Using MVCC for Clustered Databases

Markus Wanner

Oxford, July 2010

Using MVCC for Clustered Databases

structure

- introduction, scope and terms
- life-cycle of a transaction in Postgres-R
- write scalability tests
- results and their analysis

Using MVCC for Clustered Databases

focus: cluster

- high availability, load balancing, low cost
- fast, low latency interconnect
- shared-nothing

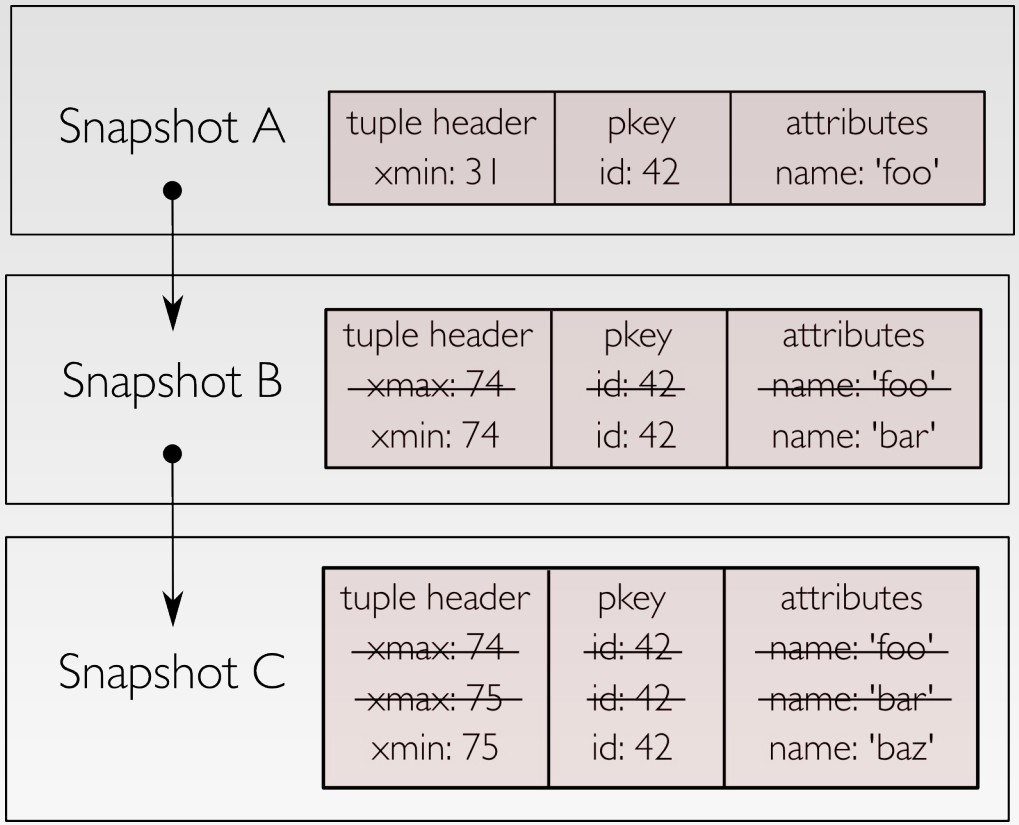
Using MVCC for Clustered Databases

focus: clustered database ideals

- application transparent
- transparent data distribution and load balancing
- ACID compliant
- easy to setup and maintain

Using MVCC for Clustered Databases

focus: multi-version concurrency control (1)



Using MVCC for Clustered Databases

focus: multi-version concurrency control (2)

- provides "point in time" consistent snapshots
- writes never block reads
- requires deferred background cleanup (vacuum)

Using MVCC for Clustered Databases

users of multi-version concurrency control

Databases: Postgres, Oracle, Microsoft SQL Server, IBM DB2, some MySQL storage engines, Ingres, Sybase SQL Anywhere, Sybase IQ, Berkeley DB, Firebird, CouchDB, HSQLDB, H2, InterBase, …

