Lecture 7 - Concurrency Control Evaluation

Setup and Intro

- 1985: primary mechanisms for CC were:
  - Locking
  - Timestamp ordering and variants (MVTO)
  - Optimistic concurrency control (validation, certification)
- Contradictory results
  - Carey & Stonebraker (VLDB84), Agrawal & DeWitt (TODS85): blocking beats restarts
  - Tay (Harvard PhD) & Balter (PODC82): restarts beat blocking
  - Franaszek & Robinson (TODS85): optimistic beats locking
- Goal of this paper:
  - go beyond "yet another performance study" - study assumptions in the underlying models and their implications
    - Importantly, **capture causes of previous conflicting results**
  - a complete and "realistic" model of DBMS; captures
    - users
    - system characteristics
    - workload characteristics
- Why are we studying this paper?
  - Tradition of using simulation for assessing database performance
  - Strong in the empirical tradition of chasing down causes

Approaches studied

- Goal: capture extremes
- Approach 1: Blocking/2PL (BL)
  - In case of deadlocks, construct waits-for graph
  - Deadlock detection runs when a txn blocks
  - Youngest txn in cycle is aborted
  - Detour to revisit Deadlocks section in Lecture 6 - Concurrency Control Schemes
    - At some granularity, deadlock detection analogous to timestamping
  - Arguably simpler, but more expensive approach compared to those sections.
- Approach 2: Immediate restart (IR)
  - If a transaction blocks, it is restarted after a delay
  - Delay is order of expected response time of a txn
    - Q: Determined how?
- Approach 3: OCC (O)
  - Txns execute unhindered and validated when reaching commit points - restarted if any object it has read has been written by another txn which committed during its lifetime
  - Detour to talk about OCC
    - Kung and Robinson paper
    - Attractive, simple idea: optimize case where conflict is rare.
    - All transactions consist of three phases:
      1. Read. Here, all writes are to private storage (shadow copies).
      2. Validation. Make sure no conflicts have occurred.

3. Write. If Validation was successful, make writes public. (If not, abort!)

- When might this make sense? Three examples:
  - All transactions are readers.
  - Lots of transactions, each accessing/modifying only a small amount of data, large total amount of data.
  - Fraction of transaction execution in which conflicts "really take place" is small compared to total pathlength.
- Validation phase guarantees that only serializable schedules result.
- Basically:
  1. Assign each transaction a TN during execution.
  2. Ensure that if you run transactions in order induced by "<" on TNs, you get an equivalent serial schedule.
- Intuitively, at validation, Tj "checks its elders"
- Two options:
  - Backward Validation: Check whether the committing txn Tj intersects its read/write sets with those of any txns that have already committed.
    - More common
  - Forward Validation: Check whether the committing txn Tj intersects its read/write sets with any active txns that have not yet committed.
- If TN(Ti) < TN(Tj) then one of the following must hold
  - Option 1:
    - Ti: |... R ...| ... V ... | ... W ...|
    - Tj: ..... |... R ...| ... V ... | ... W ...|
    - True serial execution
  - Option 2:
    - Ti: |... R ...| ... V ... | ... W ...|
    - Tj: ..... |... R ...| ... V ... | ... W ...|
    - Here, WS(Ti) ∩ RS(Tj) = ∅ and Ti completes W before Tj starts W
  - Option 3:
    - Ti: |... R ...| ... V ..... | ... W ...|
    - Tj: ..... |..... R .....| ... V .....| ... W ...|
    - Here, WS(Ti) ∩ RS(Tj) = ∅ and WS(Ti) ∩ WS(Tj) = ∅ and Ti completes its R before Tj completes its R
      - Why this last condition?
        - Because `Tj` cannot read values written by `Ti` (since `Ti` hasn't written yet).
      - But this can be improved with TWR

- Why extremes:
  - BL+IR vs OCC:
    - BL and IR detect conflicts immediately, OCC does not
  - IR+OCC vs. BL:
    - BL restarts only when necessary, IR and OCC restart always
  - OCC+BL vs. IR:
    - IR will delay restarted txn, while OCC and BL doesn't need to do so - this txn has been committed (for OCC), and same deadlock may not appear (for BL)

Simulation Model

- Carefully done - will not go in detail.
- Vary things like
  - database system model: hardware and software model (CPUs, disks), size & granule of DB, load control mechanism, CC algorithm
  - user model: arrival of user tasks, nature of tasks (e.g. batch vs. interactive)

    - transaction model: logical reference string (i.e. CC schedule), physical reference string (i.e. disk block requests, CPU processing bursts).
- Parameters mentioned:
    - sizes of transaction (percentage of objects accessed)
    - prob write | read
    - number of users
    - number of concurrent txns admitted
    - time between transactions per user
    - think time
    - IO, CPU time
    - num CPU, disks
- Q: What makes this simulation not faithful to the real world?
    - Sizes of transactions, skew, ...

Results

- Three settings:
    - very low conflicts
    - infinite resources
    - limited resources
    - interactive workload
- Lesson for experimental work: "20 batches with a large batch time to produce sufficiently tight 90% confidence intervals"
- Measurements:
    - measure throughput (tps), mostly
    - pay attention to variance of response time, too
    - try to explain through other metrics:
        - blocking ratio - how many times does a txn need to block to commit
        - restart ratio - similar
    - pick a DB size so that there are noticeable conflicts (else you get comparable performance)

Low conflicts

- Not very interesting, as all schemes perform the same (up to a point!)

