ApacheCon

Apache Derby

Configuring Apache Derby for Performance and Durability

Olav Sandstå
Sun Microsystems

EU 2007

ApacheCon

EU 2007

Learn how to configure and use **Apache Derby** to get the best **performance** and the required **durability** for your data.

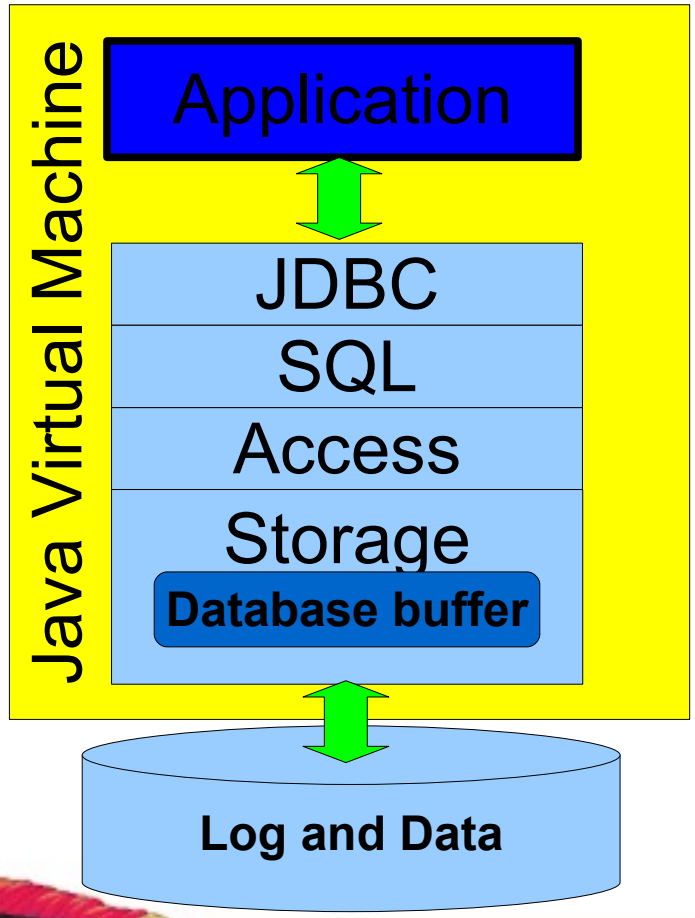
Agenda

- Apache Derby introduction
- Performance and durability
- Performance tips
- Open source database performance

Apache Derby

- Complete SQL engine including:
  - views, triggers, stored procedures, foreign keys
- Multi-user transaction support:
  - all major isolation levels
  - ACID
- Security:
  - data encryption, client authentication, GRANT/REVOKE
- Standard based:
  - JDBC 4.0 and SQL92/99/2003/XML

Architecture: Embedded



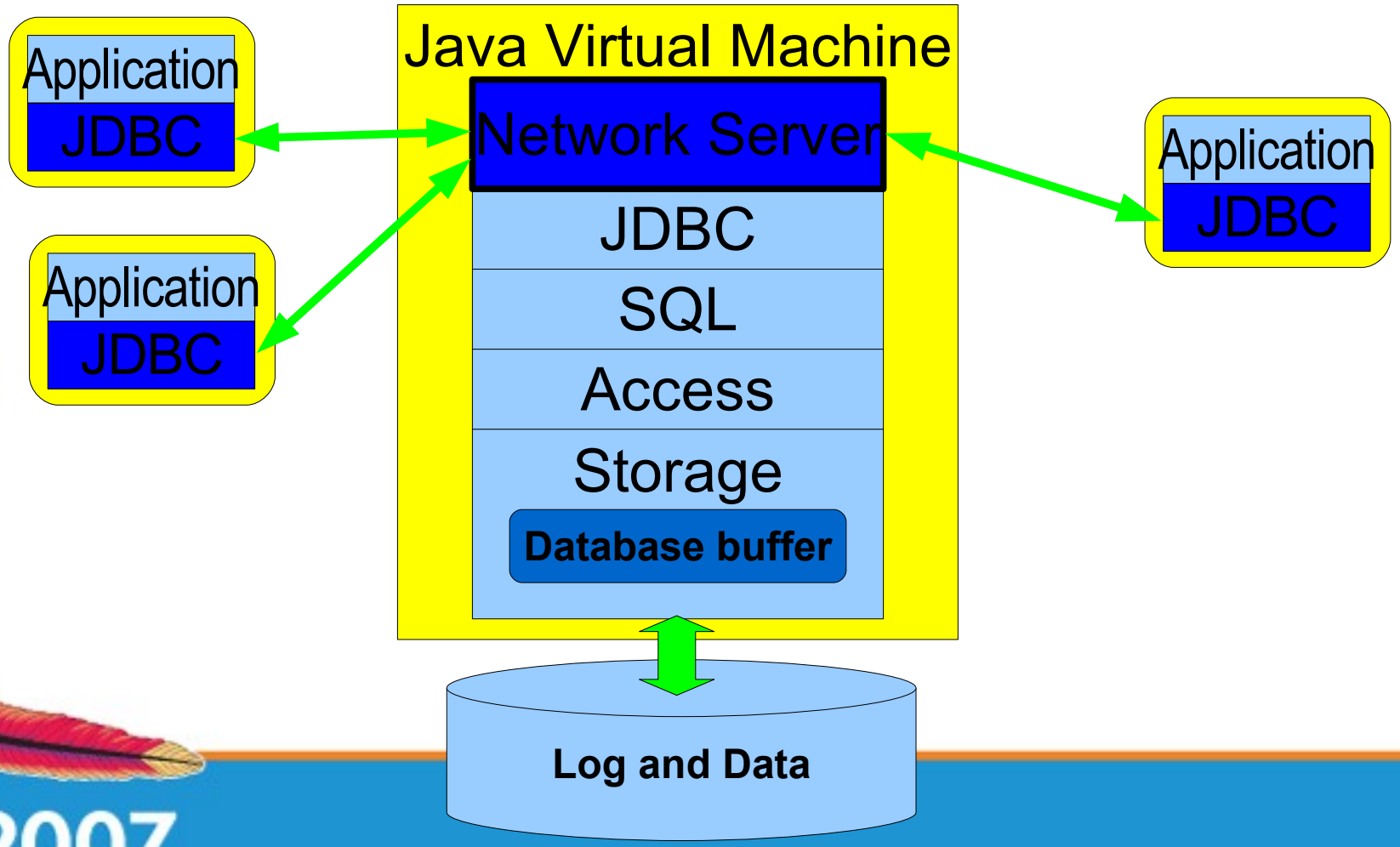
- Include derby.jar in your classpath
- Boot Derby:

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

- Create new database:

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

Architecture: Client-Server



Agenda

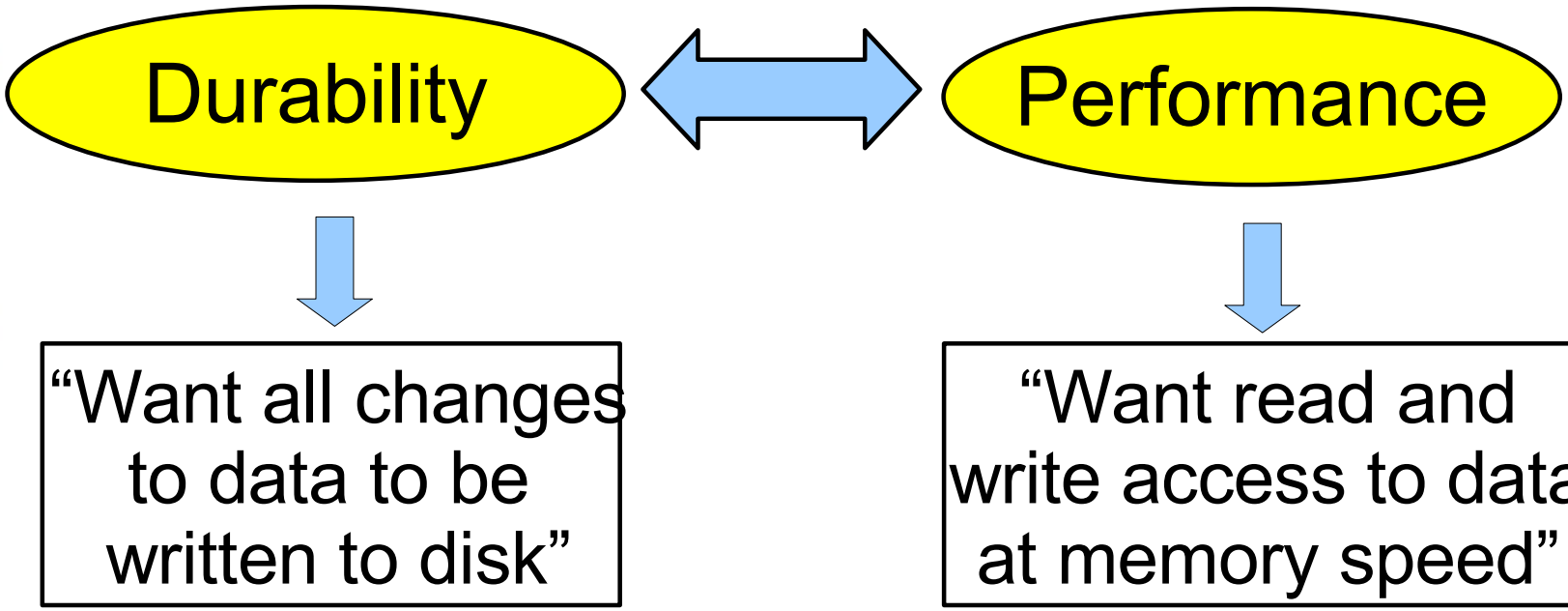
Derby introduction

Performance and durability

Performance tips

Open source database performance

Performance and Durability



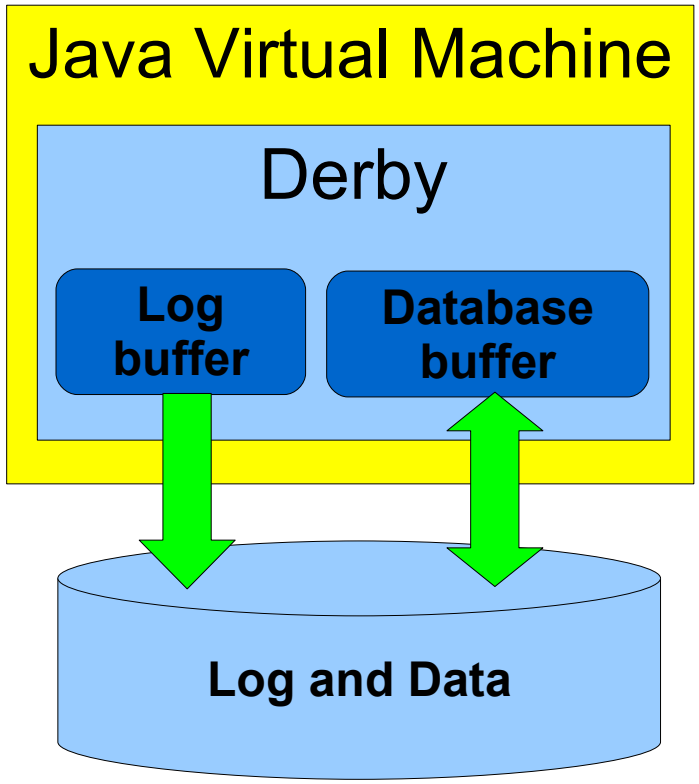
Data and Log Devices

Log device:

- Sequential write of transaction log
- Synchronous as part of commit
- Group commit

Data device:

- Data regularly written to disk as part of checkpoint
- Data read from disk on demand



Performance and Durability

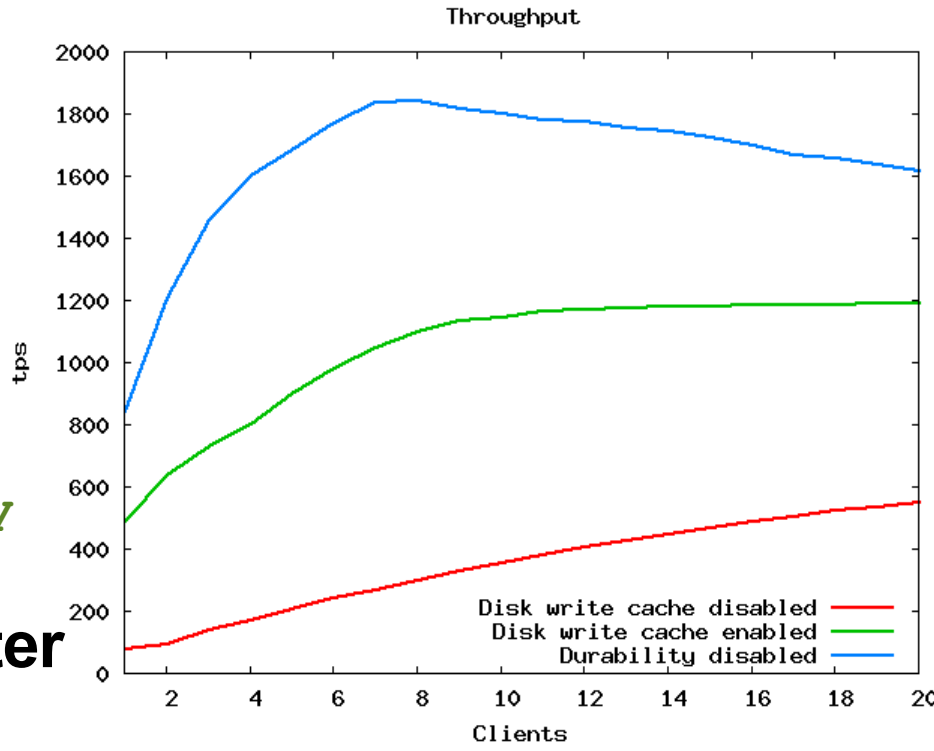
Log Device Configuration

Log device:

- Disk's write cache:
  - disabled
  - enabled
- Disable durability:

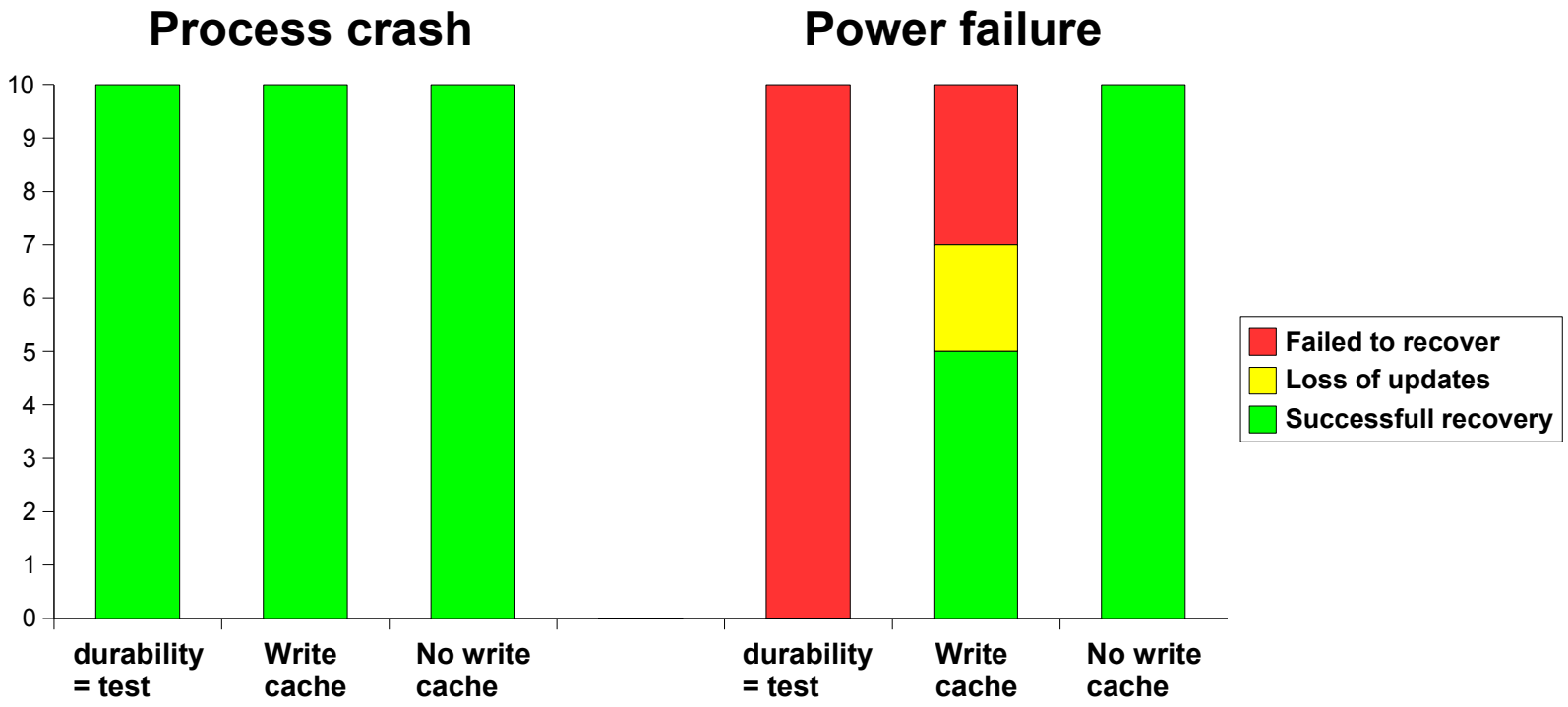
```
derby.system.durability
  =test
```

  - log written to disk **after** commit



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

Crash Recovery



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

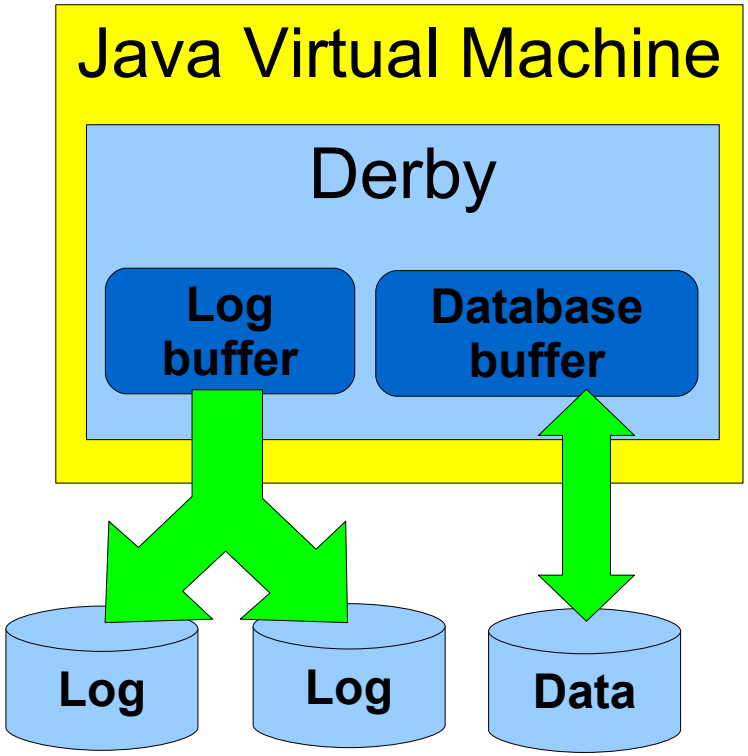
Durability:
Preparing for Hardware Failures

Log device:

- Mirror log on two disks (RAID-1)
- Must use OS support for mirroring

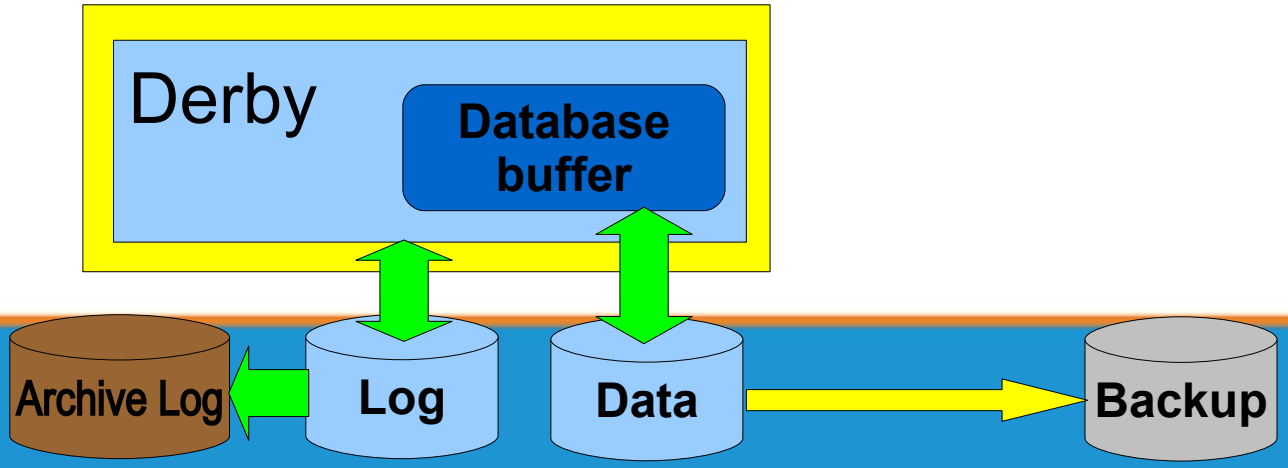
Data device:

- Offline backup
- Online backup



Online Backup

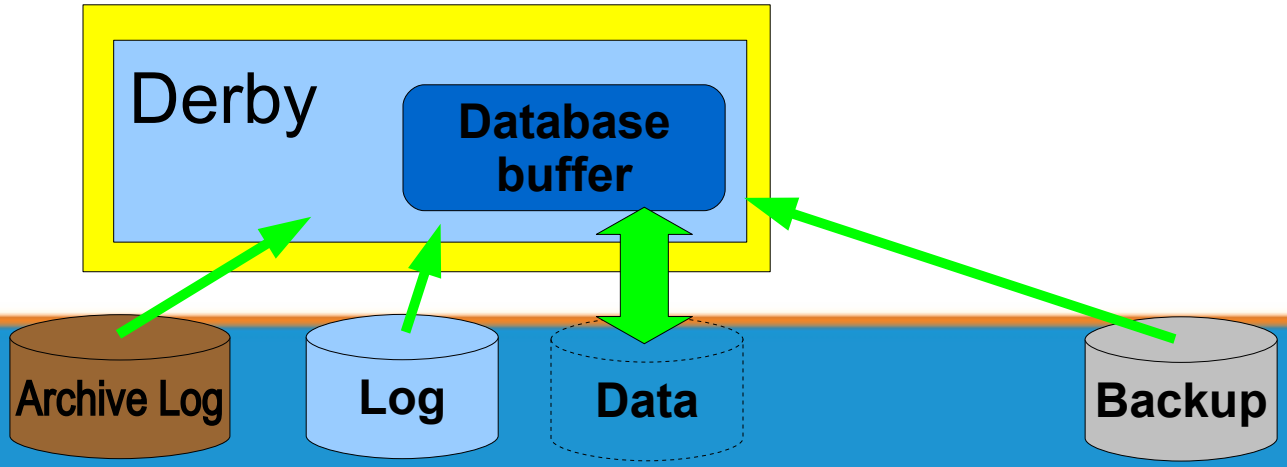
- Non-blocking backup while Derby is running
- Backup:
  - `SYSCS_UTIL.SYSCS_BACKUP_DATABASE('/home/backup/061012')`
- Backup and archive log:
  - `SYSCS_UTIL.SYSCS_BACKUP_AND_ENABLE_LOG_ARCHIVE_MODE('/home/backup/061012', 1)`



Restore and Roll-Forward Recovery

- Situation:
  - Database is corrupted
  - Disk with database has errors
- Restore and roll-forward recovery using:
  - JDBC connection url:

```
'jdbc:derby:myDB;rollForwardRecoveryFrom=
/home/backup'
```



Agenda

Derby introduction

Performance and durability

Performance tips

Open source database performance

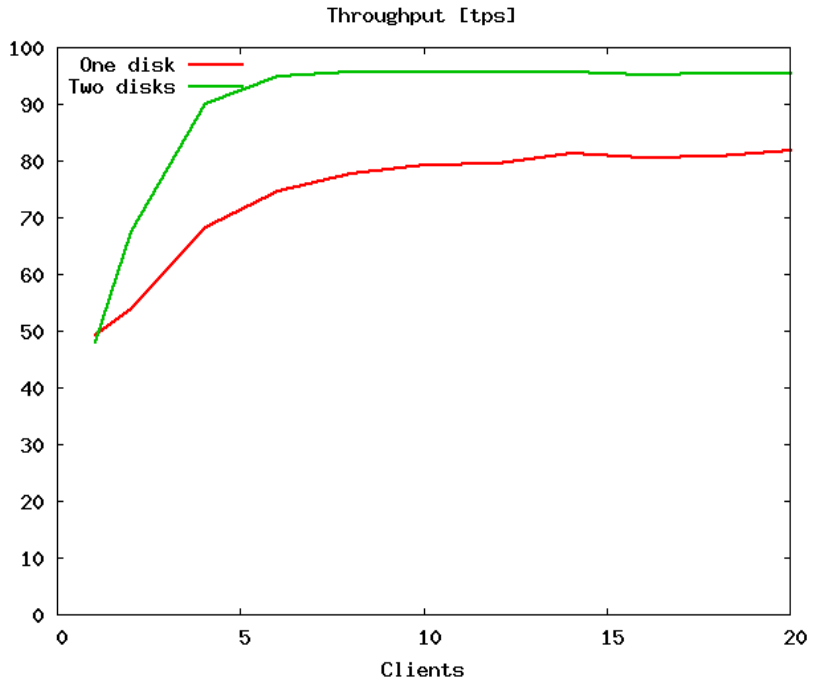
Performance Tip 1:
Separate Data and Log Devices

Log on separate disk:

- Utilize sequential write bandwidth on disk
- Configuration:

  JDBC connection url:

  `logDevice=<path>`



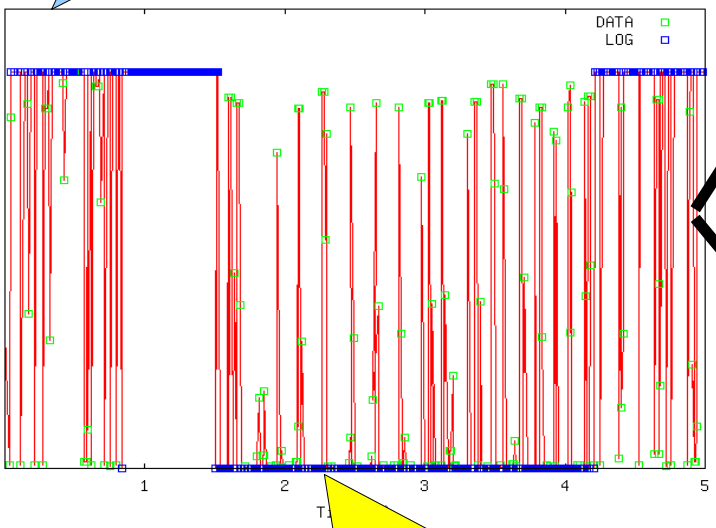
Performance tip:

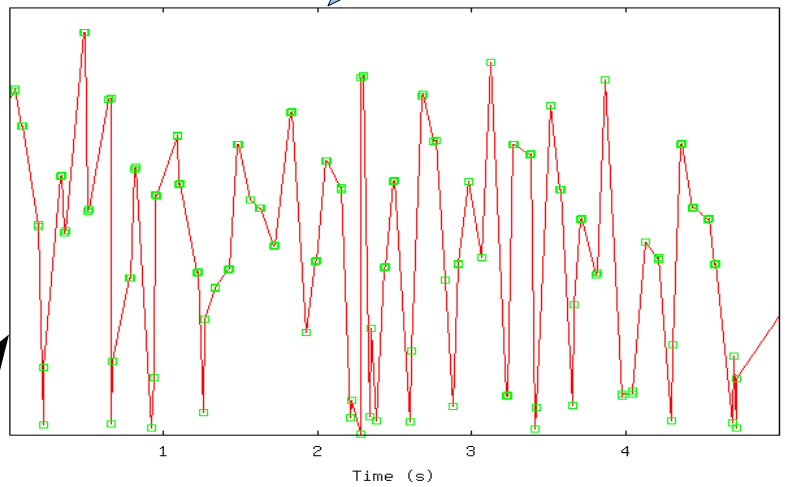
- Use separate disks for data and log device

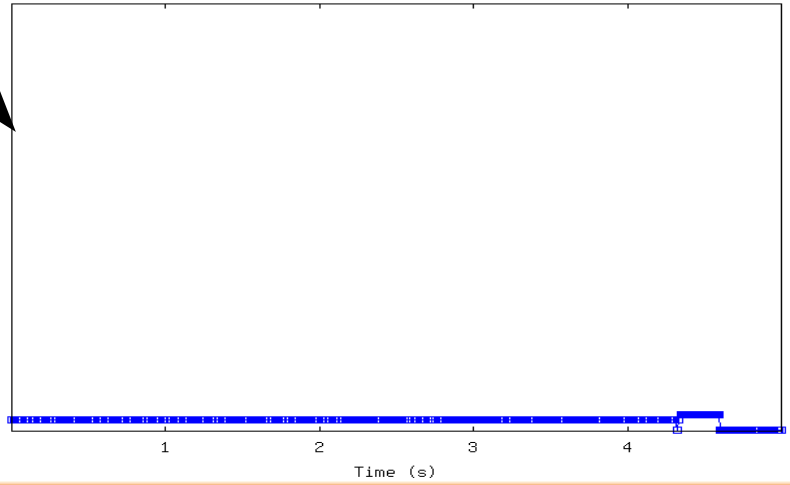
Disk Activity

Data and log on separate disks

Data and log on same disk



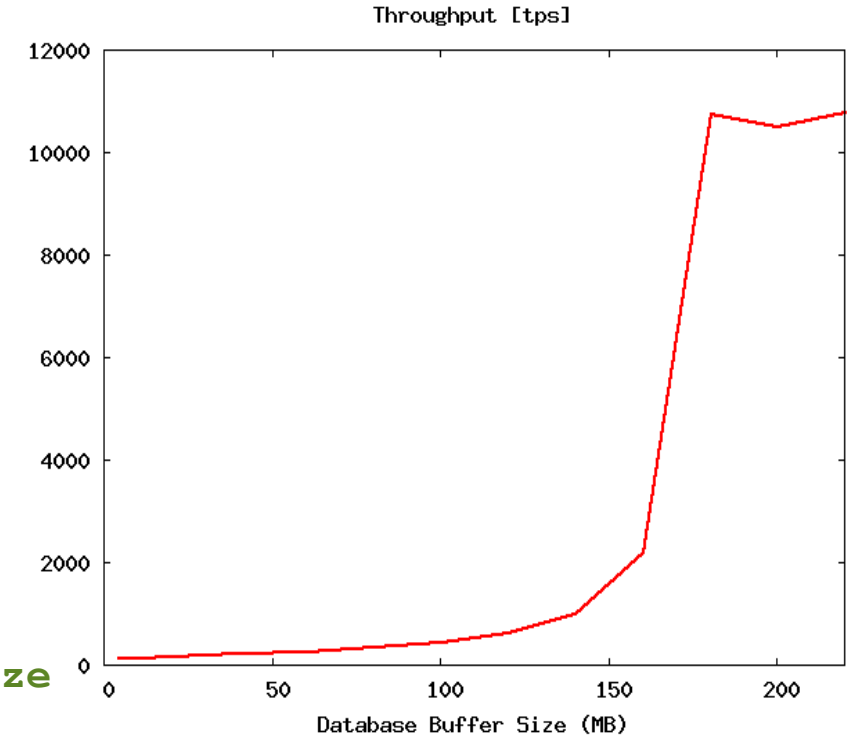




Disk head movement for 5 seconds of database activity

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:
Use Prepared Statements

- Compilation of SQL statements is expensive:

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

  - generates Java byte code and loads classes

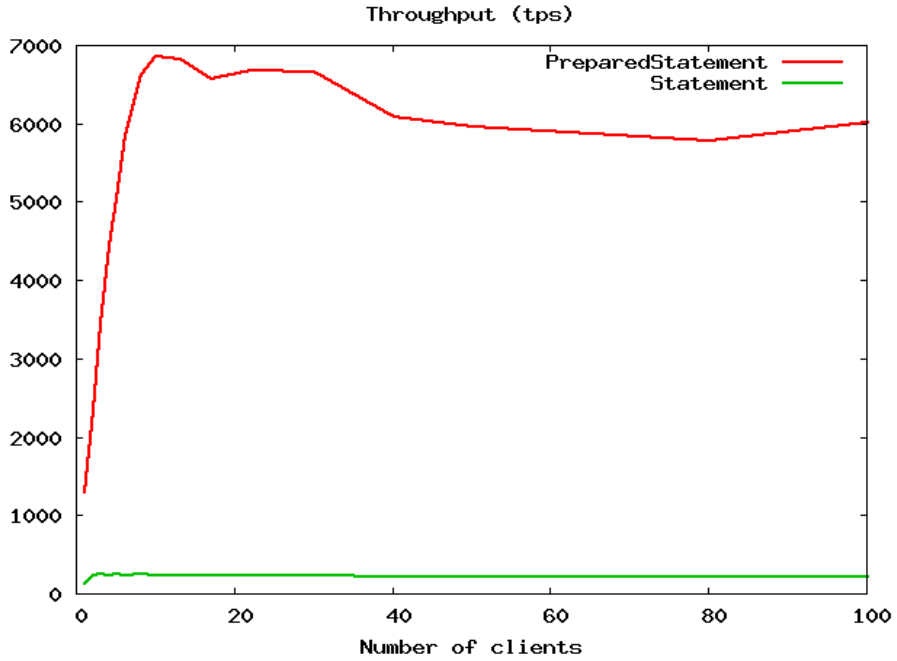
- Prepared statements eliminate this cost:

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

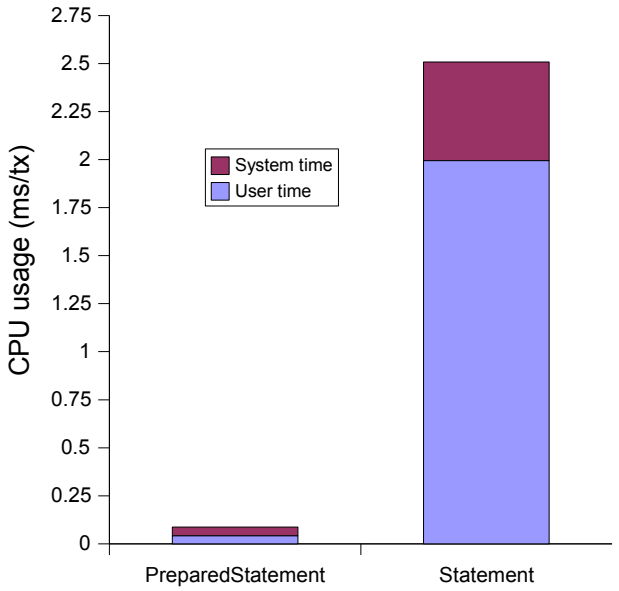
  - generated Java byte code can be JIT compiled

Performance Tip 3:
Use Prepared Statements, example

Throughput:



CPU usage:



Performance tip:

- **USE** prepared statements - and **REUSE** them

Performance Tip 4:
Avoid Table Scans

Two ways of locating data:

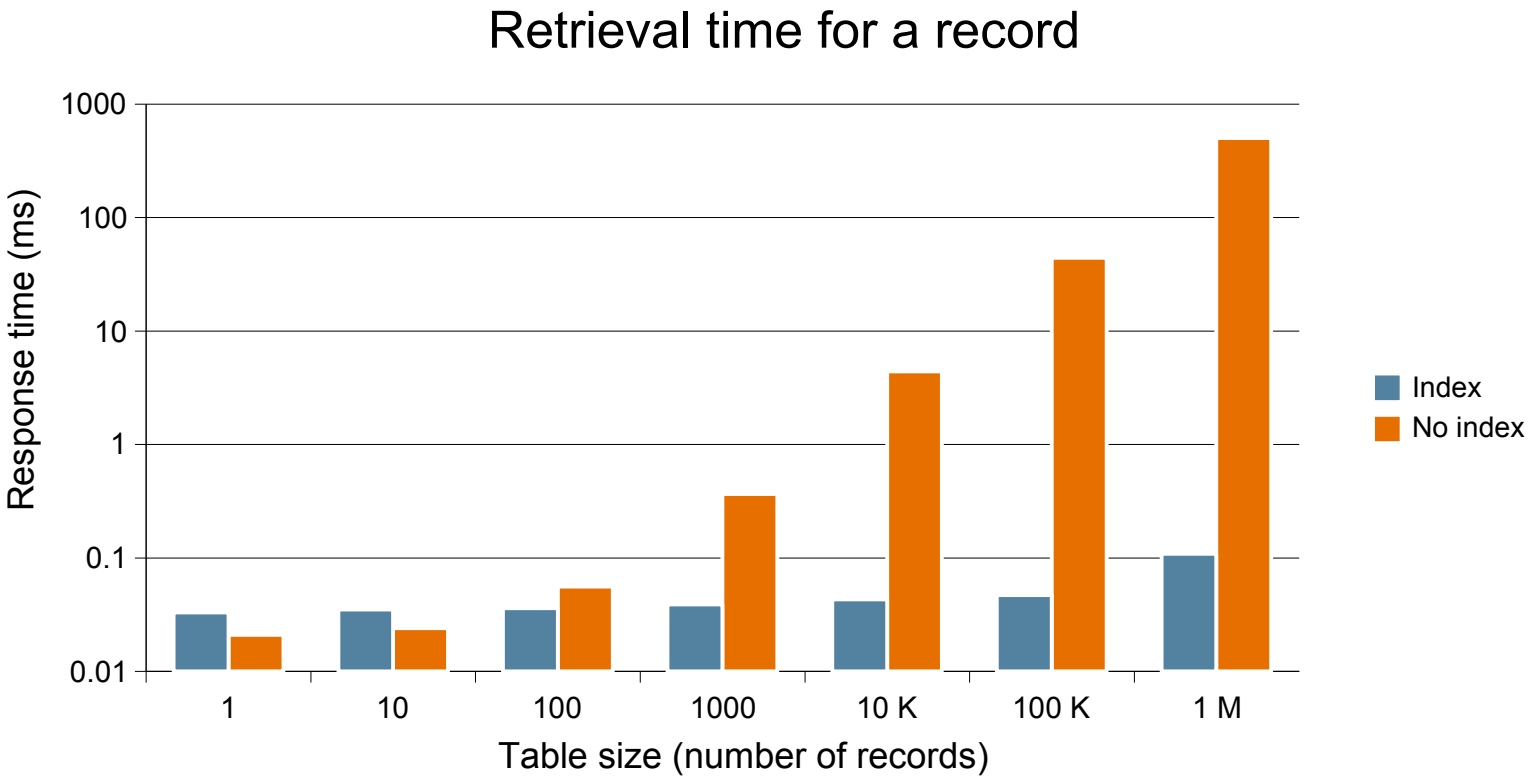
- Table scan: reads the entire table
- Index: finds the data by reading a few blocks

Avoid table scans:

- Use indexes to optimize frequently used access paths:

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

Performance Tip 4:
Avoid Table Scans, example



Performance tip:

- Create and use indexes

Performance Tip 5:
JDBC Tips

- Close JDBC objects after in use
  - Connections, Statements, ResultSets, Streams
- Use transactions - do not rely on auto-commit
  - Particularly for insert/update/delete operations
- Batch updates reduce network traffic

