ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

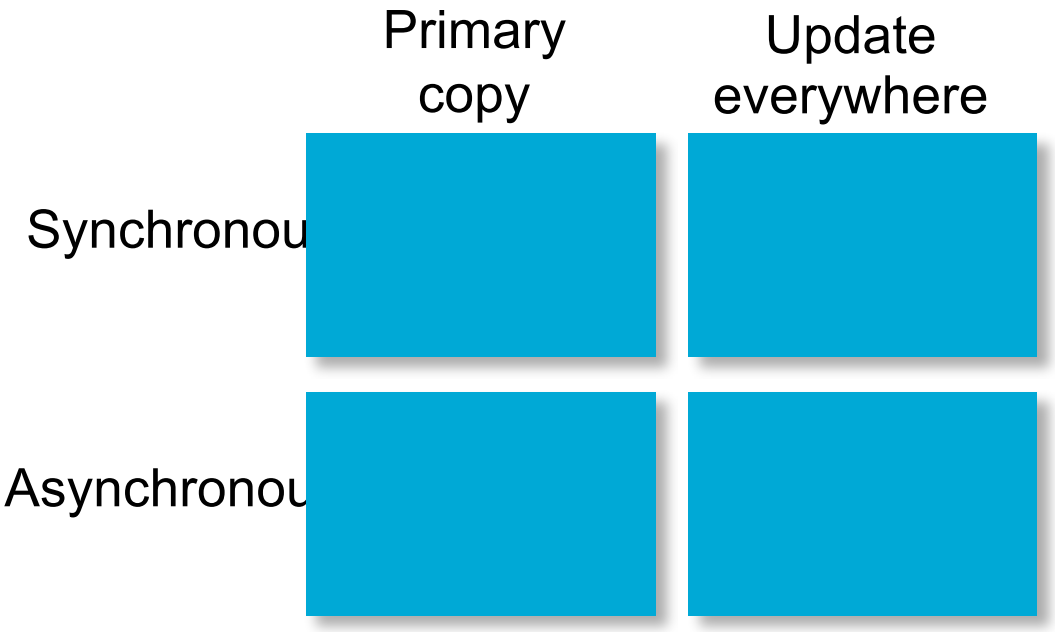
Database replication for commodity database services

Gustavo Alonso
Department of Computer Science
ETH Zürich
alonso@inf.ethz.ch
http://www.iks.ethz.ch

Replication as a problem

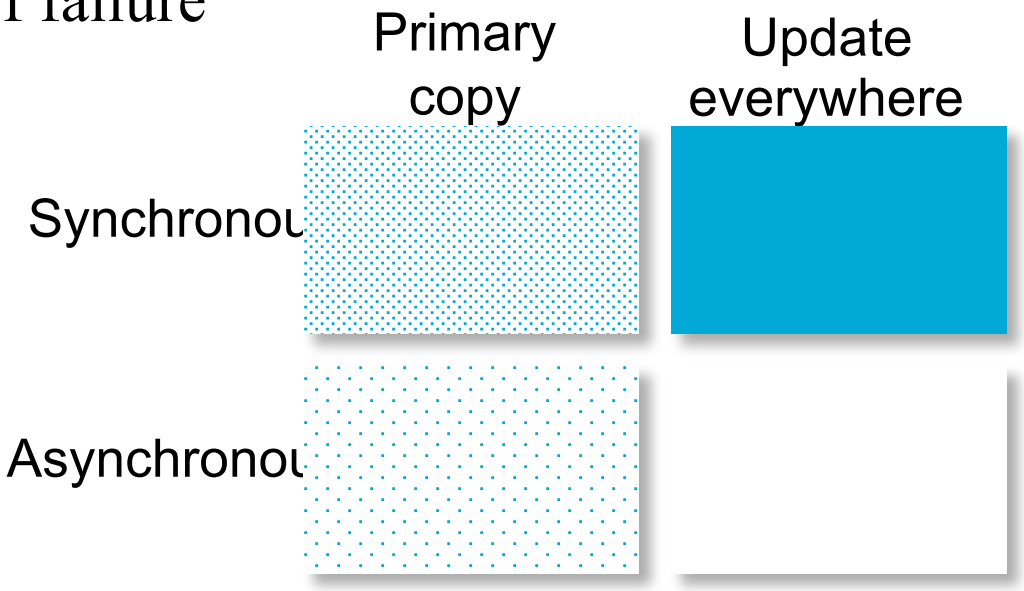
How to replicate data?

- Depending on when the updates are propagated:
  - Synchronous (eager)
  - Asynchronous (lazy)
- Depending on where the updates can take place:
  - Primary Copy (master)
  - Update Everywhere (group)



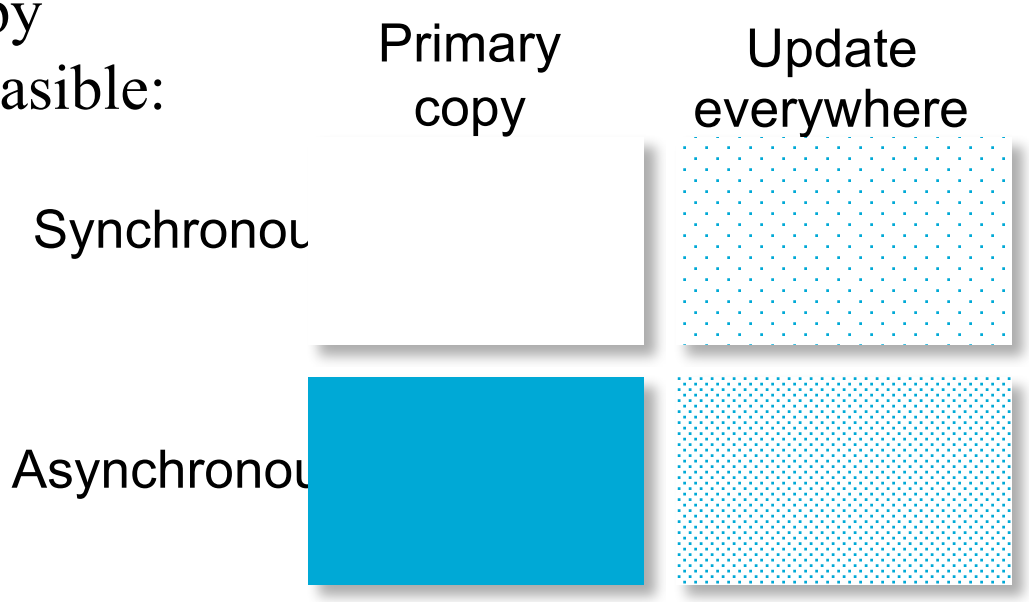
Theory …

- The name of the game is **correctness and consistency**
- Synchronous replication is preferred:
  - copies are always consistent (1-copy serializability)
  - programming model is trivial (replication is transparent)
- Update everywhere is preferred:
  - system is symmetric (load balancing)
  - avoids single point of failure
- Other options are ugly:
  - inconsistencies
  - centralized
  - formally incorrect



… and practice

- The name of the game is **throughput and response time**
- Asynchronous replication is preferred:
  - avoid transactional coordination (throughput)
  - avoid 2PC overhead (response time)
- Primary copy is preferred:
  - design is simpler (centralized)
  - trust the primary copy
- Other options are not feasible:
  - overhead
  - deadlocks
  - do not scale



The dangers of replication ...

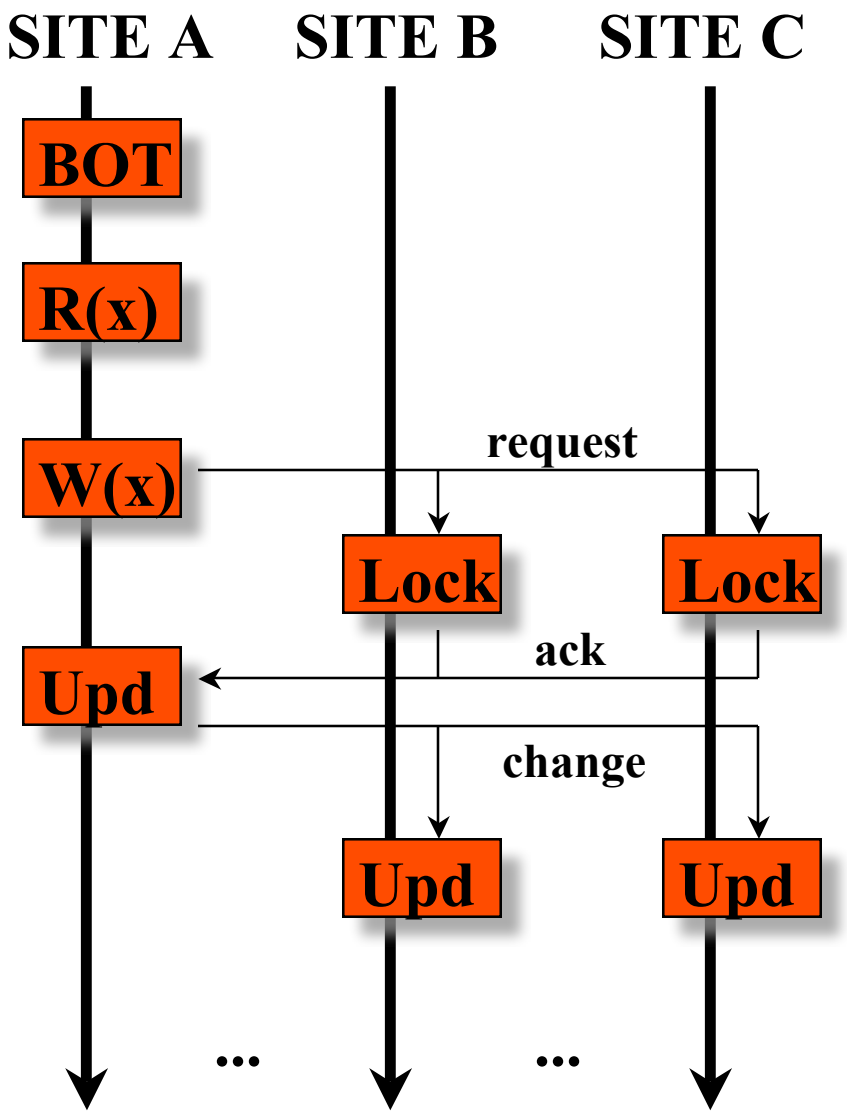
SYNCHRONOUS

- Coordination overhead
  - distributed 2PL is expensive
  - 2PC is expensive
  - prefer performance to correctness
- Communication overhead
  - 5 nodes, 100 tps, 10 w/txn = 5'000 messages per second !!

