Portable Scale-Out Benchmarks for MySQL

MySQL User Conference 2008

Robert Hodges – CTO Continuent, Inc.

continuent

Agenda

- Introductions
- Scale-Out Review
- Bristlecone Performance Testing Tools
- Scale-Out Benchmarks in Action
- Final Words

About Continuent

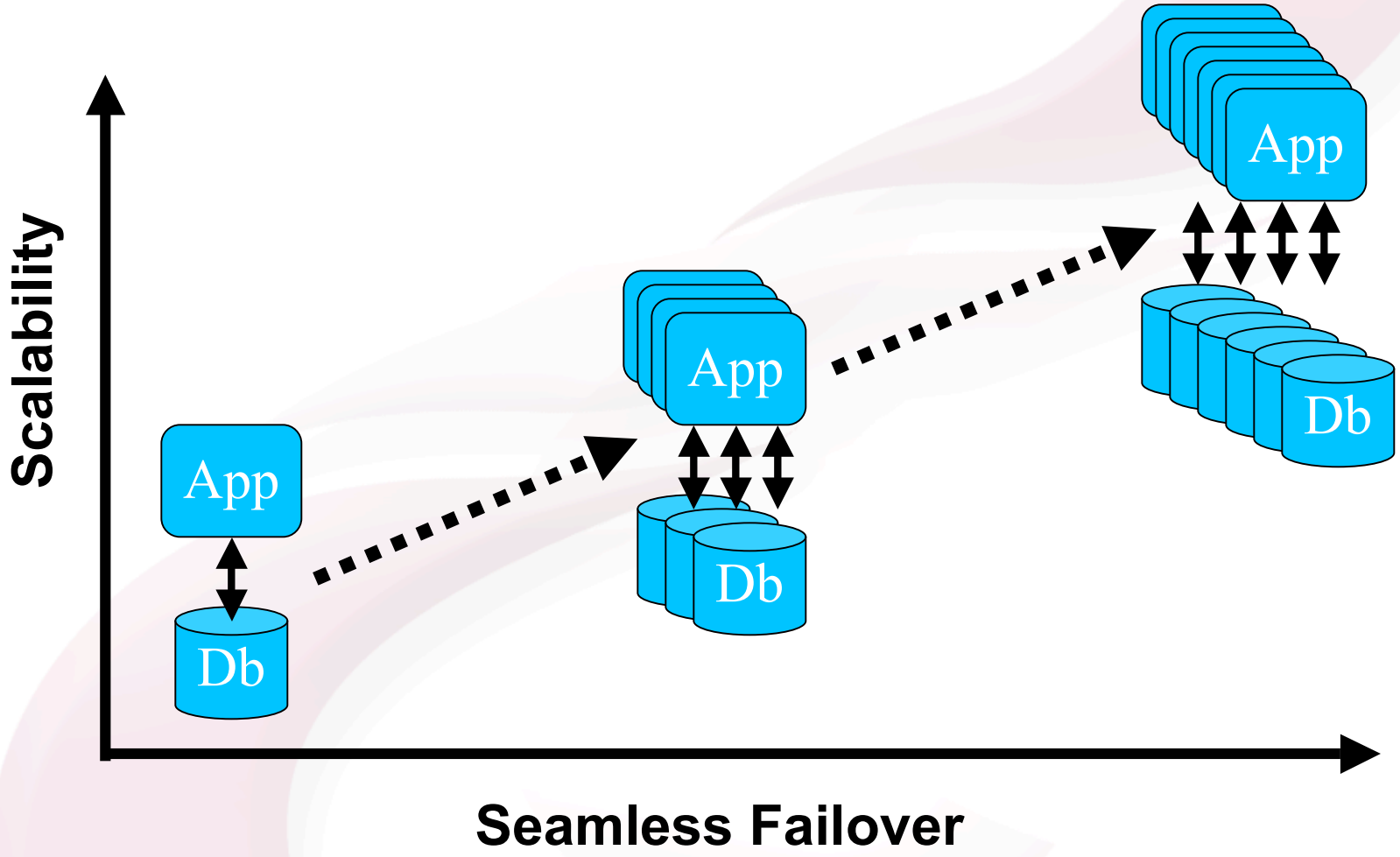
- **Company**
  - The leading provider of open source database availability and scaling solutions
- **Solutions**
  - uni/cluster – multi-master database clustering that replicates data across multiple databases and load balances reads
  - Uses “database virtualization” to provide a seamless client interface
- **Value**
  - Low-cost open source business critical solutions
  - Highly available data
  - Raise performance and hardware utilization through load balancing
  - Remove chance of data loss
- **Open Source**
  - Sequoia - generic middleware clustering for JDBC databases
  - Hedera - Group communications adapter library
  - Myosotis - Java proxy for native PostgreSQL and MySQL clients
  - Bristlecone - Performance benchmarks for database clusters
- **Collaborations: GORDA (Open DB Replication Architecture)**
  - http://gorda.di.uminho.pt

Scale-Out Review

Scale-Out Design Motivation

- ***Everybody*** **wants more database availability**
  - Protection from database failure
  - Protection from site failures
  - Continuous operation during upgrades
- ***Many people*** **want more throughput or better response times**
  - Ability to deal with peak loads
  - Ability to scale large numbers of writes
  - Ability to scale large numbers of reads
- **A simple way to meet both requirements is to scale out using multiple copies of data**
  - If one database fails other databases take over
  - Extra copies can be used to distribute reads
- ***Scale-out designs*** **offer the promise of linear scaling of costs vs. capacity starting from a small base**

The Dream - Flexible Availability and Scaling



If Everyone Wants Scale-Out, then Why…

- **Doesn't everyone have it already?**
- **Creating a set of identical replicas on different hosts is hard**
  - Distributed serialization is less powerful than serialization within DBMS
  - Brewer's conjecture
  - Promising algorithms require DBMS support
- **Scale-out approaches have trade-offs of one kind or another**
  - DDL support
  - Inconsistent reads between replicas
  - Sequences
  - Non-deterministic SQL
  - Deadlocks
- **Many scale-out approaches are therefore non-transparent**
  - No surprise: scale-out more common in web applications written in light-weight scripted languages rather than enterprise systems
- **Different scale-out approaches have different performance issues**

Three Basic Scale-Out Technologies

- **Data replication -- Copies data between databases**
  - Where are updates processed? Master/master vs. master/slave
  - When are updates replicated? Synchronous vs. asynchronous
- **Group communication -- Coordinates messages between distributed processes**
  - Views -- Who is active, who is crashed, do we have quorum, etc.
  - Message Delivery -- Ordering and delivery guarantees
- **Proxying - "Virtualizes" databases and hides database locations from applications**
  - Failover, load-balancing, caching, etc.

Three Replication Algorithms

- **Master/Slave -- Accept updates at a single master and replicate changes to one or more slaves**
  - Aka Primary/Backup

- **Multi-Master State Machine - Deliver a stream of updates in the same order simultaneously to a set of databases**

- **Certification - Optimistically execute transactions on one of a number of nodes and then apply to all nodes after confirming serialization**

