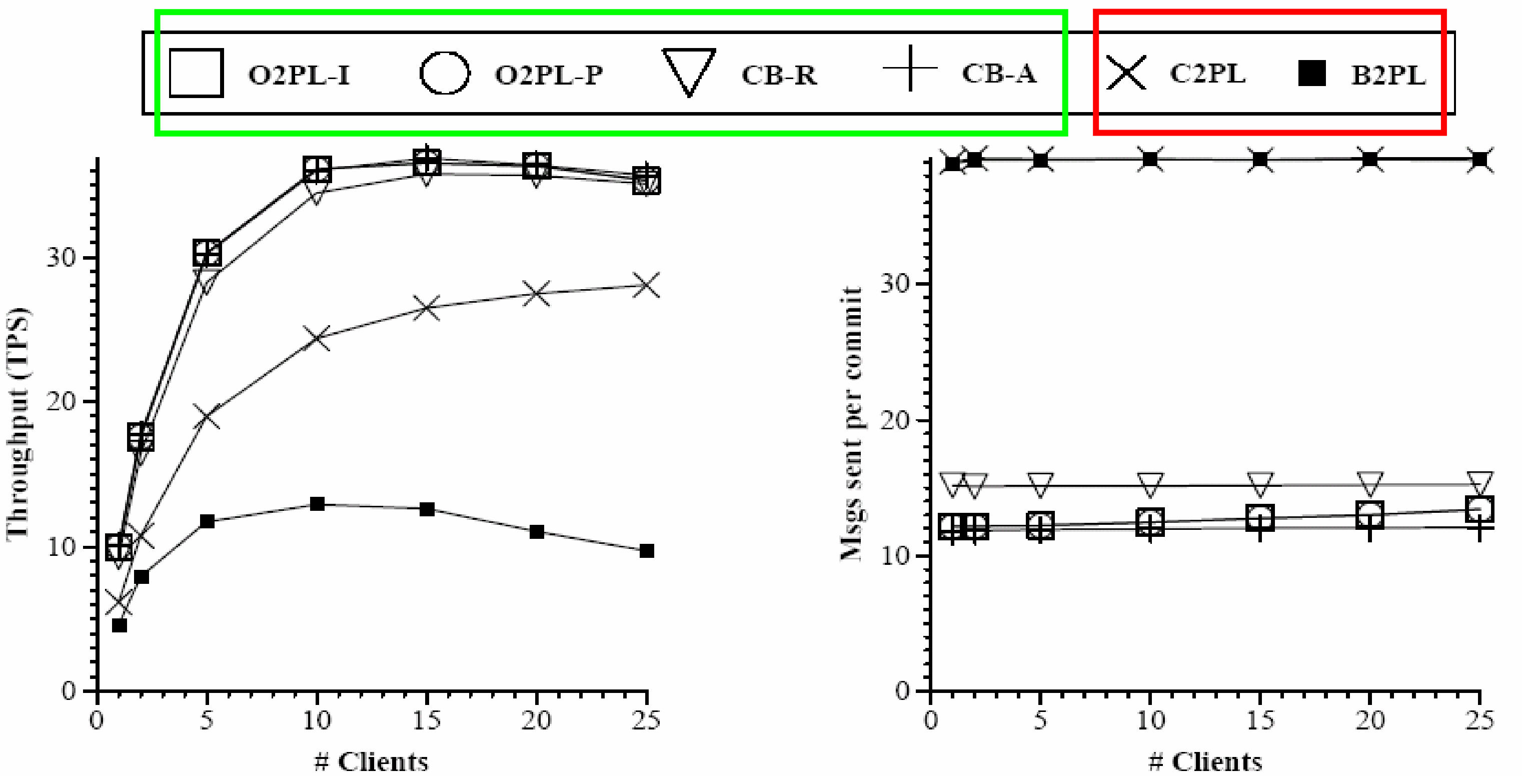


Figure 5: Throughput
(PRIVATE, 25% Client Cache, Slow Net)

Figure 6: Messages Sent/Commit
(PRIVATE, 25% Client Cache, Slow Net)

Private Model

Looser: Synch

- Sligthly worse performance in low contention env
- O2PL saves some msgs by batching write intention declarations at commit time (no concurrency induced aborts)

Tradeoff:

Synch vs Asynch Write Intention Timing

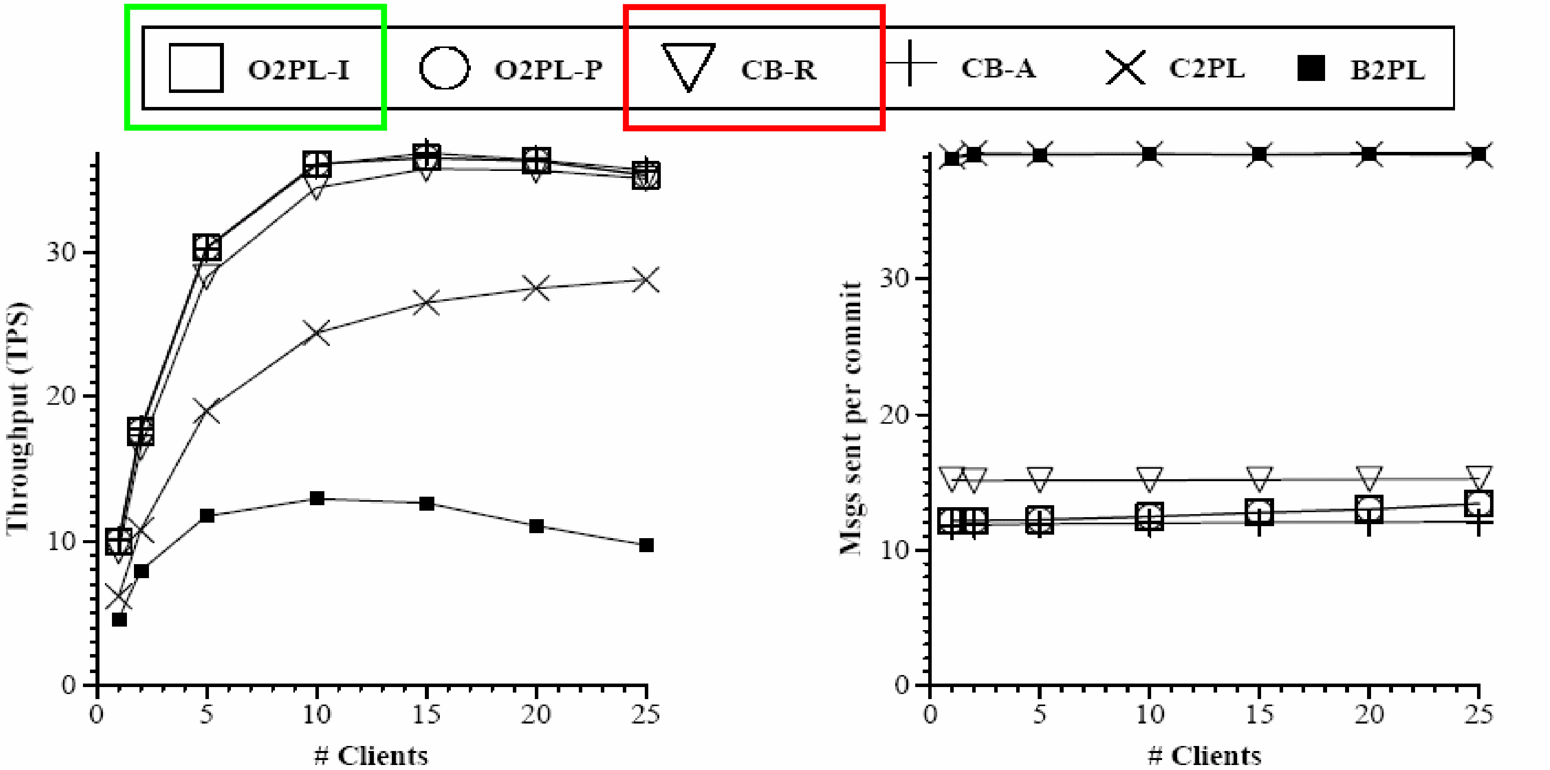


Figure 5: Throughput
(PRIVATE, 25% Client Cache, Slow Net)

Figure 6: Messages Sent/Commit
(PRIVATE, 25% Client Cache, Slow Net)

Private Model

Looser: Single Transaction

- With no data contention, CB-A never callbacks write permissions:
  - Lower message overhead

Tradeoff:

