Building Highly Available Database Applications for Apache Derby

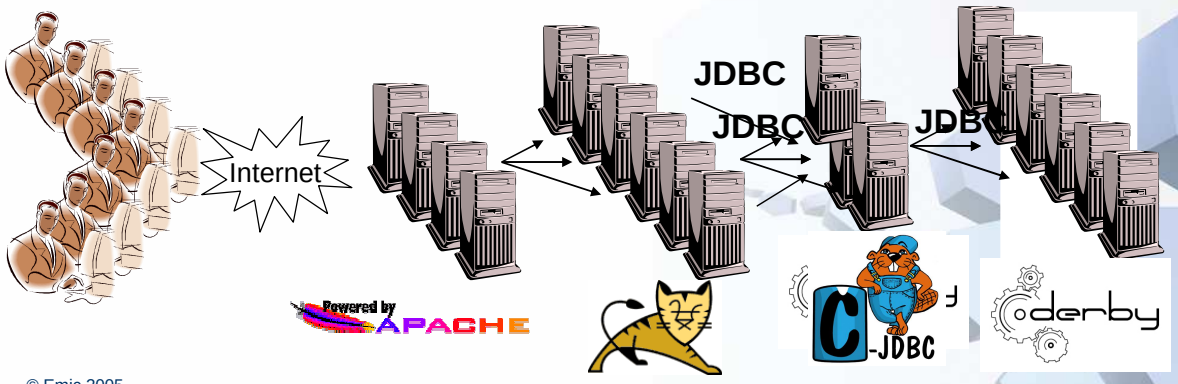
Emmanuel Cecchet

Principal architect - Emic Networks

Chief architect – ObjectWeb consortium

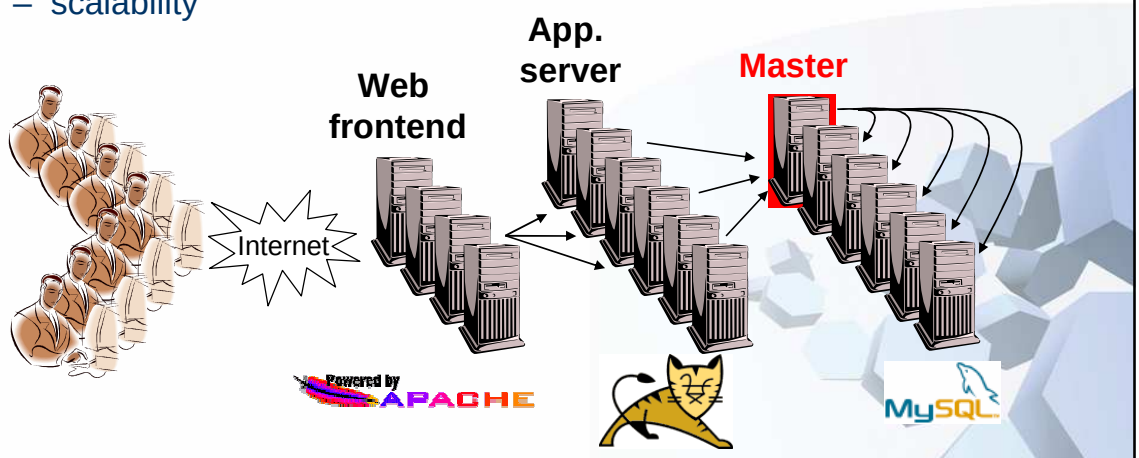
Motivations

- Database tier should be
  - scalable
  - highly available
  - without modifying the client application
  - database vendor independent
  - on commodity hardware