UPDATE EVERYWHERE

- Deadlock/Reconciliation rates
  - the probability of conflicts becomes so high, the system is unstable and does not scale
- Useless work
  - the same work is done by all
  - administrative costs paid by everybody
  - all nodes must understand replication (not trivial)

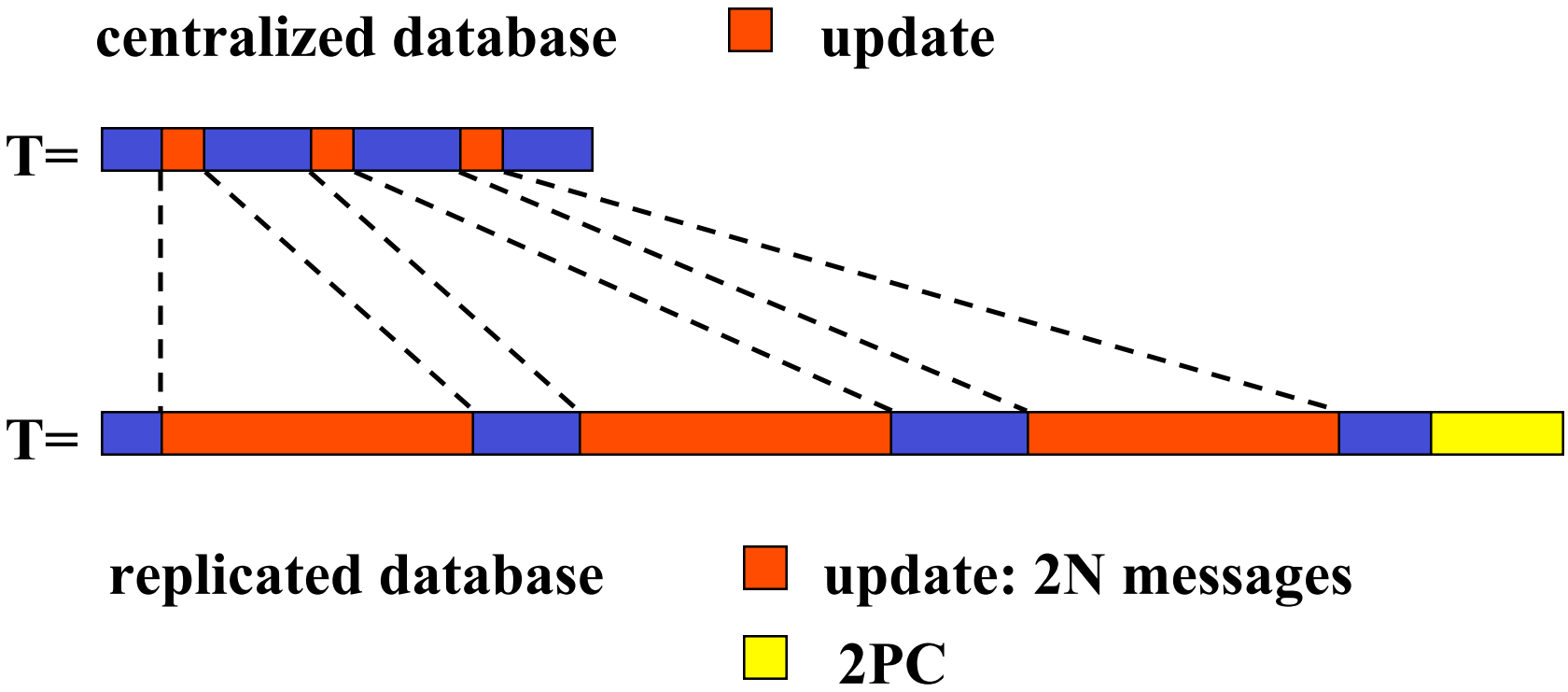
Text book replication (BHG'87)



Read One, Write All

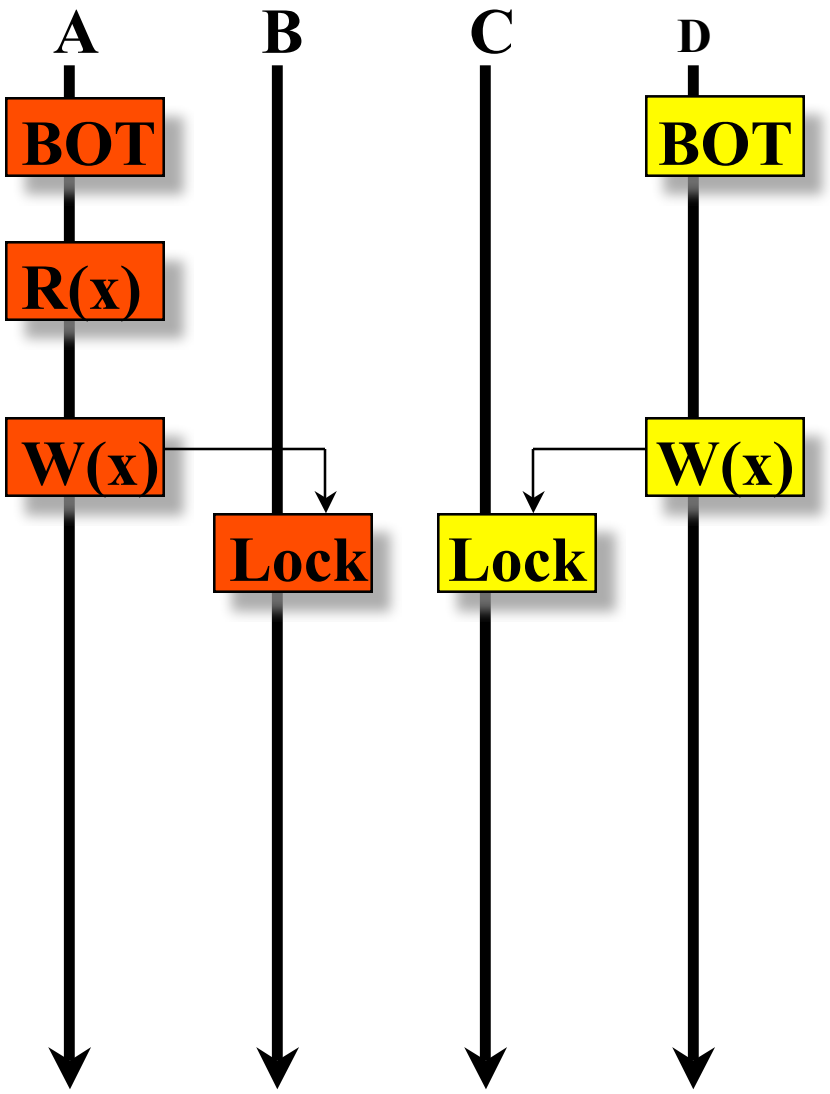
- Each site uses 2PL
- Atomic commitment through 2PC
- Read operations are performed locally
- Write operations involve locking all copies of the data item (request a lock, obtain the lock, receive an acknowledgement)
- Optimizations are based on the idea of quorums

Response Time



The way replication takes place (one operation at a time), increases the response time and, thereby, the conflict profile of the transaction. The message overhead is too high (even if broadcast facilities are available).

Deadlocks (Gray et al. SIGMOD'96)



- Approximated deadlock rate:

$$\frac{TPS^2 \cdot Action\_Time \cdot Actions^5 \cdot N^3}{4 \cdot DB\_Size^2}$$

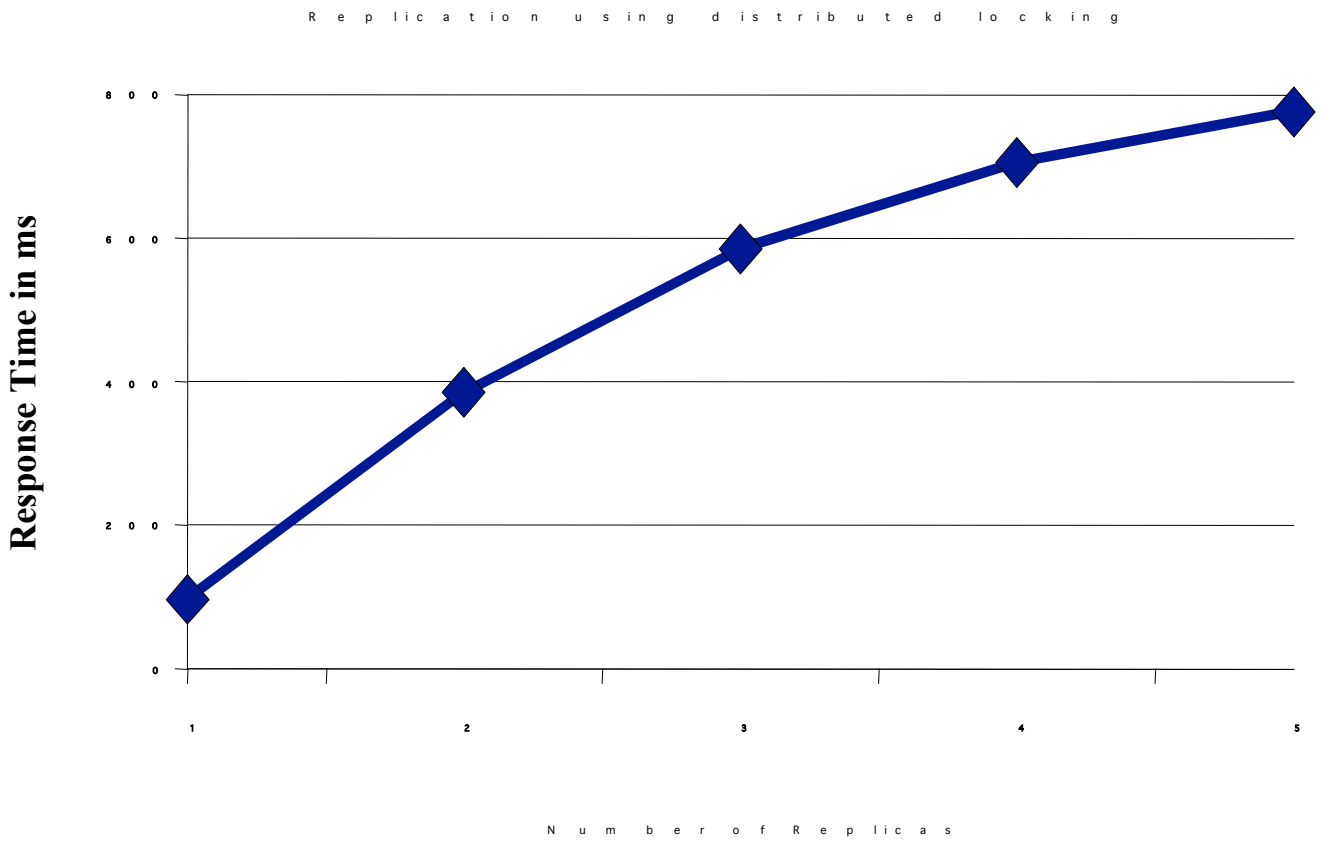
if the database size remains constant, or

$$\frac{TPS^2 \cdot Action\_Time \cdot Actions^5 \cdot N}{4 \cdot DB\_Size^2}$$

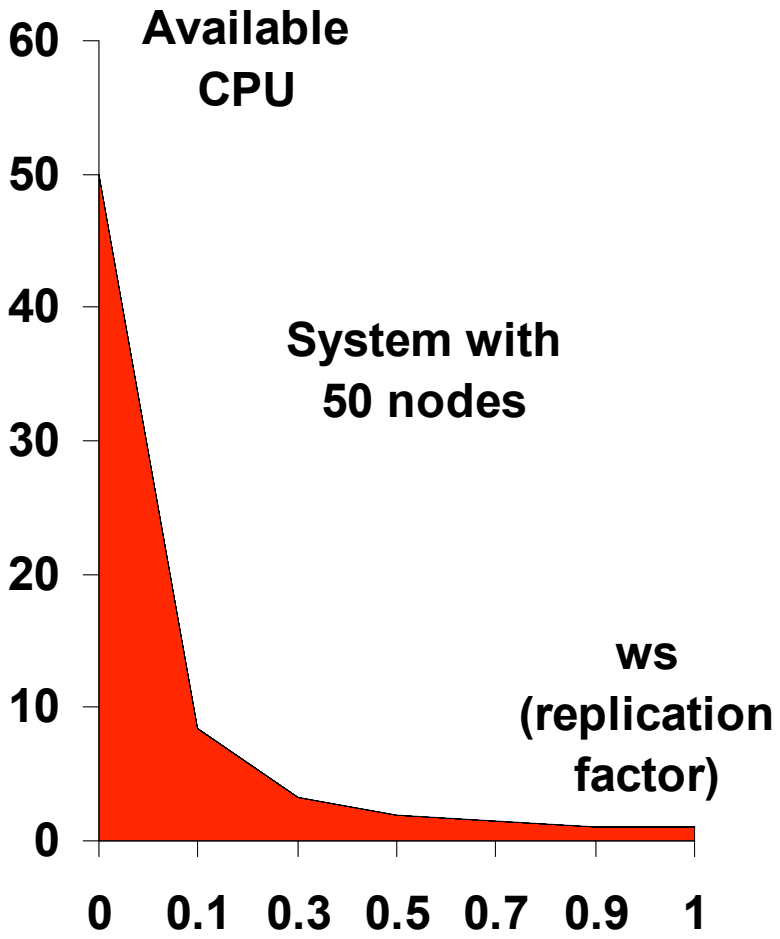
if the database size grows with the number of nodes.