Some conflicts, Infinite resources

- Expectation: throughput should increase as we increase multiprog level
    - however, conflicts also increase
    - for blocking: increased conflicts -> more blocking, more deadlocks, more restarts
    - for restart-oriented: more restarts
- Actual figure:

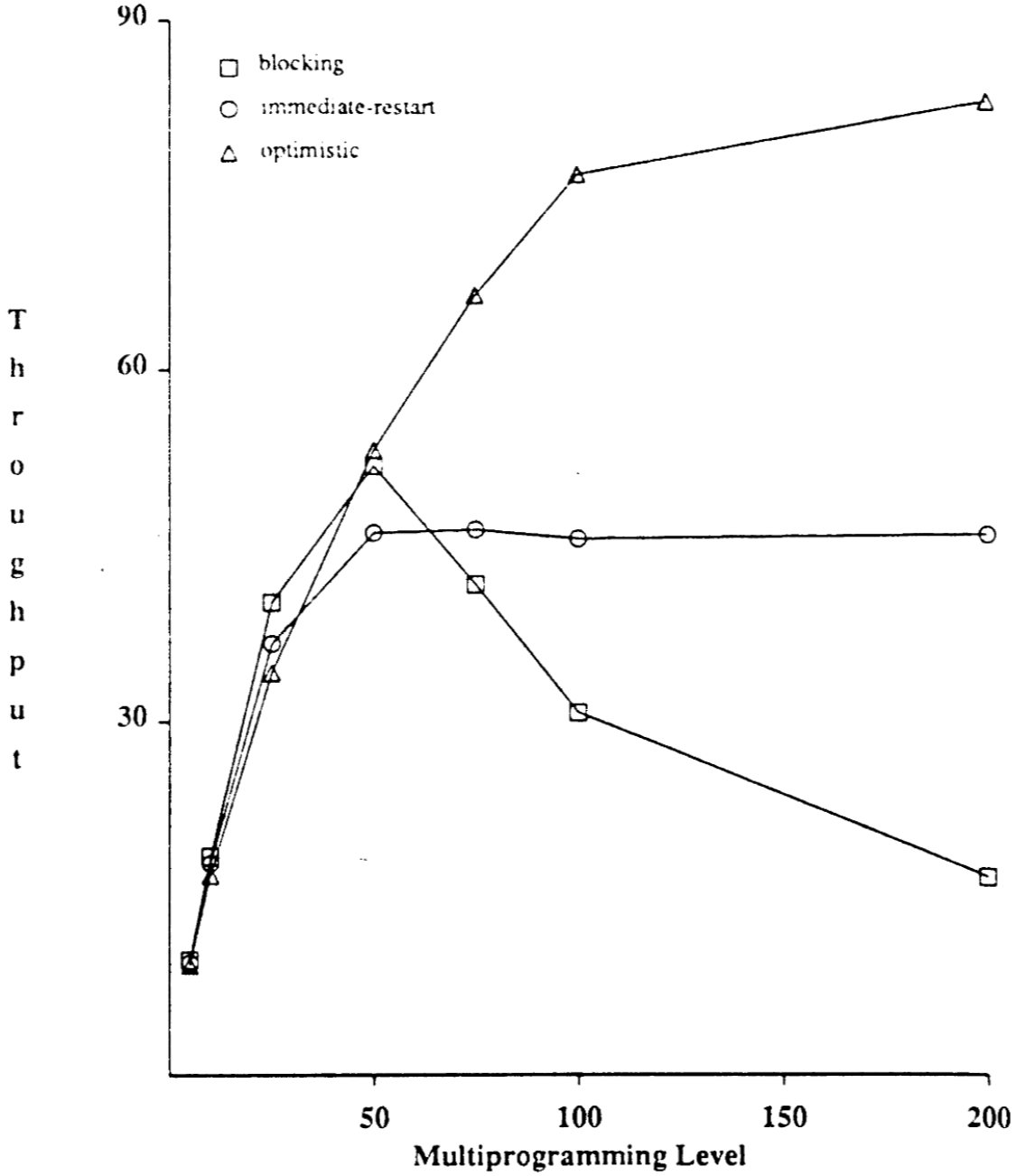


Fig. 5. Throughput (∞ resources).

- Blocking starts thrashing beyond a point, but OCC's throughput keeps increasing logarithmically, and IR becomes stable beyond a point

- Explanation through conflict ratios:

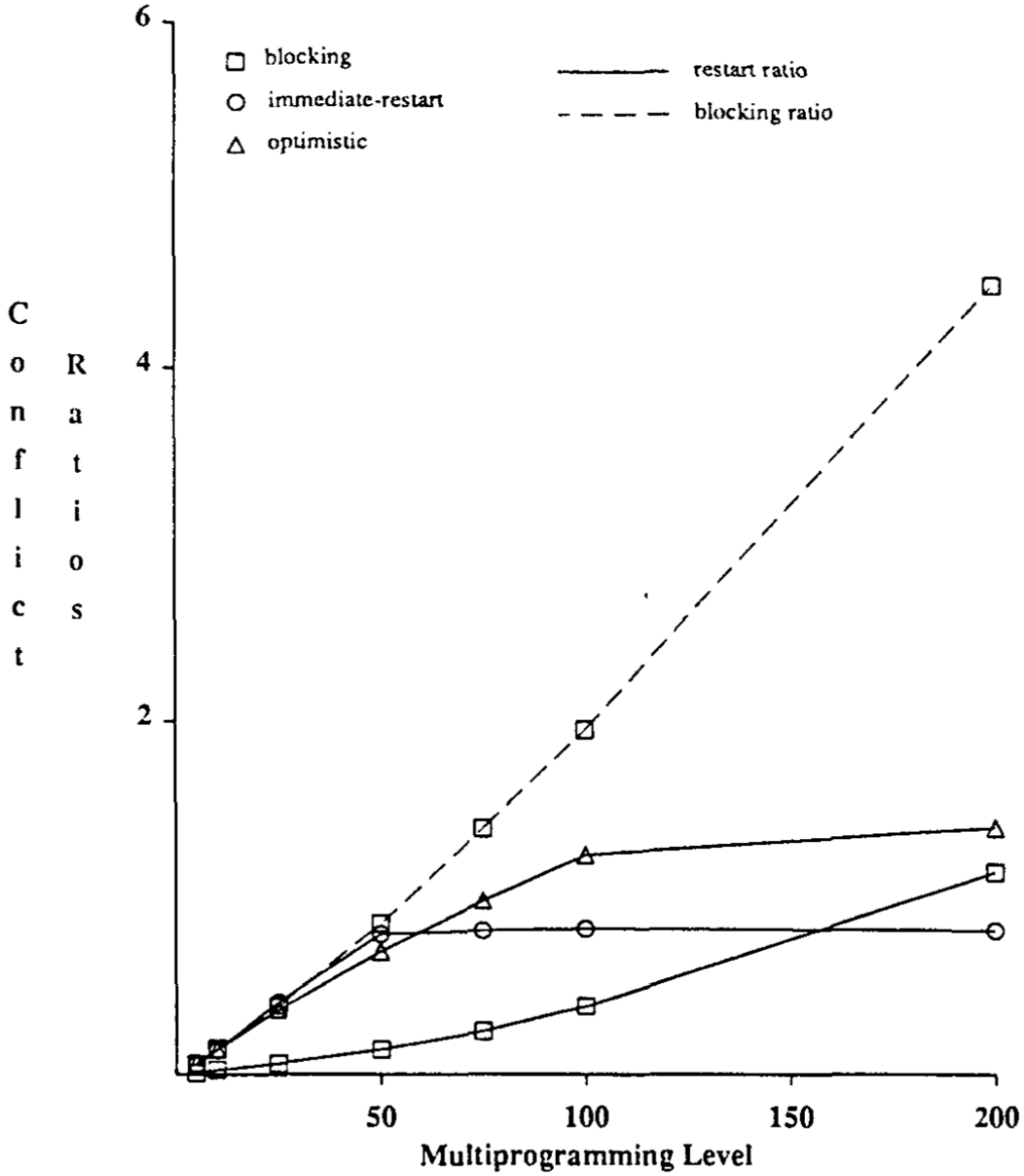


Fig. 6. Conflict ratios (∞ resources).

- Thrashing in blocking is because of more times a transaction gets blocked, reduces transactions that are running and make forward progress - *rather than restarts* (which are lower than other schemes)
- For OCC, while restarts do increase as conflicts increase, new transactions take the place of restarted ones, keeping MP level high
- For IR, throughput plateaus - because the delay is proportional to average response time.
  - hits stable state where all txns that are not active are waiting or waiting to issue next txns
  - "the number of active transactions in the system is such that a new transaction is basically sure to conflict with an active transaction and is therefore sure to be quickly restarted and then delayed."
  - restarts increase delay time which in turns reduces active txns, etc. reaching "plateau" - no txns are ready and waiting hence no impact of increasing MP
  - a form of primitive load control!

- Response time:

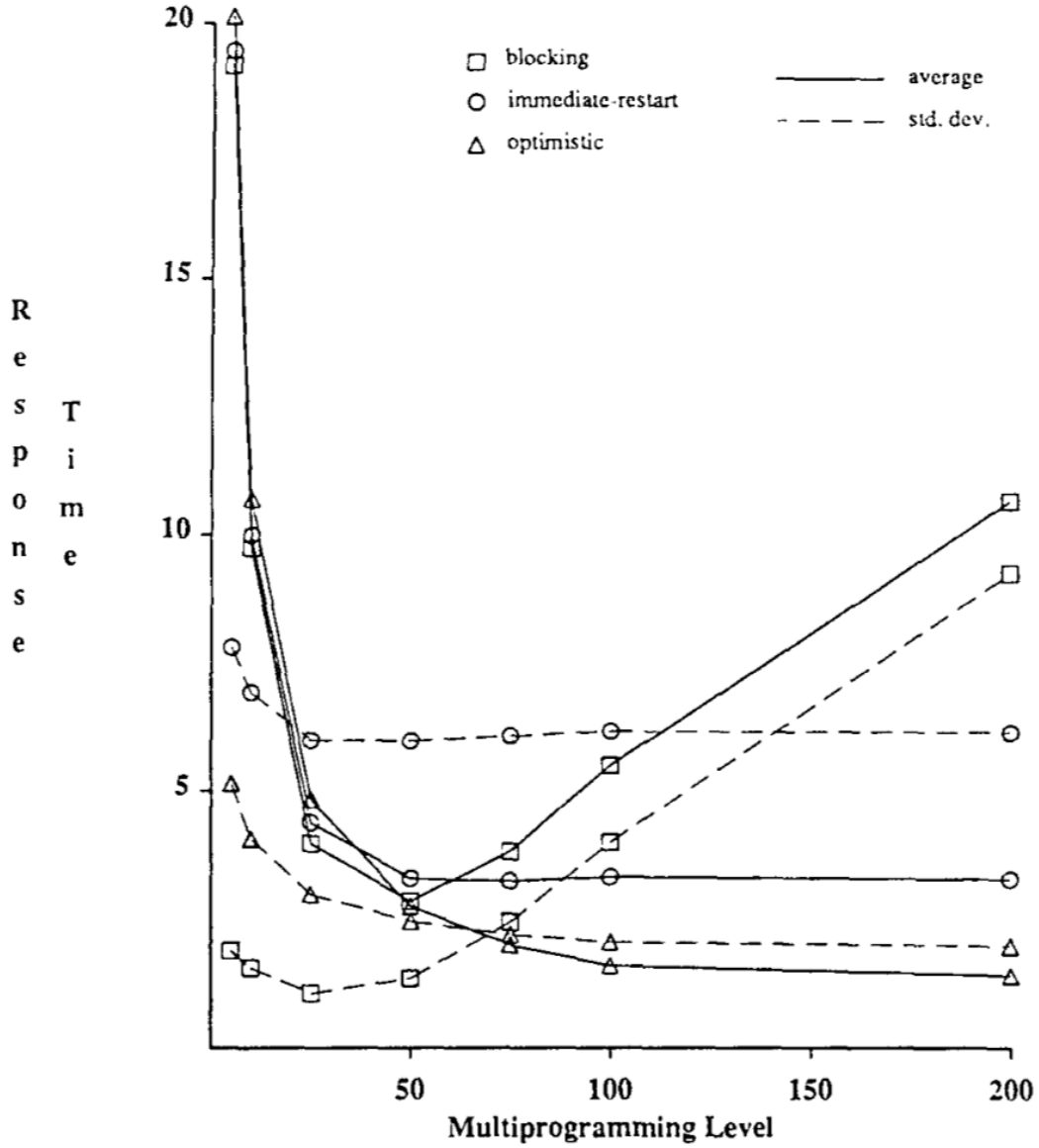


Fig. 7. Response time (∞ resources).

  - Std dev for blocking is lower in general than IR, and also for OCC until blocking performance starts degrading.
    - i,e., earlier on IR > OCC > Blocking
    - We want std-dev to be small - why?
  - Q: Why?
    - IR has large response time variation due to restart delay - k restarts - means k delays and diff txns have diff no. of restarts - also full rerun of txn as opposed to blocking which just tries to resolve deadlocks as opposed to rerun txns entirely
    - OCC restarts too but does not add a delay
    - Blocking doesn't restart as much (lock wait times are not as long as restart times) - so avoids additional work
- phew, already learned a lot!