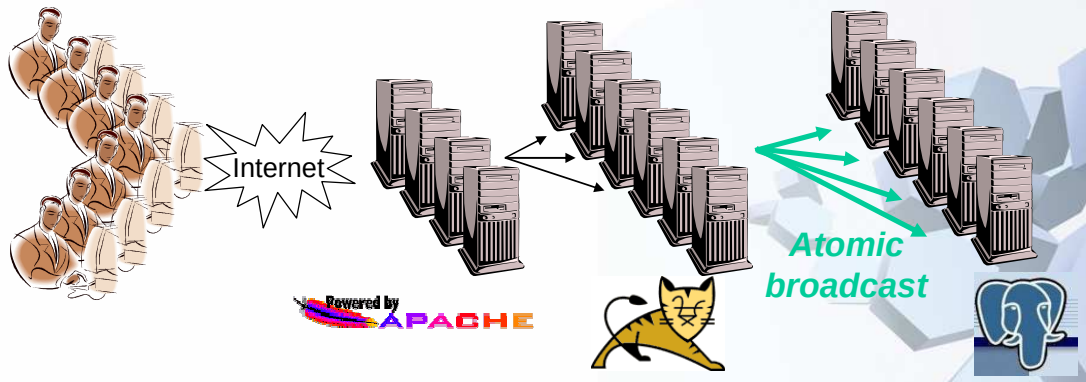
Scaling the database tier – Master-slave replication

- Cons
  - failover time/data loss on master failure
  - read inconsistencies
  - scalability



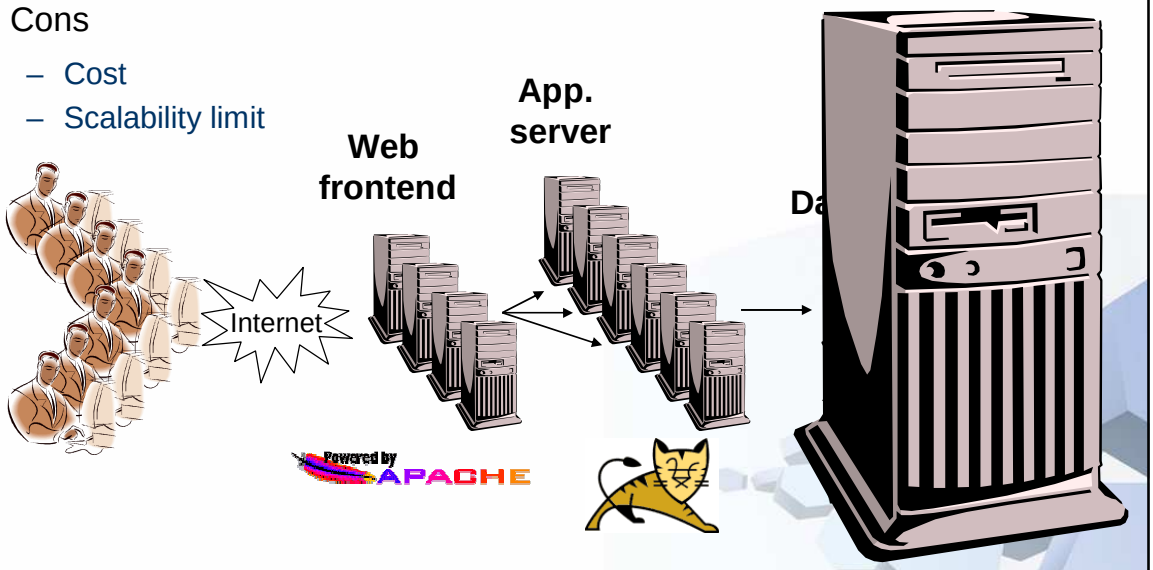
Scaling the database tier – Atomic broadcast

- Cons
  - atomic broadcast scalability
  - no client side load balancing
  - heavy modifications of the database engine