Our Goal at Continuent

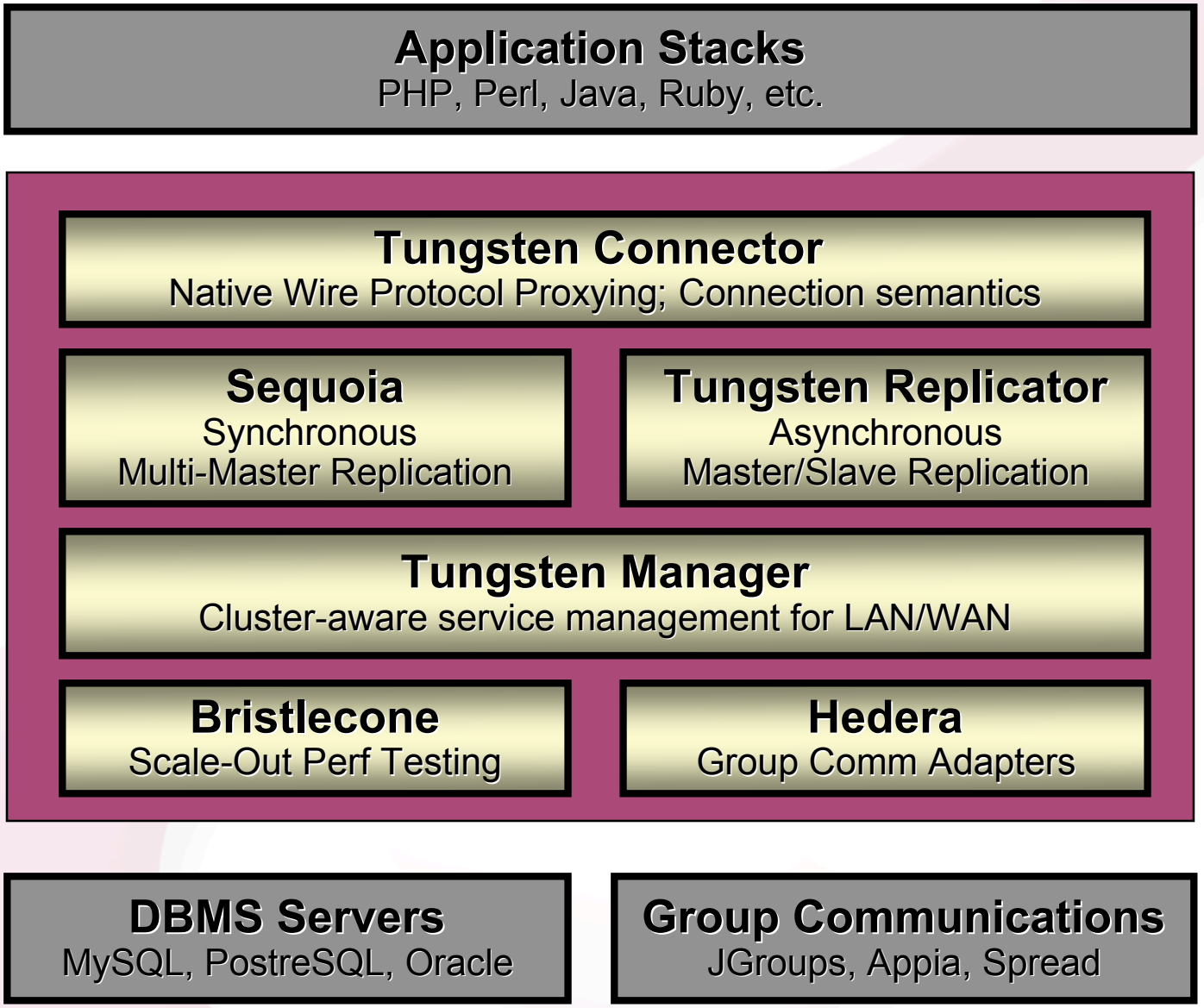
Highly reliable, transparent, easy to deploy solutions for data availability and performance scalability that work with unaltered, stock RDBMS and economical, off-the-shelf hardware

Tungsten: An architecture for scale-out that meets the goal

Tungsten Overview

- Data Services integrate multiple copies of data within and across sites into a single highly available, performant database
- Database virtualization with controlled data latency and consistency
- Database-neutral replication designed for high availability
- Cluster-aware management framework backed by excellent graphical tools
- "Stack" organization of technology to allow substitution and extensions

An Open, Database-Neutral Scale-Out Stack



Tungsten Data Service

Bristlecone Performance Testing Tools

Performance Testing Strategy

- **Run appropriate tests**
  - Mixed load tests to check overall throughput and scaling
  - Micro-benchmarks to focus on specific issues
- **Use appropriate workloads**
  - Scale-out use profiles are often read- or write-intensive
- **Cover key issues**
  - Read latency through proxies
  - Read and write scaling
  - Slave latency for master/slave configurations
  - Group communication and replication bottlenecks
  - Aborts and deadlocks
- **Generate sufficient load in the right places**
  - Many transactions/queries
  - Large data sets
  - CPU, buffer cache, sort space, disk speeds
  - Different datatypes (especially BLOB/TEXT data)
  - A diversity of operations
  - Sufficient warm-up time
  - Batch operations

Bristlecone Performance Test Tools

- **Continuent needed a set of benchmarks that would be easy to set up and cover a wide range of scale-out issues**
- **We created Bristlecone (http://bristlecone.continuent.org)**
- **Goal is to provide a portable performance framework that:**
  - Covers basic performance as well as issues of interest for scale-out
  - Works generically across database types
  - Is easy to use and extend to add new tests
- **Roadmap for three main types of tests**

Type	Description
Load	Simulate use profile with mixed transactions
Batch	Batch transaction loading
Micro-benchmarks	Focused tests to check specific use cases

- **We have load and micro-benchmarks now**
  - More load tests followed by batch tests

Bristlecone Load Testing: Evaluator

- Java tool to generate mixed load on databases
- Similar to pgbench but works cross-DBMS
- Can easily vary mix of select, insert, update, delete statements
- Default select statement designed to "exercise" the database:

```
select c.* from tbl3 c join
  tbl1 a on c.k1 = a.k1 join
    tbl2 b on c.k2 = b.k2
  where a.value between 10 and 100
      and b.value between 90 and 200
```

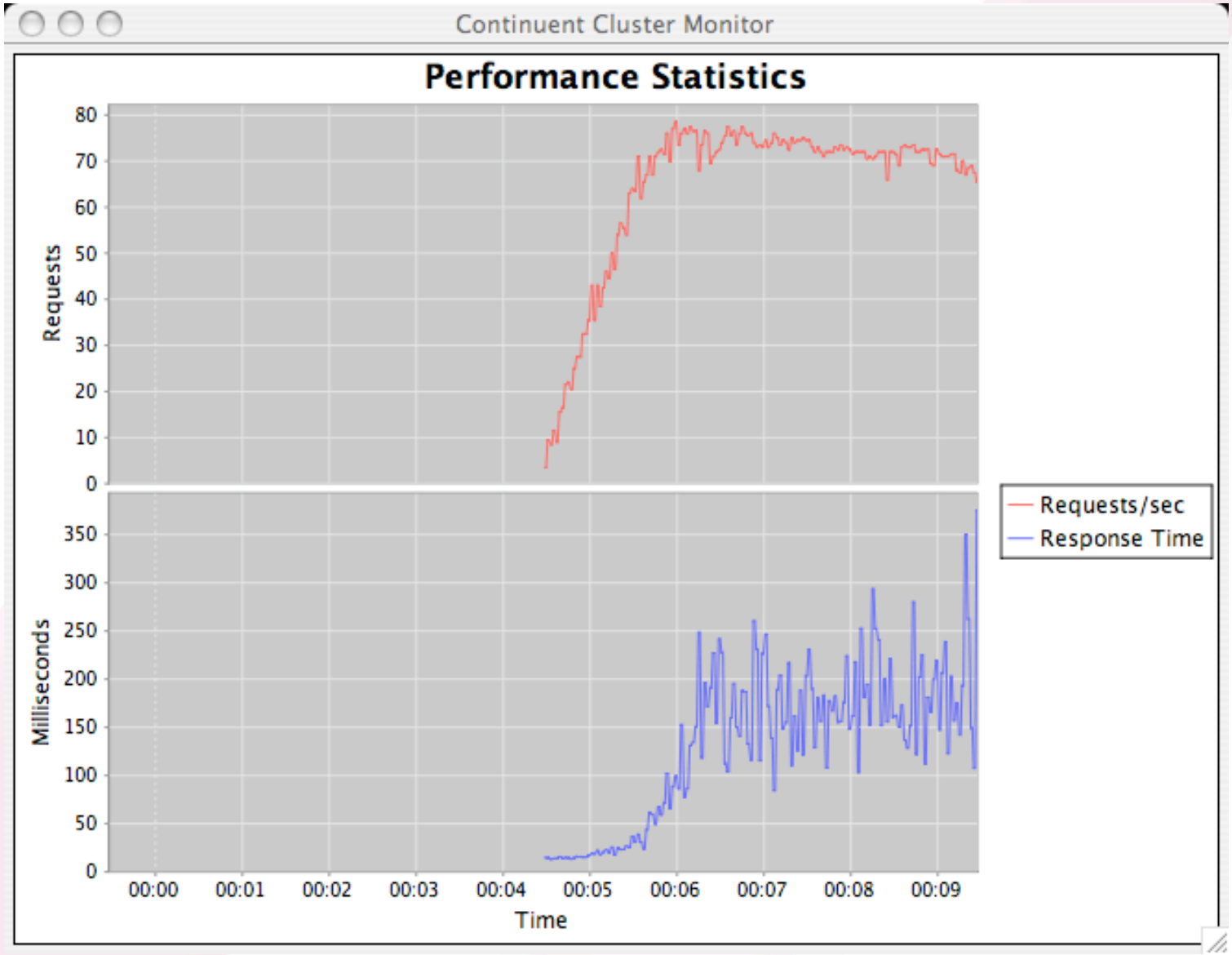
- Can choose lightweight queries as well (e.g., fetch a single row using a key)
- Parameters are defined in a simple configuration file
- Can generate graphical, XML, CSV, and HTML output

Sample Evaluator Configuration File

```
<EvaluatorConfiguration name="postgres" testDuration="600"
      autoCommit="true" statusInterval="2" xmlFile="postgresResults.xml"
      csvFile="postgresResults.csv">

      <Database driver="org.postgresql.Driver"
            url="jdbc:postgresql://coot/standalone"
            user="benchmark"
            password="secret"/>
      <TableGroup name="tbl" size="200">
            <ThreadGroup name="A" threadCount="100" thinkTime="500"
                  updates="7" deletes="1" inserts="2" readSize="10”
                  queryWeight=“medium”
                  rampUpInterval="5" rampUpIncrement="10"/>
      </TableGroup>
</EvaluatorConfiguration>
```