Resource Limited Situation

- Figure explains it

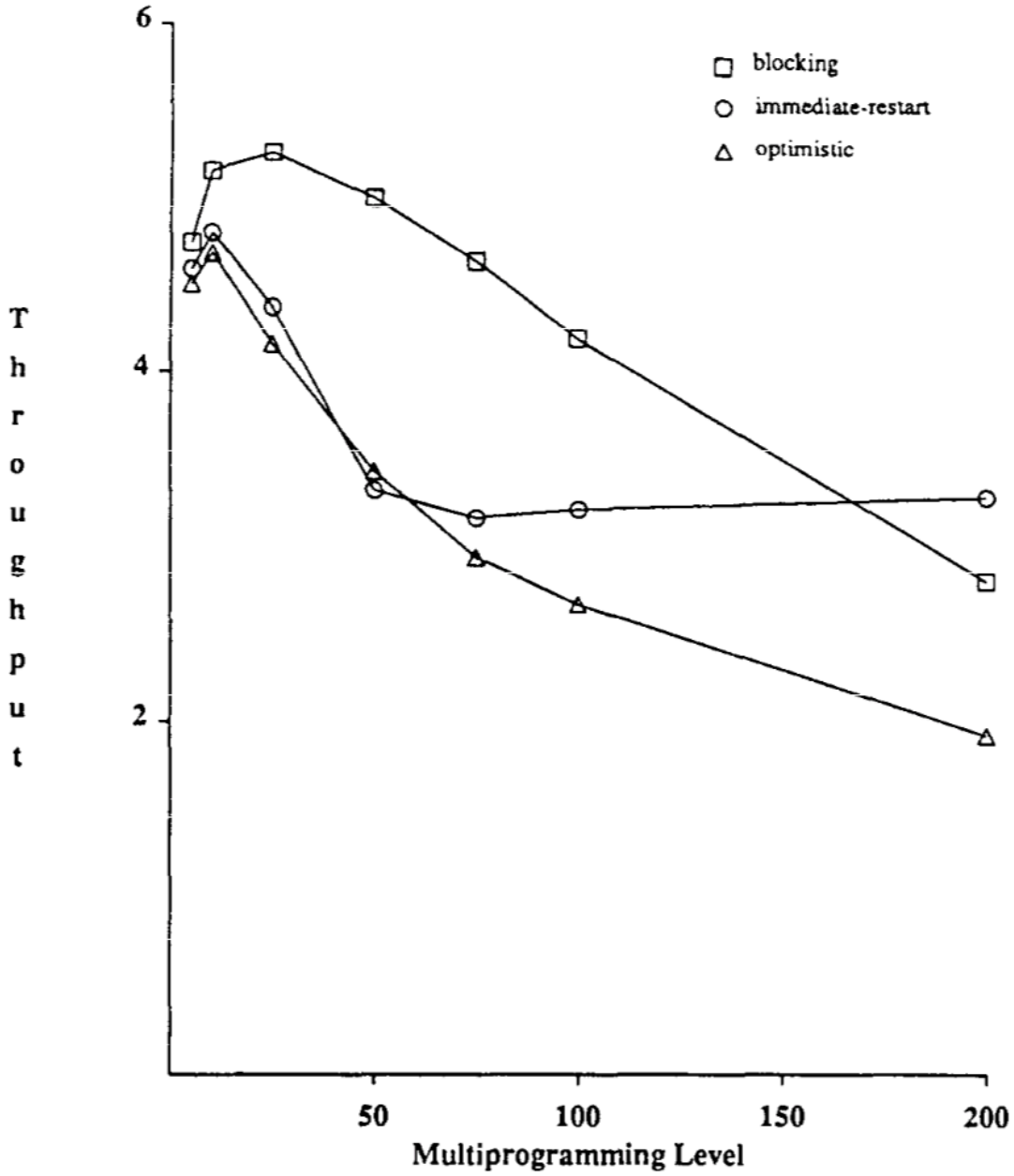


Fig. 8. Throughput (1 resource unit).

-
- ...

- All approaches indicate thrashing - some increase in throughput as level is increased, then decreases
  - throughput increased first, since there were not enough txns to keep resources utilized