Single vs Multi-Xaction Write Permission Duration

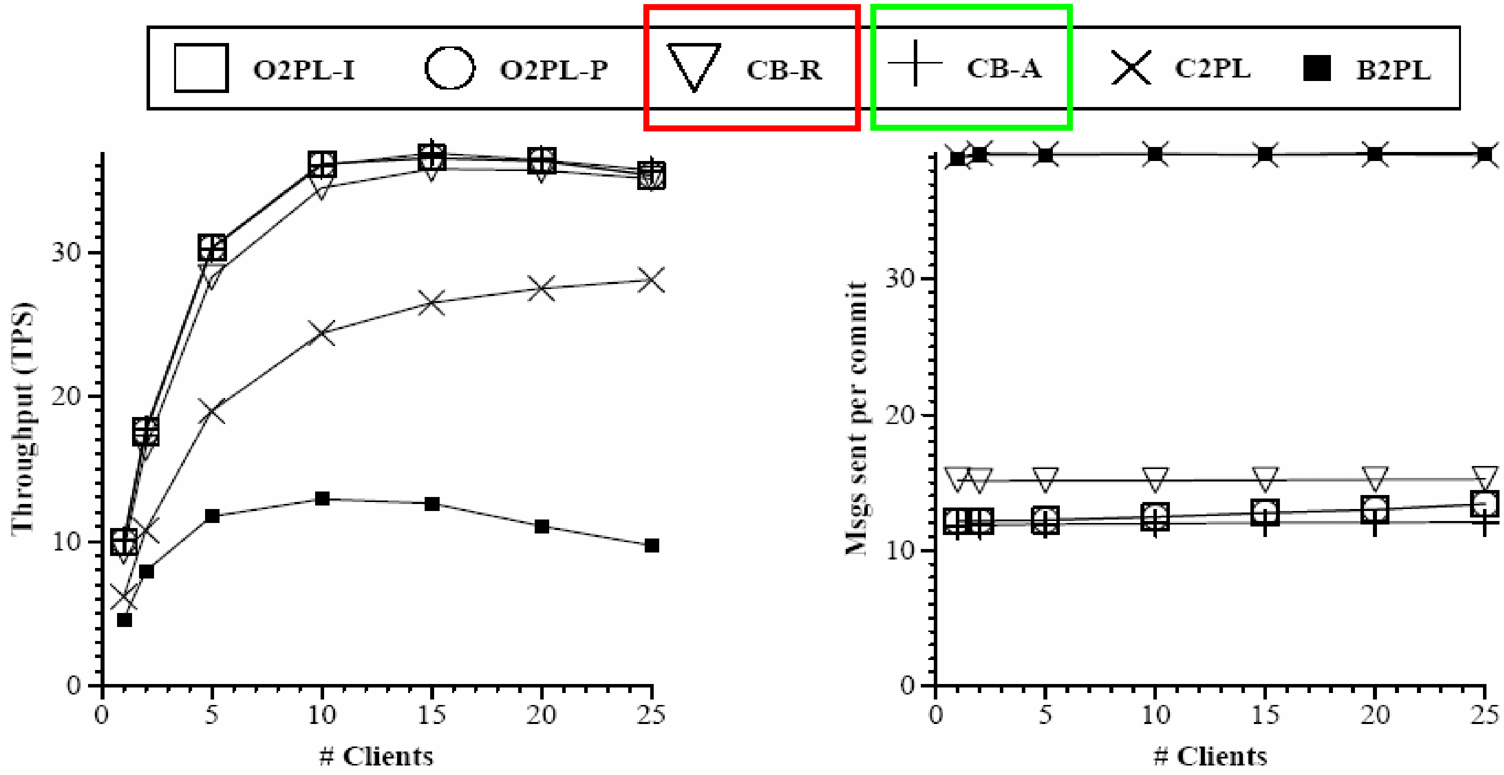


Figure 5: Throughput
(PRIVATE, 25% Client Cache, Slow Net)

Figure 6: Messages Sent/Commit
(PRIVATE, 25% Client Cache, Slow Net)

Private Model

--Tie!

- No apparent difference in absence of no read-write / write-write data conflicts:
  - no remote update ever occurs!

Tradeoff:

Invalidate vs Propagate

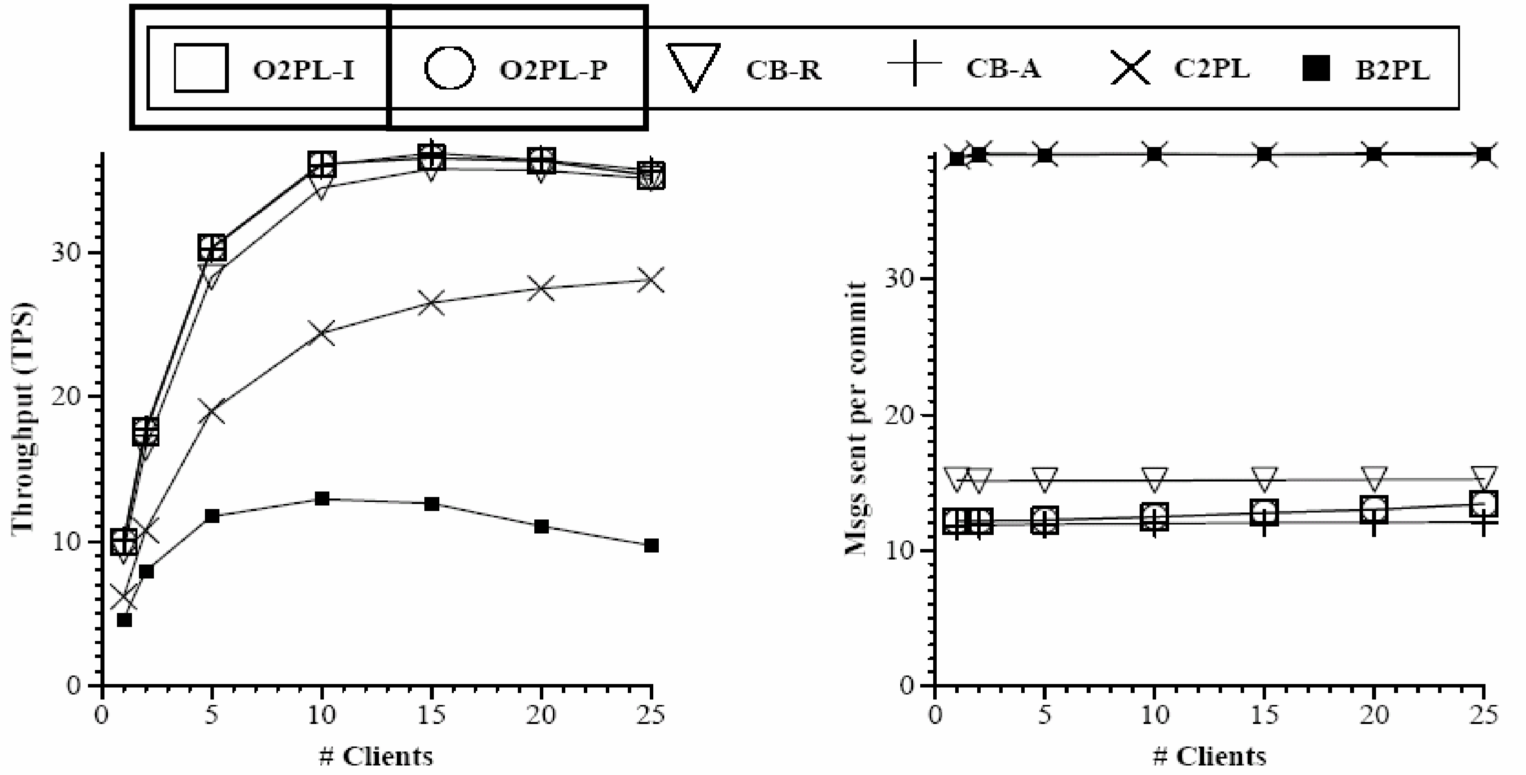


Figure 5: Throughput
(PRIVATE, 25% Client Cache, Slow Net)

Figure 6: Messages Sent/Commit
(PRIVATE, 25% Client Cache, Slow Net)

Hot-Cold Model

Results are similar to the Private Model, with some exceptions due to the introduction of read-write/write-write conflicts.