Others: JBoss Cache and Infinispan, Clojure, Subversion, Ehcache, …

Source: http://en.wikipedia.org/wiki/Multiversion_concurrency_control

Using MVCC for Clustered Databases

life-cycle of a transaction: overview

- collection and distribution of changes
- optimistic and parallel application
- conflict detection using MVCC
- commit independently but in congruent order

Using MVCC for Clustered Databases

life-cycle 1: collection of change sets

changes are collected and serialized into change sets

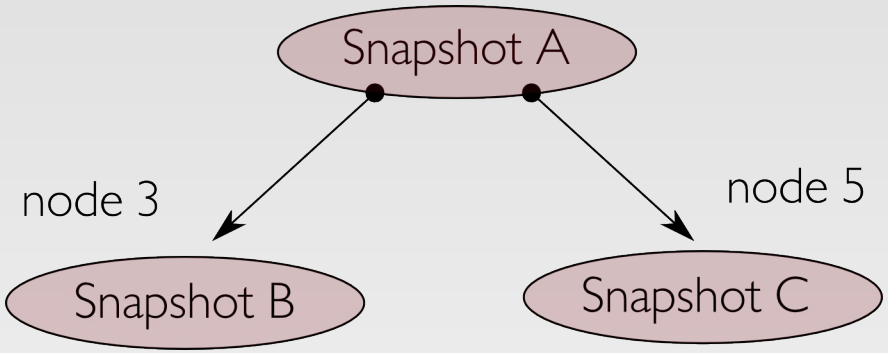
includes: primary key and origin information

tuple origin	pkey	changed atts
n:5 / xid:31	id: 42	name: 'bar'
n:2 / xid:27	id: 88	name: 'foo'

a size threshold triggers eventual broadcasts

life-cycle 2: optimistic and parallel application

background worker infrastructure allows parallel application of remote transactions



Using MVCC for Clustered Databases

life-cycle 3: conflict detection using MVCC

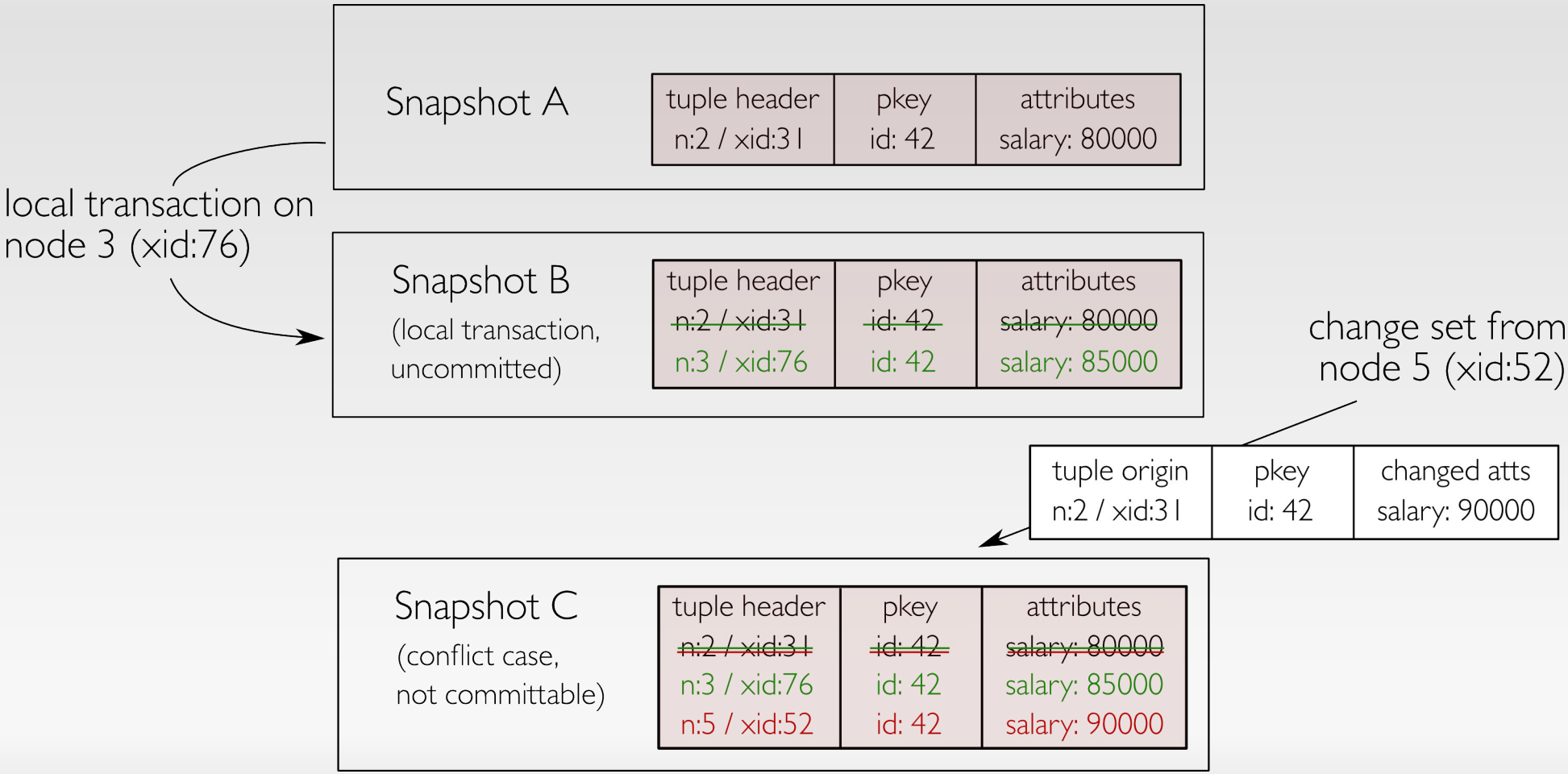
compares visible version and to-be-updated version