  - but then you still want the resources to be "usefully" utilized, e.g., due to contention, the resources that are consumed by txns that are restarted or blocked are not useful
  - direct correlation between usefulness and performance:

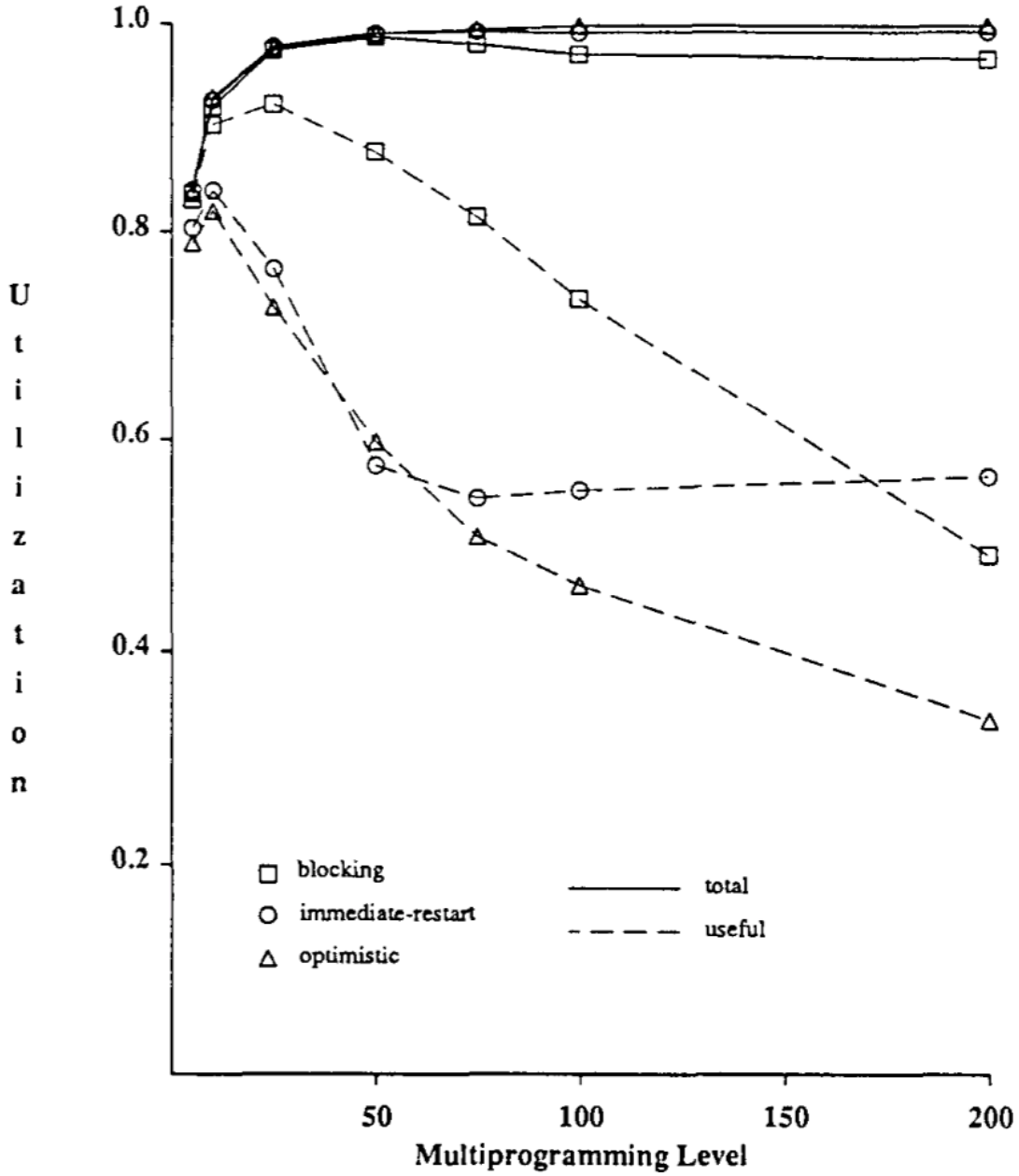
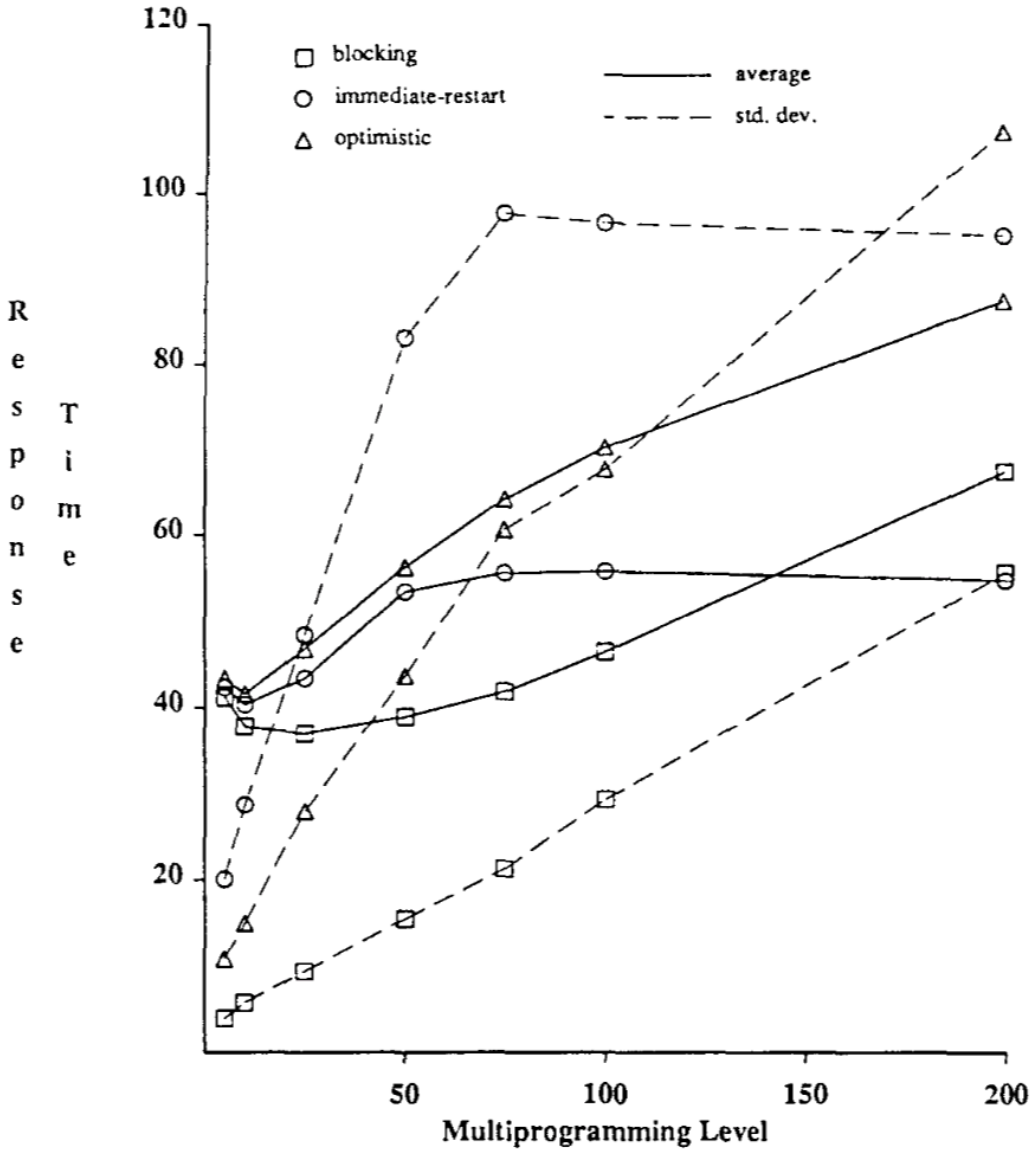