Evaluator Graphical Output



Bristlecone Micro-Benchmarks: Benchmark

- Java tool to test specific operations while systematically varying parameters
- Benchmarks run “Scenarios”--specialized Java classes with interfaces similar to JUnit
- Configuration file generates cross-product of parameters and runs benchmarks in success
- Scenario instances are spawned in separate threads - multi-client testing built in
- Bound scenario runs by iterations or duration (time)
- Utility library for manipulating sets of SQL tables, populating classes, etc., on different databases
- Can generate CSV and HTML output

Sample Benchmark Configuration File

```
# Scenario name.
scenario=com.continuent.bristlecone.benchmark.scenarios.ReadSimpleScenario

# Database connection information.
include=cluster.properties|standalone.properties

# Test duration and number of threads.
bound=duration
duration=60
threads=1|4|16

# Database table information.
tables=1
datatype=varchar
datawidth=100
datarows=10|100|1000|10000
```

Current Micro Benchmarks

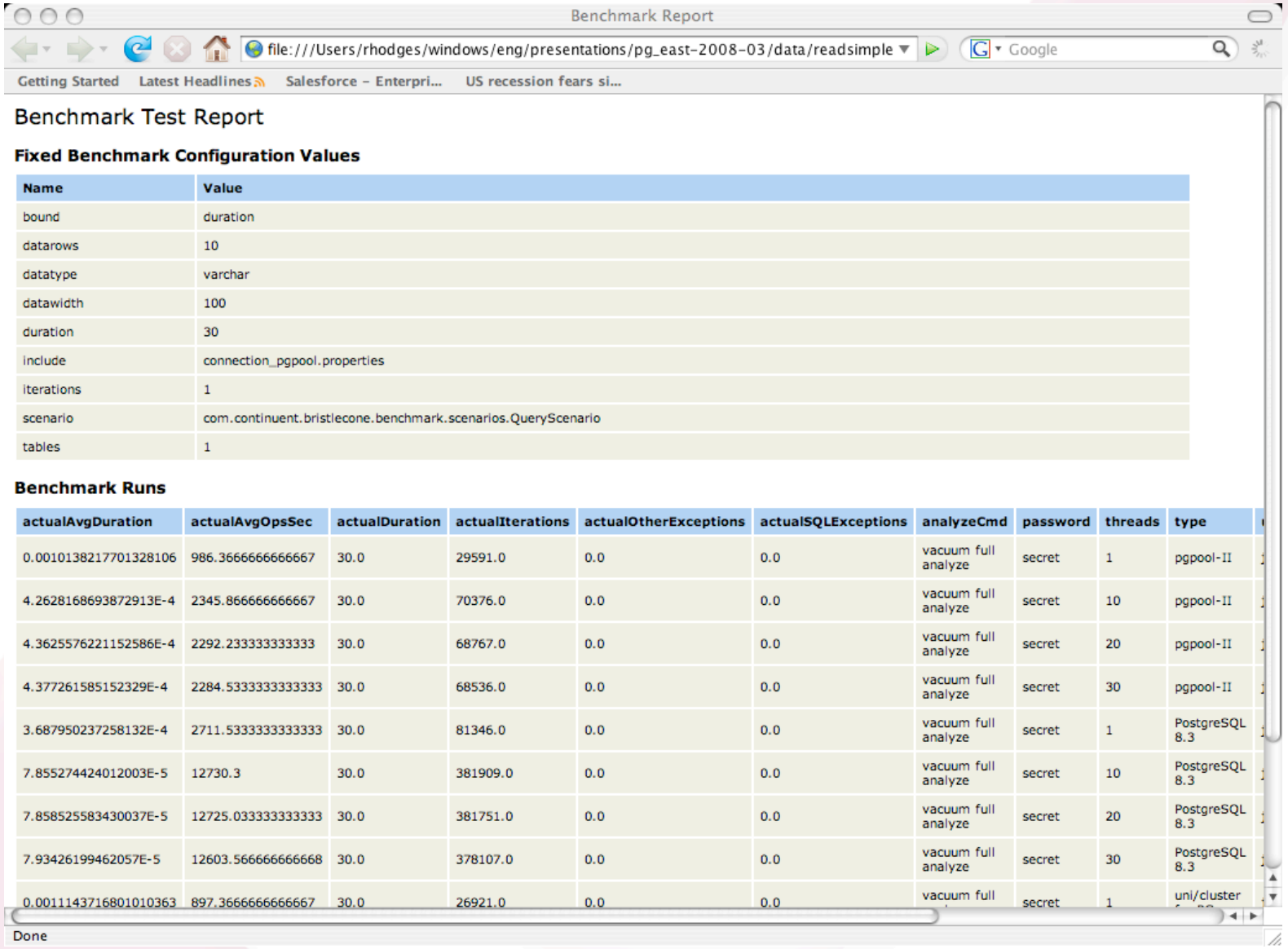
- **Basic Read Latency - Low database stress**
  - ReadSimpleScenario - Simple reads of varying sizes
  - ReadSimpleLargeScenario - Simple reads with very large result sets
- **Read Scaling - High database stress**
  - ReadScalingAggregatesScenario - CPU-intensive queries
  - ReadScalingInvertedKeysScenario - Reads on random keys to load buffer cache
- **Write latency and scaling - Low/high stress**
  - WriteSimpleScenario - Basic inserts
  - WriteComplexScenario - Updates that perform reads of varying length
  - ReadWriteScenario - Updates with complex query conditions
- **Deadlocks - Variable transaction lengths**
  - DeadlockScenario - Writes with varying likelihood of deadlocks
- **TPC-B scenario will be added shortly**
  - Thanks to a contribution from GORDA

Micro Benchmark Modifiers

- Most micro-benchmarks allow standard modifiers in addition to connection properties. Additional modifiers are provided for each scenario.

Property Name	Description
tables	Number of tables in the benchmark
datawidth	Length of column to “fatten” tables
datatype	Datatype of column to “fatten” tables
datarows	Number of rows per table
autocommit	If true use autocommit, otherwise use transactions
operations	Number of operations per transaction

Benchmark HTML Output



Benchmark Test Report

Fixed Benchmark Configuration Values

Name	Value
bound	duration
datarows	10
datatype	varchar
datawidth	100
duration	30
include	connection_pgpool.properties
iterations	1
scenario	com.continuent.bristlecone.benchmark.scenarios.QueryScenario
tables	1