Scaling the database tier – SMP

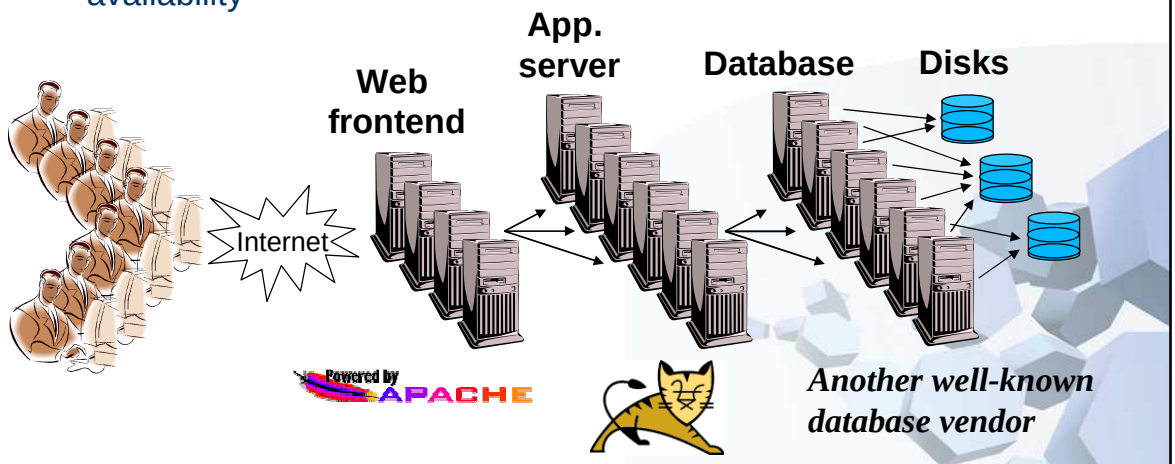
- Cons
  - Cost
  - Scalability limit



Well-known hardware + database vendors here

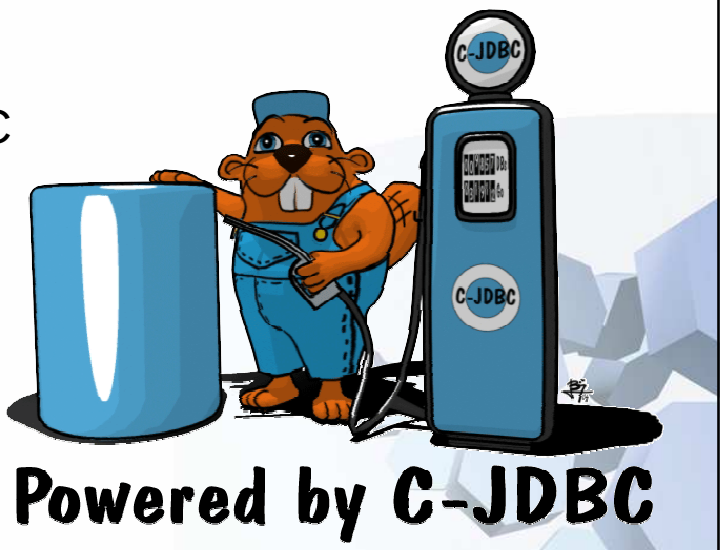
Scaling the database tier – Shared disks

- Cons
  - still expensive hardware
  - availability



Outline

- RAIDb
- C-JDBC
- Derby and C-JDBC
- Scalability
- High availability

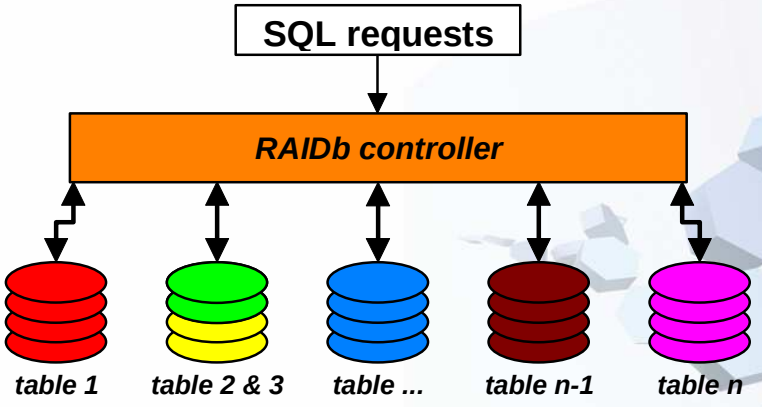


RAIDb concept

- Redundant Array of Inexpensive Databases
- RAIDb controller
  - gives the view of a single database to the client
  - balance the load on the database backends
- RAIDb levels offers various tradeoff of performance and fault tolerance