mismatches signal a conflict

MVCC allows identifying the conflicting transaction

Using MVCC for Clustered Databases

life-cycle 3: conflict detection using MVCC (example)



Using MVCC for Clustered Databases

life-cycle 4: congruent commit decision

before commit, an agreement on the ordering of transactions is required

nodes may progress at different speeds

applying the same changes in the same order guarantees eventual consistency

Using MVCC for Clustered Databases

advantages of using low-level MVCC interface

built-in support for:

- triggers
- UDFs
- rules

easy support for:

- savepoints

Using MVCC for Clustered Databases

write scalability

overall computing power, memory and storage capacity scale easily

the interconnect will inevitably be the bottleneck at larger scale

Using MVCC for Clustered Databases

write scalability: how to fight network congestion

- minimize number of nodes to replicate to (partitioning)
- minimize communication overhead between nodes
- minimize transaction runtime

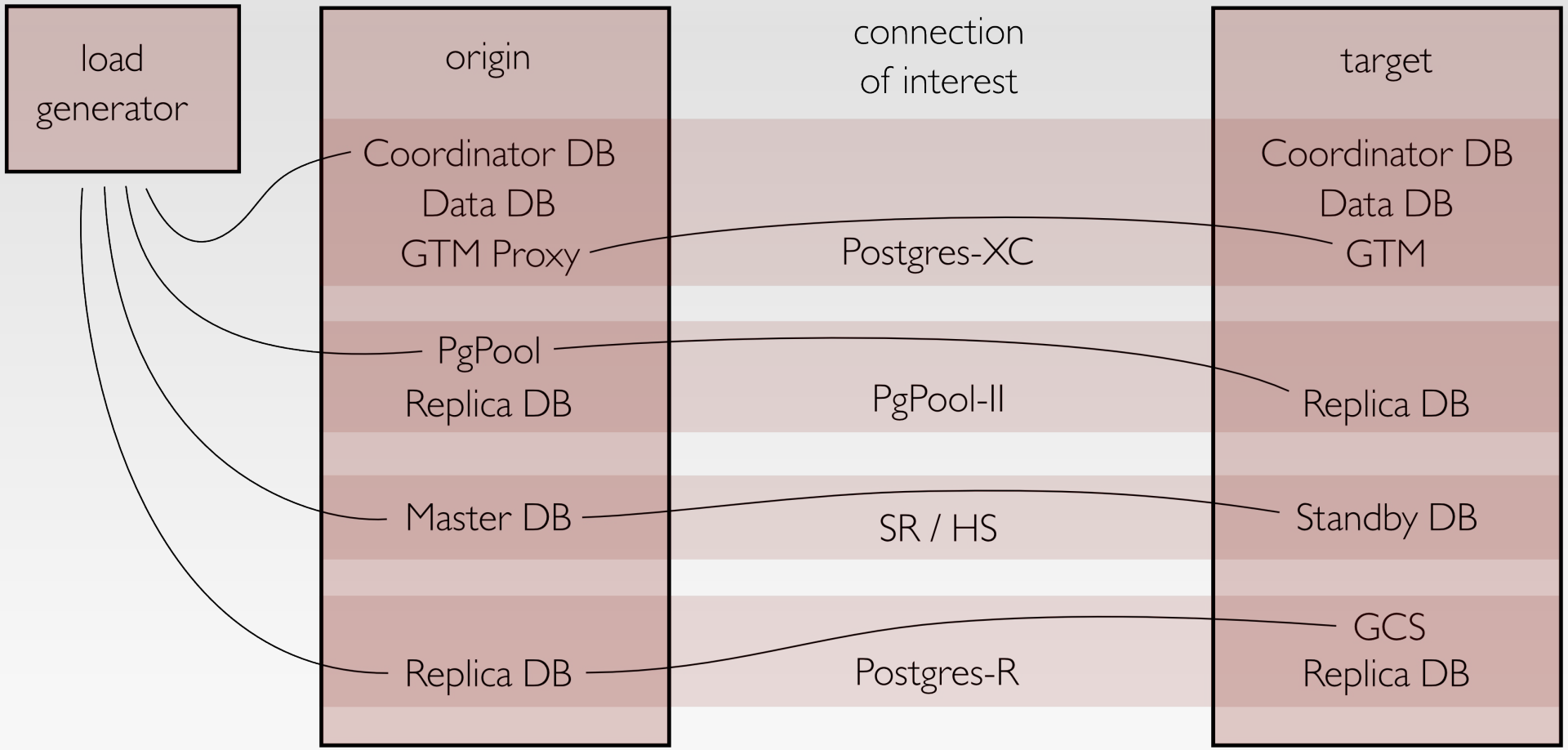
Using MVCC for Clustered Databases

traffic test: systems tested

- PgPool-II (2.3.3)
- Postgres-XC (0.9.1)
- Postgres-R (06-2010)
- Postgres Hot-Standby (9.0 Beta 2)

Using MVCC for Clustered Databases

traffic test: setup



Using MVCC for Clustered Databases

traffic test: example transaction

new-order transaction from dbt2 (used 45% there)