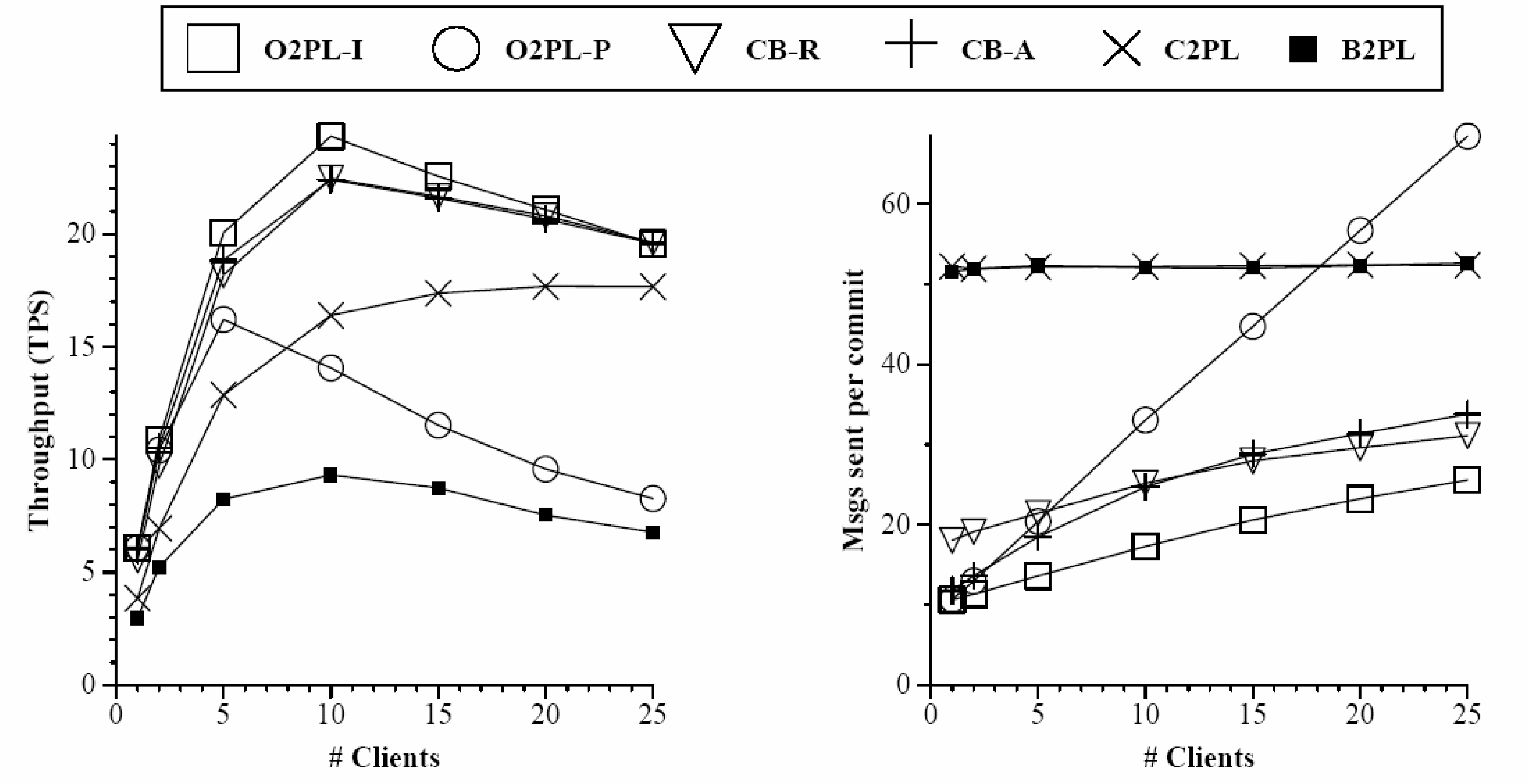


Figure 7: Throughput
(HOTCOLD, 25% Client Cache, Slow Net)

Figure 8: Messages Sent/Commit
(HOTCOLD, 25% Client Cache, Slow Net)

Hot-Cold Model

LOOSER: Detection

- High message overhead, but <u>constant</u>!
- Avoidance based approach requires remote update actions at client holding copies of updated items:
  - reduced scalability

Tradeoff: Detection vs Avoidance

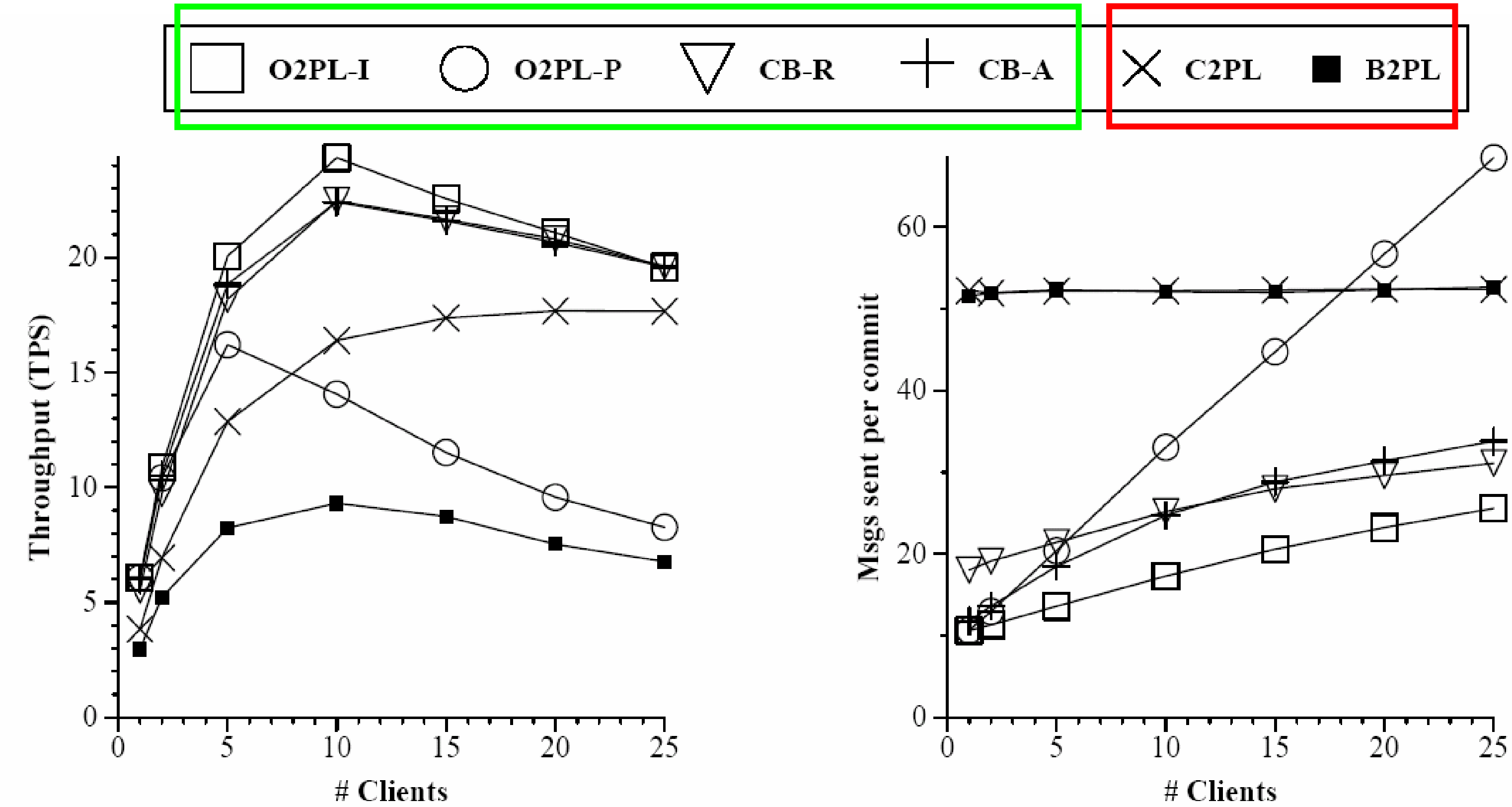


Figure 7: Throughput
(HOTCOLD, 25% Client Cache, Slow Net)

Figure 8: Messages Sent/Commit
(HOTCOLD, 25% Client Cache, Slow Net)

Hot-Cold Model

Tradeoff:
Synch vs Asynch Write Intention Timing

Looser: Synch

- Worse performance due to higher #msgs:
  - reduced difference when clients increase and the server disk becomes the bottleneck
- Few aborts due to deferred write intention: low data contention level

