

Java™ DB Performance

Olav Sandstå

Senior Staff Engineer
Sun Microsystems
http://developers.sun.com/javadb/

TS-45170

Goal of My Talk

Learn how to configure and use **Java™ DB** to get the best **performance** and the required **durability** for your data.

Agenda

Java DB Introduction

Configuring Java DB for Performance

Programming Tips

Understanding Java DB Performance

Open Source Database Performance

Java DB

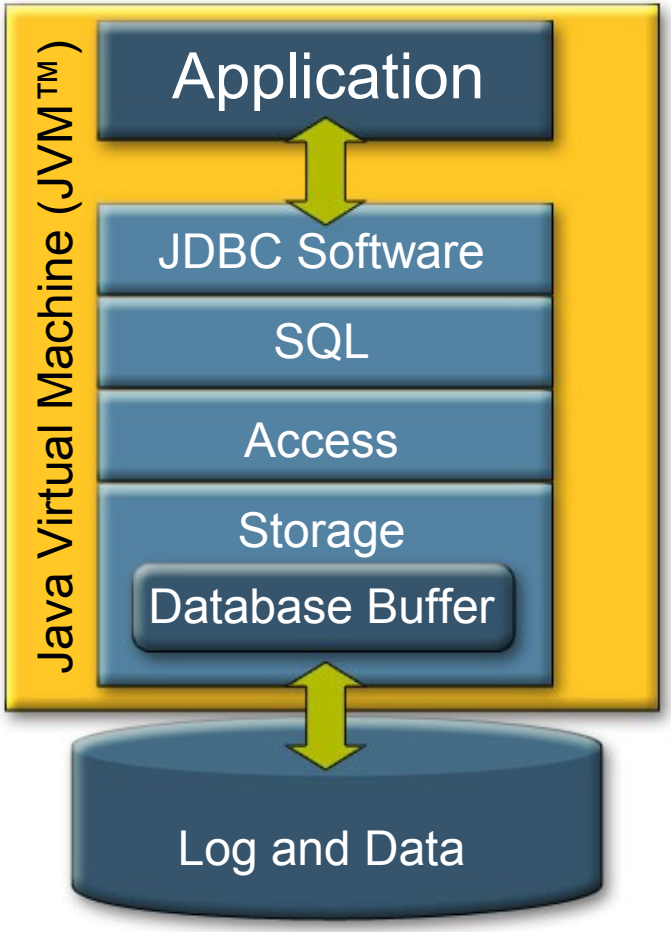


- Sun's supported distribution of Apache Derby
  - All development done in the Apache Derby community
- Complete relational database engine
- 100% Java technology
- Bundled in Sun Java Development Kit (JDK™) 6 and GlassFish™ Project
- Supported by NetBeans™ Software, Sun Java Studio Enterprise, Eclipse
- The database for Java applications Java™ DB

Java DB Features

- Complete SQL engine including:
  - Views, triggers, stored procedures, foreign keys
- Multi-user transaction support:
  - All major isolation levels
  - ACID properties
- Security:
  - Data encryption, client authentication, GRANT/REVOKE
- Standard based:
  - Java DataBase Connectivity (JDBC™) 4.0 and SQL92/99/2003/XML

Java DB Architecture: Embedded



- Include derby.jar in your classpath
- Boot the Java DB engine[1)]

```
Class.forName(
  "org.apache.derby.jdbc.
  EmbeddedDriver");
```

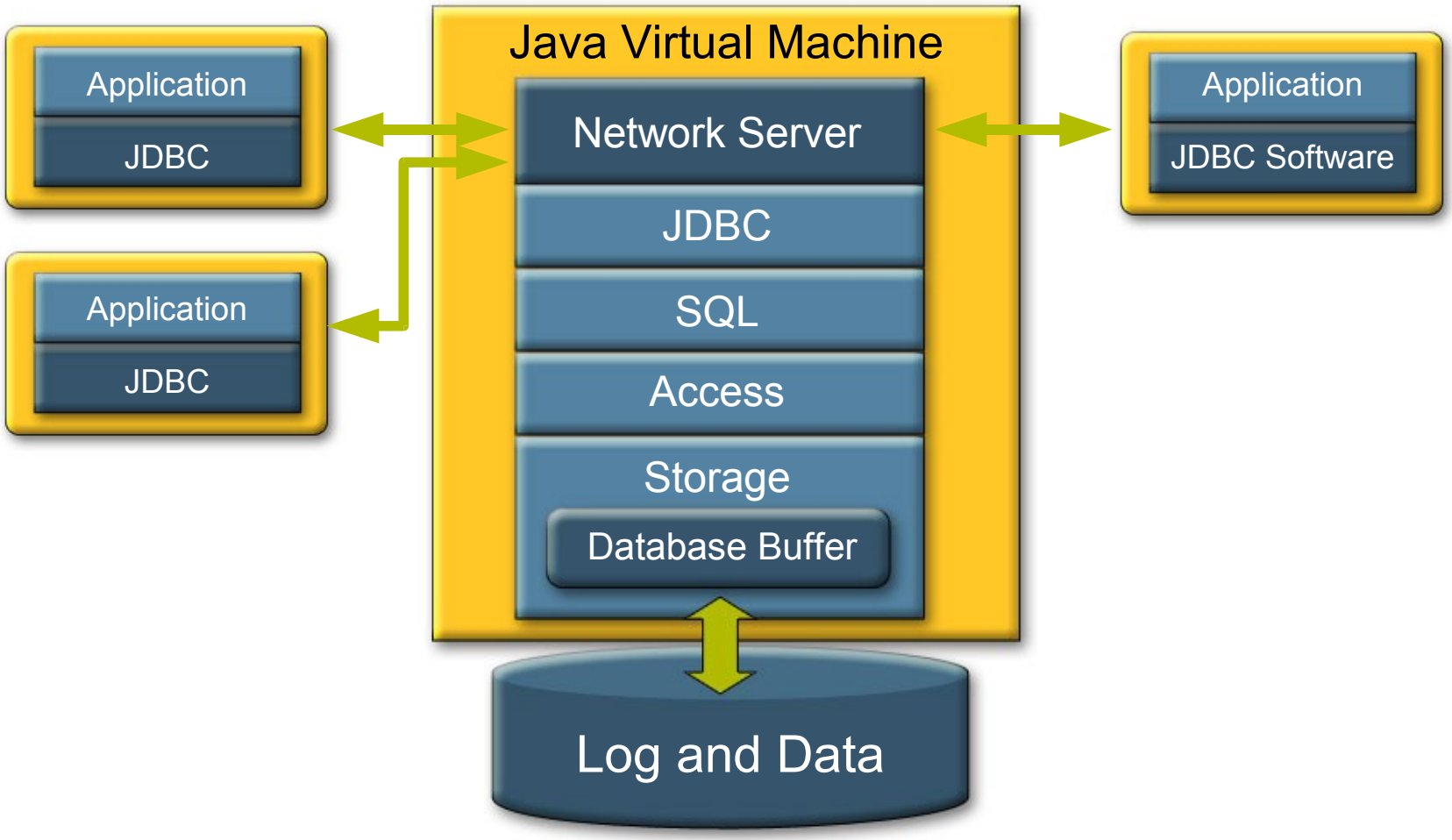
- Create a new database

```
Connection conn =
DriverManager.getConnection(
  "jdbc:derby:dbName; " +
  "create=true");
```

1) Optional when running with JDK version 6

The terms "Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java™ platform.