- Optimistic approaches result in too many aborts.

Commercial systems



Cost of Replication



- Overall computing power of the system:

$$\frac{N}{1 + w \cdot s \cdot (N - 1)}$$

- No gain with large ws factor (rate of updates and fraction of the database that is replicated)
- Quorums are questionable, reads must be local to get performance advantages.

GANYMED: Solving the replication problem

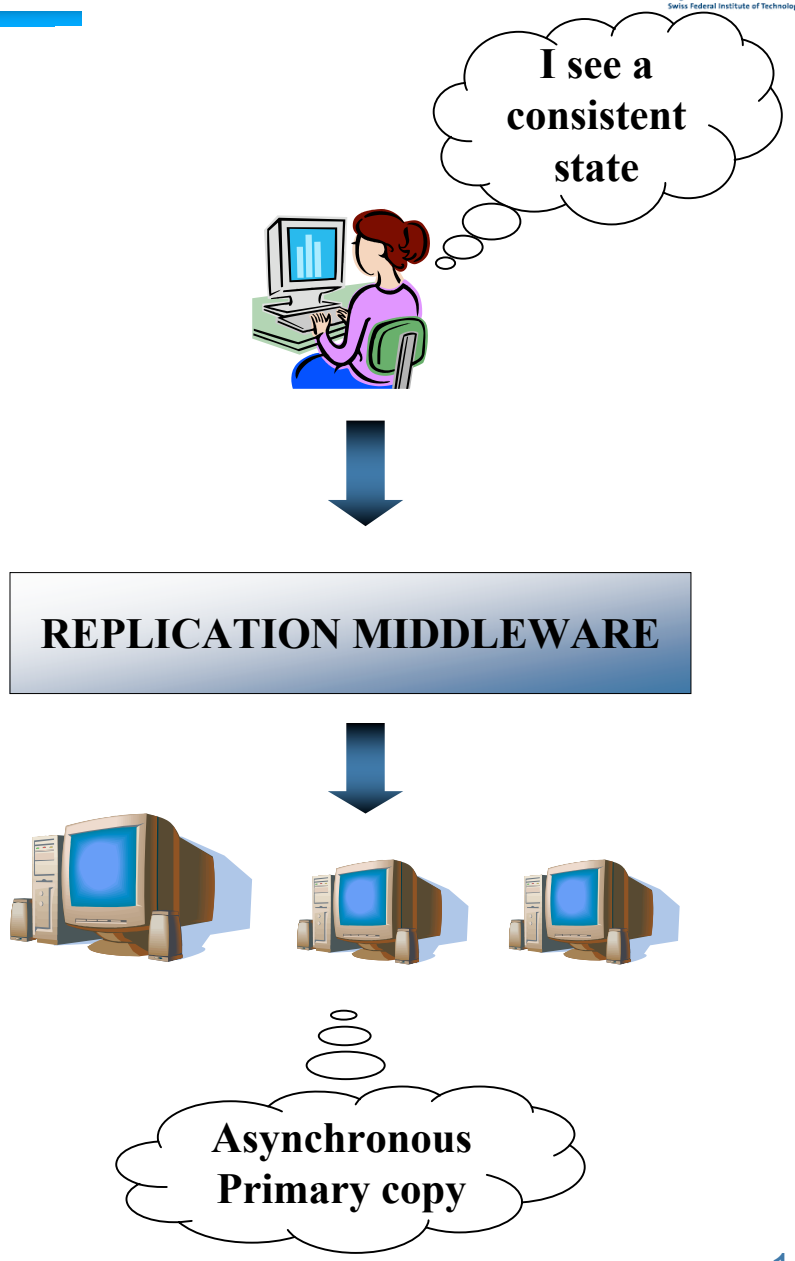
What can be done?

- Are these fundamental limitations or side effects of the way databases work?
  - Consistency vs. Performance: is this a real trade-off?
  - Cost seems to be inherent: if all copies do the same, no performance gain
  - Deadlocks: typical synchronization problem when using locks
  - Communication overhead: ignored in theory, a real show-stopper in practice

- If there are no fundamental limitations, can we do better? In particular, is there a reasonable implementation of synchronous, update everywhere replication?
  - Consistency is a good idea
  - Performance is also a good idea
  - Nobody disagrees that it would be nice …
  - … but commercial systems have given up on having both !!

Consistency vs. Peformance

- We want both:
  - Consistency is good for the application
  - Performance is good for the system
- Then:
  - Let the application see a consistent state ...
  - ... although the system is asynchronous and primary copy
- This is done through:
  - A middleware layer that offers a consistent view
  - Using snapshot isolation as correctnes criteria



Two sides of the same coin

SNAPSHOT ISOLATION

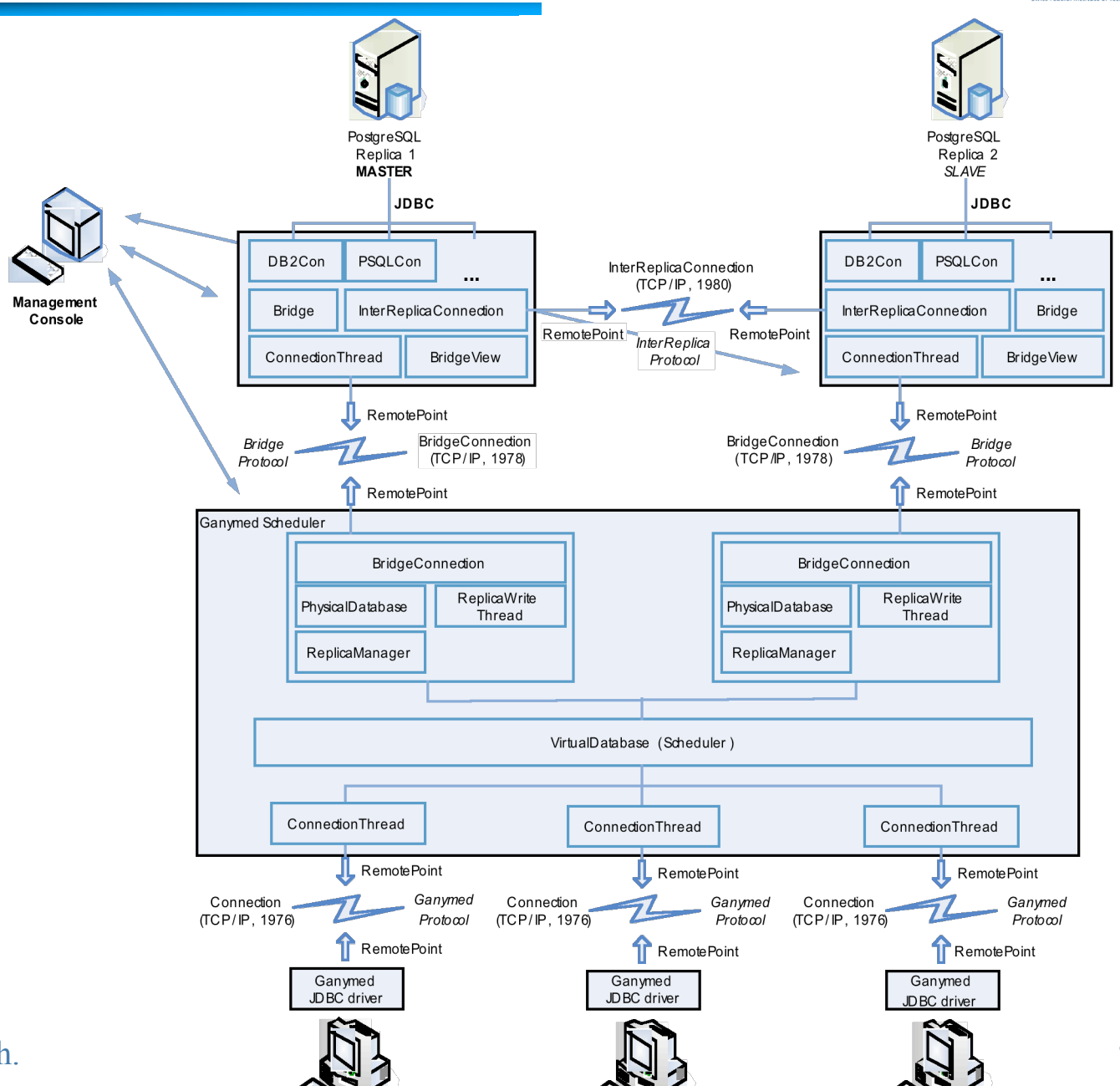
- To the clients, the middleware offers snapshot isolation:
  - Queries get their own consistent snapshot (version) of the database
  - Update transactions work with the latest data
  - Queries and updates do not conflict (operate of different data)
  - First committer wins for conflicting updates
- PostgreSQL, Oracle, MS SQL Server