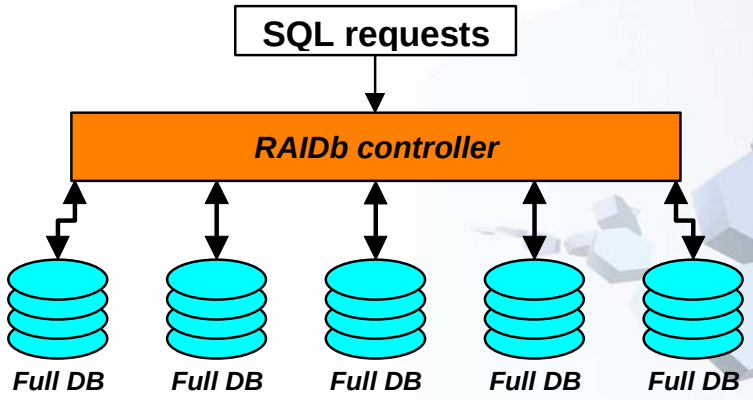
RAIDb levels

- RAIDb-0
  - partitioning
  - no duplication and no fault tolerance
  - at least 2 nodes



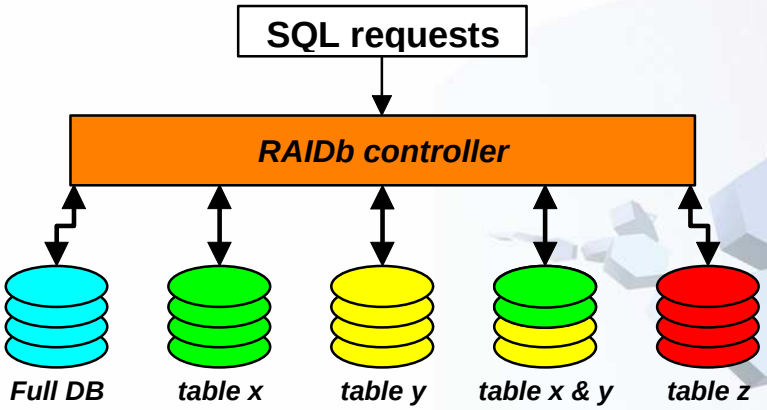
RAIDb levels

- RAIDb-1
  - mirroring
  - performance bounded by write broadcast
  - at least 2 nodes



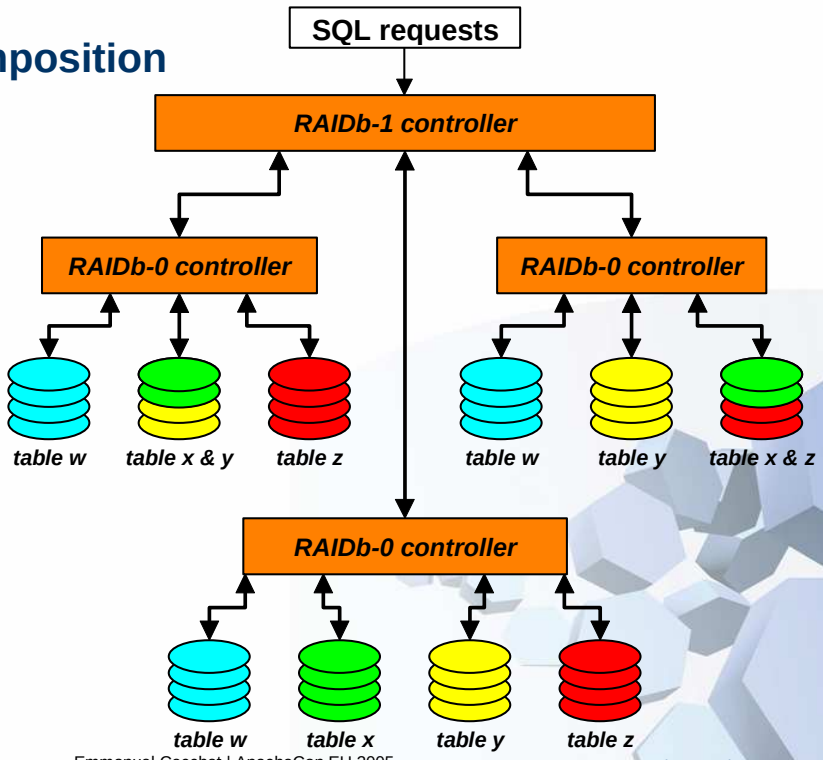
RAIDb levels

- RAIDb-2
  - partial replication
  - at least 2 copies of each table for fault tolerance
  - at least 3 nodes