Java DB Architecture: Client-Server



Agenda

Java DB Introduction

Configuring Java DB for Performance

Programming Tips

Understanding Java DB Performance

Open Source Database Performance

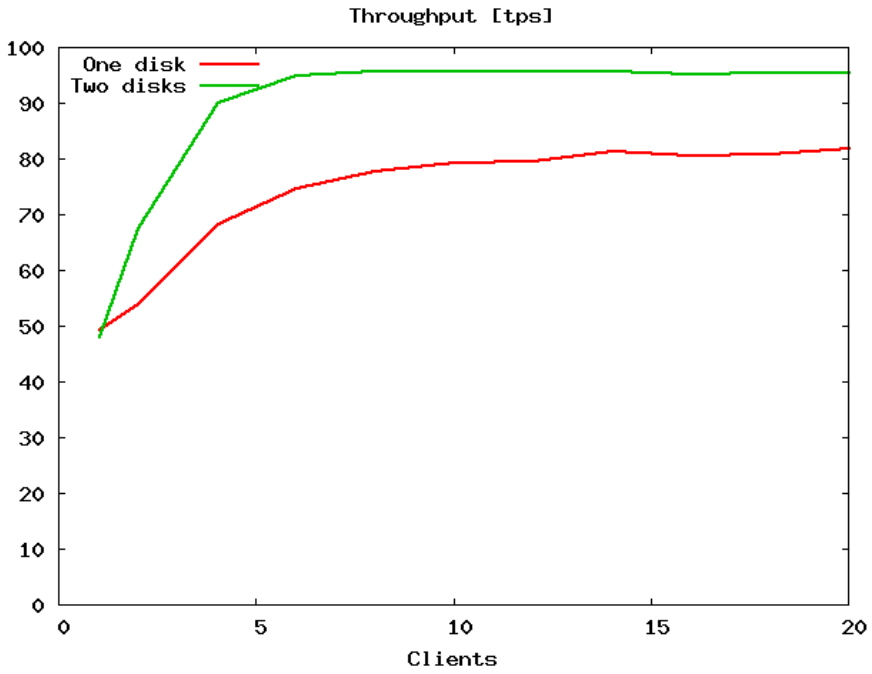
Performance Tip 1:
Separate Data and Log Devices

Log on separate disk:

- Utilize sequential write bandwidth on disk
- Configuration:

  JDBC driver connection url:

  `logDevice=<path>`



Performance tip:

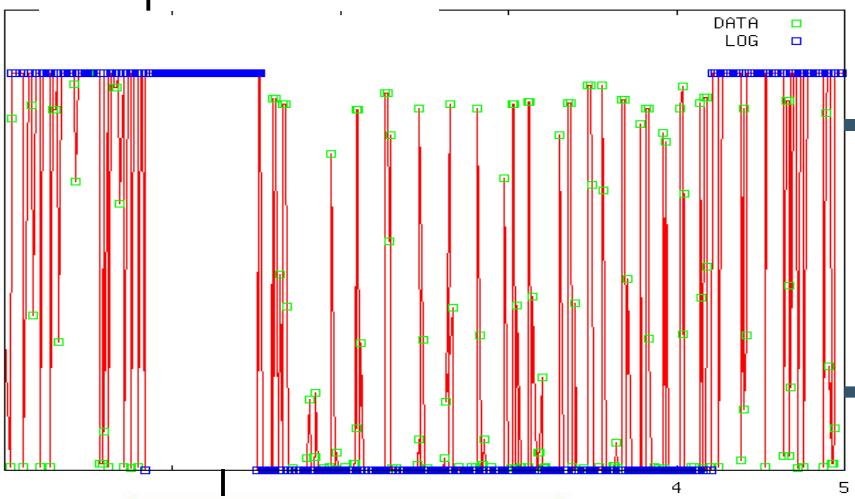
Use separate disks for data and log device

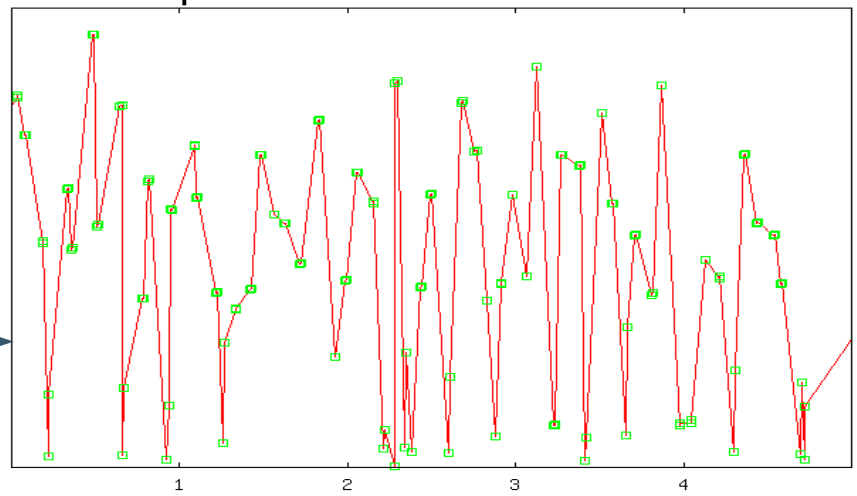
JDBC Driver = a driver supporting the JDBC™ API