ASYNCH – PRIMARY COPY

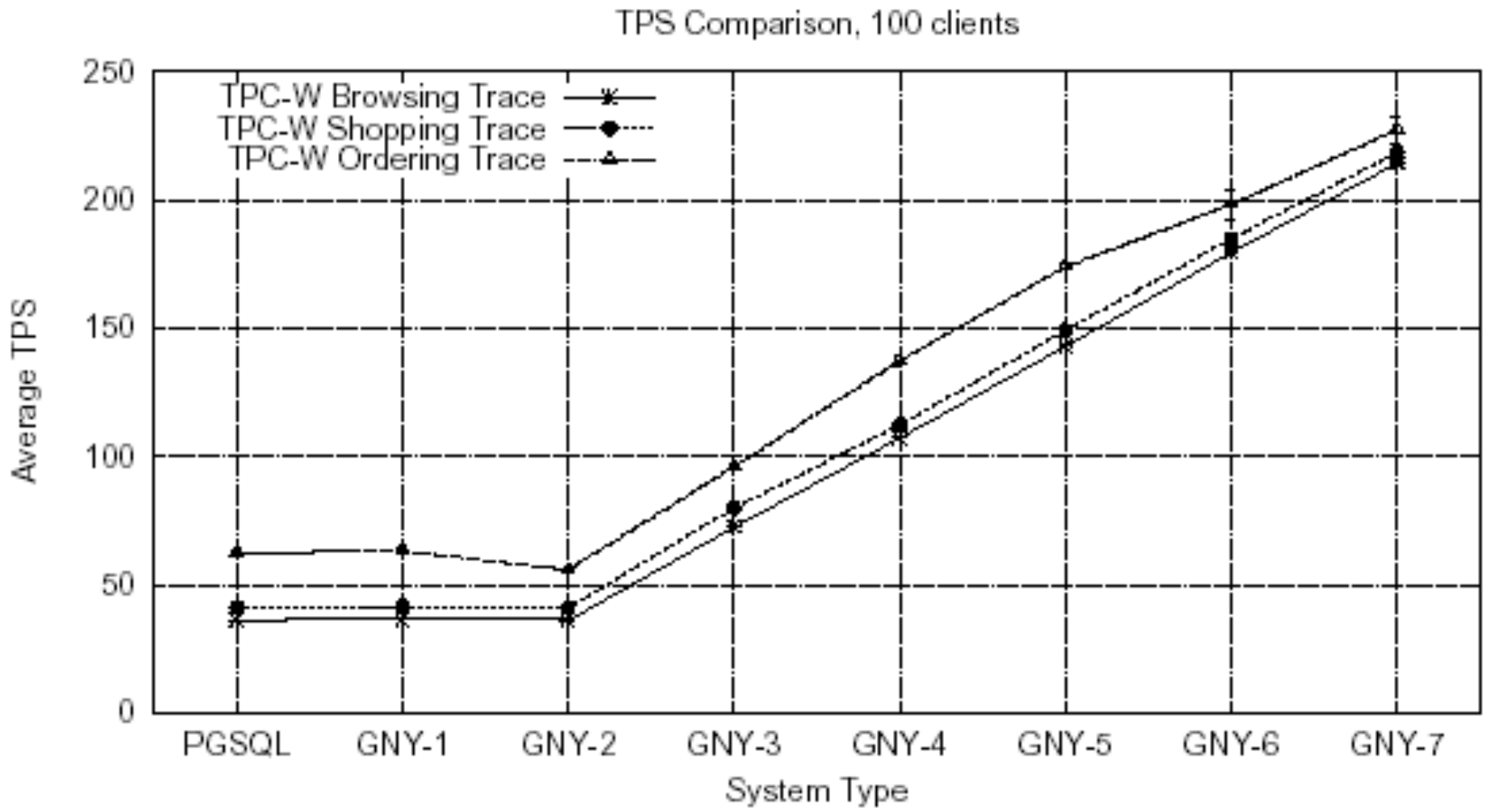
- Primary copy: master site where all updates are performed
- Slaves: copies where only reads are peformed
- A client gets a snapshot by running its queries on a copy
- Middleware makes sure that a client sees its own updates and only newer snapshots
- Updates go to primary copy and conflicts are resolved there (not by the middleware)
- Updates to master site are propagated lazily to the slaves

Ganymed: Putting it together

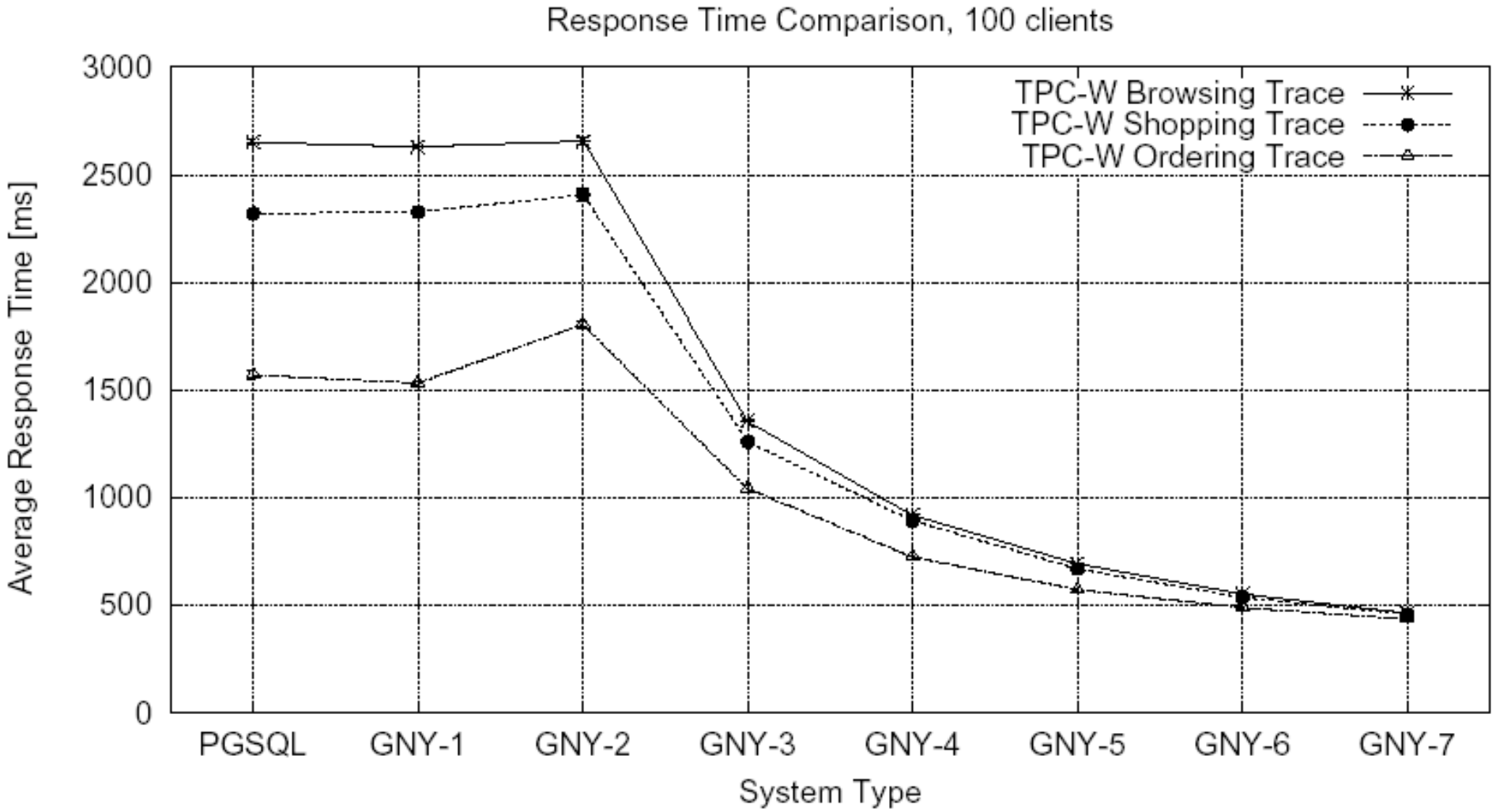
- **Based on JDBC drivers**
- **Only scheduling, no concurrency control, no query processing ...**
- **Simple messaging, no group communication**
- **Very much stateless (easy to make fault tolerant)**
- **Acts as traffic controller and bookkeeper**
- **Route queries to a copy where a consistent snapshot is available**
- **Keep track of what updates have been done where (propagation is not uniform)**