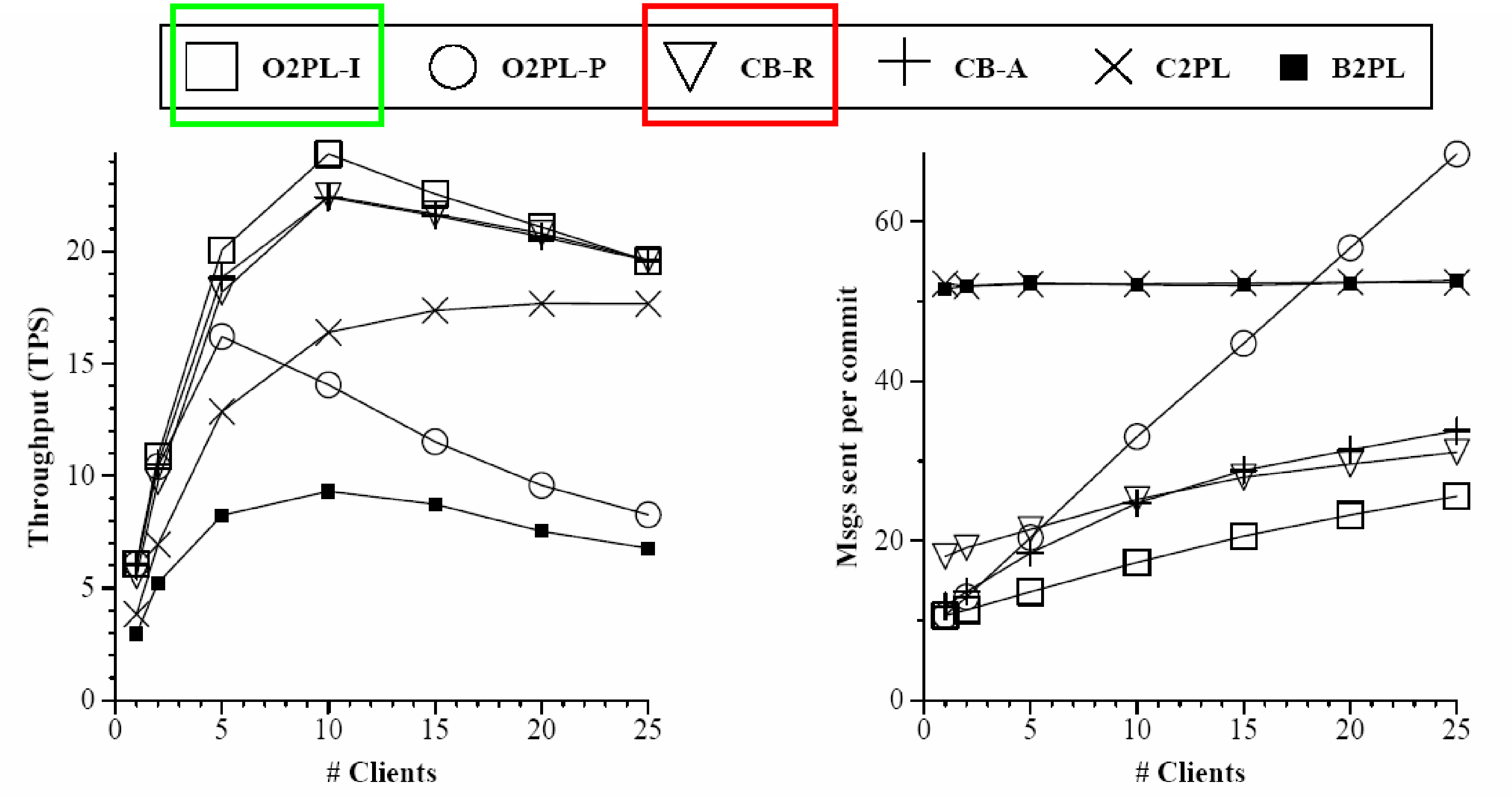


Figure 7: Throughput
(HOTCOLD, 25% Client Cache, Slow Net)

Figure 8: Messages Sent/Commit
(HOTCOLD, 25% Client Cache, Slow Net)

Hot-Cold Model

Looser: Single Transaction

- Few clients, lowest contention level:
  - CB-A saves msgs by retaining locks
- As clients increase, so does contention level:
  - CB-A ends up requiring more callbacks than CB-R

Tradeoff:

Single vs Multi-Xaction Write Permission Duration

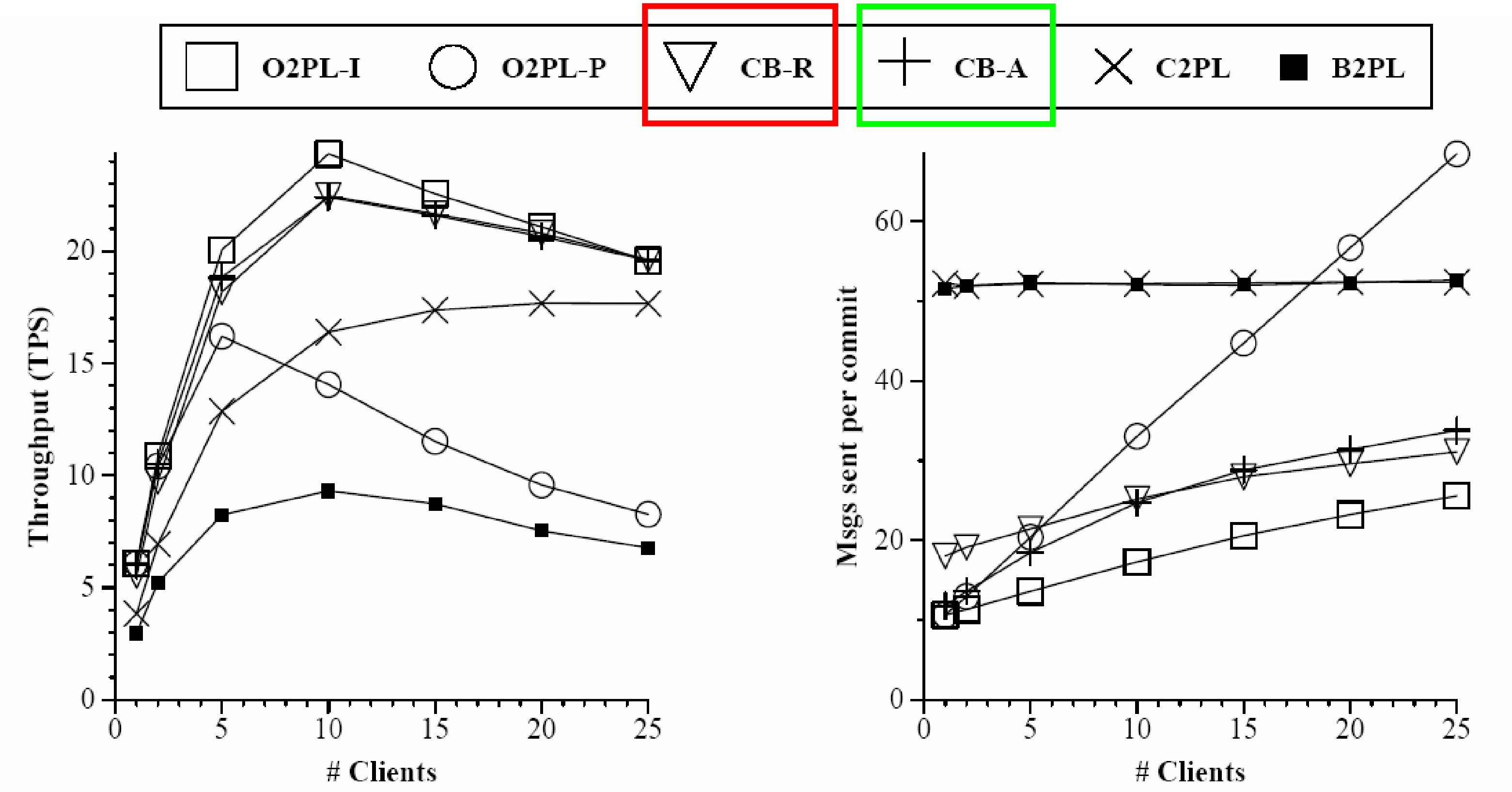


Figure 7: Throughput
(HOTCOLD, 25% Client Cache, Slow Net)

Figure 8: Messages Sent/Commit
(HOTCOLD, 25% Client Cache, Slow Net)

Hot-Cold Model

Tradeoff:

Invalidate vs Propagate

Looser: Update Propagation

- Much higher data traffic as clients increase
- At 25 clients:
  - 13 remote clients need updates
  - 120KB vs 43KB per commit
  - Many propagations are wasted:
    - re-propagated or dropped!