Benchmark Runs

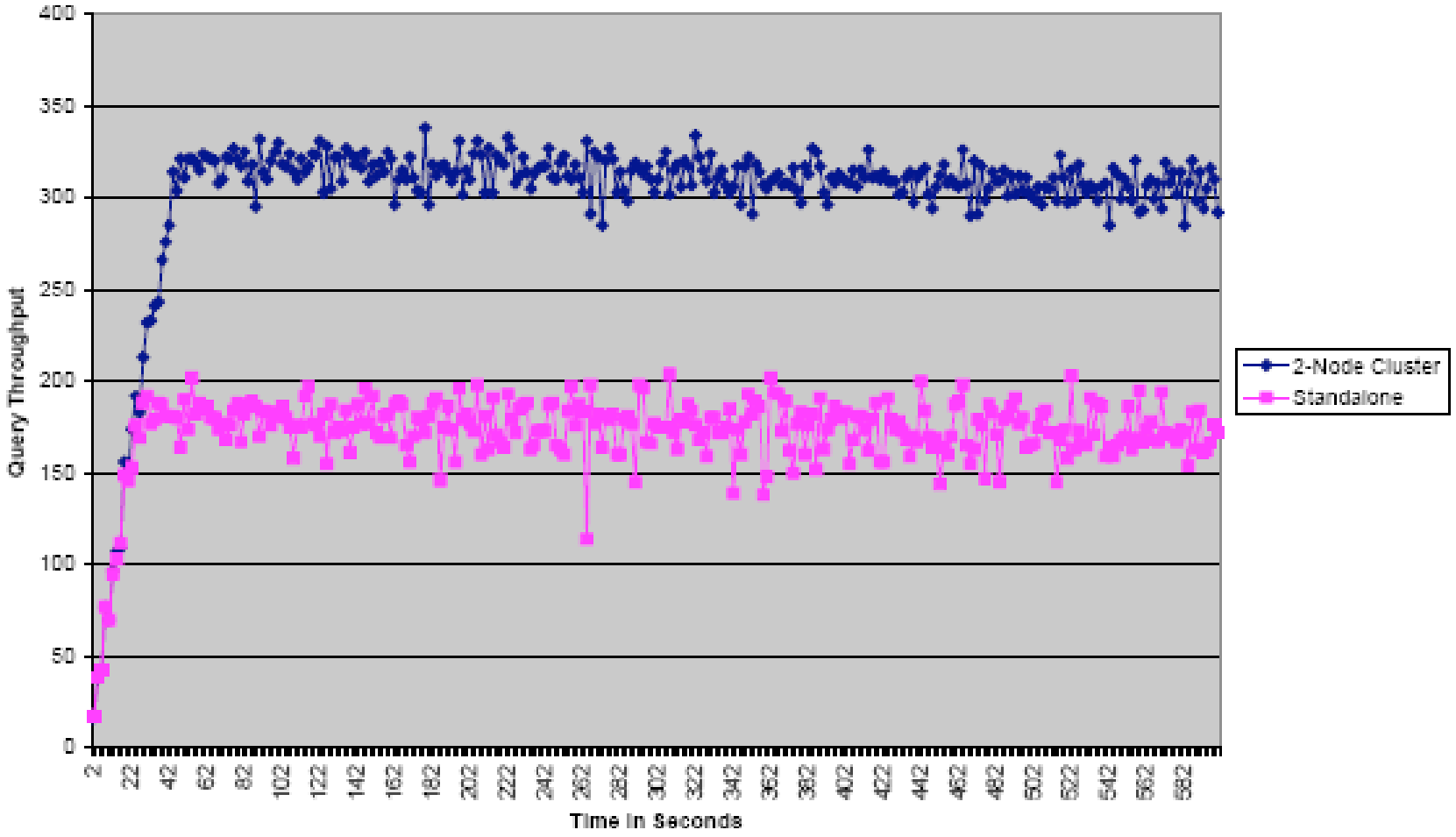
actualAvgDuration	actualAvgOpsSec	actualDuration	actualIterations	actualOtherExceptions	actualSQLExceptions	analyzeCmd	password	threads	type
0.0010138217701328106	986.3666666666667	30.0	29591.0	0.0	0.0	vacuum full analyze	secret	1	pgpool-II
4.2628168693872913E-4	2345.866666666667	30.0	70376.0	0.0	0.0	vacuum full analyze	secret	10	pgpool-II
4.3625576221152586E-4	2292.233333333333	30.0	68767.0	0.0	0.0	vacuum full analyze	secret	20	pgpool-II
4.377261585152329E-4	2284.5333333333333	30.0	68536.0	0.0	0.0	vacuum full analyze	secret	30	pgpool-II
3.687950237258132E-4	2711.5333333333333	30.0	81346.0	0.0	0.0	vacuum full analyze	secret	1	PostgreSQL 8.3
7.855274424012003E-5	12730.3	30.0	381909.0	0.0	0.0	vacuum full analyze	secret	10	PostgreSQL 8.3
7.858525583430037E-5	12725.033333333333	30.0	381751.0	0.0	0.0	vacuum full analyze	secret	20	PostgreSQL 8.3
7.93426199462057E-5	12603.566666666668	30.0	378107.0	0.0	0.0	vacuum full analyze	secret	30	PostgreSQL 8.3
0.0011143716801010363	897.3666666666667	30.0	26921.0	0.0	0.0	vacuum full	secret	1	uni/cluster

Bristlecone Testing Examples

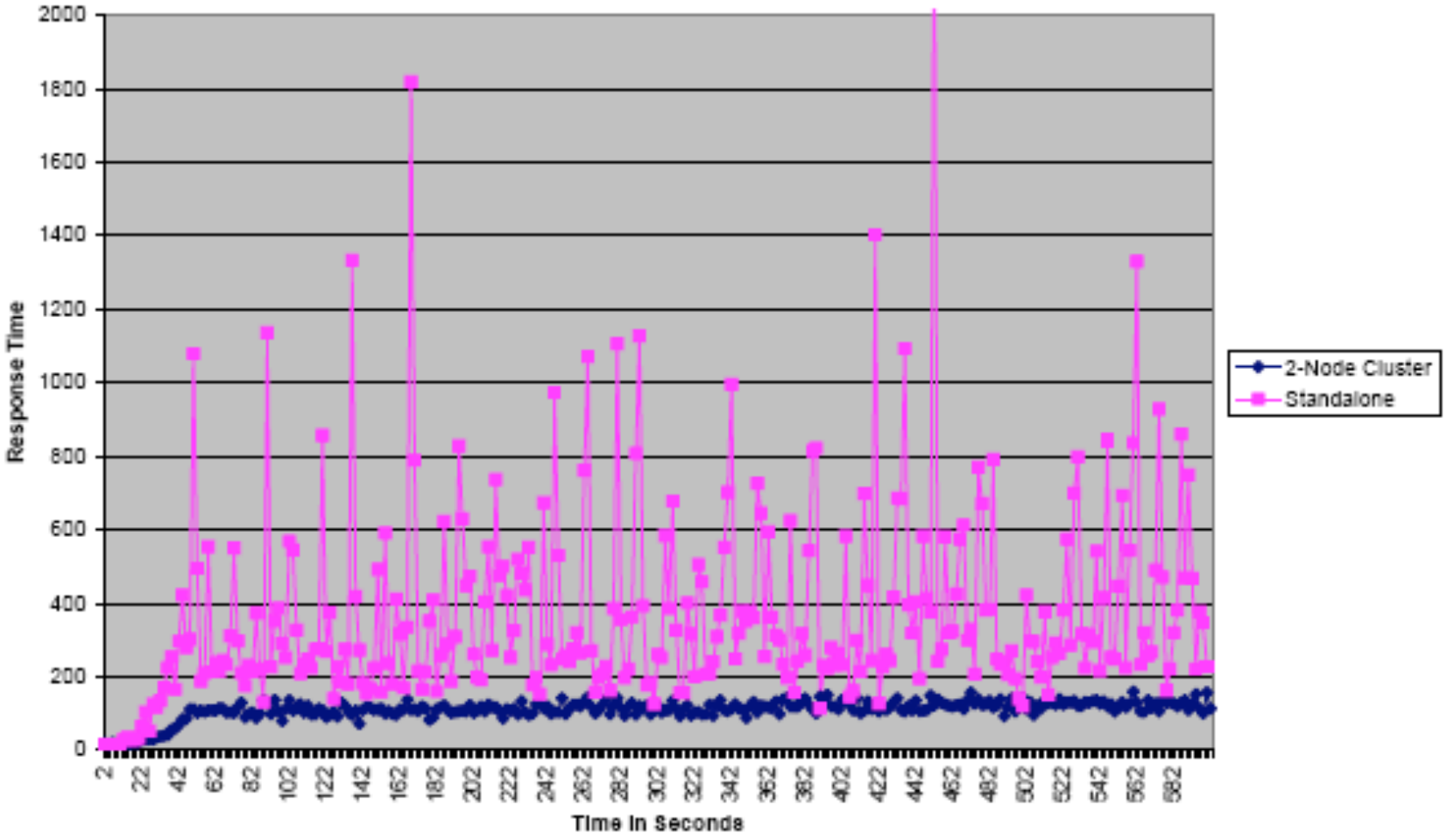
Measuring Mixed Loads With Evaluator

- **Question: How does middleware clustering compare with direct database access for a 10/90 write/read load**
- **Approach: Use separate Evaluator runs with appropriate load settings**
  - Try different numbers of clients
  - Try different “weights” on queries to generate more server load
- **Test Resources: SC 850 host(s), Hyperthreaded Pentium-4, 1GB memory, SATA disks**