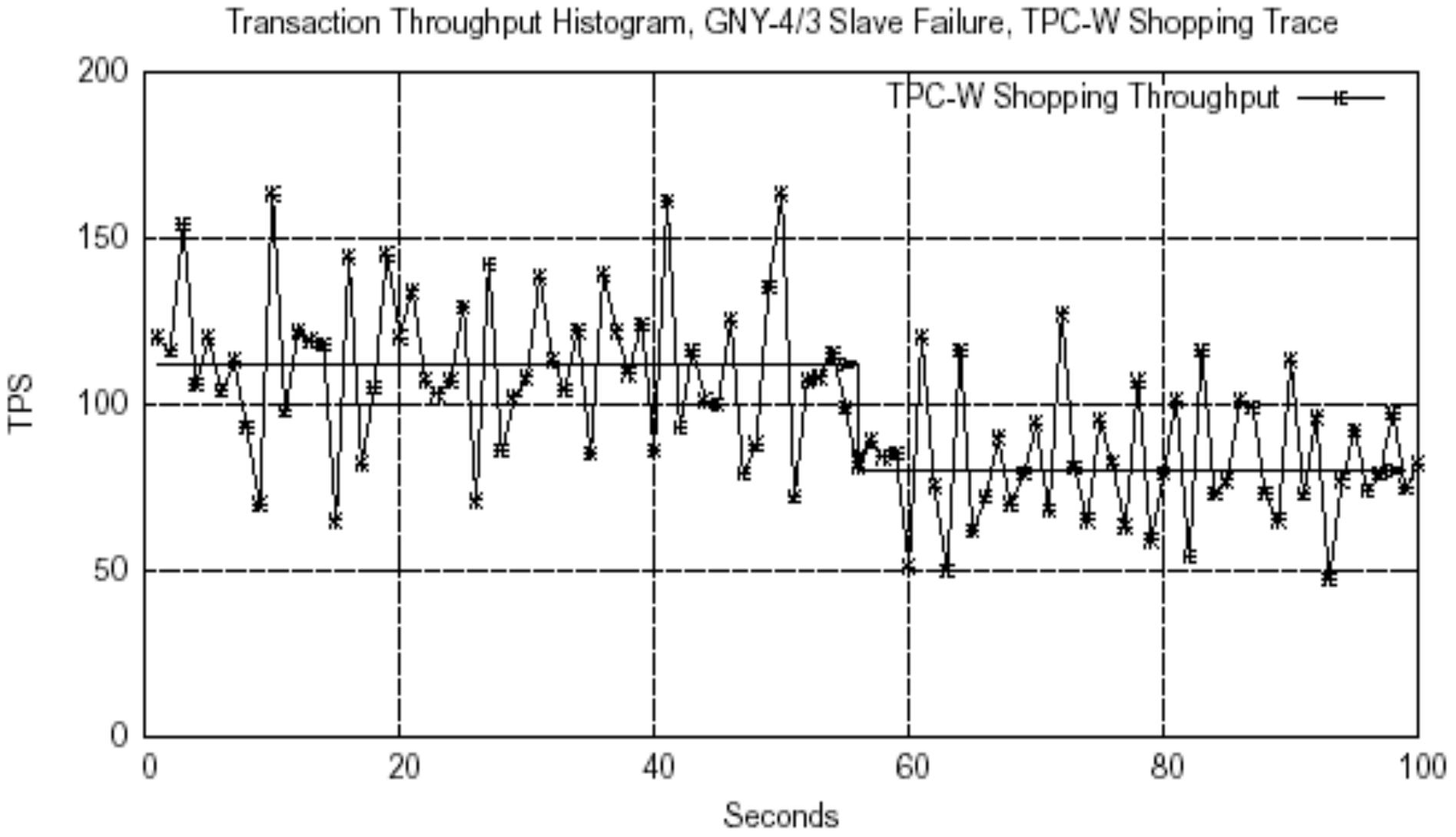
Linear scalability



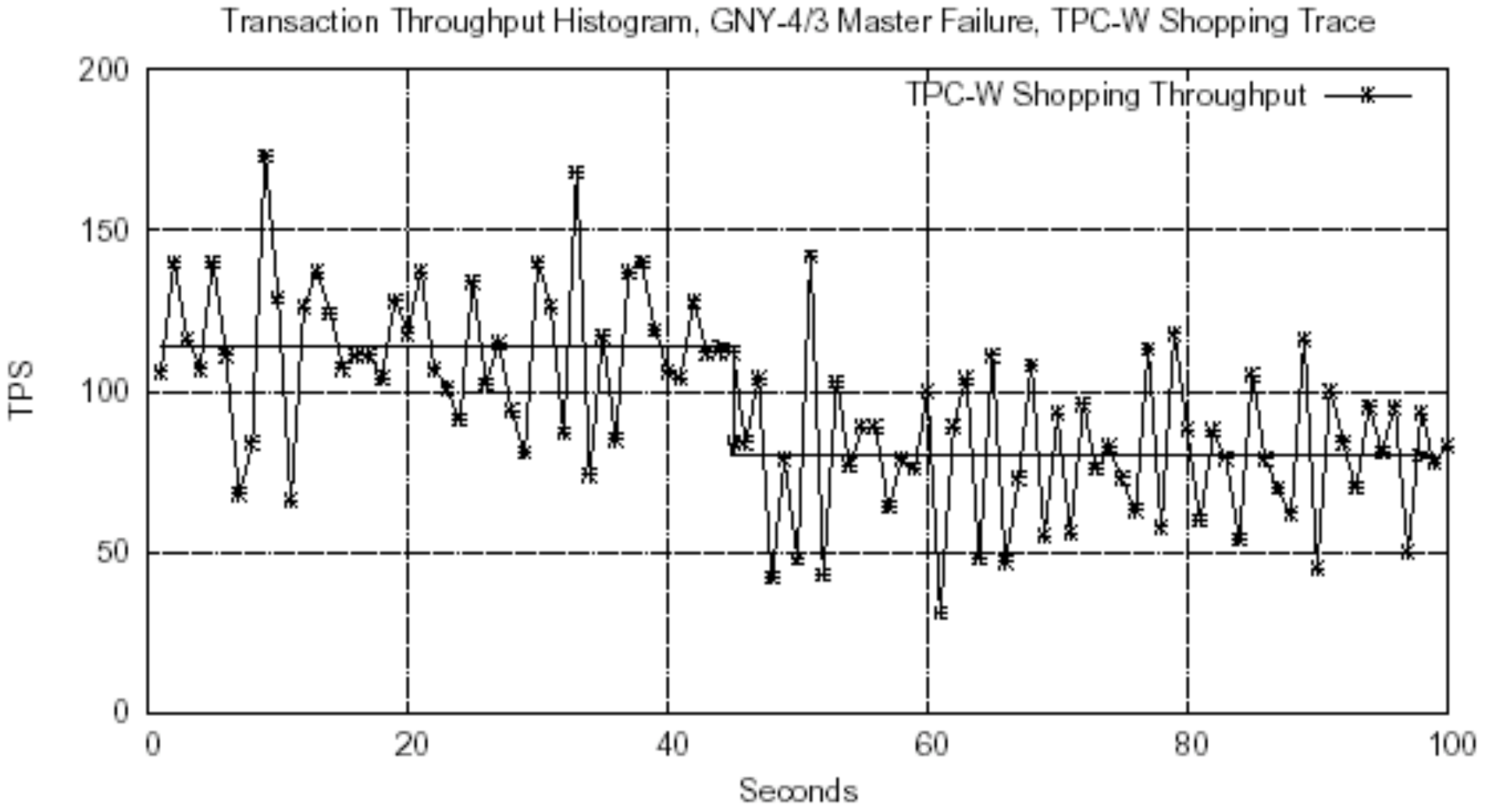
Improvements in response time (!!!)



Fault tolerance (slave failure)

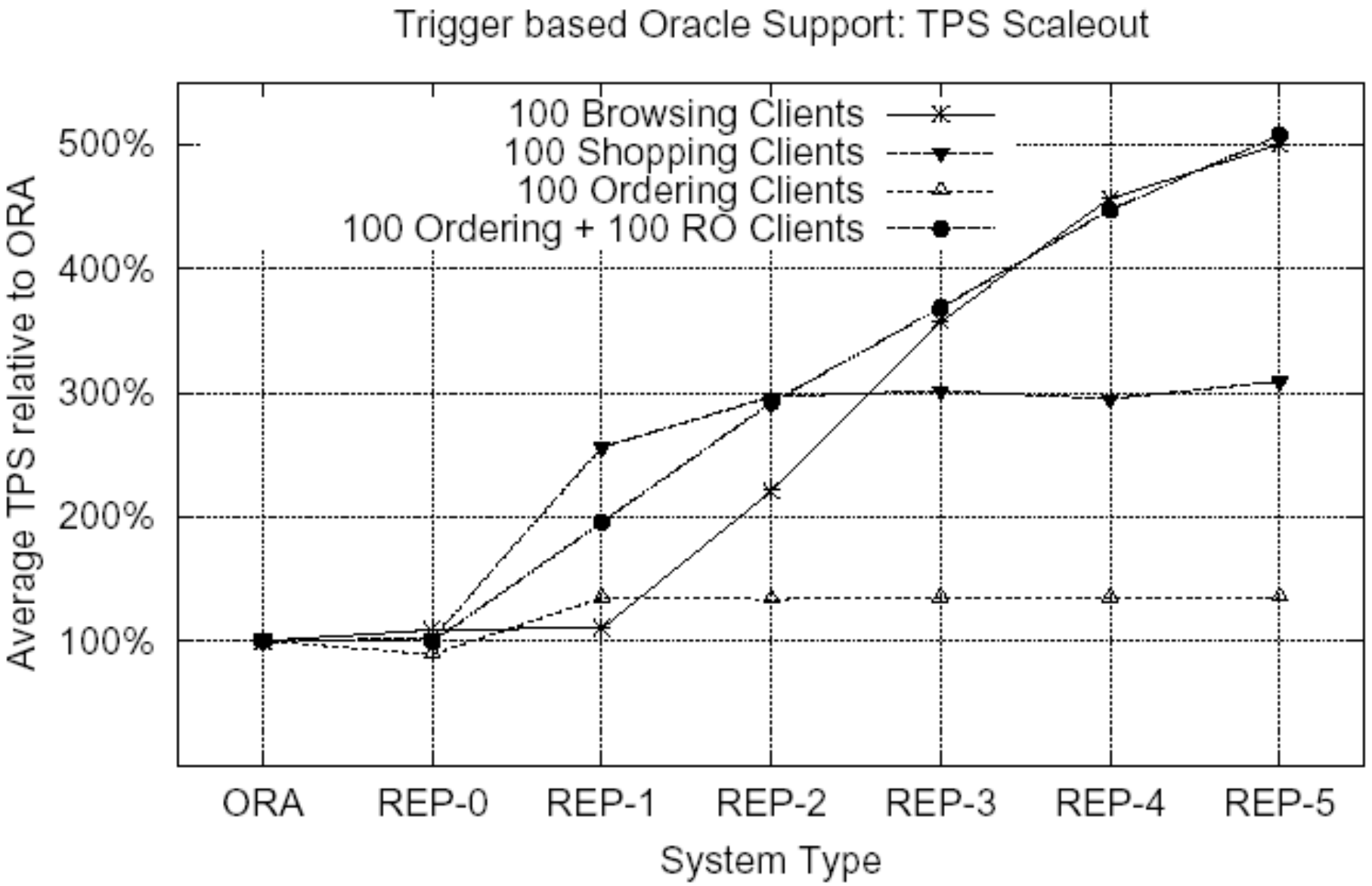


Fault tolerance (master failure)