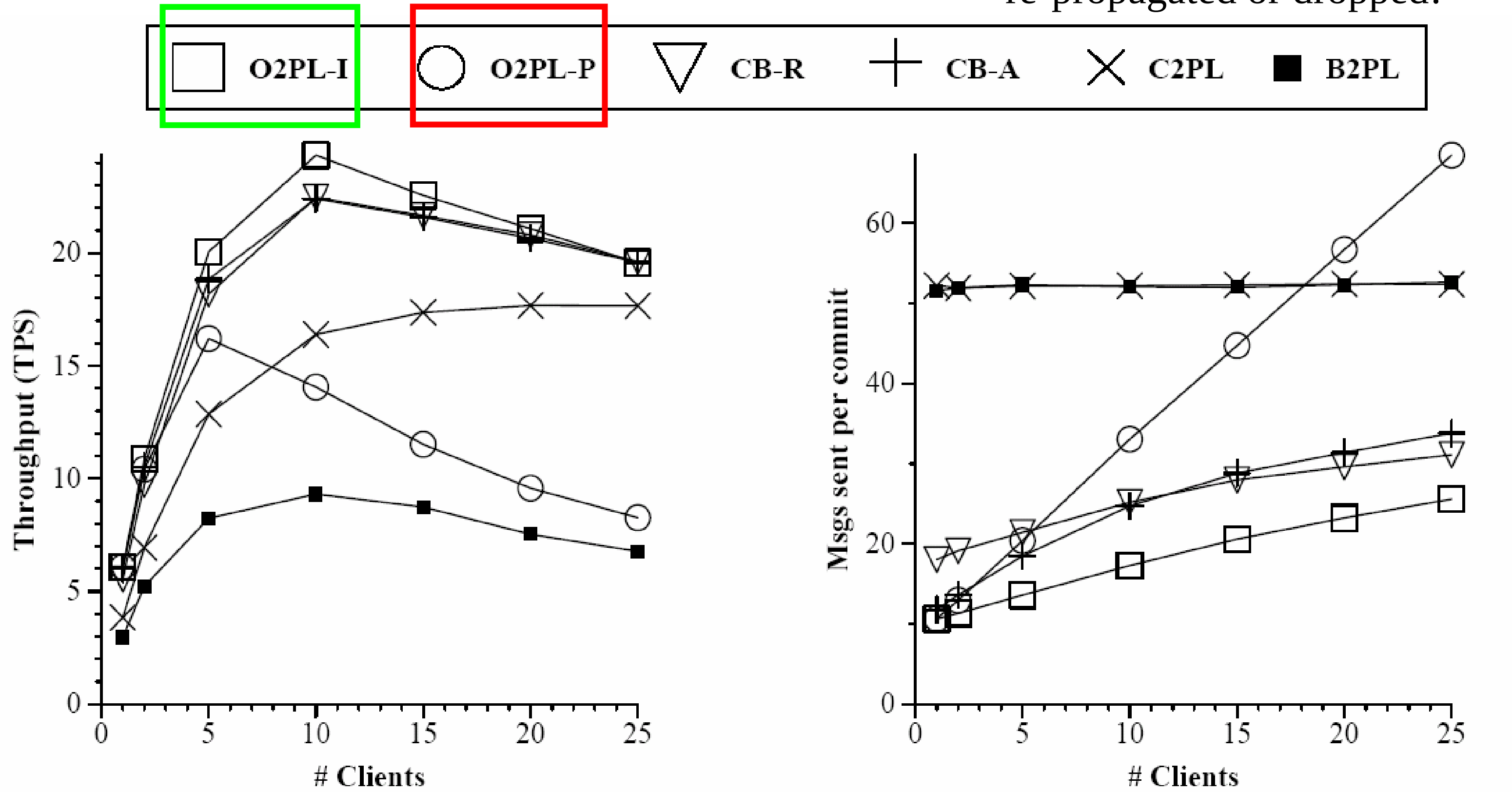


Figure 7: Throughput
(HOTCOLD, 25% Client Cache, Slow Net)

Figure 8: Messages Sent/Commit
(HOTCOLD, 25% Client Cache, Slow Net)

Uniform Model

No per-client locality: higher data contention, less benefits from caching

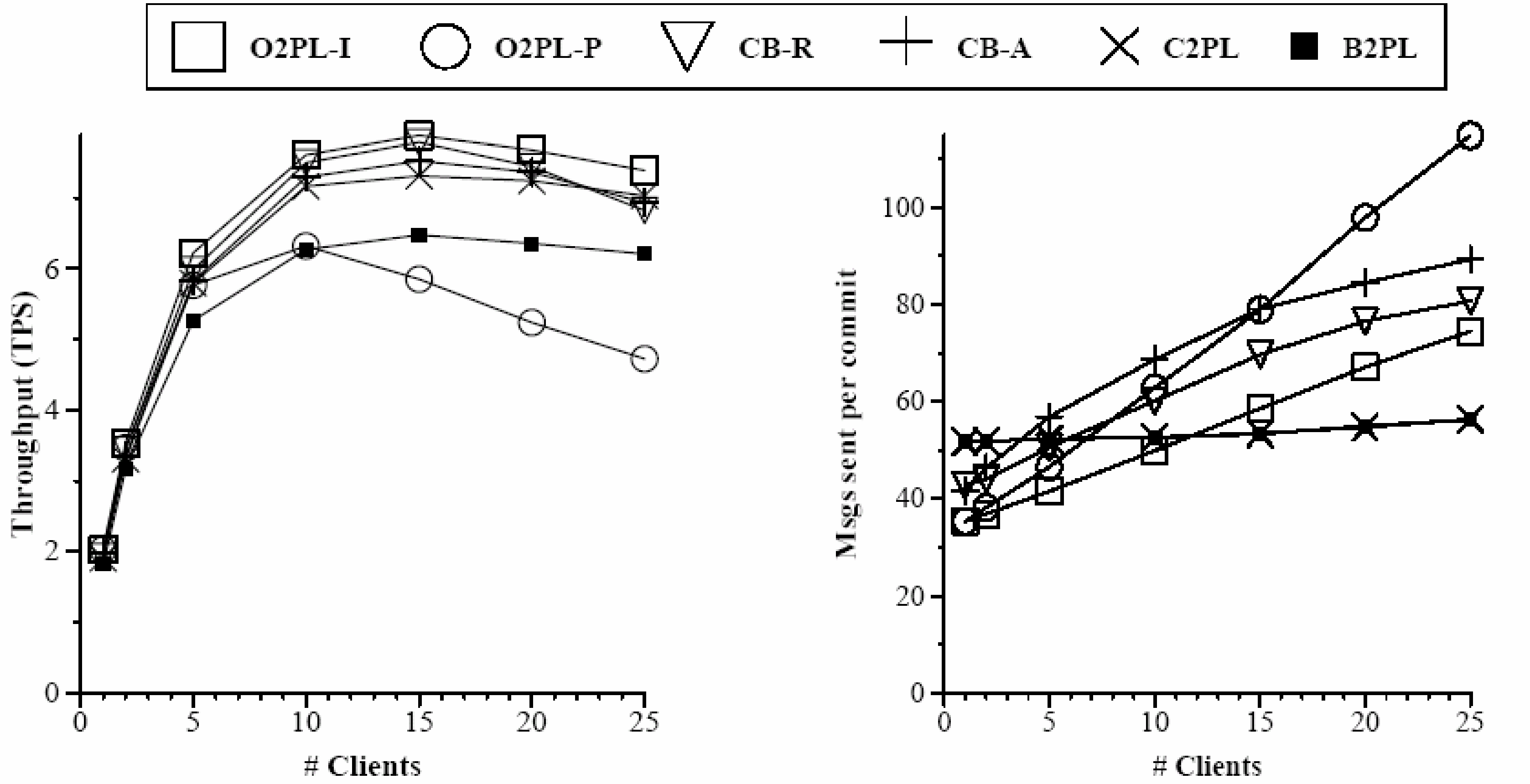


Figure 9: Throughput
(UNIFORM, 25% Client Cache, Slow Net)

Figure 10: Messages Sent/Commit
(UNIFORM, 25% Client Cache, Slow Net)

Uniform Model

Tradeoff: Detection vs Avoidance

LOOSER: Detection, but almost tie

- Avoidance based approaches require more msgs as clients increase:
  - CB-R/A require expensive callbacks which are useless in absence of (temporal) locality