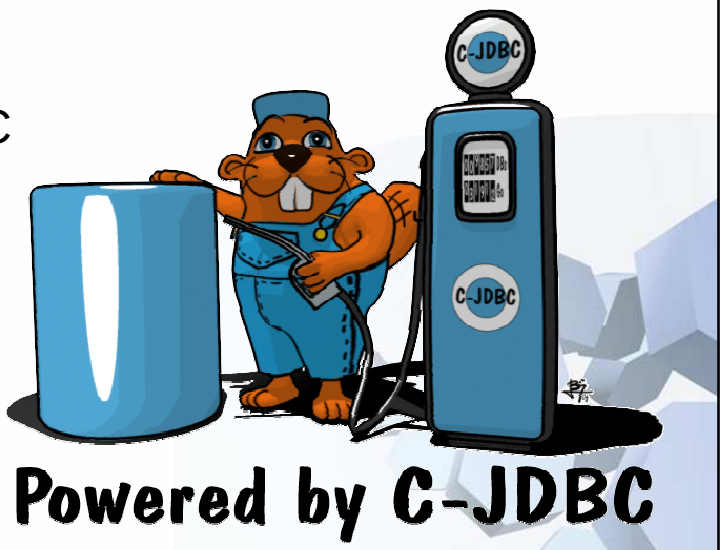
RAIDb levels composition

- RAIDb-1-0
  - no limit to the composition deepness



Outline

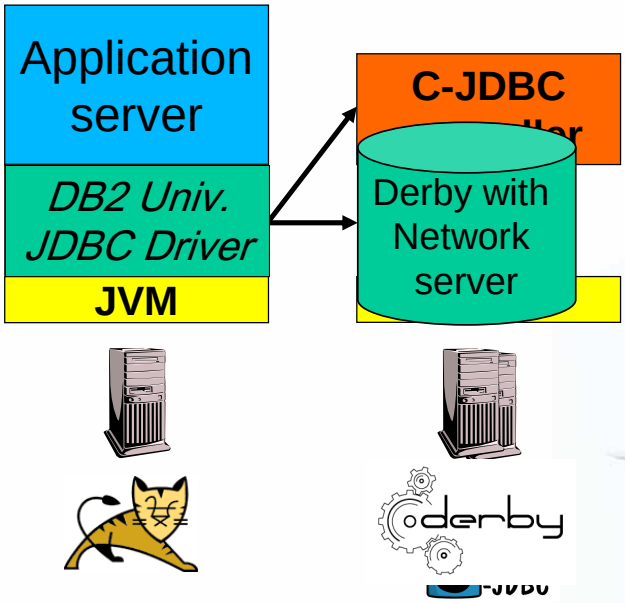
- RAIDb
- C-JDBC
- Derby and C-JDBC
- Scalability
- High availability