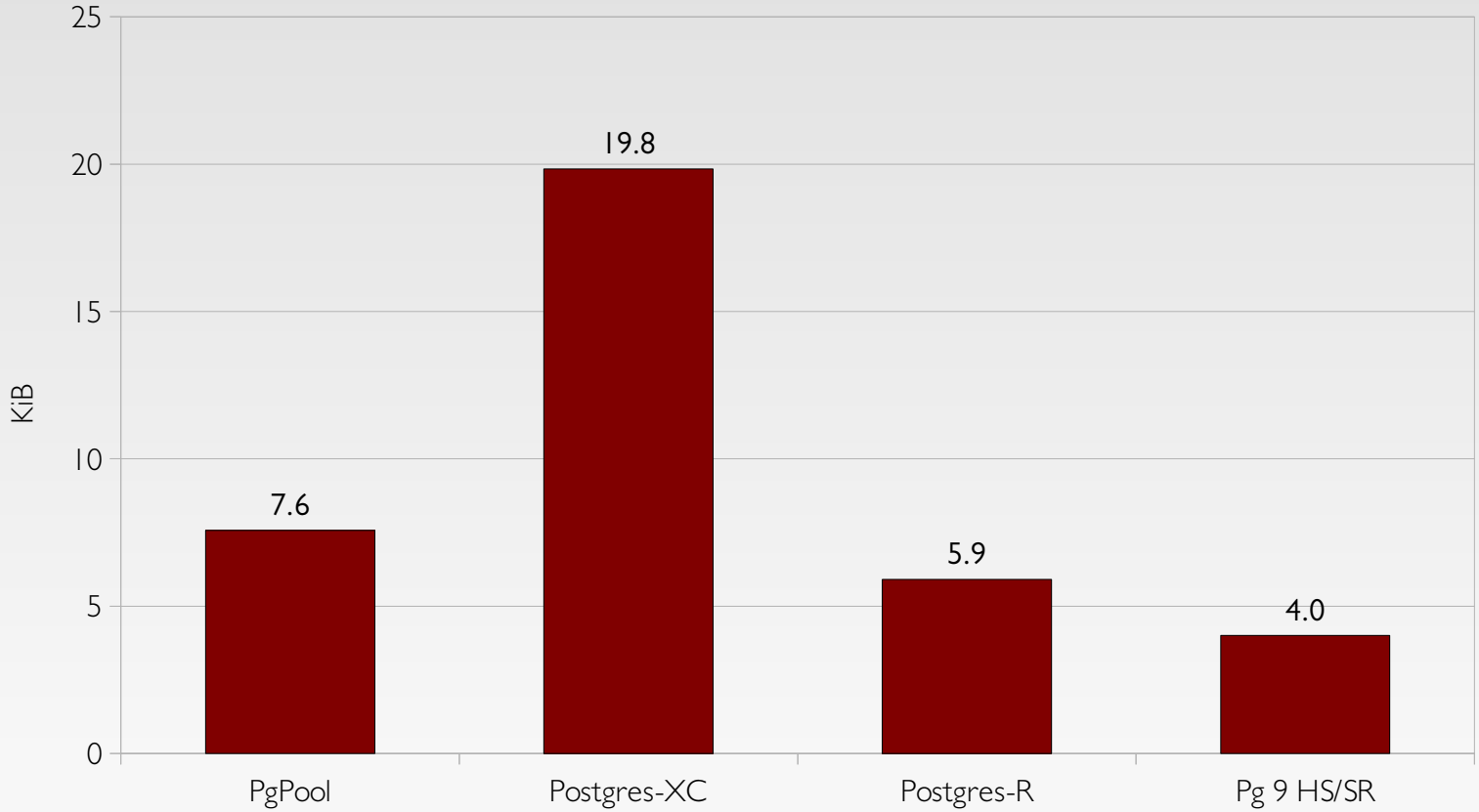
- 7 - 17 row selections (no impact on replicas)
- 6 - 16 selects followed by updates
- 7 - 17 inserts

simulated avg. new-order transaction:

11 updates and 12 inserts, scripted, no UDF

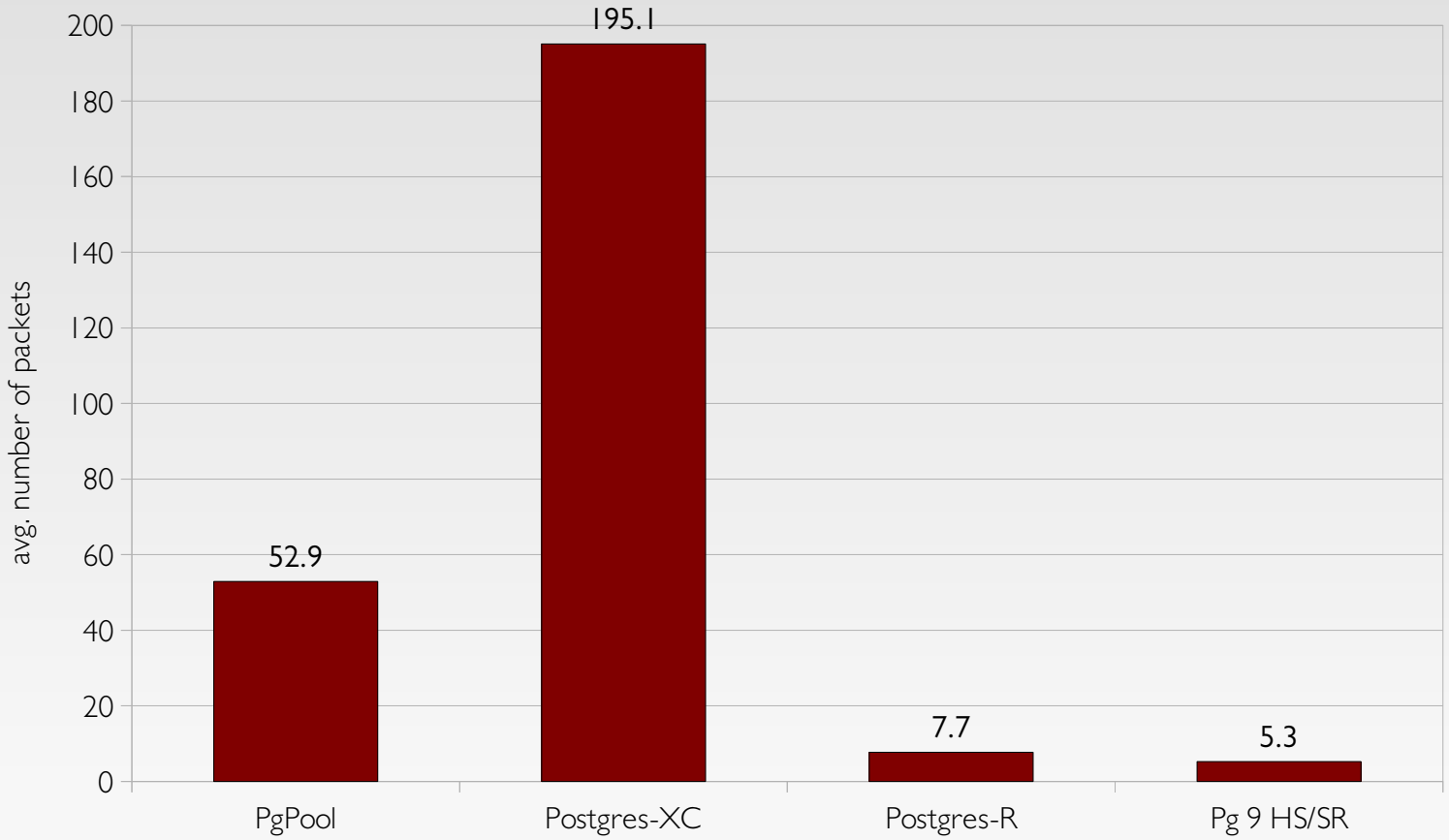
Using MVCC for Clustered Databases

traffic test results: avg. data transmitted per txn



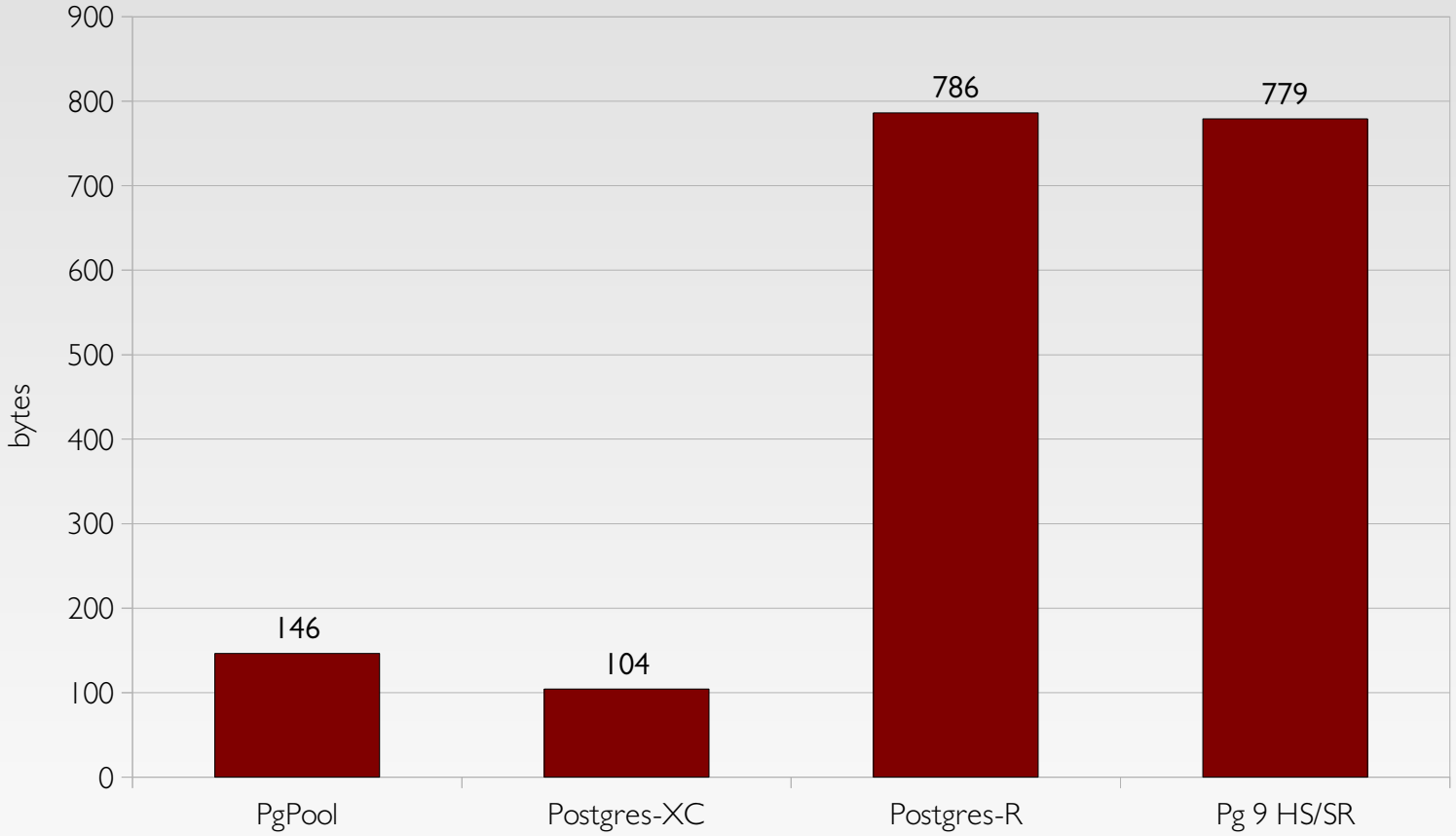
Using MVCC for Clustered Databases

traffic test results: packets exchanged per transaction



Using MVCC for Clustered Databases

traffic test results: avg. packet size



Using MVCC for Clustered Databases

latency test: description

- added artificial 1000ms delay
- measured latency for single operations
- result figure: amount of round-trips required

Using MVCC for Clustered Databases

latency test results: round-trip count comparison

	txn start	txn commit (read-only)	txn commit (writing)	txn rollback	savepoint	release savepoint	rollback to savepoint	insert	update	delete	sequence nextval	sequence setval
PgPool-II (2.3.3)	1	1	1	1	1	1	1	1/2[1]	1	1	1	1
Postgres-XC (0.9.1)	2[2]	2	5-7	2	-	n/a	-[3]	2-3	2	2	3-5[4]	2-3
Postgres-R (2010-06)	0	0	1	0	0	0	0	0	0	0	1	1