Fig. 9. Disk utilization (1 resource unit).

-
- ...
- Blocking peaks at about mpl = 25, with 97% util and useful at 92% - increasing further only increases contention (blocking, restarts, etc)
- OCC peaks at 10, beyond that there's a lot of restarts
  - a good fraction of work is not useful - will be redone later
- Same issue with IR, though again it ends up flattening because of steady state: increasing beyond 50 has no effect because all non-active txns will be in delay state or thinking
- Best throughput: blocking; IR does better than OCC in general and also better than blocking at the end

- Response time
  - Chart:



-
  - Similar to previous
- Response time:
  - Blocking has lowest delay, then IR, then OCC - mirroring throughput
- Std dev
  - IR highest, then OCC, then blocking

- Is IR a better approach for high MP?
  - not really - we (as the database) need to control the MP
    - does not relate to user "parallelism"
  - in effect, it basically sets effective MP to be 60, doesn't actually take advantage of increased MP
  - restart delay provides a crude mechanism to limit MP when restarts become frequent - and we can apply a delay to other approaches as well
  - to test this:

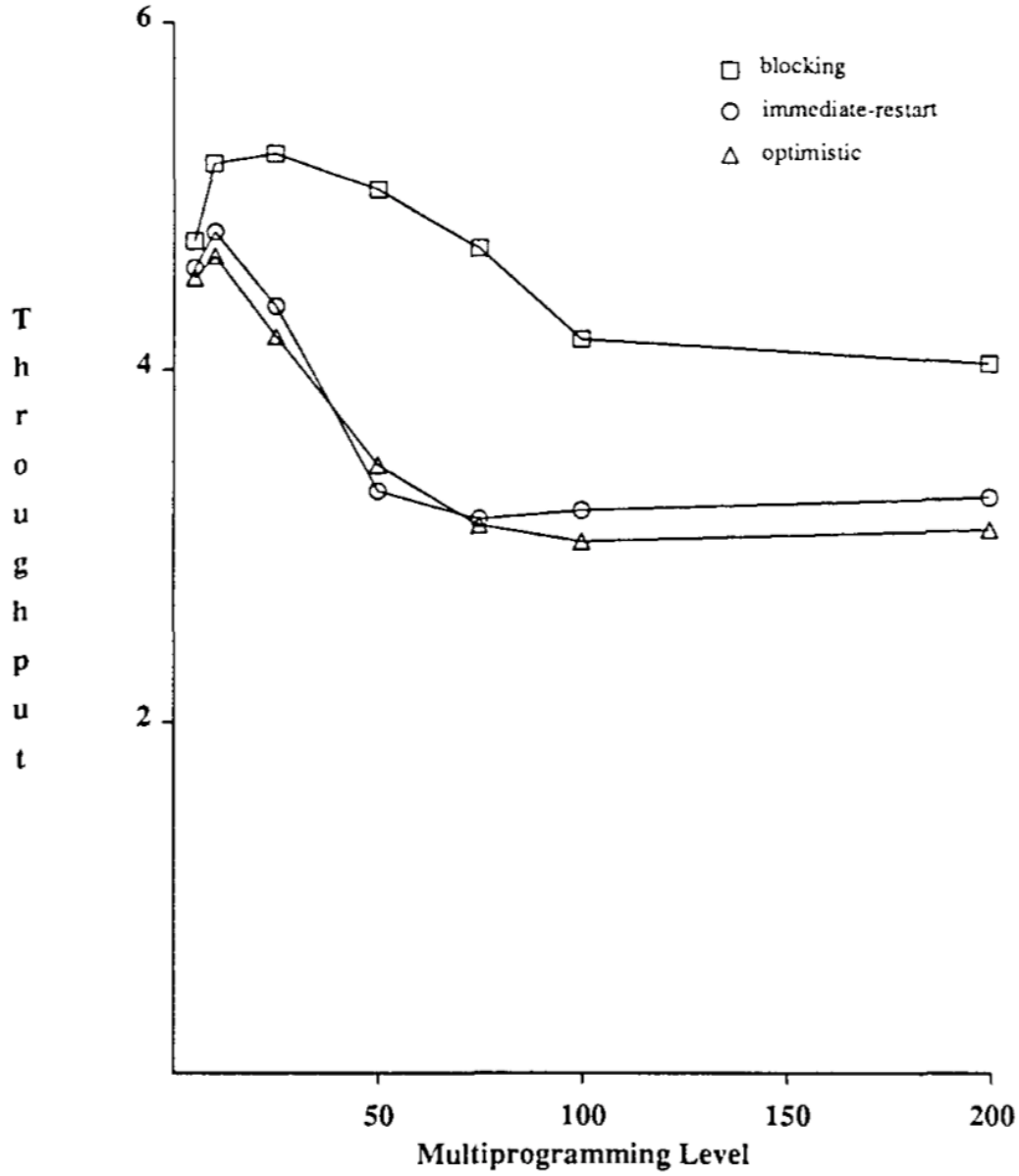


Fig. 11. Throughput (adaptive restart delays).

-
- ...

- Introducing adaptive delays/load control limits MP level for both Blocking and OCC, and here blocking emerges as a clear winner
  - But, delays increase

Somewhere in the middle

- Where do you go from limited to infinite

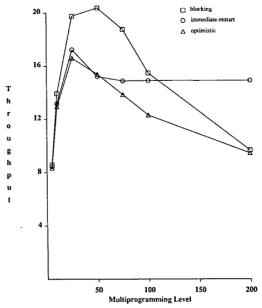


Fig. 14. Throughput (5 resource units).

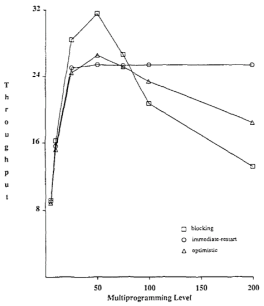


Fig. 16. Throughput (10 resource units).

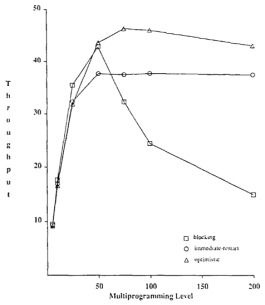


Fig. 18. Throughput (25 resource units).

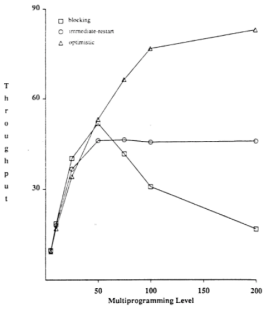


Fig. 5. Throughput (∞ resources).

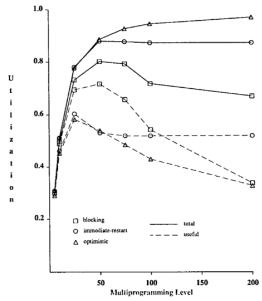


Fig. 15. Disk utilization (5 resource units).

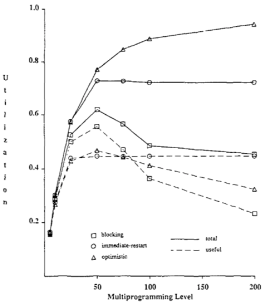


Fig. 17. Disk utilization (10 resource units).

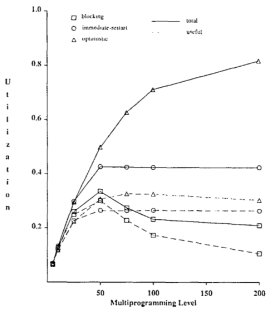


Fig. 19. Disk utilization (25 resource units).