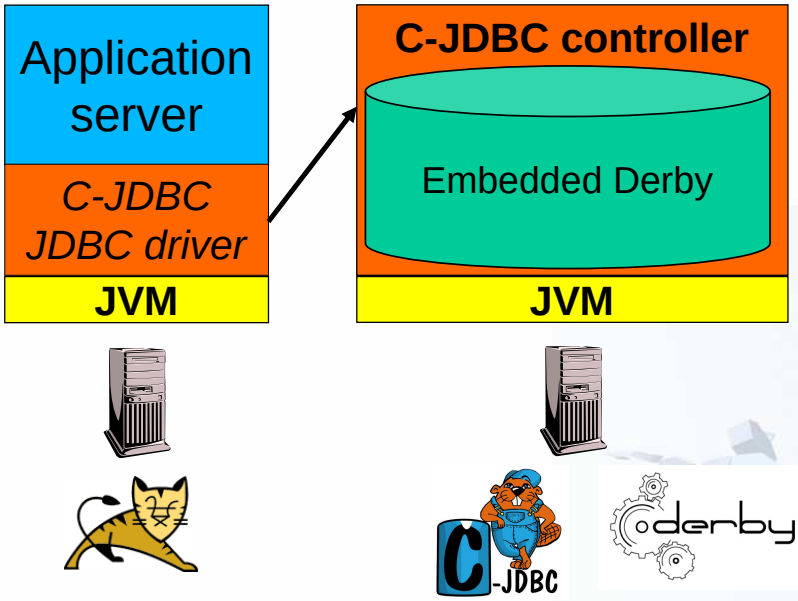
C-JDBC overview

- Middleware implementing RAIDb
  - 100% Java implementation
  - open source (LGPL)
- Two components
  - generic JDBC driver (C-JDBC driver)
  - C-JDBC Controller
- Read-one, Write all approach
  - provides eager (strong) consistency
- Supports heterogeneous databases

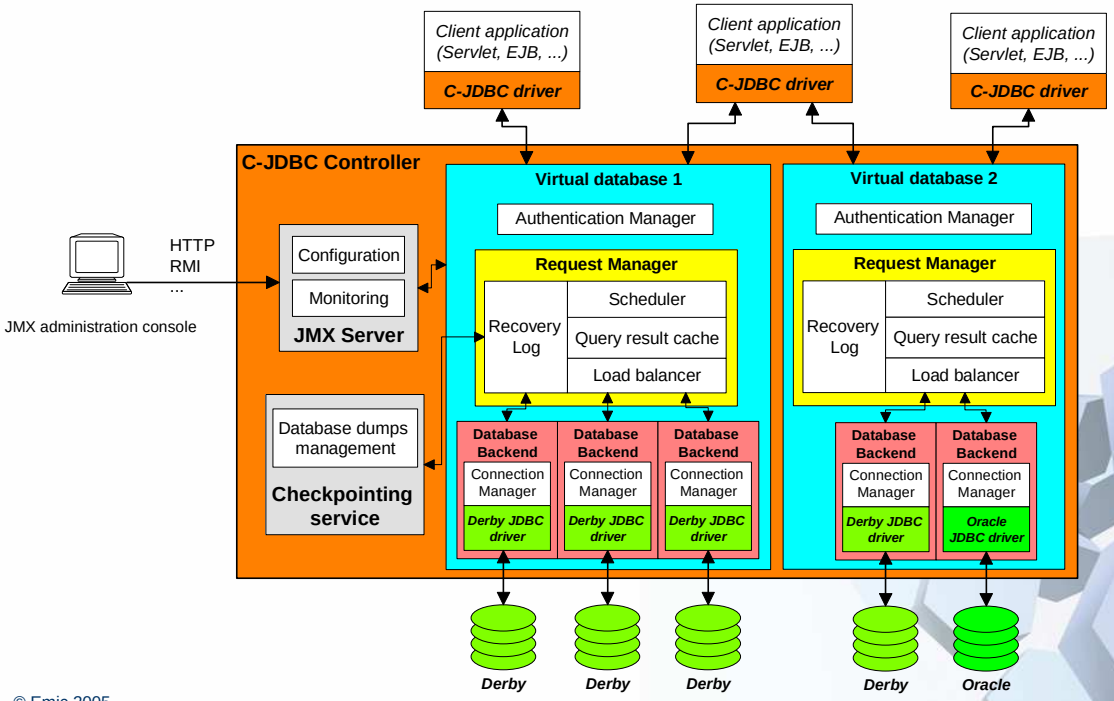
architectural overview



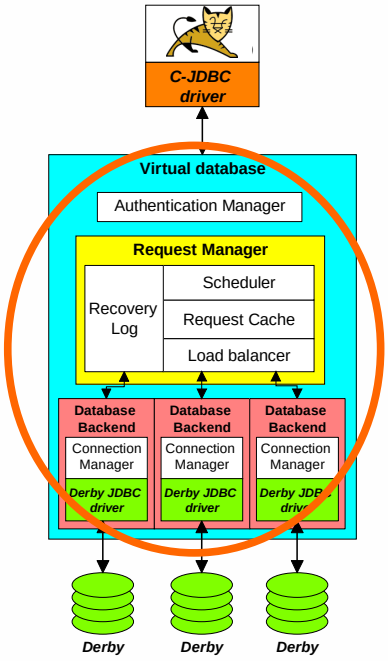
Using C-JDBC as an open source driver for Derby