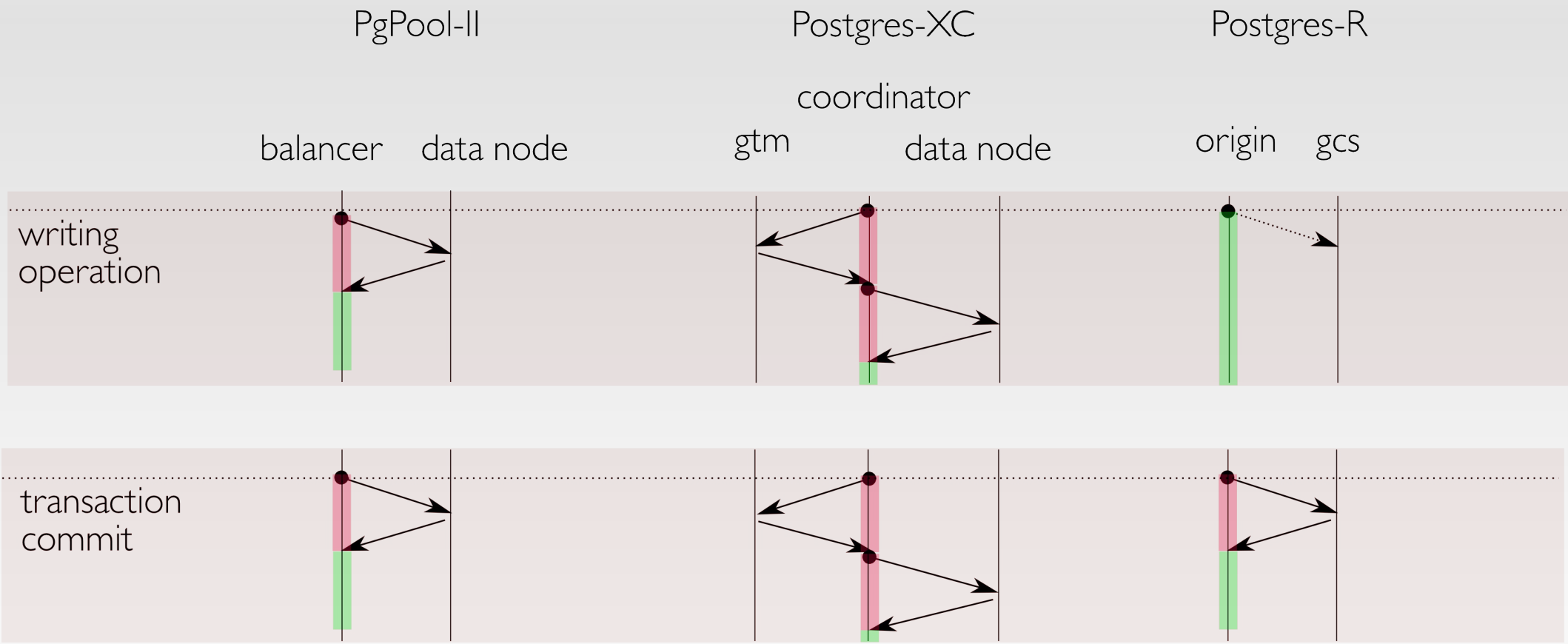
[1] 2 RTT if a SERIAL is involved and insert_lock == true

[2] deferred until a snapshot is requested

[3] savepoints are not supported in Postgres-XC 0.9.1

[4] nextval() is free if used from an INSERT

latency test analysis: time-lines



Using MVCC for Clustered Databases

new order transaction latency

	txn start	inserts (12)	updates (11)	txn commit		estimated total round-trips	measured runtime (at 1000ms latency)
PgPool-II (2.3.3)	1	12	11	1		25	26.25
Postgres-XC (0.9.1)	2	24	22	6		54	58.45
Postgres-R (2010-06)	0	0	0	1		1	1.19

thank you