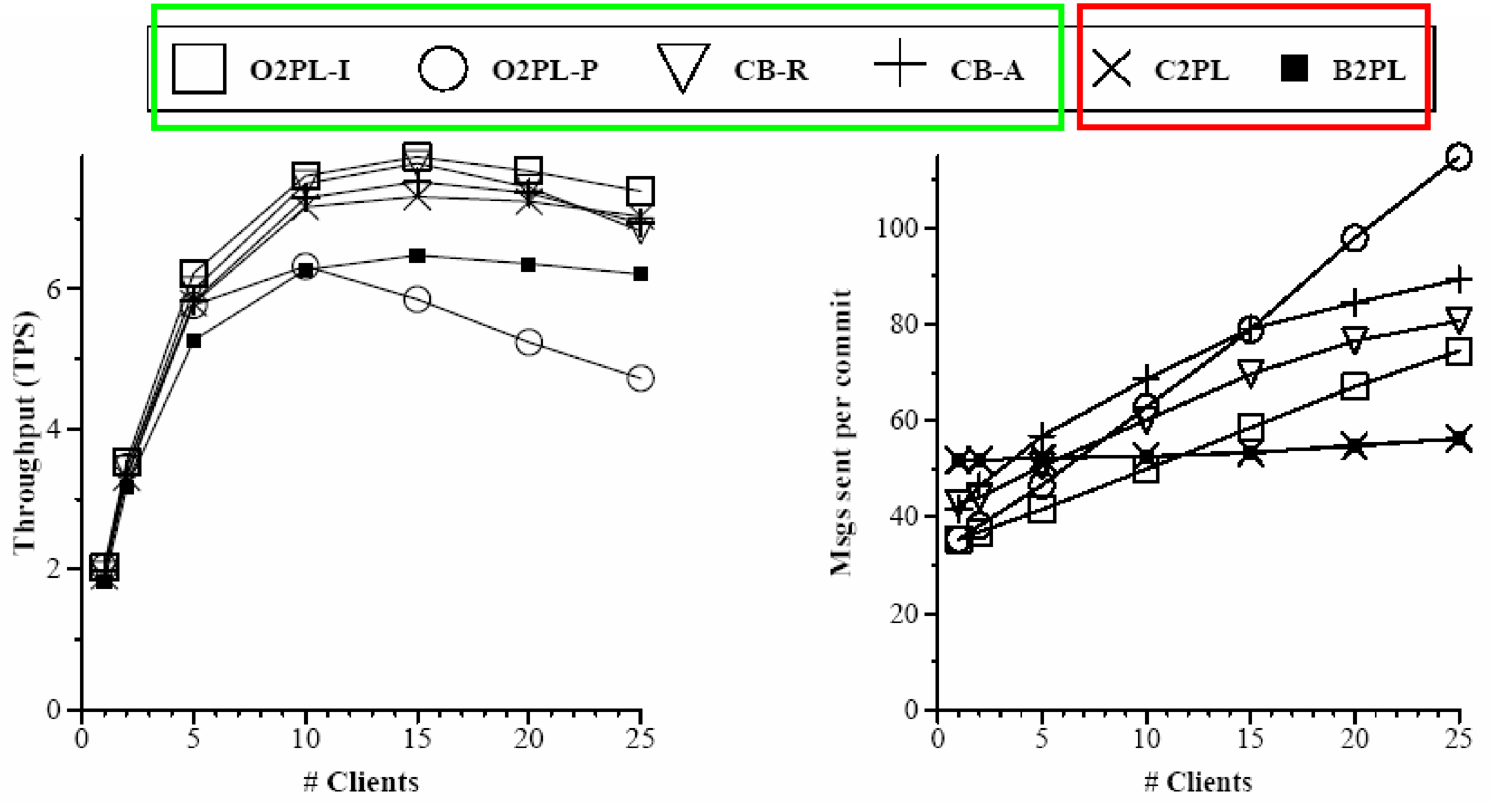


Figure 9: Throughput (UNIFORM, 25% Client Cache, Slow Net)

Figure 10: Messages Sent/Commit (UNIFORM, 25% Client Cache, Slow Net)

Uniform Model

LOOSER: Detection, but almost tie

- Detection causes lower hit rates, due to the presence of invalid data in the client caches.

Tradeoff: Detection vs Avoidance

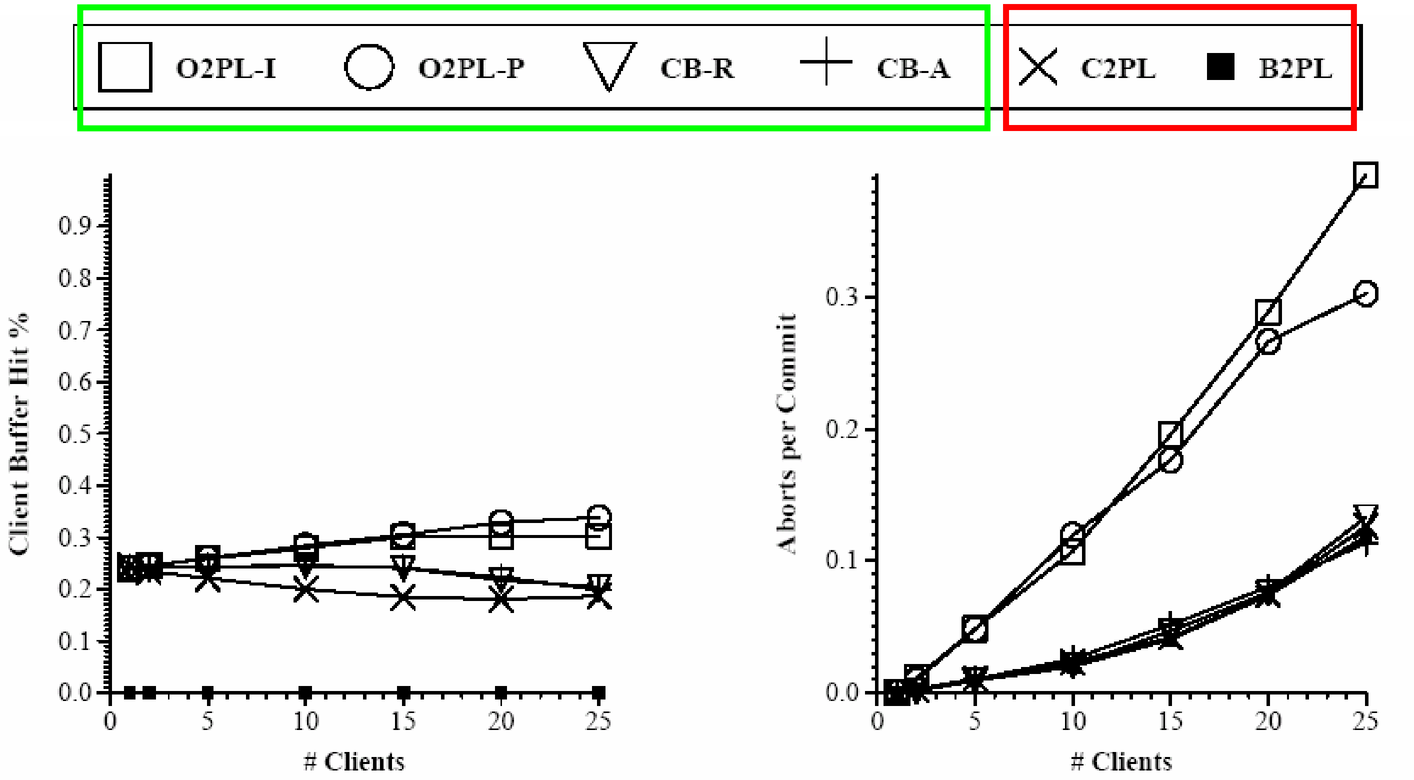


Figure 11: Client Hit Rate
(UNIFORM, 25% Client Cache, Slow Net)

Figure 12: Aborts/Commit
(UNIFORM, 25% Client Cache, Slow Net)

Uniform Model

Almost a tie….

Tradeoff: optimism vs pessimism

- O2PL-I/A incurs high abort rates (40%)
- O2PL-I still performs well due to cache hits as transactions re-run: low abort cost!