Mixed Load Query Throughput



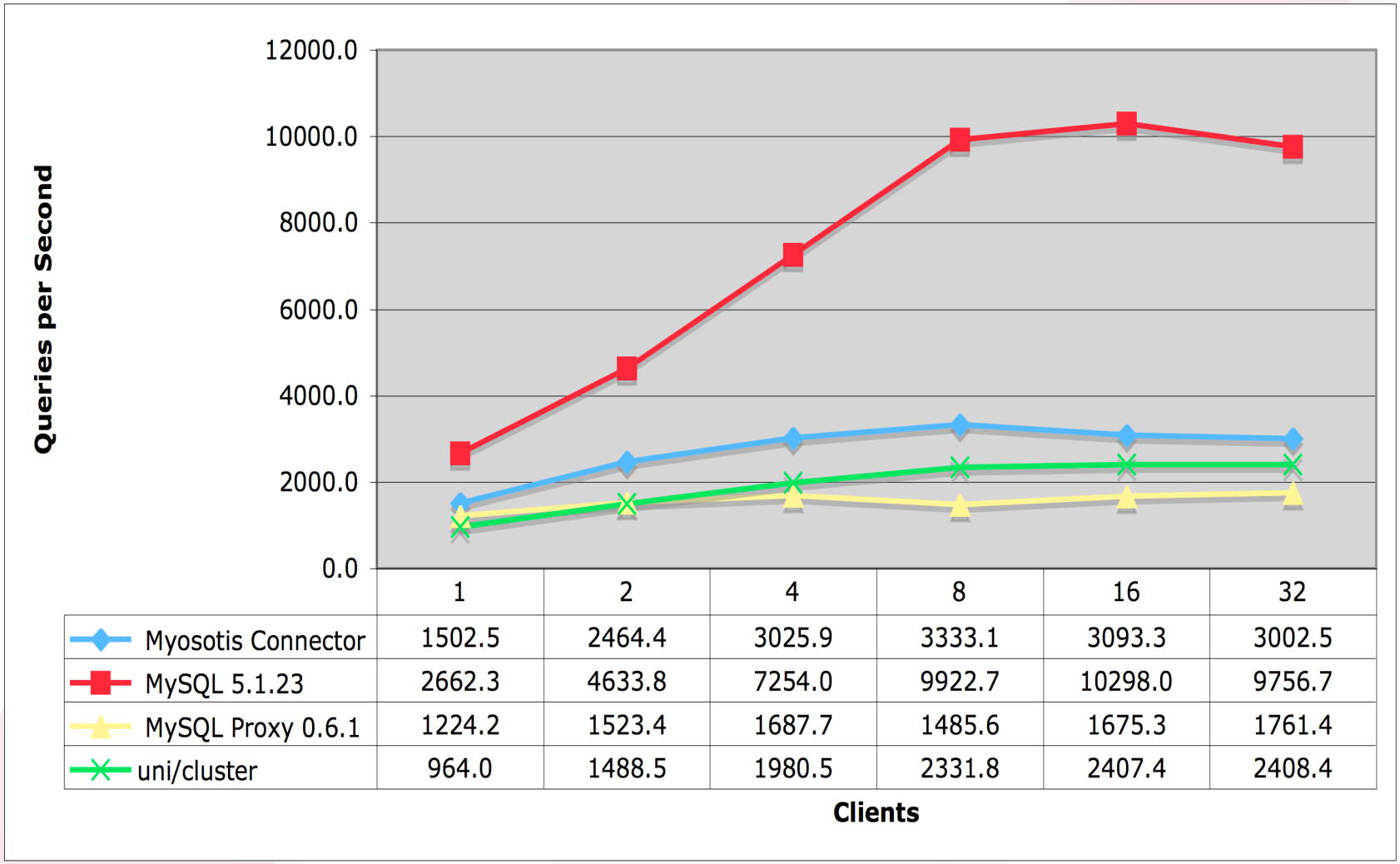
Mixed Load Query Response



Measuring Read Latency Costs

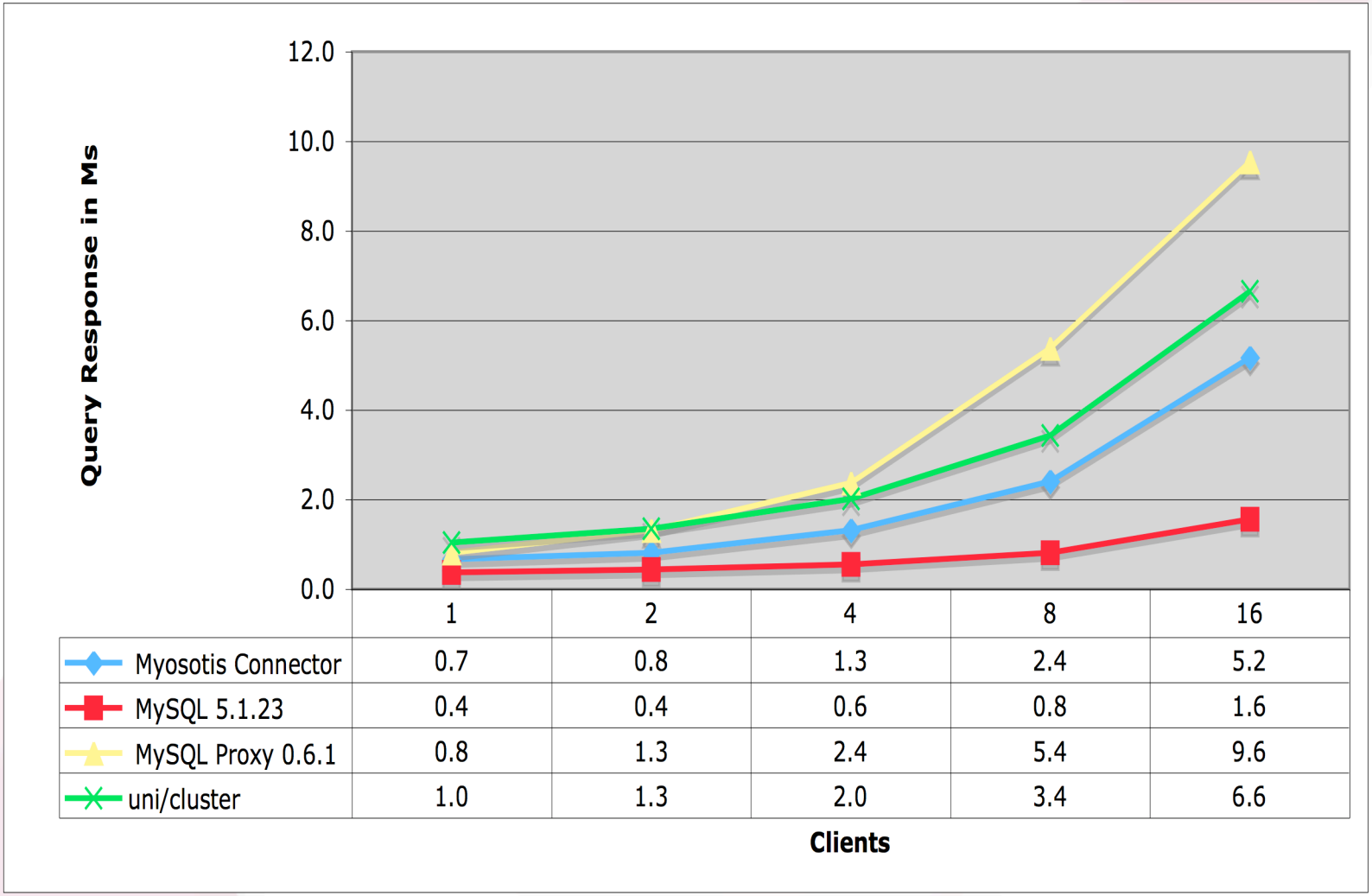
- **Question: How much latency is induced by middleware and/or proxies?**
- **Approach: Run queries of varying lengths with varying numbers of clients**
  - Queries should use data in buffer cache to maximize database speed
- **Test Scenario: ReadSimpleScenario**
  - Fix rows and measure queries per second as number of clients increases
  - Fix clients and measure number of queries per second as rows selected increase
  - Query very simple: 'select * from benchmark_scenario_0'
- **Test Resources: Dell SC 1425, Xeon 2 CPU x 2 Cores, 2.8Mhz, 1GB Memory**

Proxy Query Throughput (MySQL 5.1=Base)