Performance Tip 6:
Understand the Load on Derby

- Know the load on the database:
  - `derby.language.logStatementText=true`
- Check the query plan:
  - `derby.language.logQueryPlan=true`
- Use run-time statistics:
  - `SYSCS_UTIL.SYSCS_SET_RUNTIMESTATISTICS(1)`
  - `SYSCS_UTIL.SYSCS_SET_STATISTICS_TIMING(1)`
  - `SYSCS_UTIL.SYSCS_GET_RUNTIMESTATISTICS()`

Performance tip:

- Understand the query execution and where resources are spent

Performance Tip 7:
Optimizer Overrides

- 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

Performance tip:
- Use optimizer overrides
- - **but only** when needed

Performance Tip 8:
Understand Locking Issues

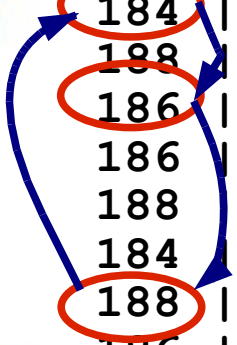
- Isolation level:
  - Reducing isolation level increases concurrency
- Lock escalation:
  - Default: escalation from **row locks** to **table locks** when 5000 locks are set on the table
  - `derby.locks.escalationThreshold=100`
  - `LOCK TABLE t1 IN {SHARE|EXCLUSIVE} MODE`
- Deadlock tracing:
  - `derby.locks.monitor=true`
  - `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	SQL07042502
188	ROW	S	T1	(1,1)	GRANT	SQL07042502
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

Performance and durability

Performance tips

Open source database performance

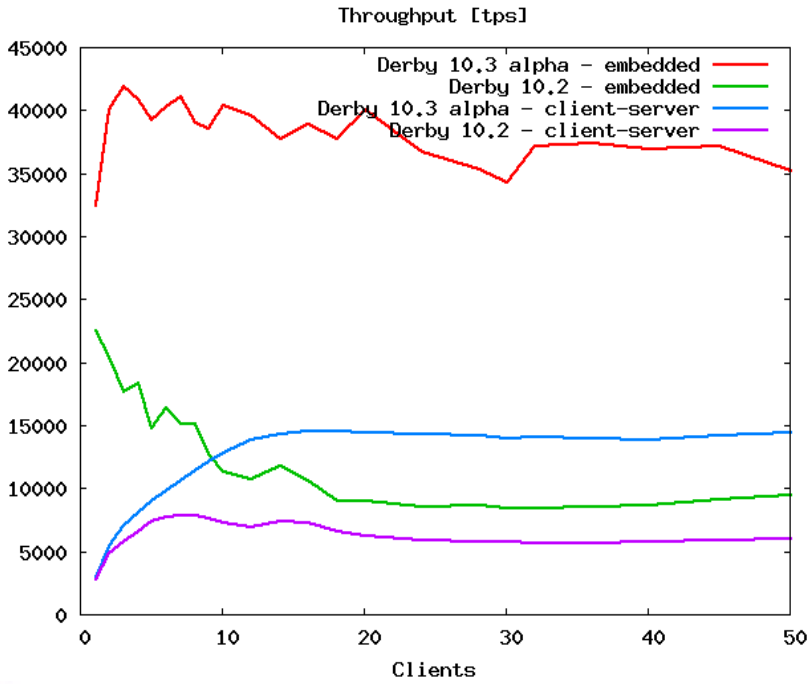
Derby 10.3: Performance Improvements

- Embedded:
  - reduced synchronization & 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

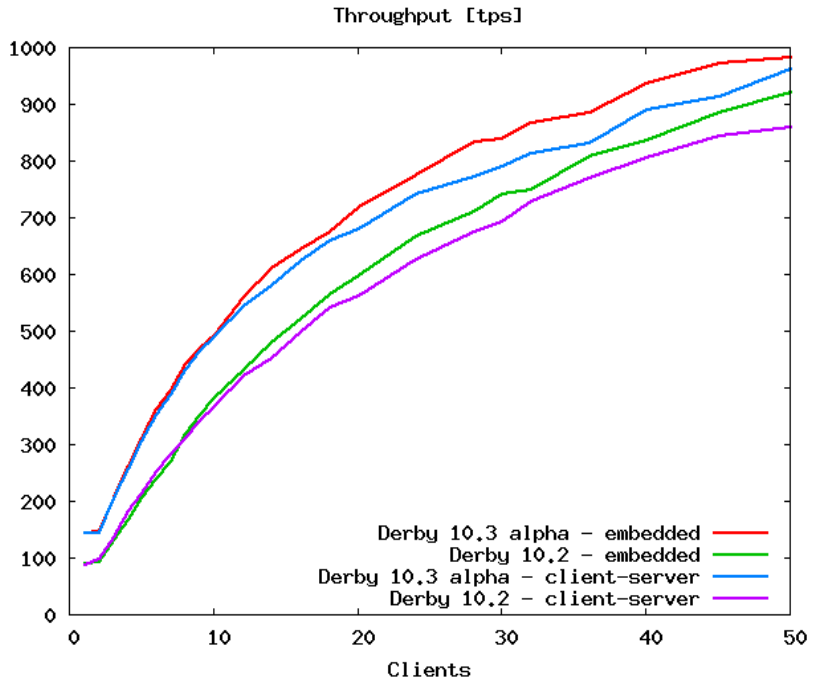
30-150% increased throughput on simple queries

Derby 10.3: Performance Improvements

Single-record select:



Update load:



Comparing Performance

Databases:

- Derby 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:

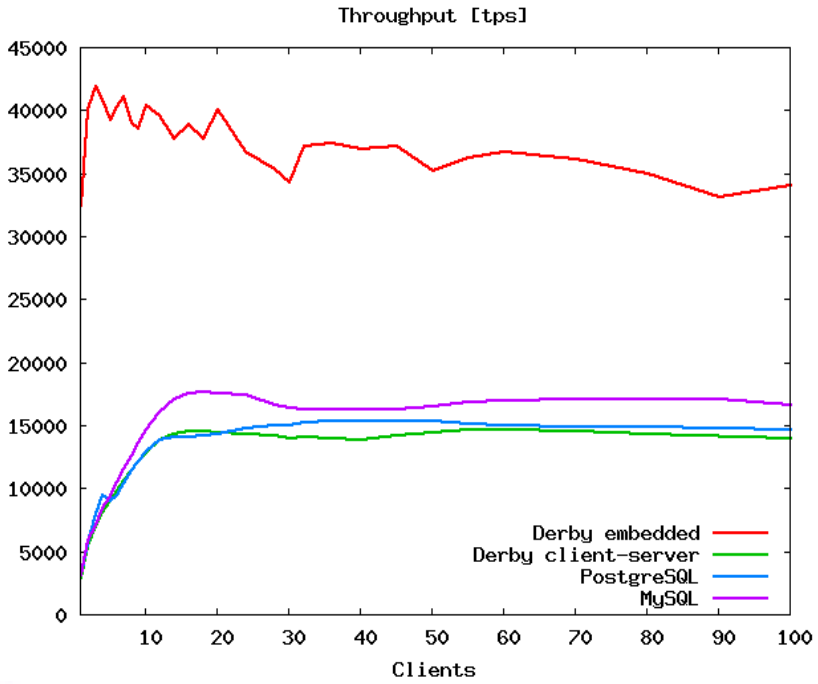
3 upd., 1 ins., 1 select

Test platform:

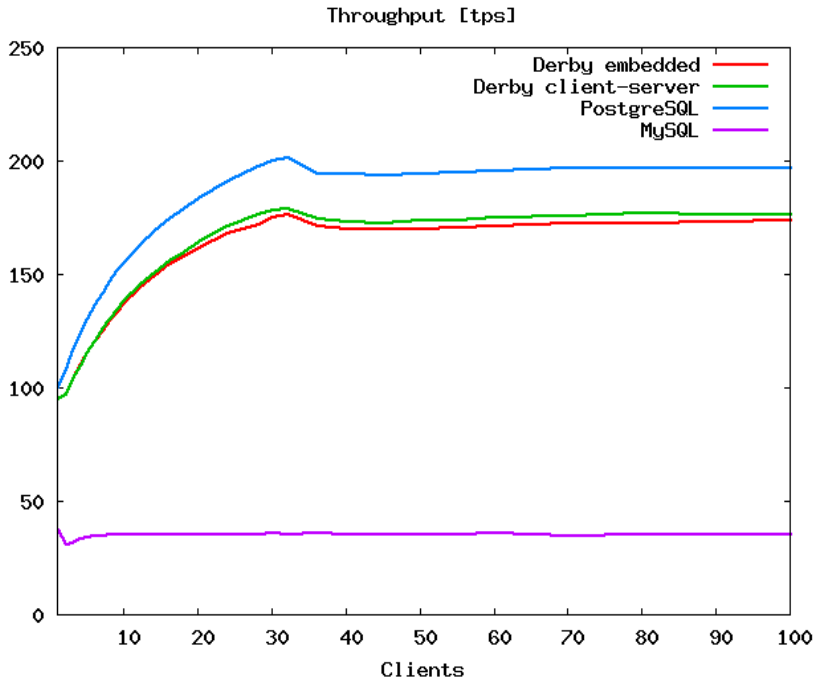
- 2 AMD Opteron CPUs
- Solaris 10
- Sun Java SE 6

Throughput: Single-record Select

Main-memory database (10 MB):

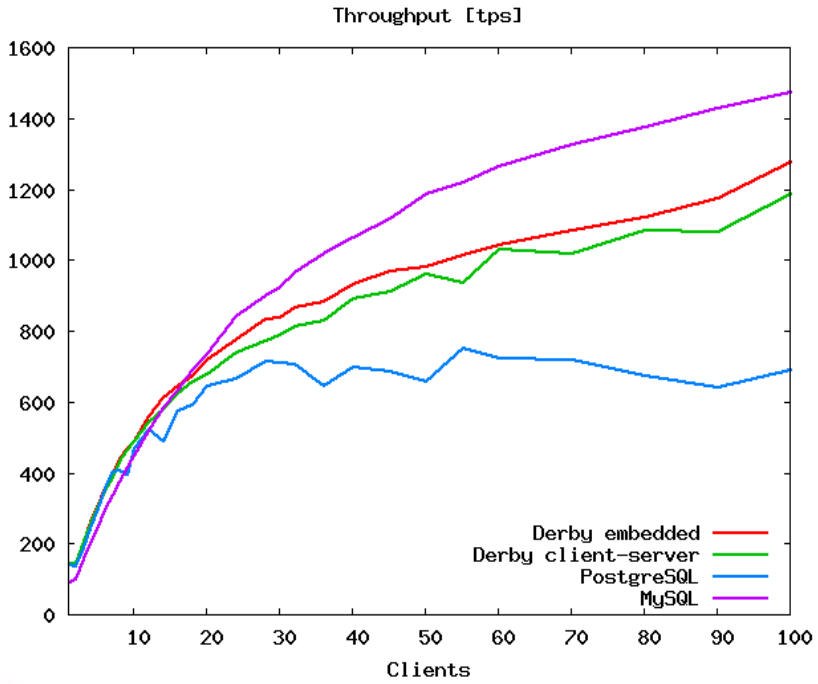


Disk-based database (10 GB):

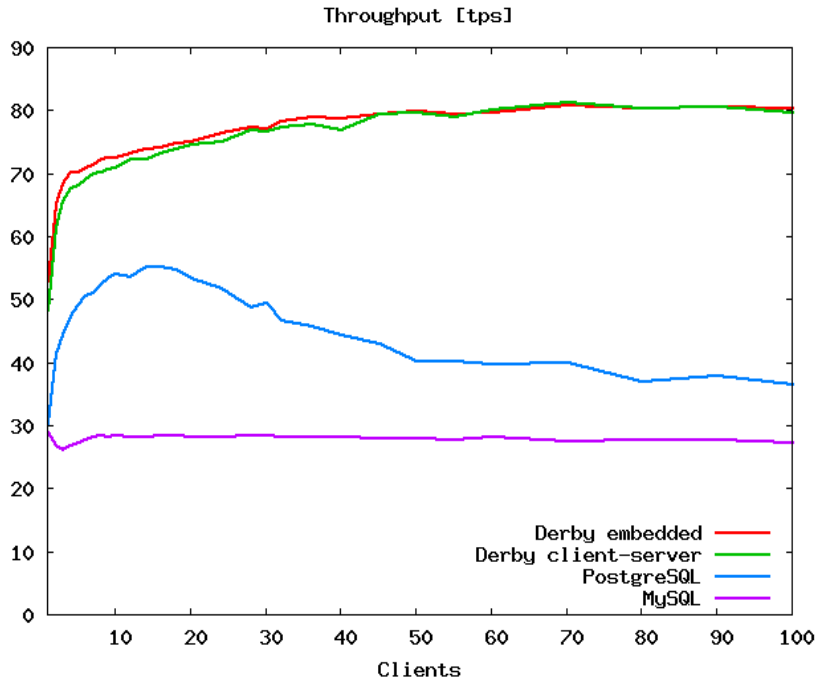


Throughput: Update Load

Main-memory database (10 MB):



Disk-based database (10 GB):



Summary

- Trade-offs between durability and performance
  - Know your requirements and select carefully
- Have a strategy for backup and recovery
  - Test it!!
- Know what influence performance
  - Derby configuration
  - User application
- Tips and tools to find and solve performance bottleneck