Tradeoff:

Synch vs Asynch Write Intention Timing

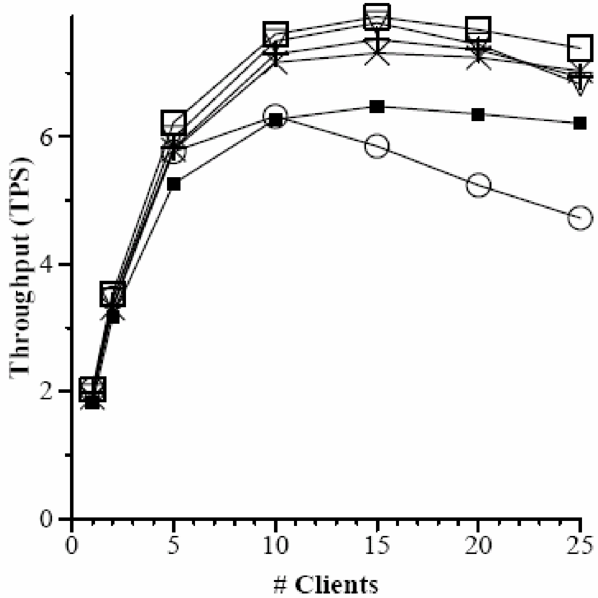


Figure 9: Throughput
(UNIFORM, 25% Client Cache, Slow Net)

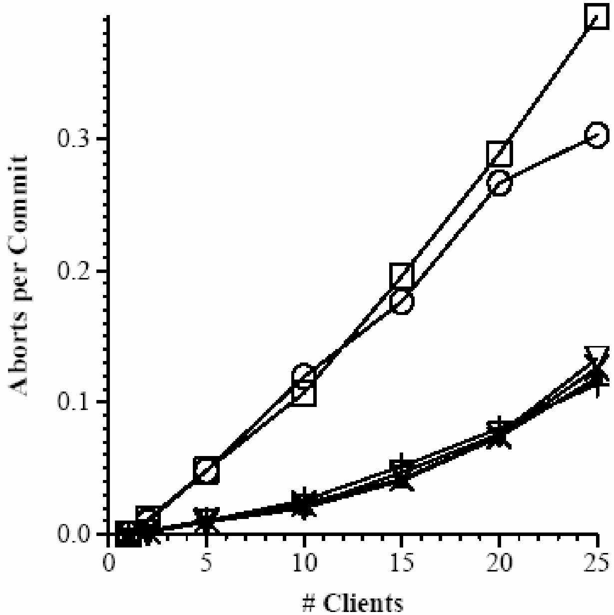


Figure 12: Aborts/Commit
(UNIFORM, 25% Client Cache, Slow Net)

Uniform Model

Looser: Multi-Transaction

- CB-A requires more messages than CB-R, since we're in a low locality scenario:
  - Retaining write permissions across transactions is expensive (due to subsequent callbacks) if data are not likely to be written again locally

Tradeoff:

Single vs Multi-Xaction Write Permission Duration

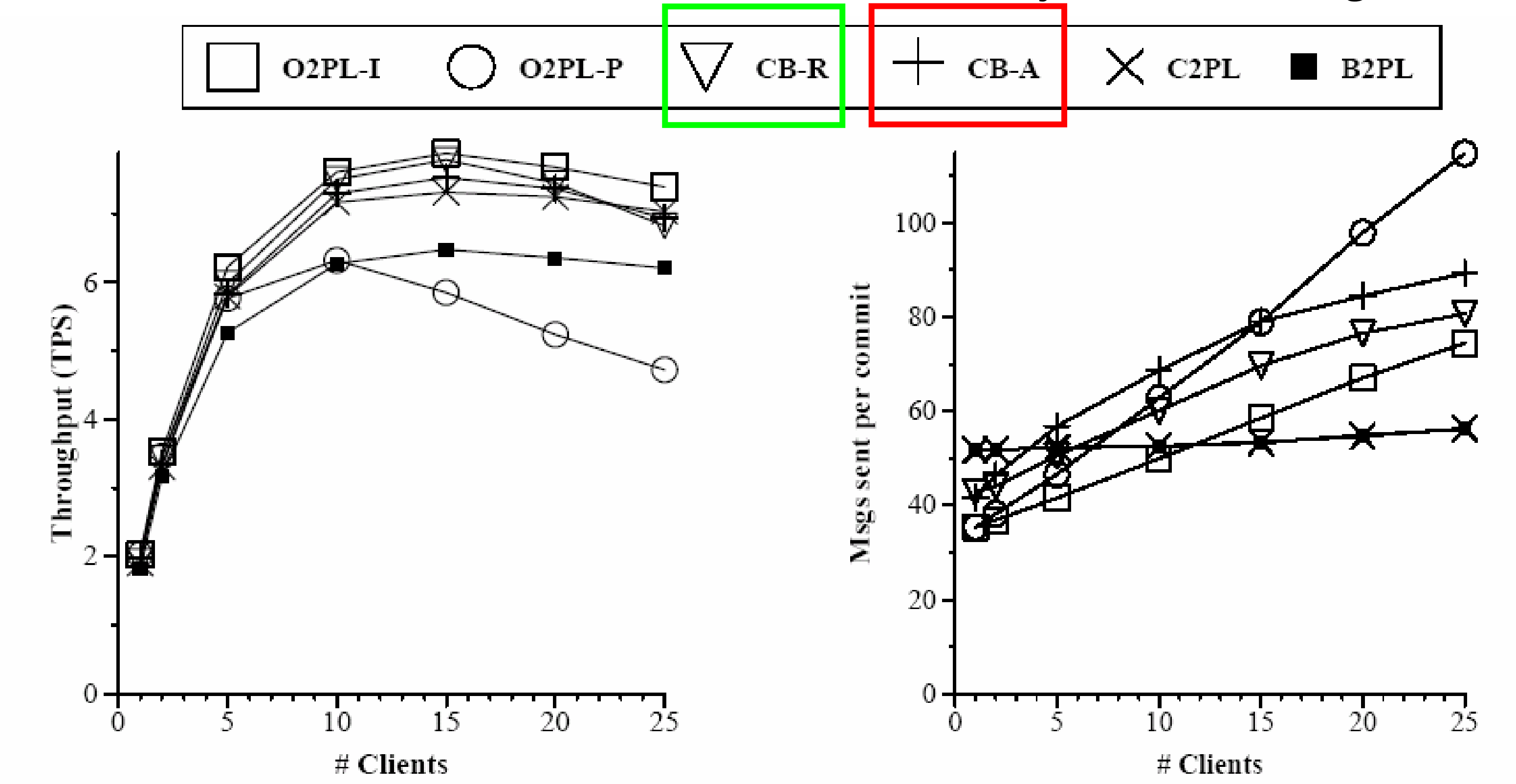


Figure 9: Throughput (UNIFORM, 25% Client Cache, Slow Net)

Figure 10: Messages Sent/Commit (UNIFORM, 25% Client Cache, Slow Net)

Uniform Model

Looser: Update Propagation

- Like in previous scenarios propagation produces much higher data traffic as clients increase

Tradeoff:

Invalidate vs Propagate

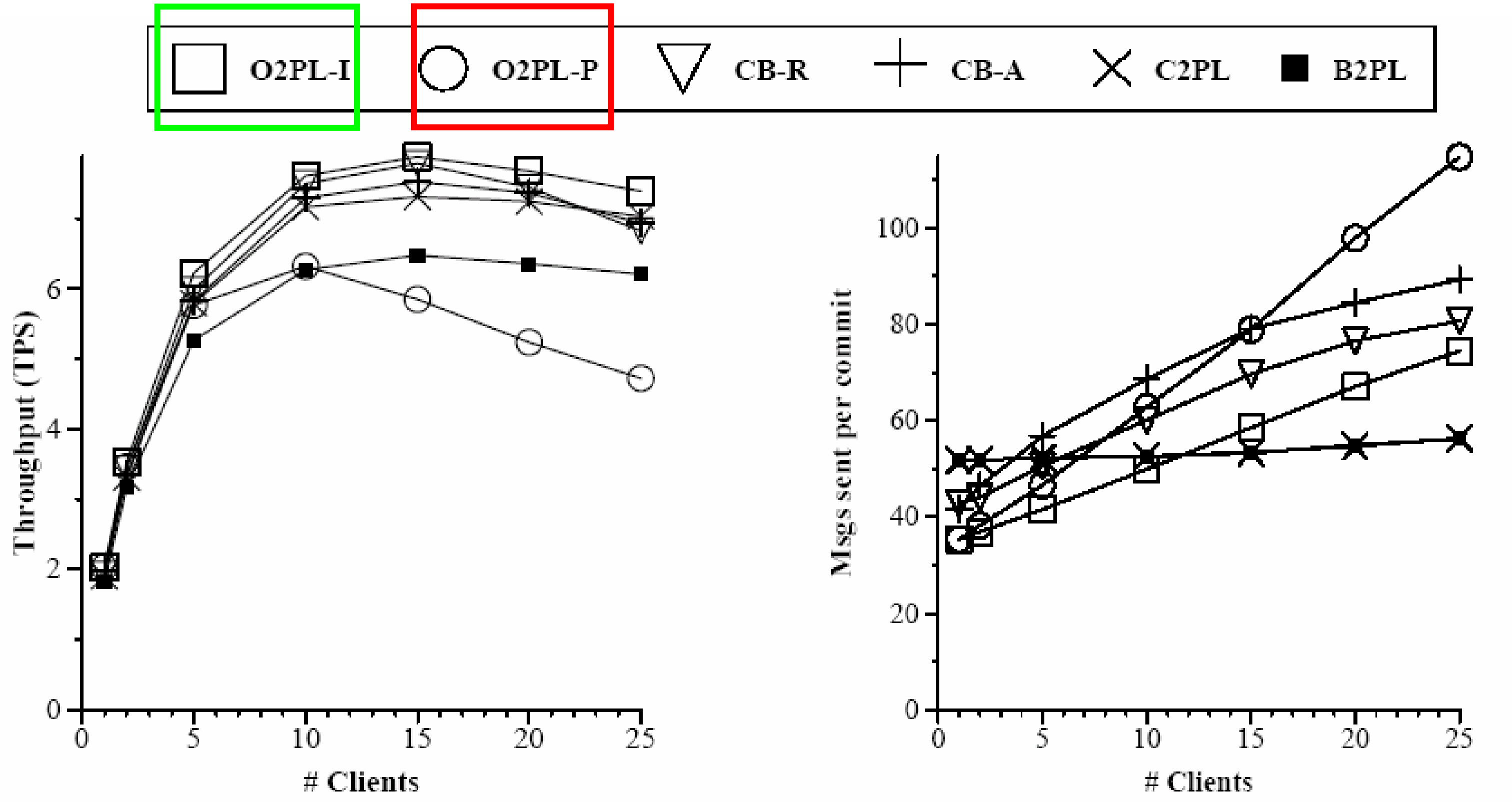


Figure 9: Throughput
(UNIFORM, 25% Client Cache, Slow Net)

Figure 10: Messages Sent/Commit
(UNIFORM, 25% Client Cache, Slow Net)

Feed Model

Single writer, many readers: here update propagation pays off:

- **increased cache hit rate**
- **few wasted propagations due to high locality in clients accesses**

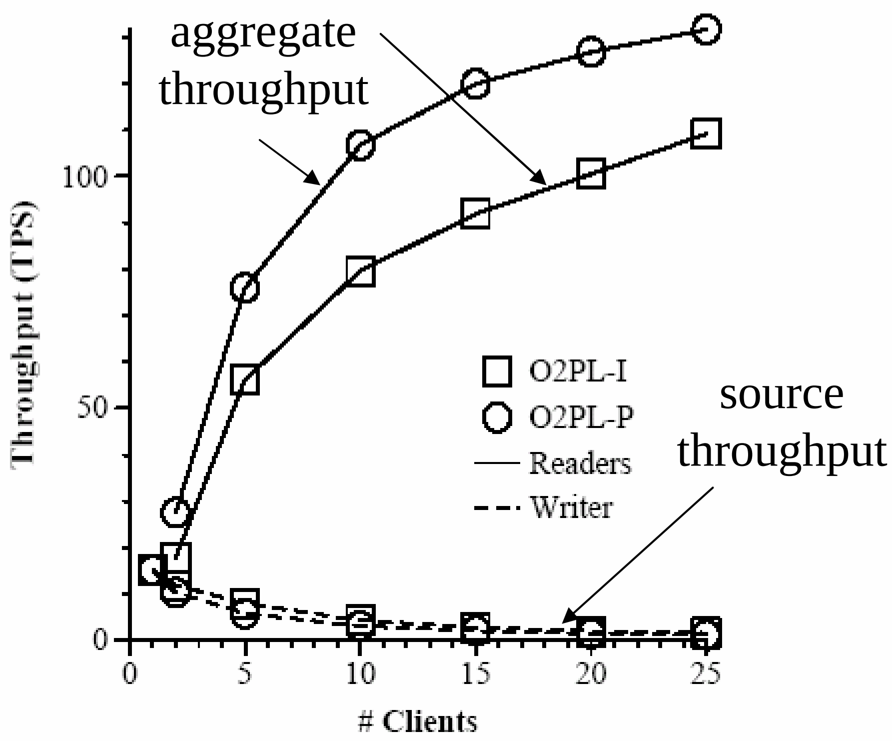


Figure 13: Throughput
(FEED, 25% Client Cache, Slow Net)

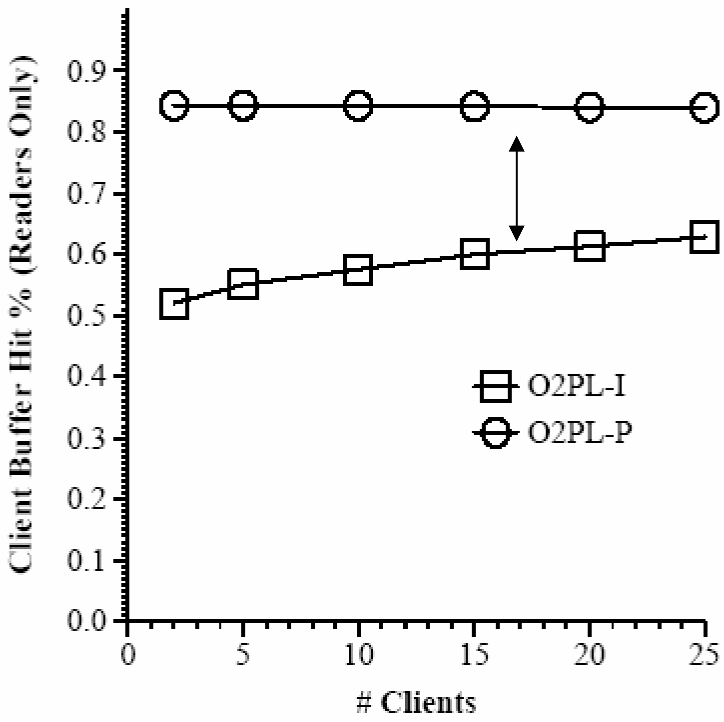


Figure 14: Client Cache Hit % (Readers only)
(FEED, 25% Client Cache, Slow Net)

Overall considerations

- Detection vs Avoidance:
  - considered detection-based approaches are pessimistic (on access detection) only:
    - This keeps the abort-rate low, but strongly increases the message traffic & dependence on server
    - Anyway, message traffic is roughly independent on the number of clients
    - More optimism (deferred validity check initiation, e.g. at commit time) would have:
      - Consistenly reduced the exchanged messages
      - Increased the abort rate in high contention
      - It can be shown [Adya95] that in low contention scenarios optimistic detection based approaches outperform avoidance based approaches

Overall considerations

- Detection vs Avoidance:
  - A noteworthy side-effect of detection based algorithms is that, allowing invalid pages in client caches, they typically achieve lower hit rates:
    - “Effective” cache size is reduced by invalid pages in detection based alg.
    - Avoidance-based ones avoid caching invalid pages and end up in high contention scenarios with more empty (i.e. usable) slots.

Overall considerations

- Write Intention Declaration (O2PL vs CB):
  - Pessimism vs Optimism tradeoff in avoidance based algorithms
  - No sharing:
    - same performance
  - Limited sharing:
    - Optimism wins: less msgs thanks to batching at commit
  - Higher sharing & contention:
    - Optimistic approaches lead to high transaction abort rates:
      - which may be unacceptable in interactive applications
      - in the simulation abort cost is rather low (cache hit upon restart)

Overall considerations

- Write Permission Duration:
  - High contention levels + low locality make unworthy retaining write permission across transactions:
    - Such an effort pays off only in case a page is more likely to be written locally than read remotely!
- Remote Update Action:
  - Update propagations can lead to high resource wastage and is highly sensitive to the contention level
  - Invalidation seems the best choice in the majority of cases
  - Adaptive approaches were also proposed.

Overall considerations

- There’s no winning solution for all the possible workload scenarios:
  - Reduced contention levels make “optimistic” approaches more attractive in general, but…
  - at higher contention levels too much “optimism” translates into high abort rates!
  - General purpose DBMS must provide good performance in <u>all</u> the workload scenarios:
    - <u>Need for robust solutions</u>!

Granularity of Consistency Actions

- Consistency actions (callbacks/lockings) can take place either for each accessed row/object or at the page level:
  - **Page granularity:**
    - + reduced message overhed in case of spacial locality
    - - false conflicts may be detected
  - **Object granularity:**
    - Exactly the opposite!
  - **Adaptive solutions:**
    - Normally use page granularity
    - If a read-write conflict is detected, switch to object granularity

What we did not cover…

- Geographically distributed transactional cache schemes:
  - Performance study was focused on LAN environments…
  - What if network latencies get predominant and highly variant?
  - What if we need to scale to thousands of clients?
    - e.g. edge server performing caching of data originally hosted at the origin site DBMS

<u>Open Research Questions</u>