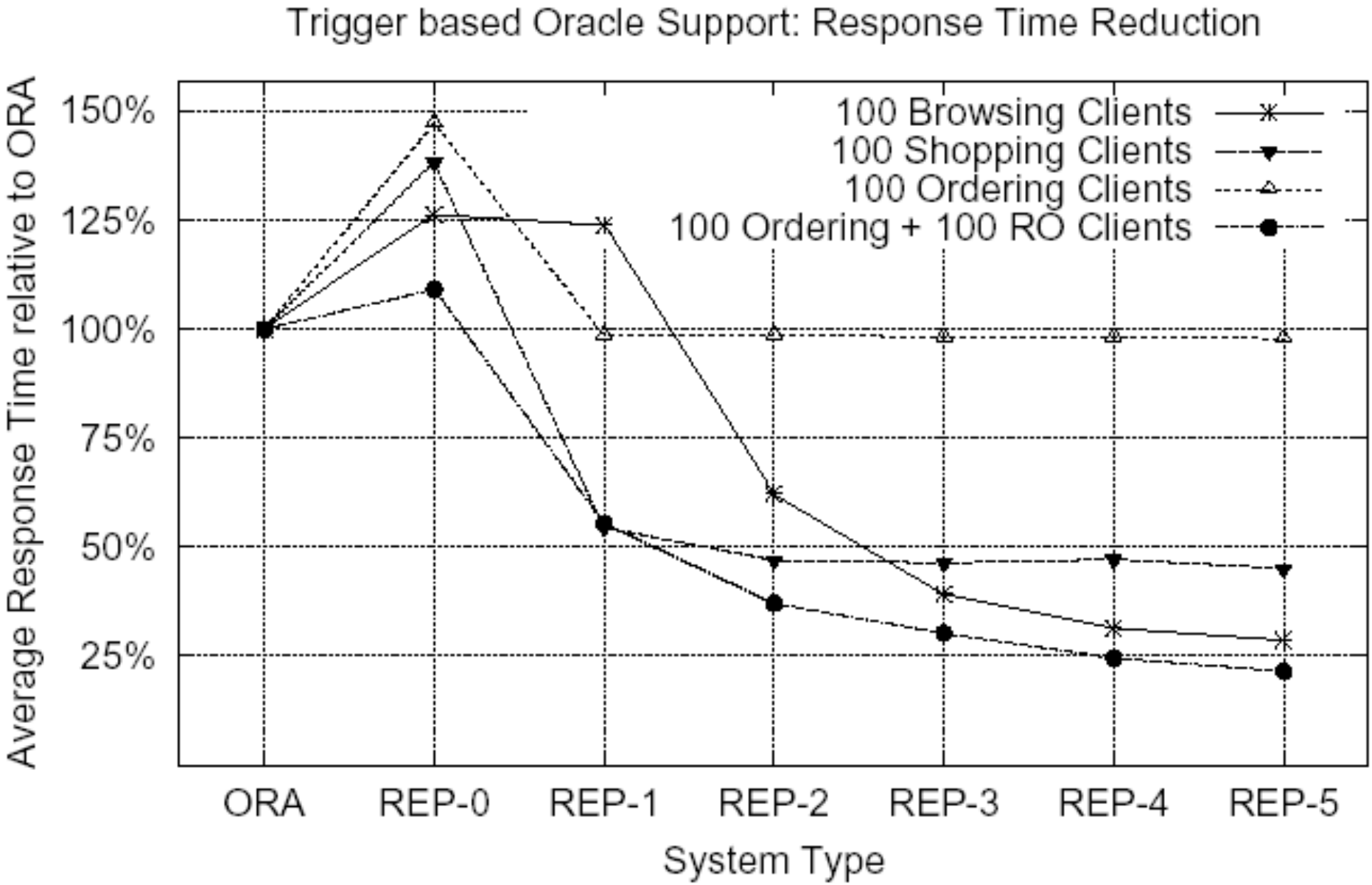


GANYMED: Beyond conventional replication

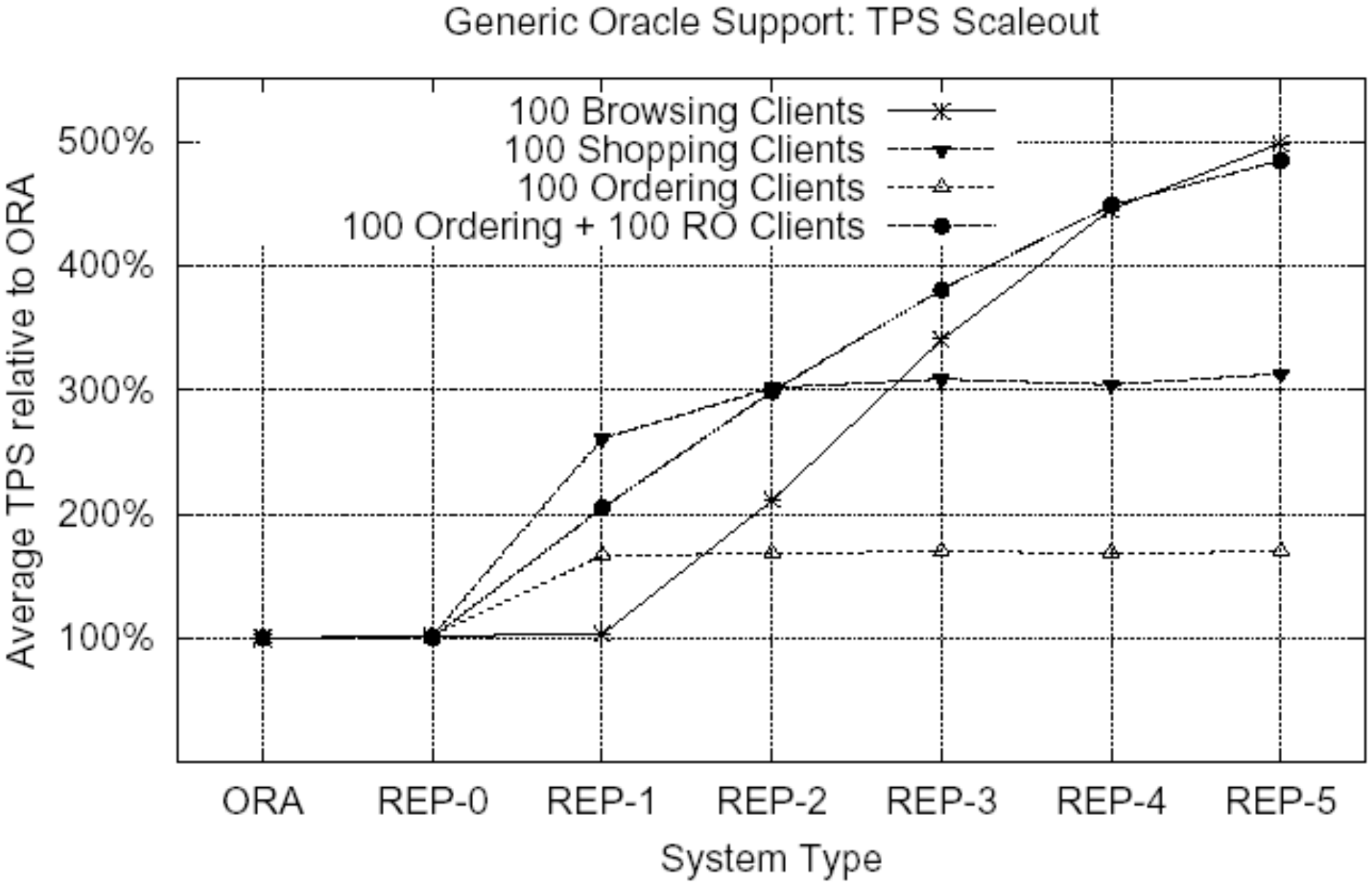
Oracle master – PostgreSQL slaves



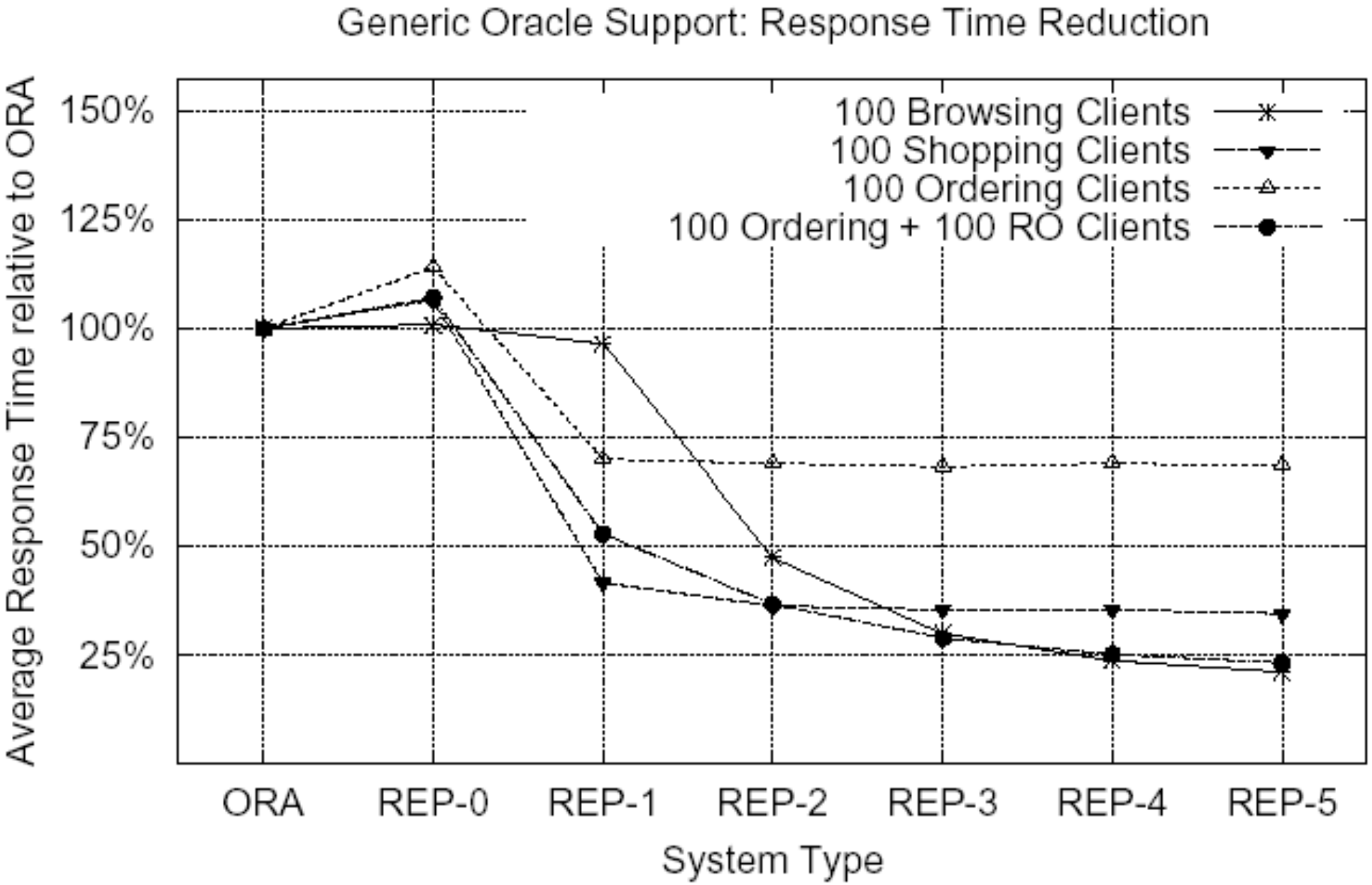
Oracle master – PostgreSQL slaves



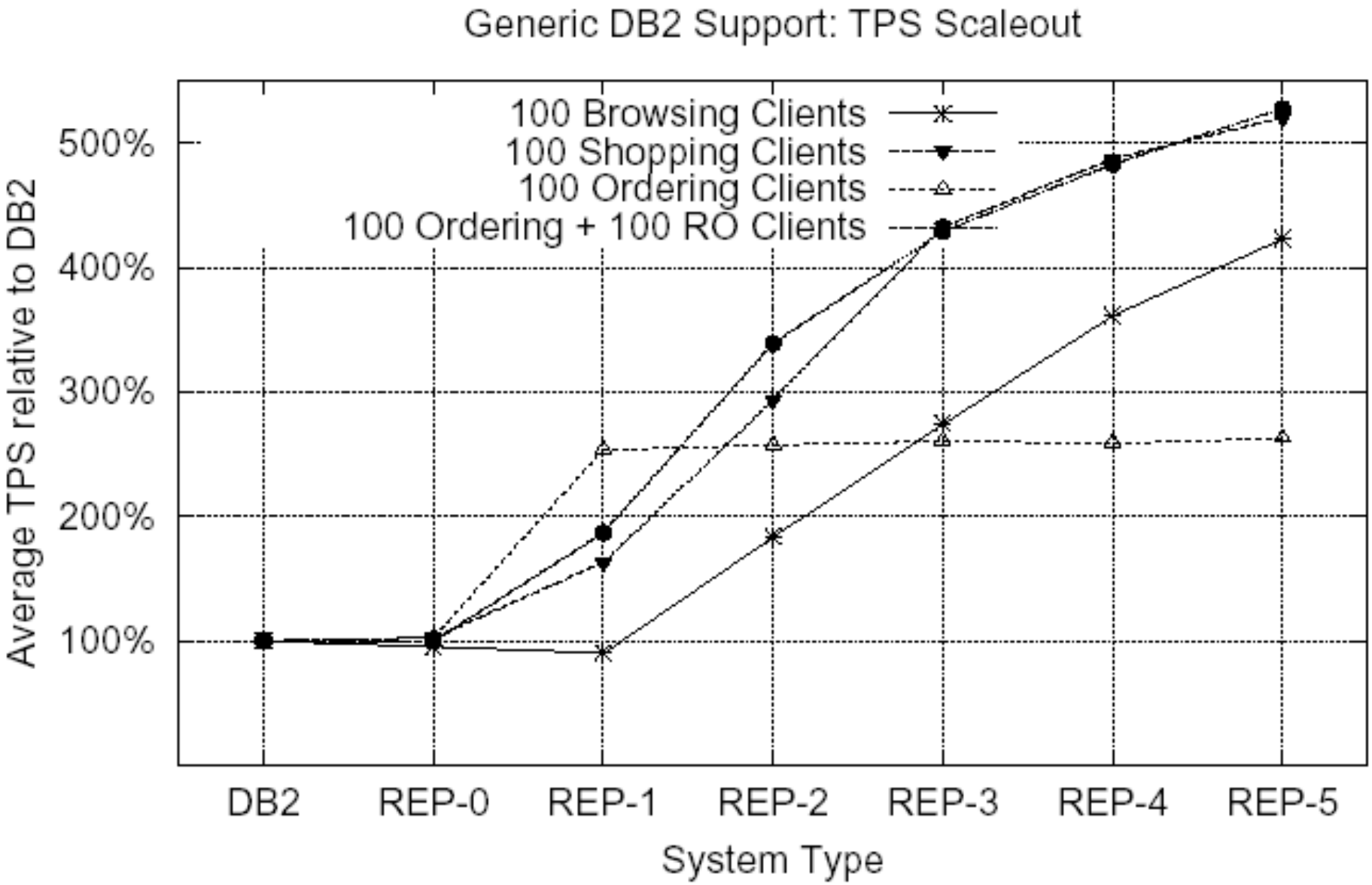
Updates through SQL (Oracle-Postgres)