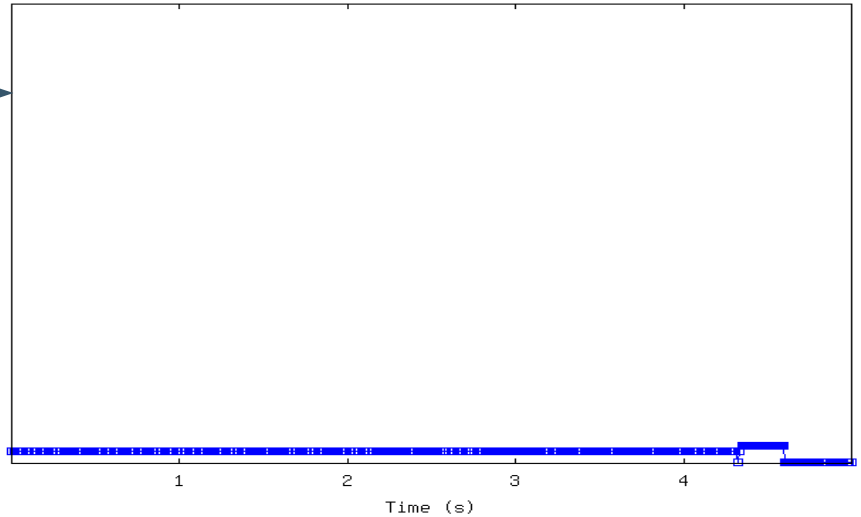
Disk Activity

Data and log on separate disks

Data and log on same disk



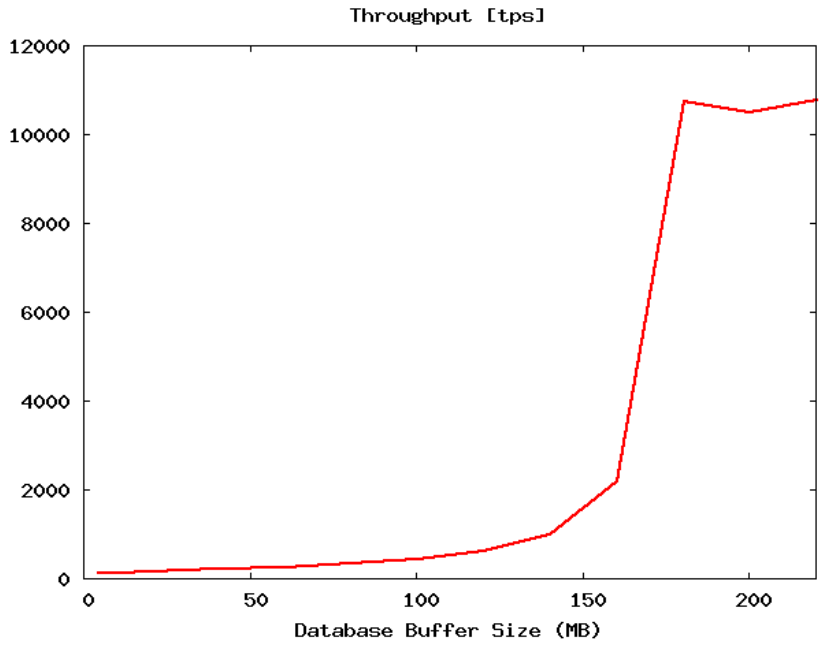




Disk head movement for 5 seconds of database activity

Performance Tip 2:
Tune Database Buffer Size

- Cache of frequently used data pages in memory
- Cache-miss leads to read from disk
- Size:
  - default 4 MB
  - `derby.storage.pageCacheSize`



Performance tip:

Increase the size of the database buffer to get frequently accessed data in memory

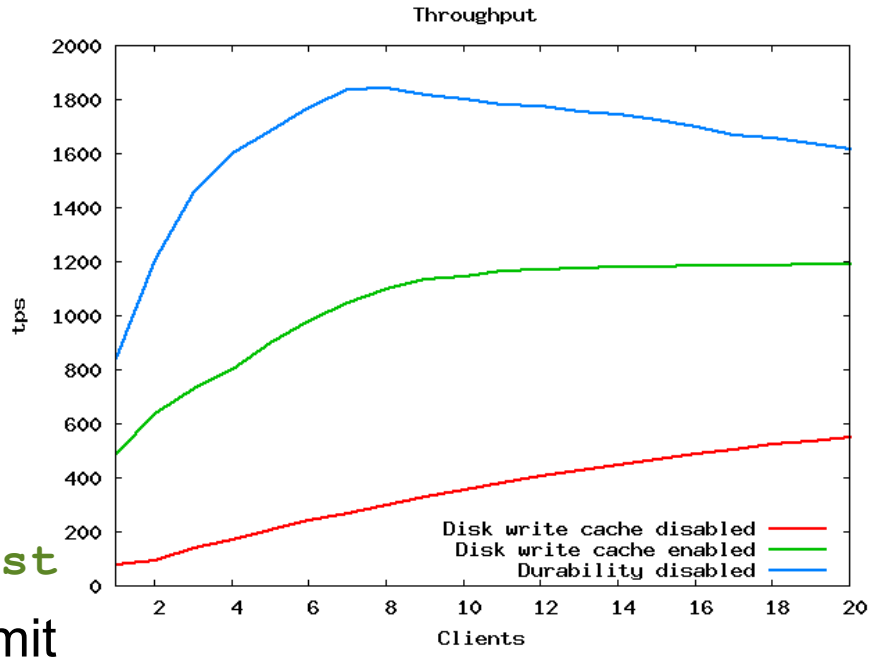
Performance Tip 3:
Trade Durability for Performance

Log device configuration:

- Disk's write cache:
  - Disabled
  - Enabled
- Disable durability:

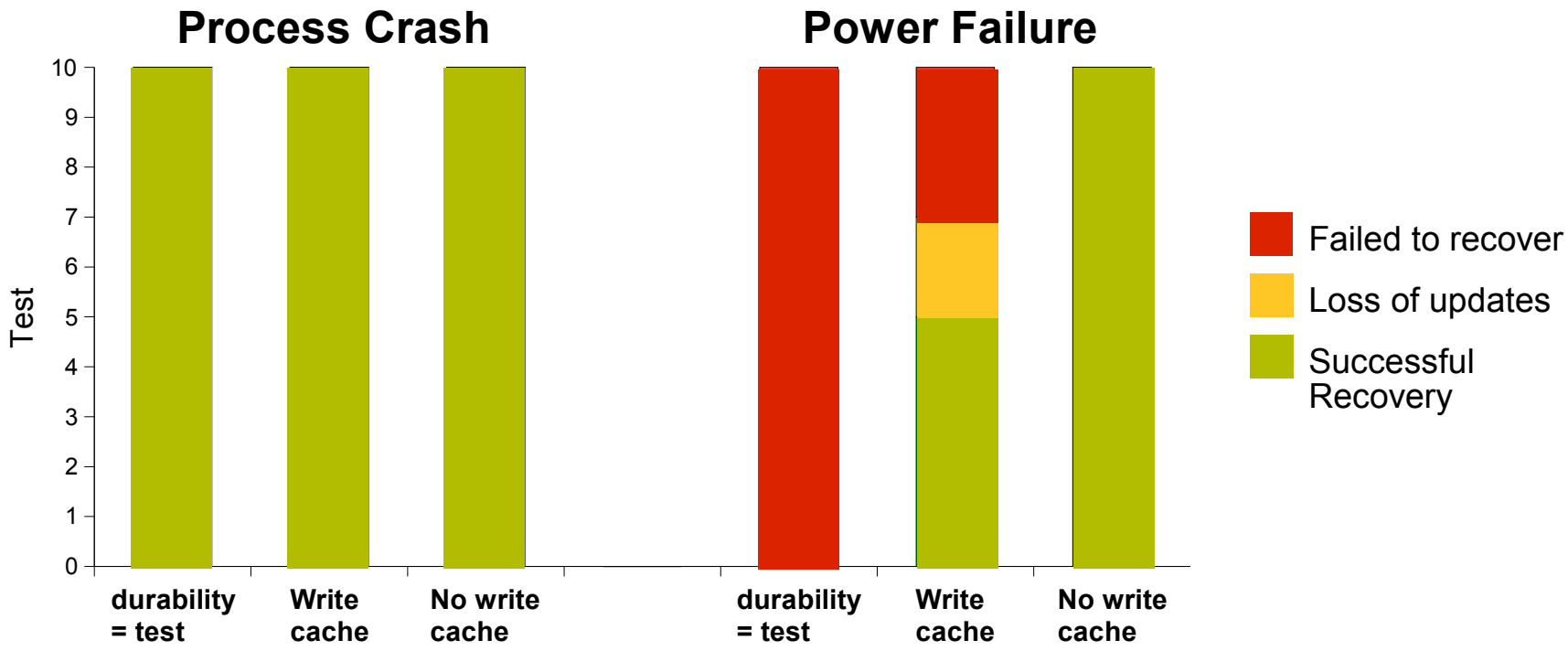
```
derby.system.durability=test
```

  - log flushed to disk **after** commit



WARNING: Write cache reduces probability of successful recovery after power failure