- Notice how OCC moves the most, rest stay similar?
- With 5 units, maximum useful disk util is 72 (B), 60 (IR), 58 (OCC)
- With 10, 56 (B), 45 (IR), 47 (OCC)
  - Disk util for restarted approaches are all higher than that for blocking because these resources are wasted on txns that will get restarted
- With 25 units,
  - OCC max throughput beats max throughput by Blocking, though not by much
  - max useful disk util is similar 30 (B), 30 (OCC)
  - OCC starts to become more attractive "*because a large amount of otherwise unused resources are available, and thus the waste of resources due to restarts does not adversely affect performance. In other words, with useful utilizations in the 30 percent range, the system begins to behave somewhat like it has infinite resources*."
- As mp increases:
  - Thrashing in blocking is always (even in infinite case) - due to waiting for locks
  - Thrashing in OCC is because of restarts, but this is only a problem if the resources are finite and restarts are not consuming a huge fraction of the resources - so no thrashing for OCC at 50 and beyond
  - For IR, plateau is reached always due to restart delay
- Summary figure: if you could pick best throughput:
  - Elegant way of showing how to pick!

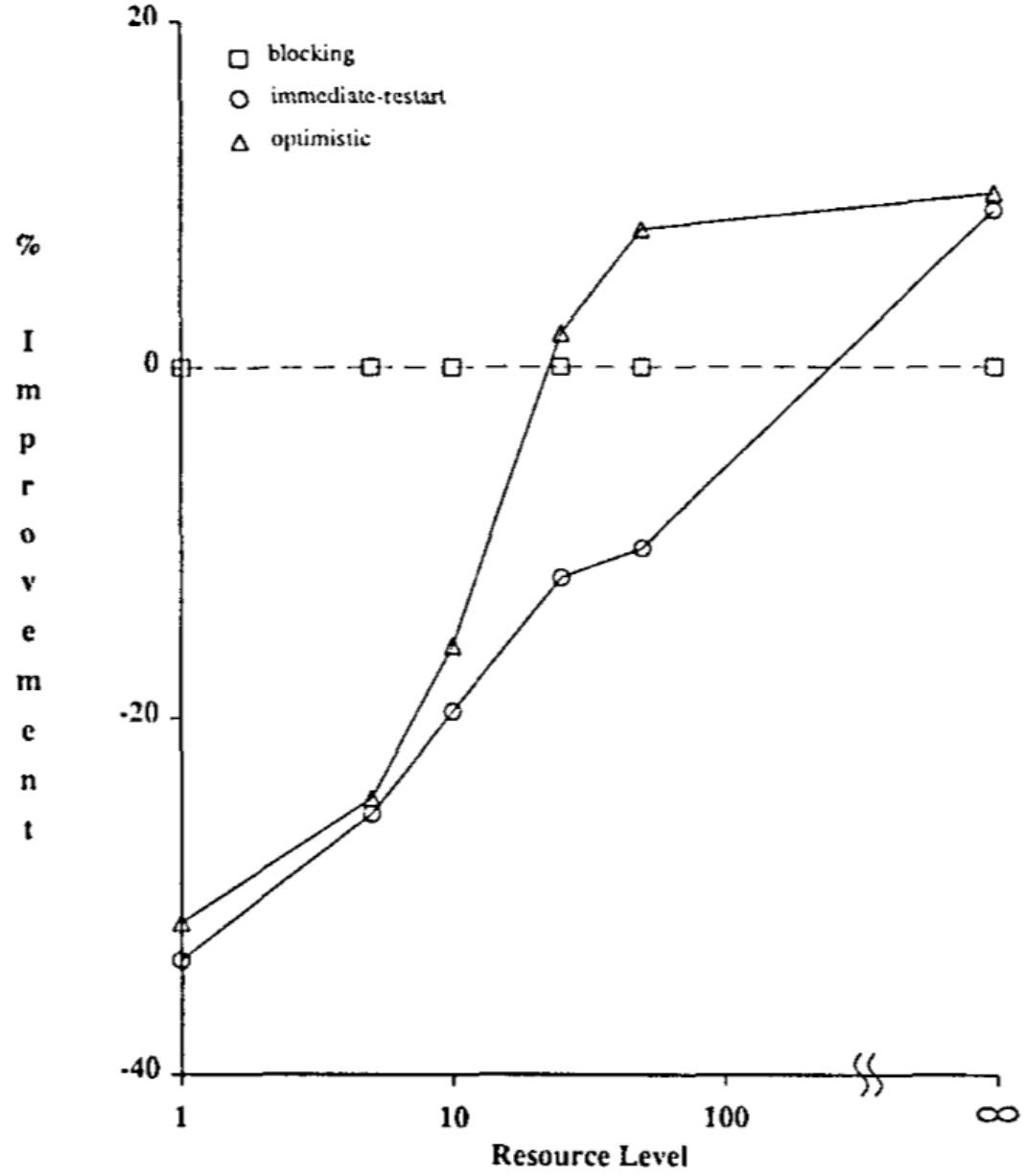


Fig. 22. Improvement over blocking (MPL = 50).

-
- ...

Think Time / Interactive Workloads

- Gap in time between the reads and the writes
- Not much to say except that think time tends to make resources seem infinite

Takeaways:

- for med/high utilization - use blocking
- for low utilization or high think-time, better to use restarts - and OCC is preferable
- each prior work result matches a setting
- important to control the mp level as thrashing for locking can make blocking (or OCC) both degrade in performance

- This can make blocking/OCC appear worse than they are
- Q: can you do better than IR?