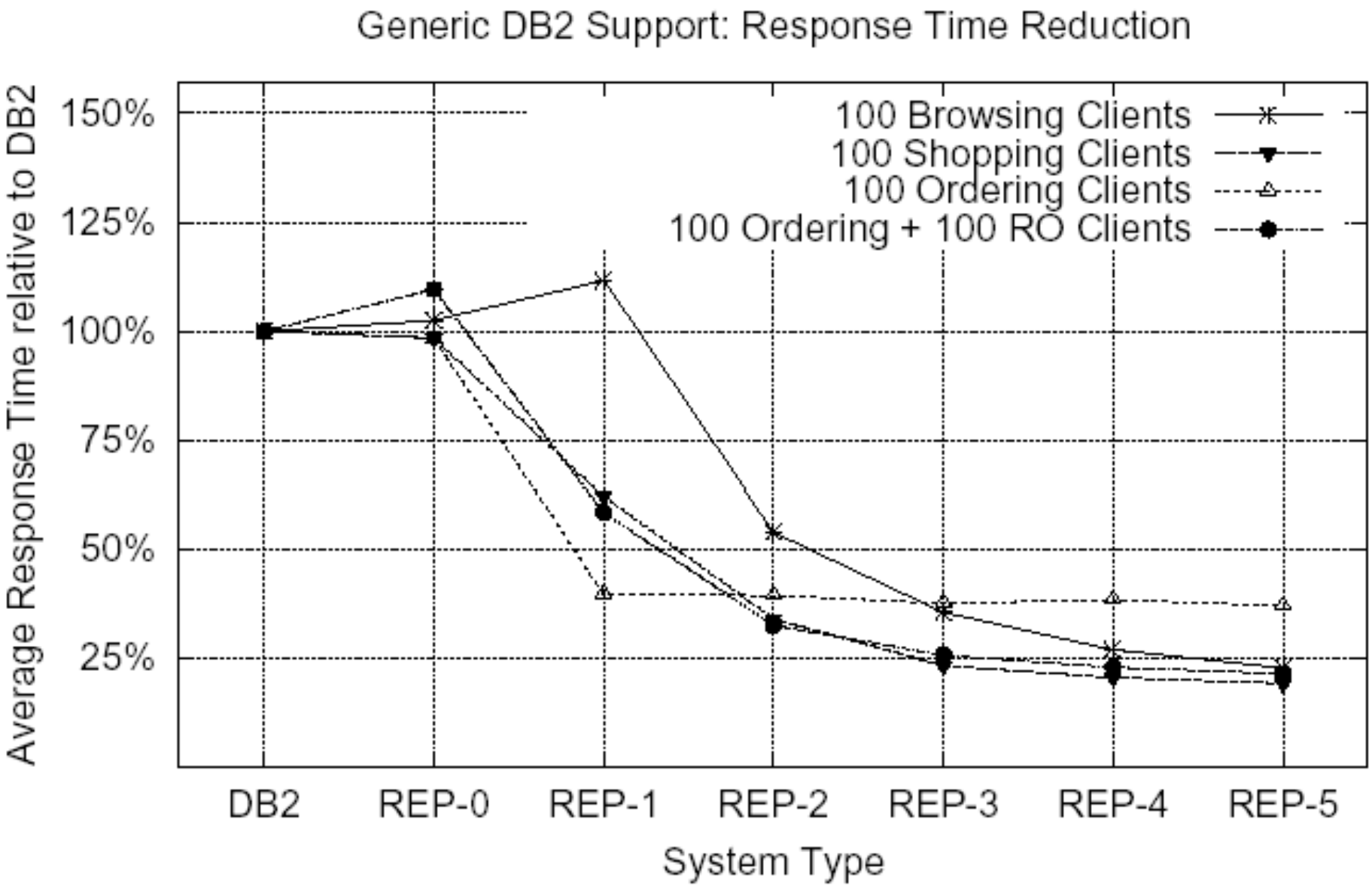
Updates through SQL (Oracle-Postgres)



DB2 master – PostreSQL slaves



DB2 master – PostreSQL slaves

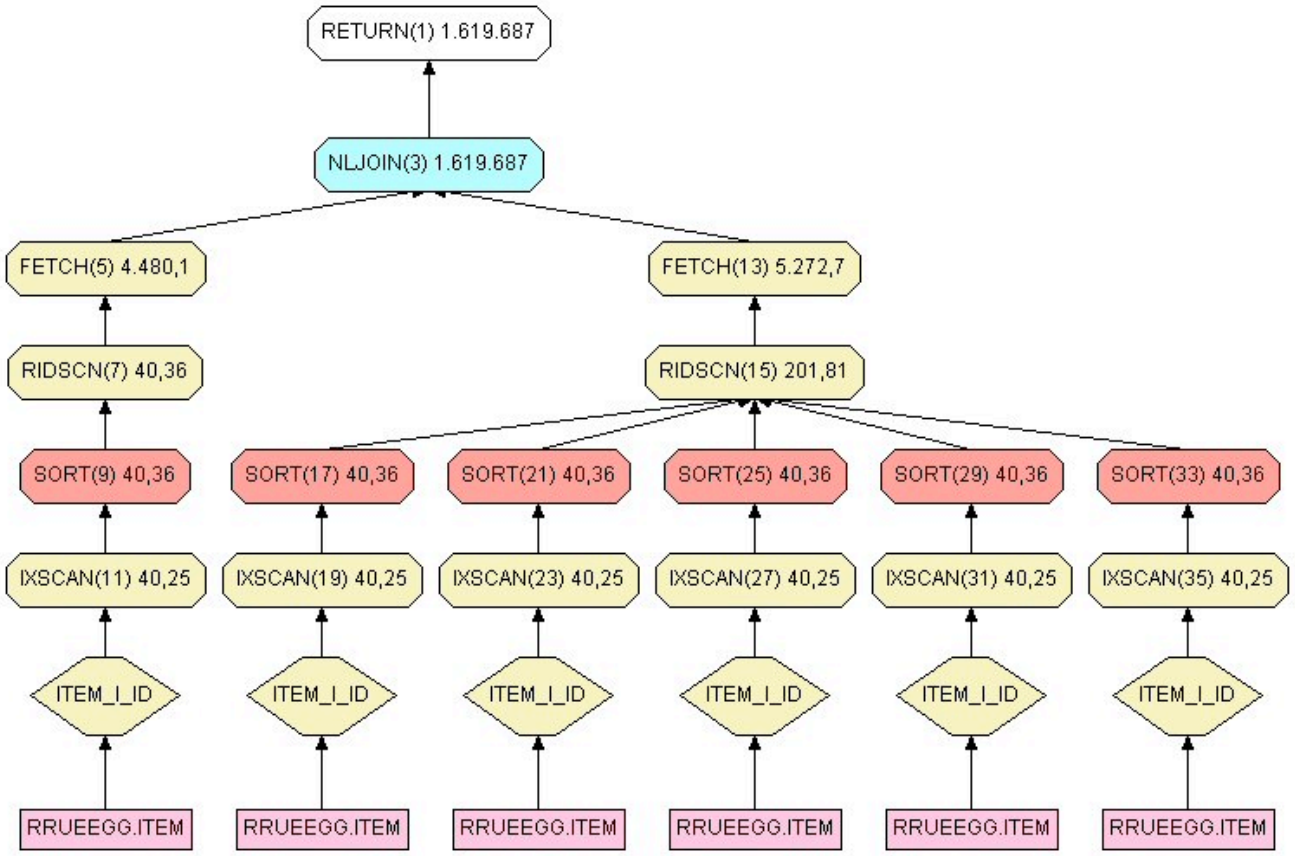


Critical issues

- The results with a commercial master and open source slaves is still a proof of concept but a very powerful one
- More work needs to be done (in progress)
  - Update extraction from the master
    - Trigger based = attach triggers to tables to report updates (low overhead at slaves, high overhead at master)
    - Generic = propagate update SQL statements to copies (high overhead at slaves, no overhead at master, limitations with hidden updates)
  - Update propagation = tuple based vs SQL based
  - SQL is not standard (particularly optimized SQL)
  - Understanding workloads (how much write load is really present in a database workload)
  - Replicate only parts of the database (table fragments, tables, materialized views, indexes, specialized indexes on copies ...)

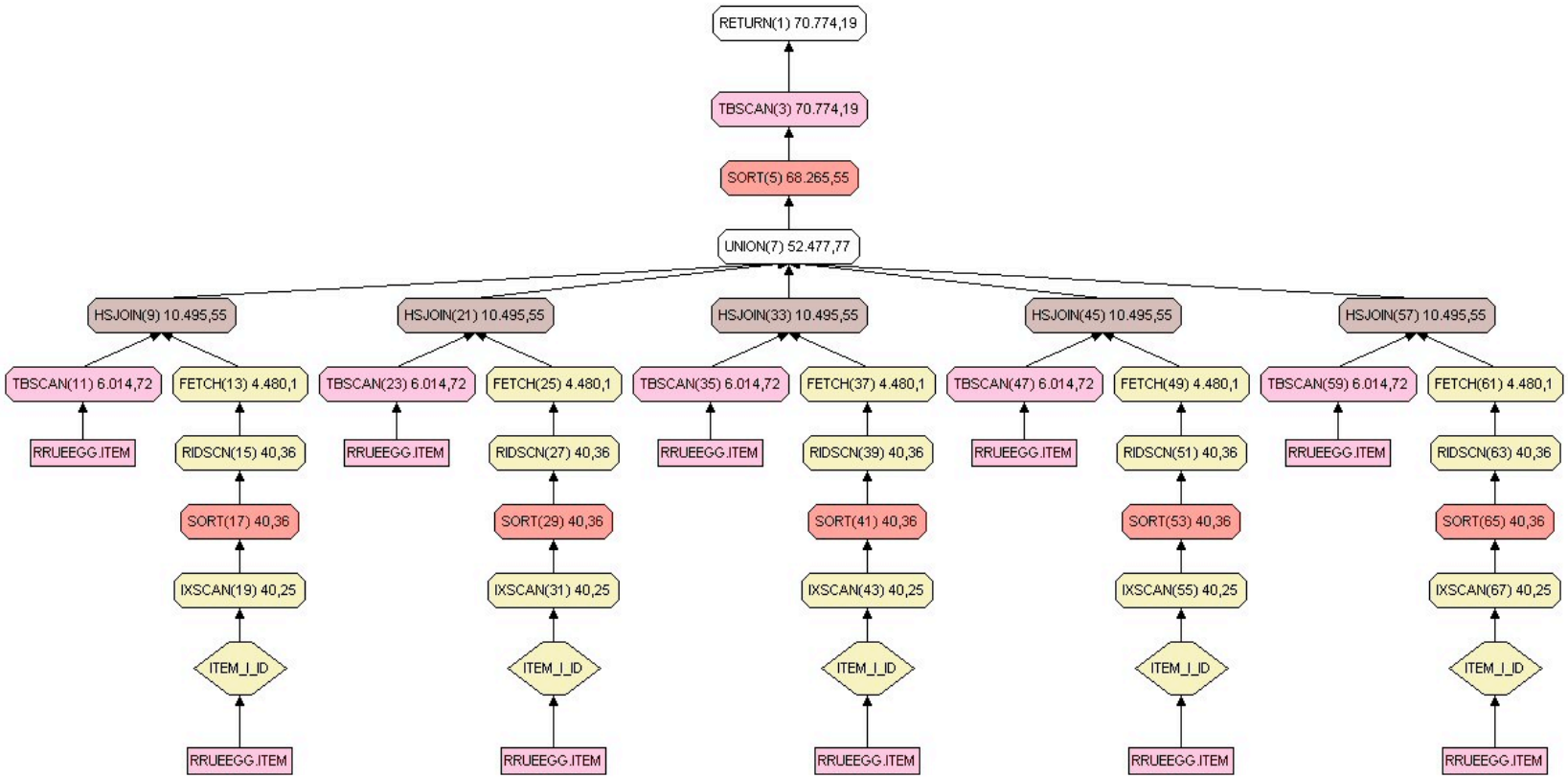
Query optimizations (DB2 example)

SELECT J.i_id, J.i_thumbnail
FROM item I, item J
WHERE (I.i_related1 = j.i_id **OR** I.i_related2 = j.i_id **OR** I.i_related3 = j.i_id **OR** I.i_related4 = j.i_id **OR** I.i_related5 = j.i_id) **AND** i.i_id = 839;