Log Device Configuration:
Effect on Durability



Durability tip:
Disable the disk's write cache on the log device

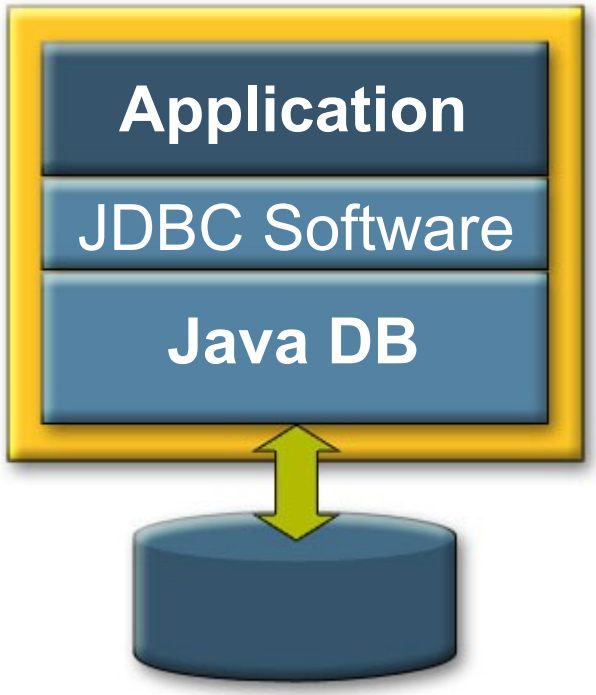
Performance Tip 4:
Use Embedded Java DB

Performance advantages:

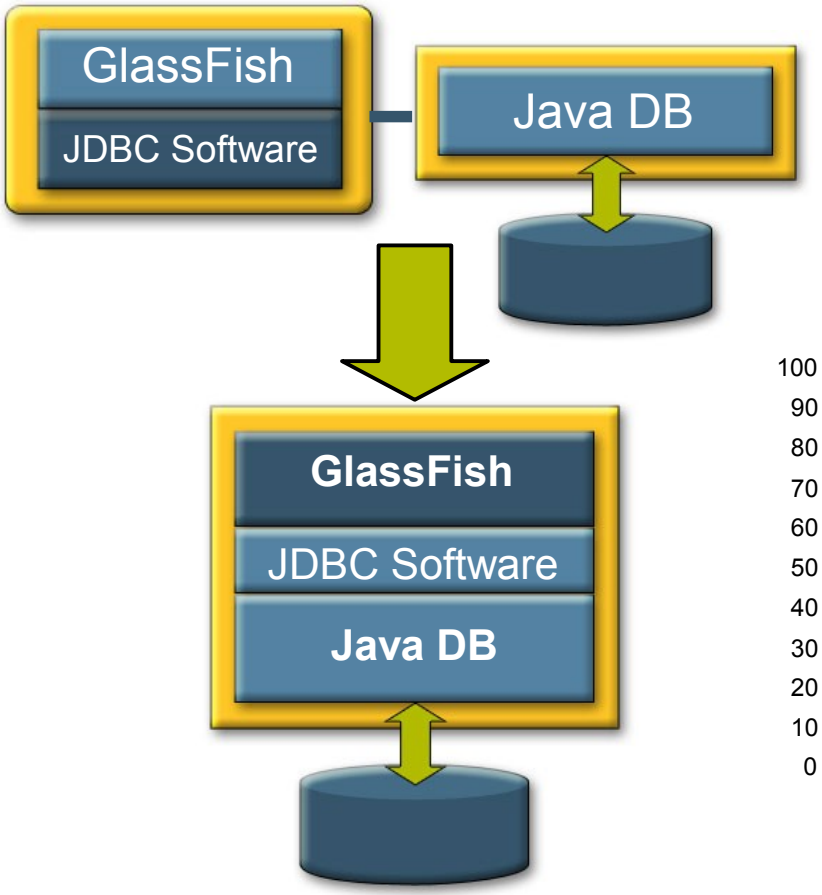
- Saves inter-process or server communication
- Reduces CPU usage
- Reduces hardware cost

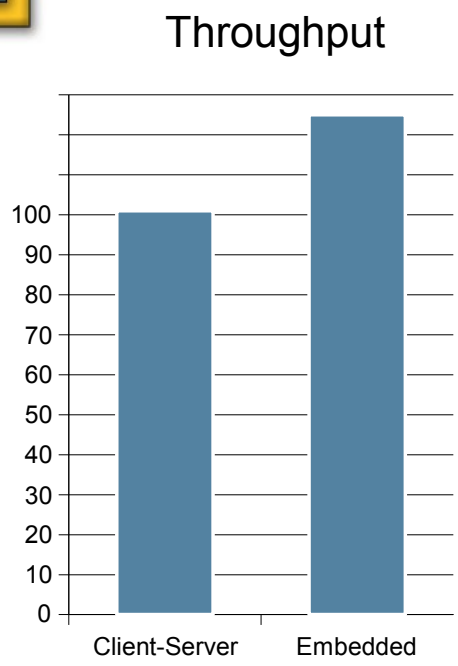
Potential issues:

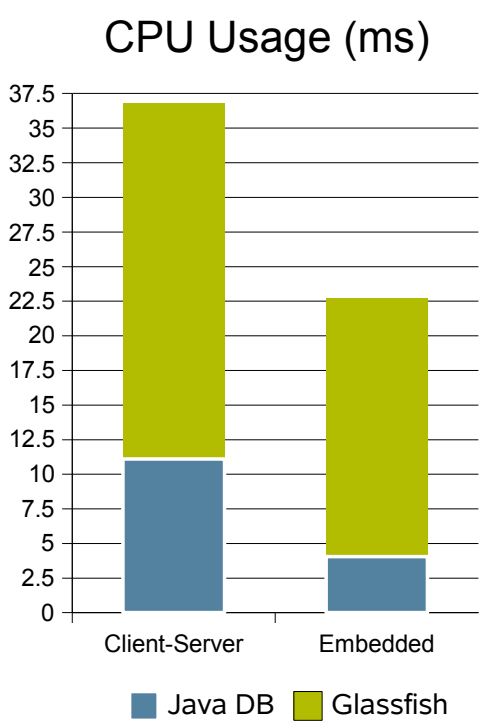
- Scalability (one machine)
- JVM software configuration



GlassFish Project and Java DB:
Client-Server vs. Embedded: Example







Agenda

Java DB Introduction

Configuring Java DB for Performance

Programming Tips

Understanding Java DB Performance

Open Source Database Performance

Performance Tip 5:
Use Prepared Statements

- Compilation of SQL statements is expensive:

```
Statement s = c.createStatement();
while (...) {
    s.executeQuery("SELECT * FROM t WHERE a = " + id);
}
```