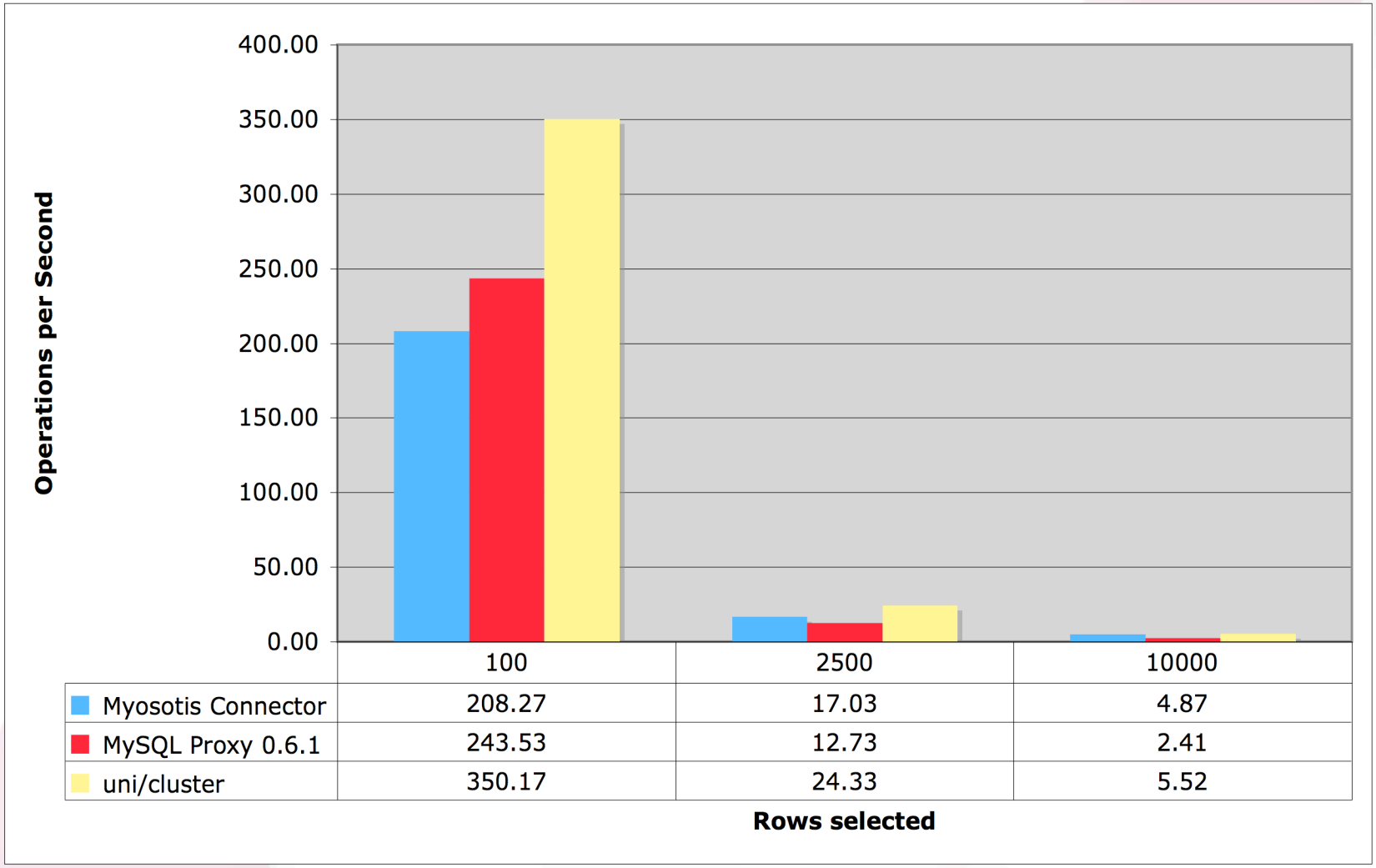
	1	2	4	8	16	32
Myosotis Connector	1502.5	2464.4	3025.9	3333.1	3093.3	3002.5
MySQL 5.1.23	2662.3	4633.8	7254.0	9922.7	10298.0	9756.7
MySQL Proxy 0.6.1	1224.2	1523.4	1687.7	1485.6	1675.3	1761.4
uni/cluster	964.0	1488.5	1980.5	2331.8	2407.4	2408.4

Proxy Query Response (MySQL 5.1=Base)



	1	2	4	8	16
Myosotis Connector	0.7	0.8	1.3	2.4	5.2
MySQL 5.1.23	0.4	0.4	0.6	0.8	1.6
MySQL Proxy 0.6.1	0.8	1.3	2.4	5.4	9.6
uni/cluster	1.0	1.3	2.0	3.4	6.6

Proxy Throughput as Rows Returned Vary



	100	2500	10000
Myosotis Connector	208.27	17.03	4.87
MySQL Proxy 0.6.1	243.53	12.73	2.41
uni/cluster	350.17	24.33	5.52

Measuring Read Scaling

- **Question: How does scaling out to multiple hosts increase read capacity? At what point does a cluster do better?**
- **Approach: Compare using standalone database vs. cluster using queries of varying "weights" and numbers of clients**
- **Test Scenario: ReadScalingAggregatesScenario**
  - Runs a CPU intensive cross-product query:

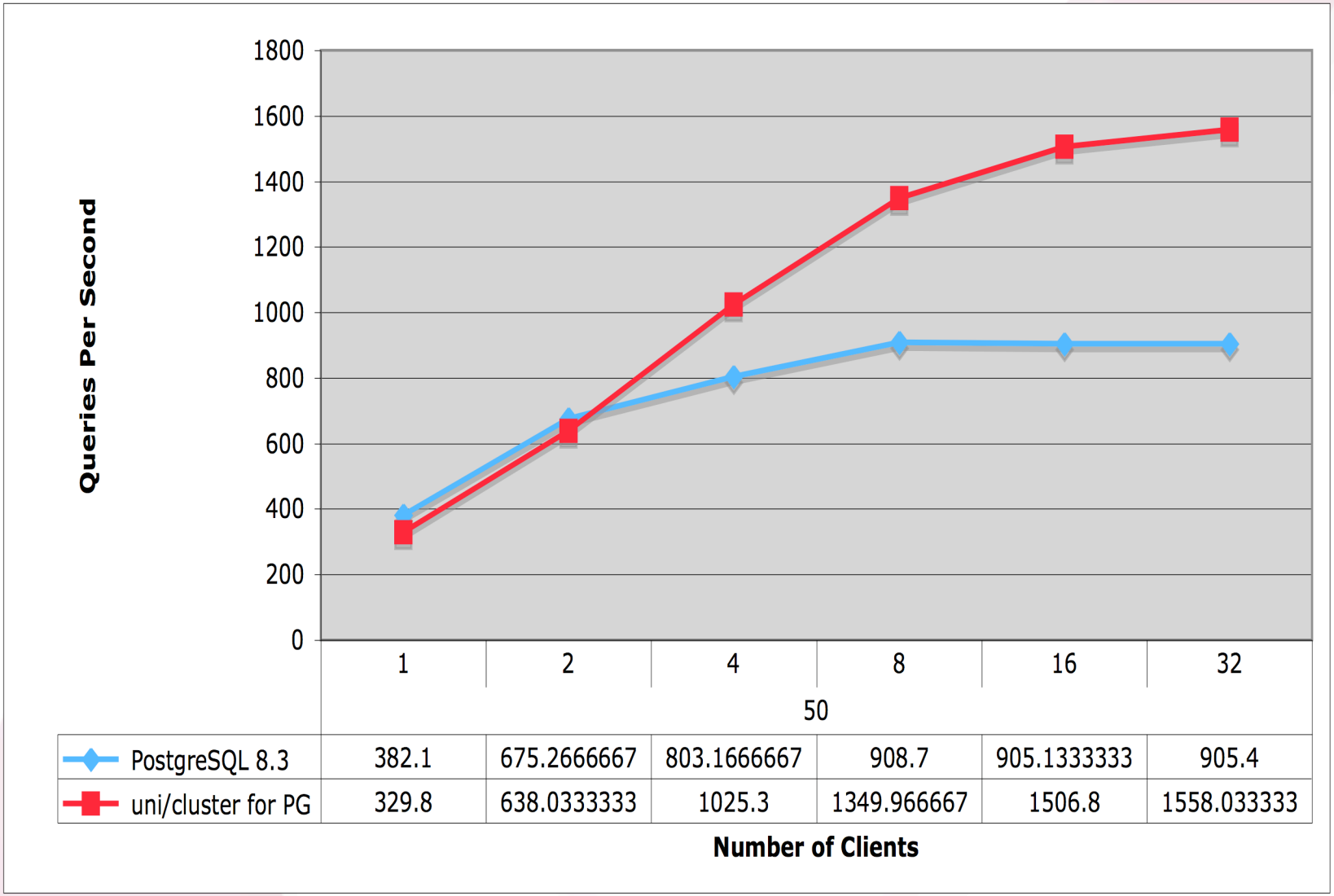
```
select count(*) from benchmark_scenario_0 t0
  join benchmark_scenario_1 t1 on t0.mykey = t1.mykey
    where t0.mykey between ? and ?
```

  - Select 200 rows
  - See how PostgreSQL does as well!
- **Test Resources: Dell SC 1425, Xeon 2 CPU x 2 Cores, 2.8Mhz, 1GB Memory**

MySQL 5.1 Read Scaling (200 Rows)

	1	2	4	8	16	32
MySQL 5.1.23	1069.53	1475.83	2057.70	2032.17	1980.13	1962.40
uni/cluster	597.00	1200.73	1878.83	2176.77	2472.37	2515.97

PostgreSQL Read Scaling (50 Rows)



	1	2	4	8	16	32
PostgreSQL 8.3	382.1	675.2666667	803.1666667	908.7	905.1333333	905.4
uni/cluster for PG	329.8	638.0333333	1025.3	1349.966667	1506.8	1558.033333