Query optimization (DB2 example)

```
SELECT J.i_id, J.i_thumbnail
FROM item J
WHERE J.i_id IN (
        (SELECT i_related1 FROM item WHERE i_id = 839) UNION
        (SELECT i_related2 FROM item WHERE i_id = 839) UNION
        (SELECT i_related3 FROM item WHERE i_id = 839) UNION
        (SELECT i_related4 FROM item WHERE i_id = 839) UNION
        (SELECT i_related5 FROM item WHERE i_id = 839)
);
```



Understanding workloads

TPC-W	Updates	Read-only	Ratio
Browsing	3.21 %	96.79 %	1 : 30.16
Shopping	10.77 %	89.23 %	1 : 8.29
Ordering	24.34 %	75.66 %	1 : 3.11

POSTGRES — NON-OPTIMIZED SQL — OPTIMIZED SQL

COST	Ratio (avg) updates : read only	Ratio (total) updates : read only	Ratio (avg) updates : read only	Ratio (total) updates : read only
Browsing	7:50 : 50.11	7.50 : 1511.32	6.92 : 10.39	6.29 : 313.36
Shopping	6.38 : 49.35	6.38 : 409.11	6.28 : 6.59	6.28 : 54.63
Ordering	7.70 : 36.28	7.70 : 112.83	6.23 : 3.28	6.23 : 10.20

A new twist to Moore´s Law

- What is the cost of optimization?
  - SQL rewriting = several days two/three (expert) people (improvement ratio between 5 and 10)
  - Ganymed = a few PCs with open source software (improvement factor between 2 and 5 for optimized SQL, for non-optimized SQL multiply by X)

- Keep in mind:
  - Copies do not need to be used, they can be kept dormant until increasing load demands more capacity
  - Several database instances can share a machine (database scavenging)
  - We do not need to replicate everything (less overhead for extraction)

SQL is not SQL

Amongst the 3333 most recent orders, the query performs a TOP-50 search to list a category's most popular books based on the quantity sold

```
SELECT * FROM (
 SELECT i_id, i_title, a_fname, a_lname,
   SUM(ol_qty) AS orderkey
 FROM item, author, order_line
 WHERE i_id = ol_i_id AND i_a_id = a_id
   AND ol_o_id > (SELECT MAX(o_id)-3333 FROM orders)
   AND i_subject = 'CHILDREN'
 GROUP BY i_id, i_title, a_fname, a_lname
 ORDER BY orderkey DESC
) WHERE ROWNUM <= 50
```

Virtual column specific to Oracle.
In PostgreSQL = LIMIT 50

Use of MAX leads to sequential scan in Postgres, change to:

```
SELECT o_id-3333 FROM orders
   ORDER BY o_id DESC LIMIT 1
```

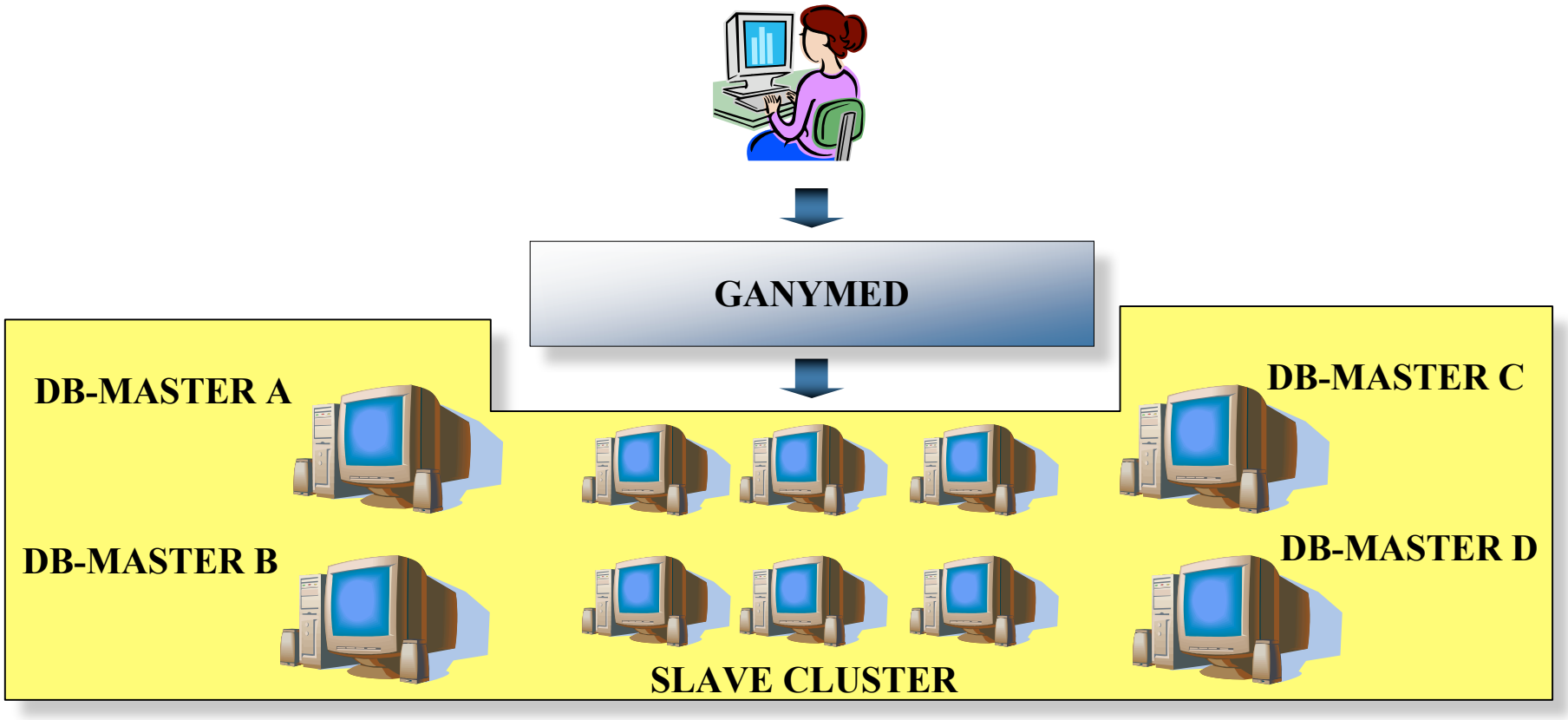
Current version does very basic optimizations on the slave side. Further work on optimizations at the middleware layer will boost performance even more

Optimizations can be very specific to the local data

GANYMED: Our real goals

Database scavenging

- Ideas similar to cluster/grid computing tools that place computing jobs in a pool of computers
- We want to dynamically place database slaves for master databases in a pool of computers
- The goal is to have a true low cost, autonomic database cluster



Steps to get there

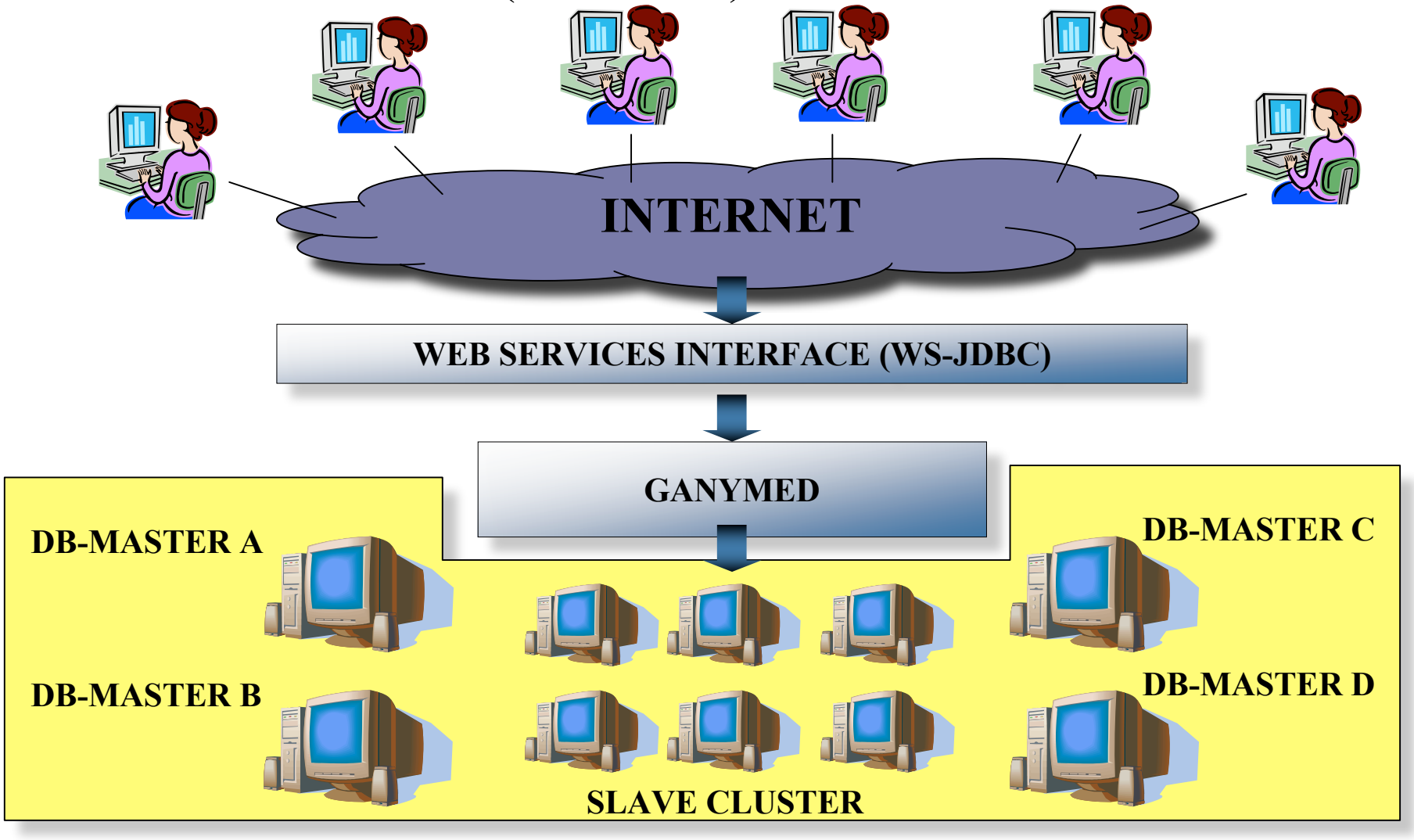
- We already have the performance and scalability gain
- We already have the ability to replicate commercial engines (Oracle, DB2, SQL Server)
- What is missing

  - Optimization of write set extraction or SQL update propagation
  - Optimization of SQL statements forwarded to slaves
  - Optimization of replication strategies in slaves

  - Dynamic creation of slaves (many possibilities)
  - Autonomic strategies for dynamic creation/deletion of slaves
  - Grid engine for resource management

Databases as commodity service

- Remote applications use the database through a web services enabled JDBC driver (WS-JDBC)



Conclusions

- Ganymed synthesizes a lot of previous work in DB replication
  - Postgres-R (McGill)
  - Middle-R (Madrid Technical Uni.)
  - Middleware based approaches (U. Of T.)
  - C-JDBC (INRIA Grenoble, Object Web)
  - ...
- Contributions
  - There is nothing comparable in open source solutions
  - Database independent
  - Very small footprint
  - Easily extensible in many context
    - Can be turned into a lazy replication engine
    - Can be used for data caching across WANs
    - Almost unlimited scalability for dynamic content \ web data
- Very powerful platform to explore innovative approaches
  - Databases as a commodity service
  - Database scavenging
  - Optimizations to commercial engines through open source slaves