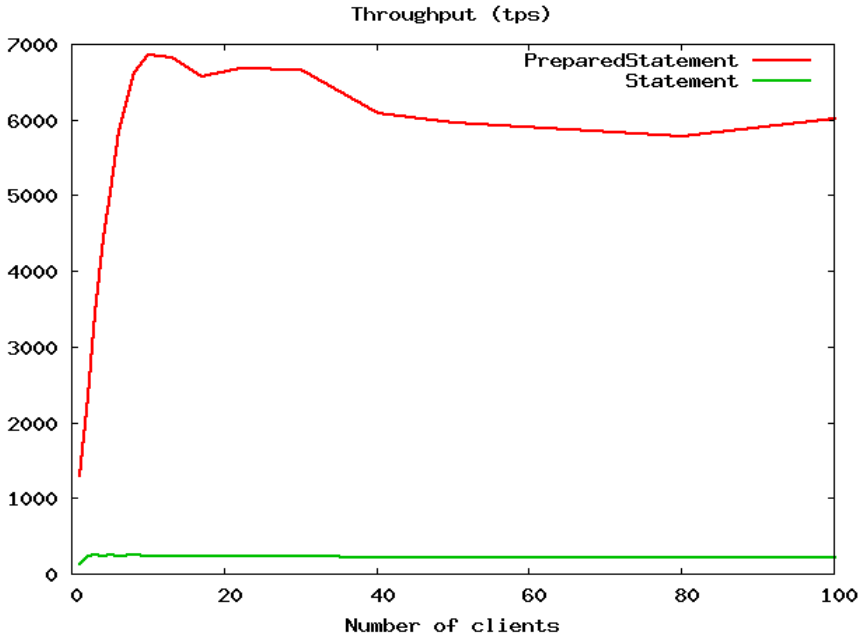
  - generates Java bytecode and loads generated classes
- Prepared statements eliminate this cost:

```
PreparedStatement s =
    c.prepareStatement("SELECT * FROM t WHERE a = ?");
while (...) {
    s.setInt(1, id);
    s.executeQuery();
}
```

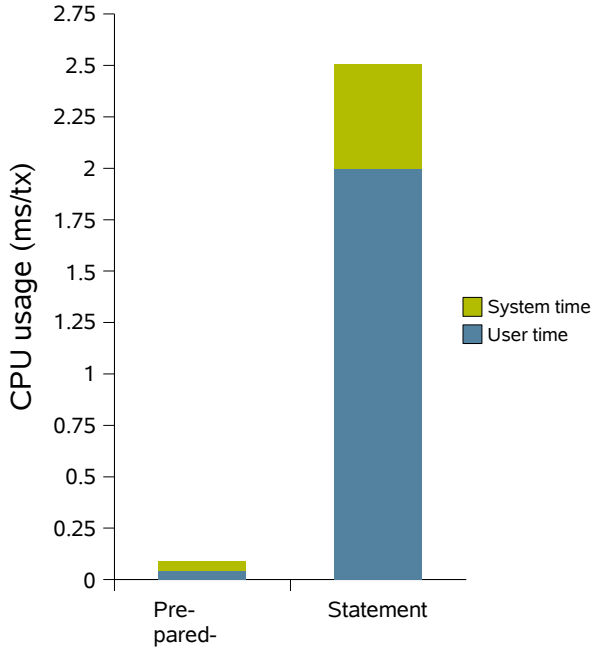
  - generated Java bytecode can be JIT compiled

Use Prepared Statements: Example

CPU Usage



Throughput



Performance tip:
USE prepared statements—and **REUSE** them

Performance Tip 6:
Help the Database to Perform

- Use indexes to optimize frequently used access paths:

```
CREATE INDEX indexName ON tableName (column)
```

  - Table scan: reads the entire table
  - Index: finds the data by reading a few blocks
- Close JDBC software objects after in use
  - Connections, Statements, ResultSets, Streams
- Use transactions—do not rely on auto-commit
  - Particularly for insert/update/delete operations

Agenda

Java DB Introduction

Configuring Java DB for Performance

Programming Tips

Understanding Java DB Performance

Open Source Database Performance

Performance Tip 7:
Know the Load

- Know the load on the database:
  - `derby.language.logStatementText=true`
  - All executed queries written to derby.log
- Know how the queries are executed:
  - `derby.language.logQueryPlan=true`
- Use OS and Java tools to find resource usage:
  - CPU, memory, disk IO for log and data device

Performance tip:

Use the available tools to understand what the database is doing and where resources are spent

Performance Tip 8:
Query Plan and Run-time Statistics

- Enable/disable tracing of query plan:
  - `SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS(1)`
  - `SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS(0)`
- Enable/disable timing information in query plan:
  - `SYSCS_UTIL.SYSCS_SET_STATISTICS_TIMING(1)`
  - `SYSCS_UTIL.SYSCS_SET_STATISTICS_TIMING(0)`
- Retrieve query plan for individual queries:
  - `SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS()`

Example: Query Plan and Run-time Statistics

```
// enable run-time statistics
Statement s = c.createStatement();
s.executeUpdate("CALL SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS(1)");
s.executeUpdate("CALL SYSCS_UTIL.SYSCS_SET_STATISTICS_TIMING(1)");

// execute query
ResultSet rs = s.executeQuery("SELECT T1.c2 from T1, T2
                               where T1.c2 = T2.c2 and T1.c2 < 800");
while (rs.next()) {}
rs.close();

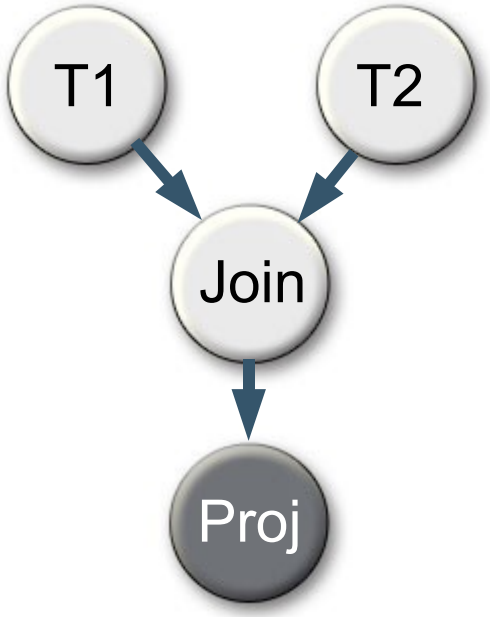
// retrieve query plan and run-time statistics
rs = s.executeQuery("VALUES
                     SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS()");
rs.next();
String str = rs.getString(1);
System.out.println("Query Plan: " + str);
```