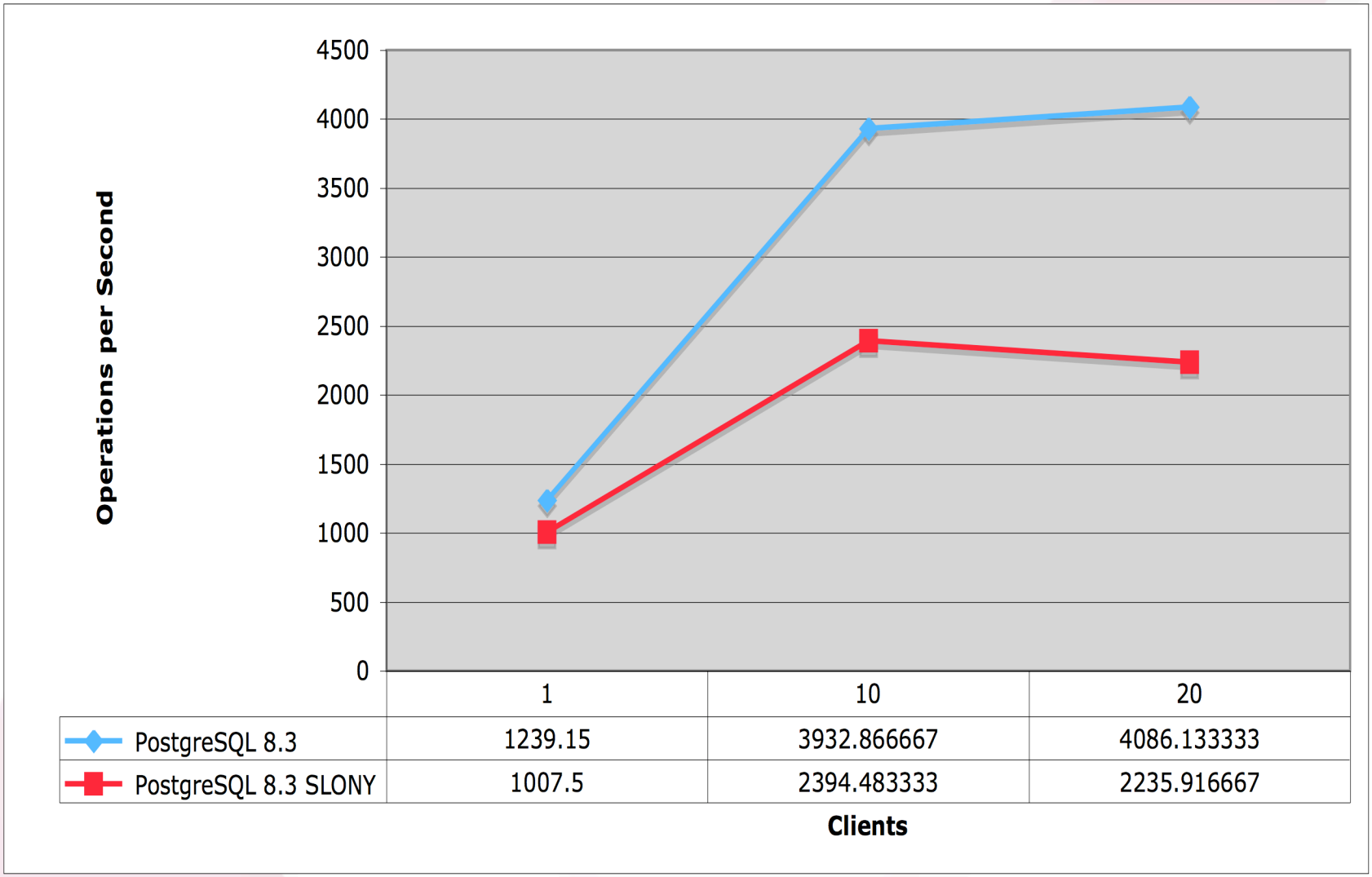
Measuring Replication Latency

- **Question: How does replication affect master performance and what is the latency for a slave?**
- **Approach: Use INSERT commands with varying numbers of clients on replicated and non-replicated databases**
- **Test: WriteSimpleScenario**
  - Use 'replicaUrl' property to measure latency--check how long it takes for the row to show up in the database
  - For non-replicated databases replicateUrl points to same database
  - For replicated databases we point to the slave
- **Test Resources: Dell SC 1425, Xeon 2 CPU x 2 Cores, 2.8Mhz, 1GB Memory**
- **Compare PostgreSQL 8.3 vs. MySQL (5.0.22)**

Replication: MySQL Master Overhead

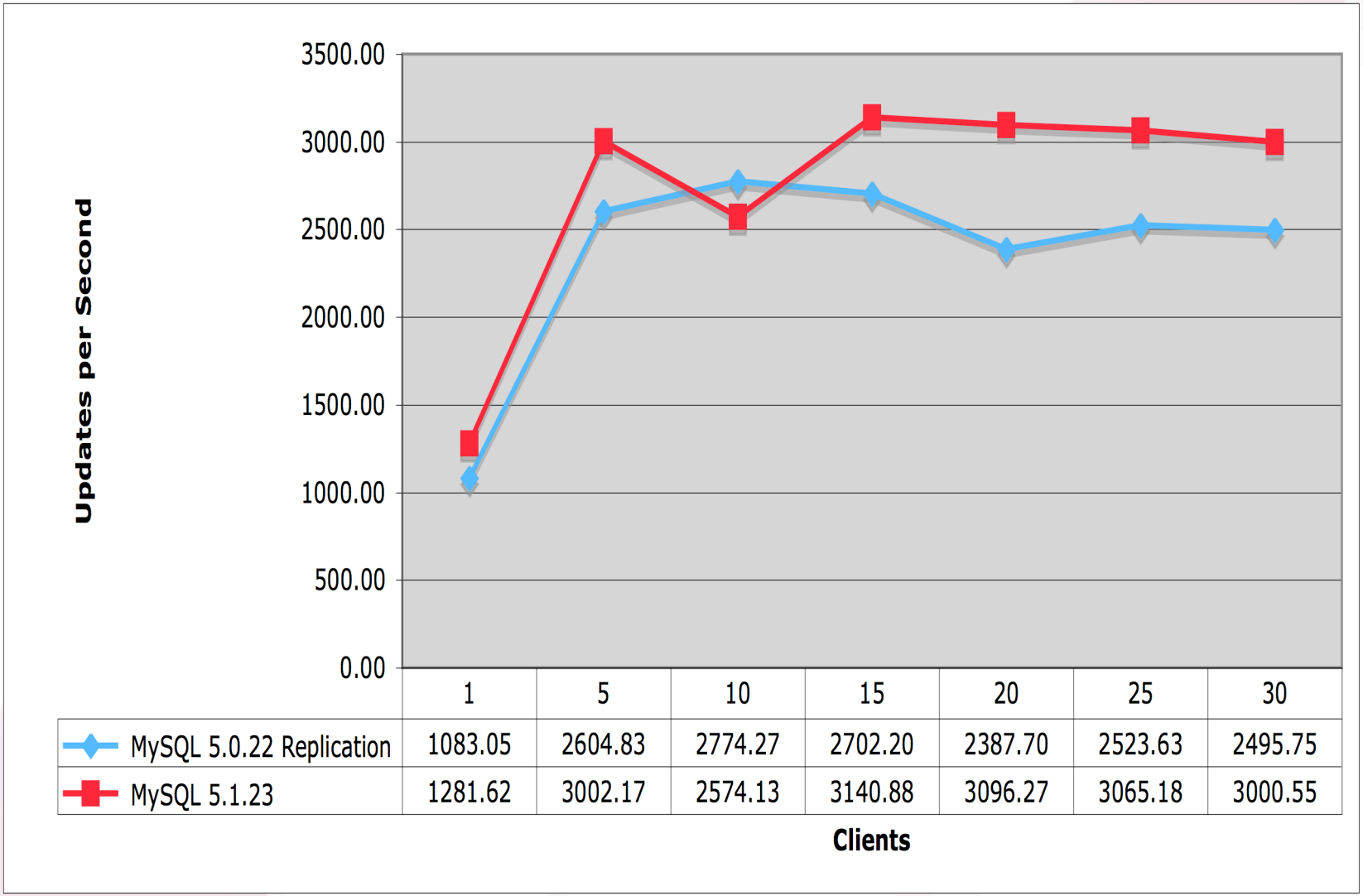
	1	5	10	15	20	25	30
MySQL 5.1.23	2571.13	7012.18	6578.10	6032.47	5892.95	5829.02	5830.80
MySQL 5.1.23 Replication	2241.33	4919.70	4636.07	4473.60	4406.55	4328.20	4352.83