Understanding the Query Plan (1)

```
CodeStatement Text:
        SELECT T1.c2 from T1, T2
        where T1.c2 = T2.c2 and T1.c2 < 800
Parse Time: 1
Bind Time: 5
Optimize Time: 16
Generate Time: 3
Compile Time: 25
Execute Time: 52

Project-Restrict ResultSet (4):
Number of opens = 1
Rows seen = 800
Rows filtered = 0

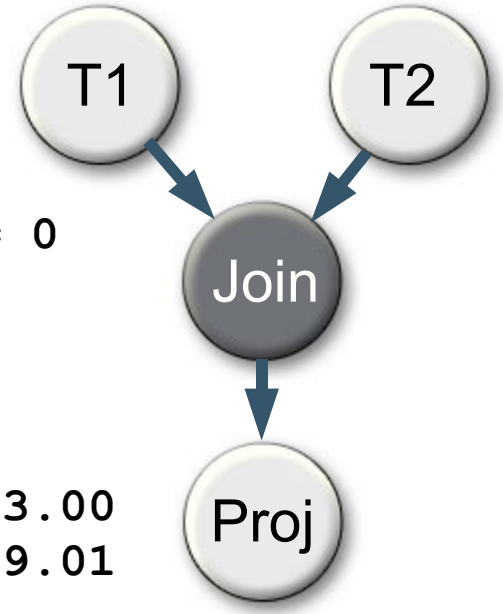
optimizer estimated row count:      753.00
optimizer estimated cost:           349.01
```



Understanding the Query Plan (2)

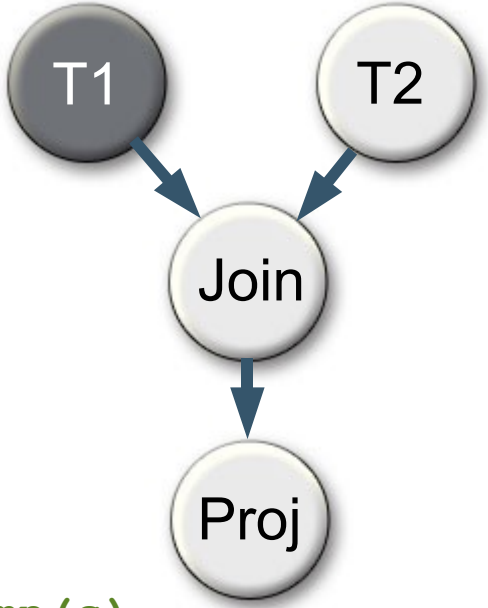
```
Source result set:
    Hash Exists Join ResultSet:
    Number of opens = 1
    Rows seen from the left = 800
    Rows seen from the right = 800
    Rows filtered = 0
    Rows returned = 800
        constructor time (milliseconds) = 0
        open time (milliseconds) = 31
        next time (milliseconds) = 21
        close time (milliseconds) = 0

    optimizer estimated row count:      753.00
    optimizer estimated cost:           349.01
```



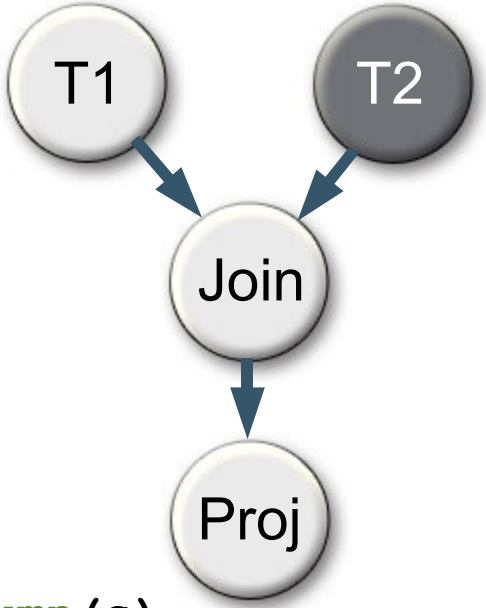
Understanding the Query Plan (3)

```
Right result set:
    Hash Scan ResultSet for T1 using index T1_i2 at read
committed isolation level using instantaneous share row
locking:
    Number of opens = 800
    Hash table size = 800
    Rows seen = 800
    scan information:
        Number of columns fetched=1
        Number of pages visited=6
        Number of rows qualified=800
        Number of rows visited=801
        Scan type=btree
        start position: None
        stop position: >= on first 1 column(s)
    optimizer estimated row count:      753.00
    optimizer estimated cost:           186.27
```



Understanding the Query Plan (4)

```
Left result set:
    Index Scan ResultSet for T2 using index T2_i2 at read
committed isolation level using instantaneous share row
locking chosen by the optimizer
    Number of opens = 1
    Rows seen = 800
    Rows filtered = 0
    scan information:
        Number of columns fetched=1
        Number of pages visited=6
        Number of rows qualified=800
        Number of rows visited=801
        Scan type=btree
        start position: None
        stop position:  >= on first 1 column(s)
    optimizer estimated row count:      753.00
    optimizer estimated cost:           162.74
```



Performance Tip 9:
Optimizer Overrides

- Override execution strategy selected by optimizer
- Force use of specific index:

```
SELECT * FROM t1 --DERBY-PROPERTIES index=t1_c1
WHERE c1=1
```

- Force use of constraint:

```
SELECT * FROM t1 --DERBY-PROPERTIES constraint=c
WHERE c1=1 and c2=3
```

- Force specific JOIN order and JOIN strategy:

```
SELECT * FROM --DERBY-PROPERTIES joinOrder=FIXED
t1,t2 --DERBY-PROPERTIES joinStrategy=NESTEDLOOP
WHERE t1.c1=t2.c1
```

  - Join strategies: HASH and NESTEDLOOP

Optimizer Overrides Example

- ```
  SELECT t1.c2 FROM --DERBY-PROPERTIES joinOrder=FIXED
  t1, t2 --DERBY-PROPERTIES joinStrategy=NESTEDLOOP
  WHERE t1.c2 = t2.c2
  ```

Optimizer overide	Estimated cost	CPU (ms)
None	2311	12.0
joinOrder=FIXED	2505	12.0
joinStrategy=NESTEDLOOP	3404	14.7
FIXED and NESTEDLOOP	3404	14.4

Performance tip:

Use optimizer overrides—**but only** when needed
  ```

Performance Tip 10:
Understand Locking Issues

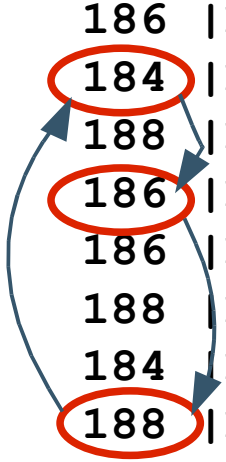
- Lock-based concurrency control
- Isolation level:
  - Reducing isolation level increases concurrency
- Lock escalation:
  - Default: escalation from **row locks** to **table locks** when 5000 locks are set on the table
  - `derby.locks.escalationThreshold=100`
  - `LOCK TABLE t1 IN {SHARE|EXCLUSIVE} MODE`
- Deadlock tracing:
  - `derby.locks.monitor=true`
  - `derby.locks.deadlockTrace=true`

Understand Locking Issues

Retrieve lock information:

```
SELECT * FROM SYSCS_DIAG.LOCK_TABLE
```

XID	TYPE	MODE	TABLENAME	LOCKNAME	STATE	INDEXNAME
186	ROW	X	T2	(1,9)	GRANT	
184	ROW	S	T2	(1,9)	WAIT	
188	ROW	X	T1	(1,11)	GRANT	
186	ROW	S	T1	(1,11)	WAIT	
186	ROW	S	T1	(1,1)	GRANT	SQL070425023
188	ROW	S	T1	(1,1)	GRANT	SQL070425023
184	ROW	X	T1	(1,7)	GRANT	
188	ROW	S	T1	(1,7)	WAIT	
186	TABLE	IX	T2	Tablelock	GRANT	
184	TABLE	IS	T2	Tablelock	GRANT	



Agenda

Java DB Introduction

Configuring Java DB for Performance

Programming Tips

Understanding Java DB Performance

Open Source Database Performance

Performance Improvements

Java DB 10.3

- Embedded:
  - Reduced synchronization and context switches
  - Reduced CPU usage
  - Reduced number of disk updates to log device
  - Concurrent read/writes on data device
- Client-server:
  - Improved streaming of LOBs
- SQL Optimizer:
  - Improved optimization

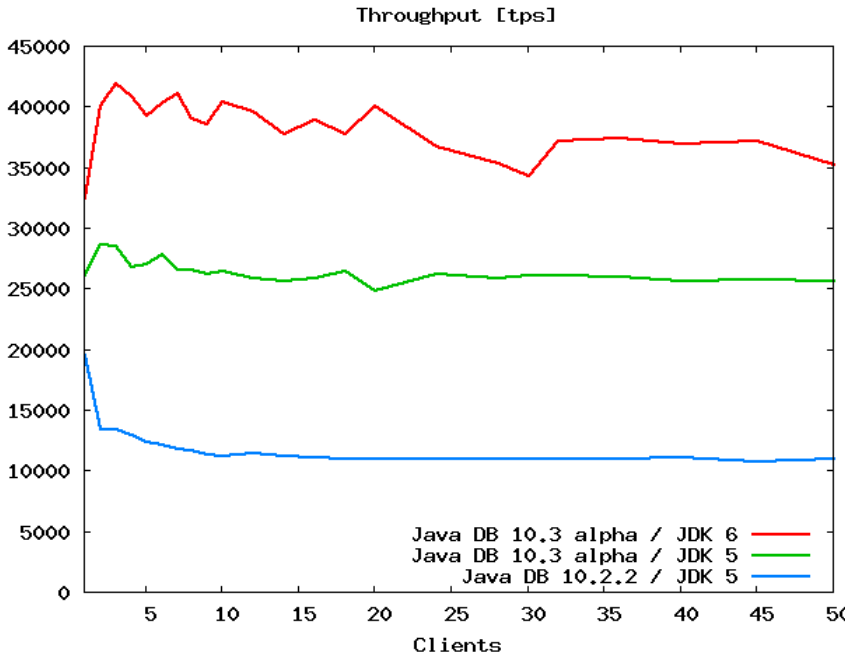


30–150% increased throughput on simple queries

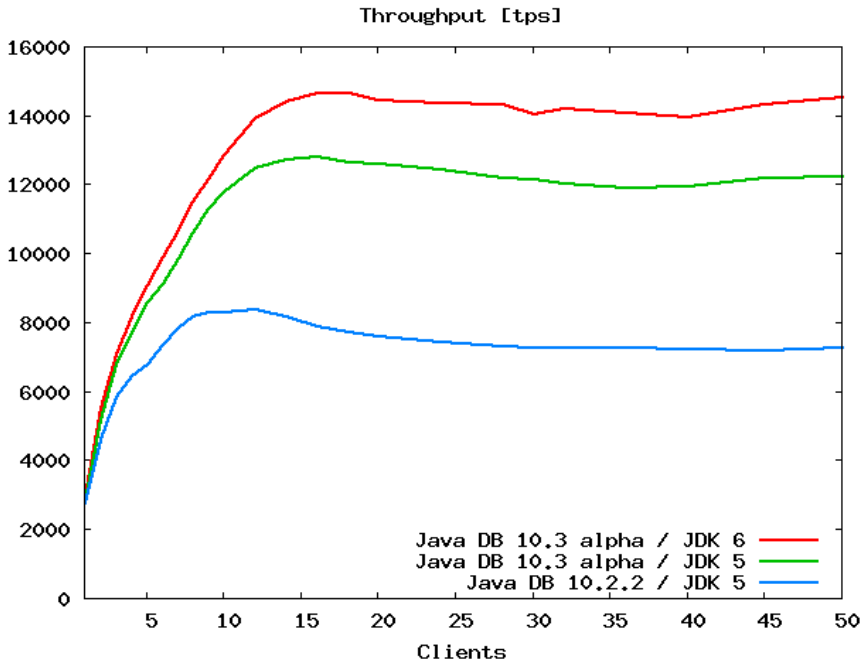
Performance Improvement: Example

Upgrading to Sun JDK version 6 and Java DB 10.3 alpha

Java DB embedded:



Java DB client-server:



Load: Select one record in a table

Comparing Performance

Open-Source Databases

Databases:

- Java DB 10.3 alpha
  - Embedded
  - Client-server
- PostgreSQL 8.1.8
- MySQL 5.0.33
  - With InnoDB







Load clients:

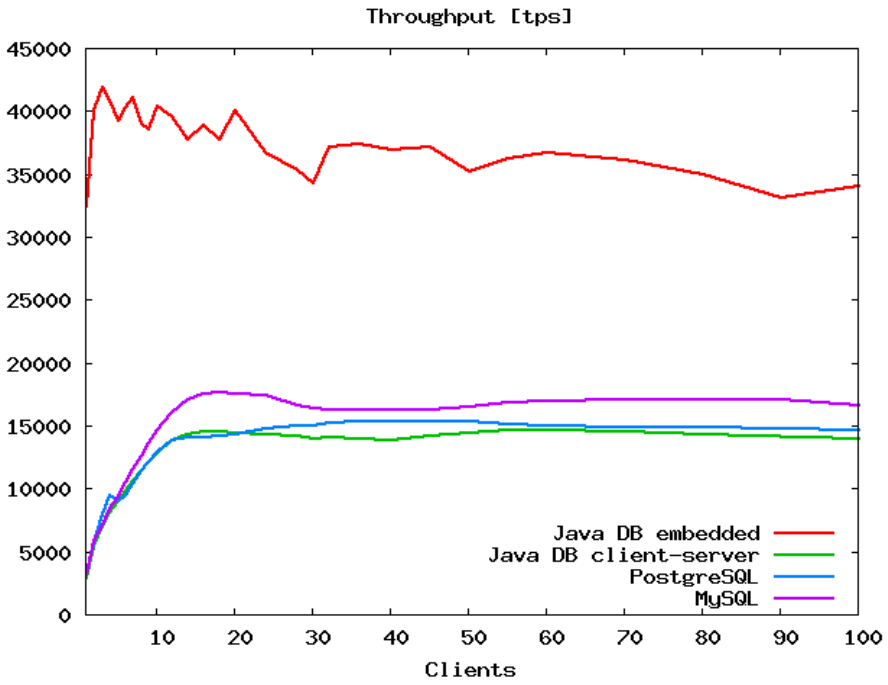
1. Select load:
   1 single-record select
2. Update load:
   3 updates, 1 insert, 1 select

Test Configuration:

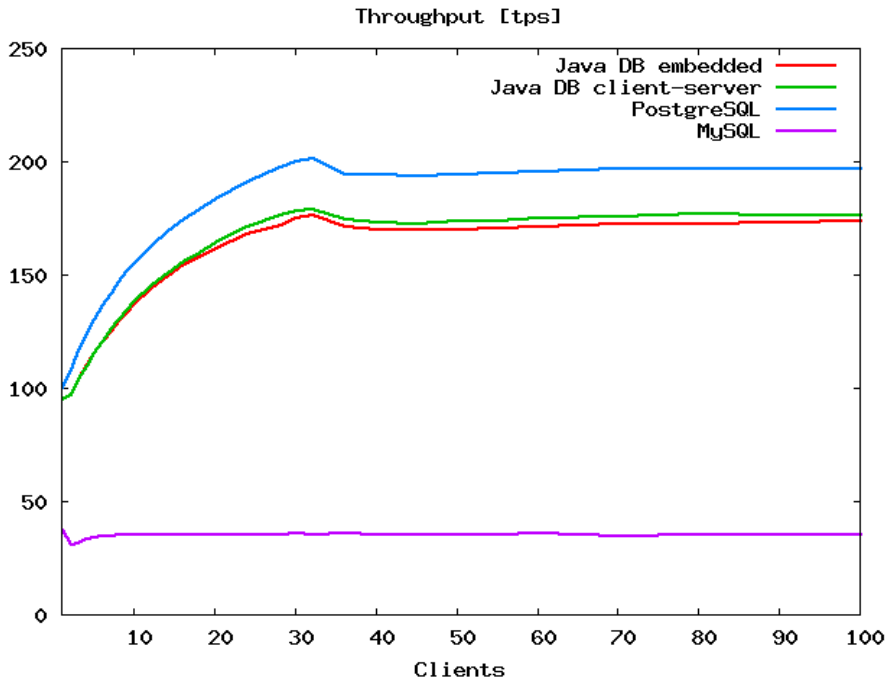
- "Out of the box"
- 50 MB database buffer
- Log and data on separate disks

Throughput: Single-record Select

Main-memory database (10 MB):

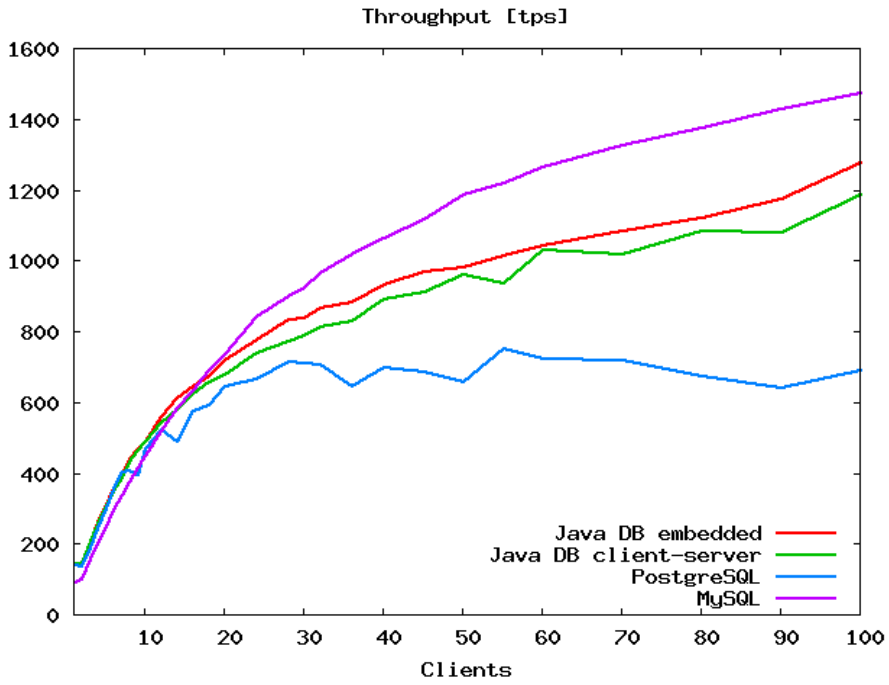


Disk-based database (10 GB):

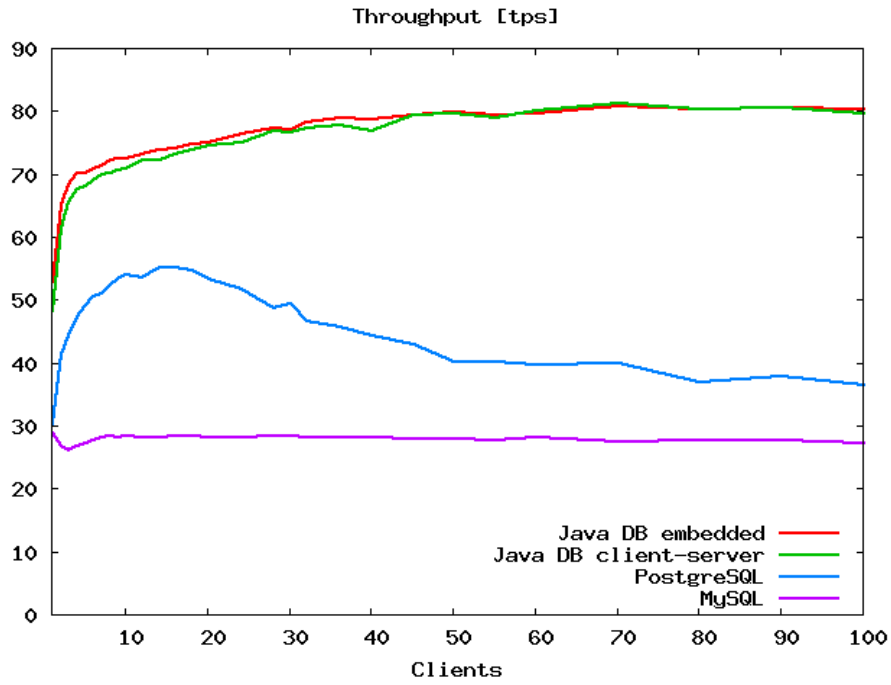


Throughput: Update Load

Main-memory database (10 MB):



Disk-based database (10 GB):



Summary

- Trade-offs between durability and performance
  - Know your requirements and select carefully
- Know what influences performance
  - Java DB configuration
  - User application
- Tips and tools to find and solve performance bottlenecks

Java DB Performs!—Comparable to competition

For More Information

- Java DB: http://developers.sun.com/javadb/
- Apache Derby: http://db.apache.org/derby/
- derby-user@apache.org
  - Discuss experiences, get help, give feedback
- derby-dev@apache.org
  - Discuss developer issues



Q&A



Java™ DB Performance

Olav Sandstå

Senior Staff Engineer
Sun Microsystems
http://developers.sun.com/javadb/

TS-45170