Replication: PostgreSQL Master Overhead



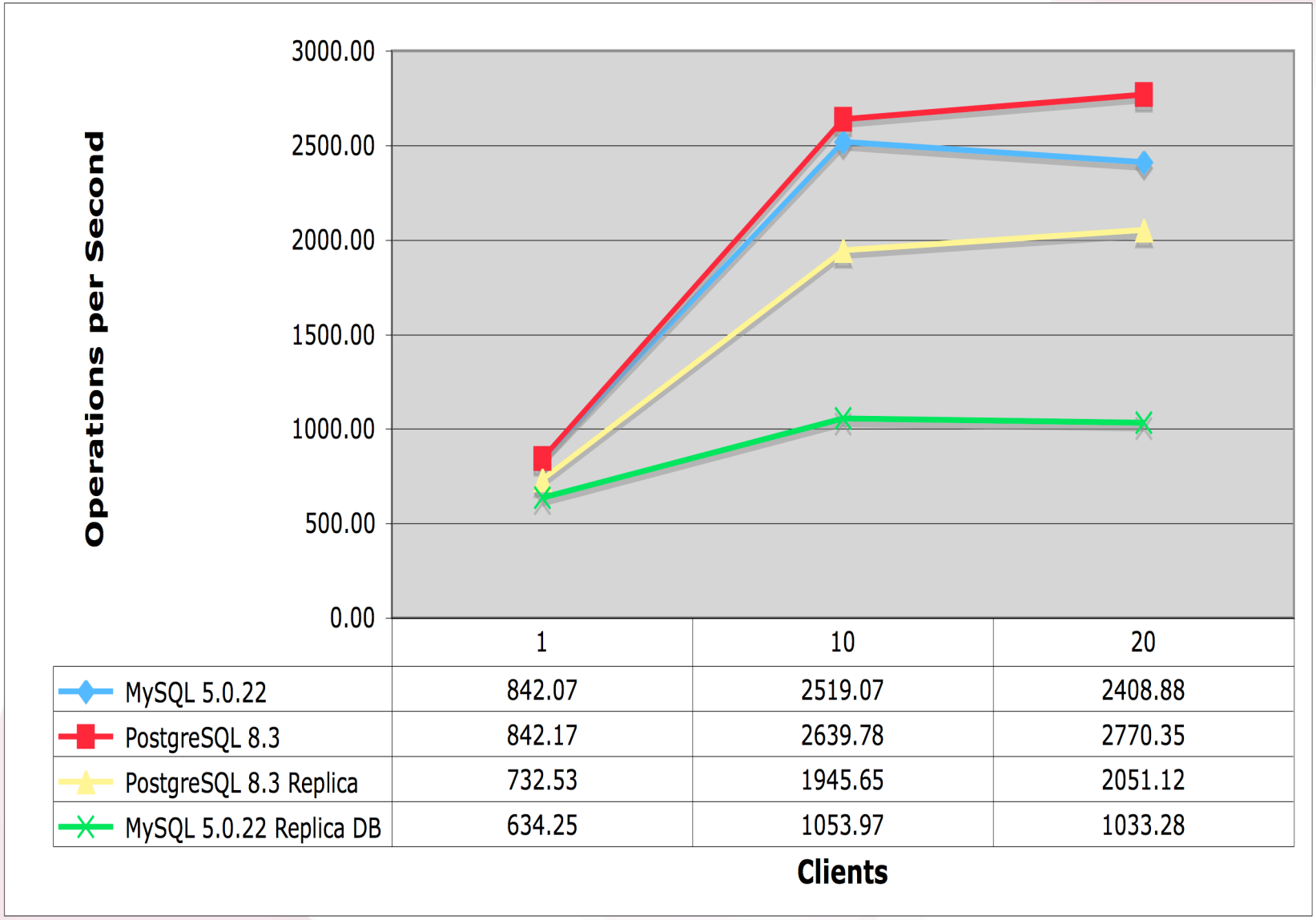
	1	10	20
PostgreSQL 8.3	1239.15	3932.866667	4086.133333
PostgreSQL 8.3 SLONY	1007.5	2394.483333	2235.916667

Replication Latency: MySQL 5.1 Slave Lag



	1	5	10	15	20	25	30
MySQL 5.0.22 Replication	1083.05	2604.83	2774.27	2702.20	2387.70	2523.63	2495.75
MySQL 5.1.23	1281.62	3002.17	2574.13	3140.88	3096.27	3065.18	3000.55

Replication Latency: MySQL vs. PG



	1	10	20
MySQL 5.0.22	842.07	2519.07	2408.88
PostgreSQL 8.3	842.17	2639.78	2770.35
PostgreSQL 8.3 Replica	732.53	1945.65	2051.12
MySQL 5.0.22 Replica DB	634.25	1053.97	1033.28

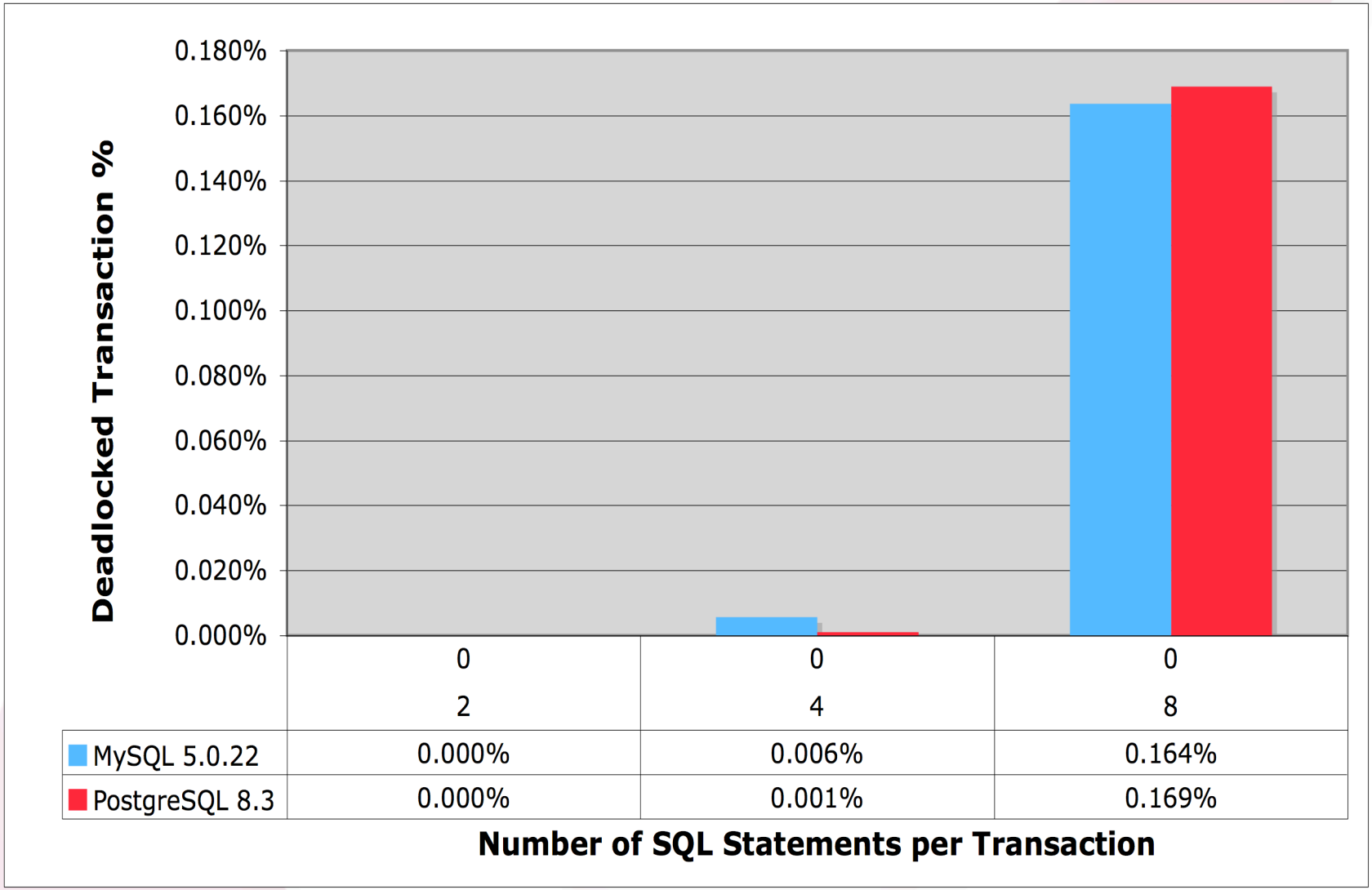
Measuring Deadlocks

- **Question: How do deadlocks increase as transaction size increases?**
- **Approach: Run transactions with random updates in random order, varying the number of updates per transaction**
- **Test: DeadlockScenario**
  - Performs row updates using random keys:

```
update benchmark_scenario_0
   set myvalue = ?, mypayload = ?
where mykey = ?
```

  - Compare against MySQL for fun!
- **Test Resources: Dell SC 1425, Xeon 2 CPU x 2 Cores, 2.8Mhz, 1GB Memory**
- **PostgreSQL 8.3 vs. MySQL 5.0.22**

Deadlock Handling - PostgreSQL vs MySQL



	0 2	0 4	0 8
MySQL 5.0.22	0.000%	0.006%	0.164%
PostgreSQL 8.3	0.000%	0.001%	0.169%

Final Thoughts

Summary

- Scale-out solutions have a number of performance trade-offs that require careful testing using mixed load as well as micro-benchmarks
- Continuent is developing a stack for scale-out that covers multiple design approaches
- Bristlecone test tools provide portable tests that check interesting cases for performance
- You can get into the action by downloading Bristlecone today
- Visit at the following URLs:

http://www.continuent.org

http://www.continuent.com

Questions?

Robert Hodges

CTO

Continuent, Inc.

robert.hodges@continuent.com