Inside the C-JDBC Controller

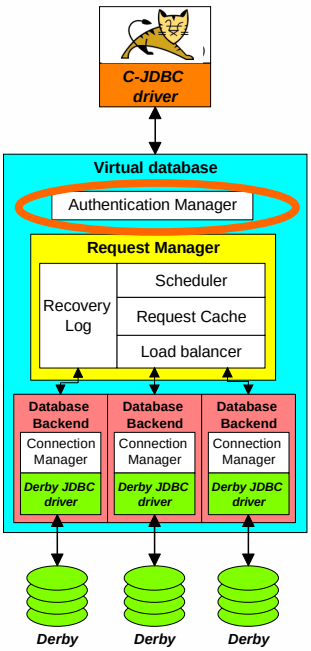


Virtual Database



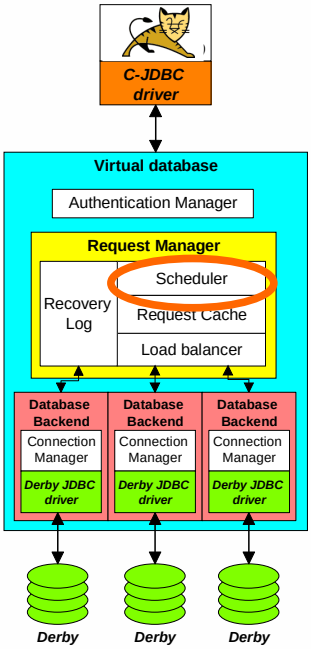
- gives the view of a single database
- establishes the mapping between the database name used by the application and the backend specific settings
- backends can be added and removed dynamically
- configured using an XML configuration file

Authentication Manager



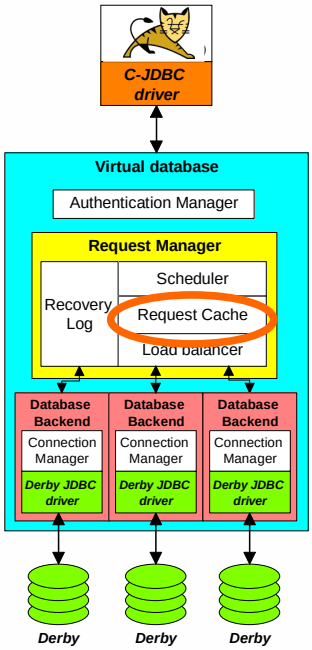
- Matches real login/password used by the application with backend specific login/ password
- Administrator login to manage the virtual database

Scheduler



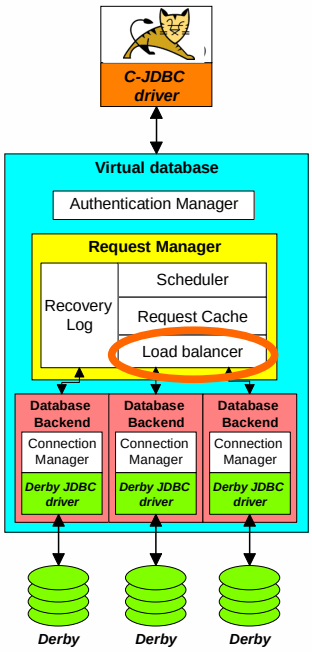
- Manages concurrency control
- Specific implementations for RAIDb 0, 1 and 2
- Pass-through
- Optimistic and pessimistic transaction level
  - uses the database schema that is automatically fetched from backends

Request cache



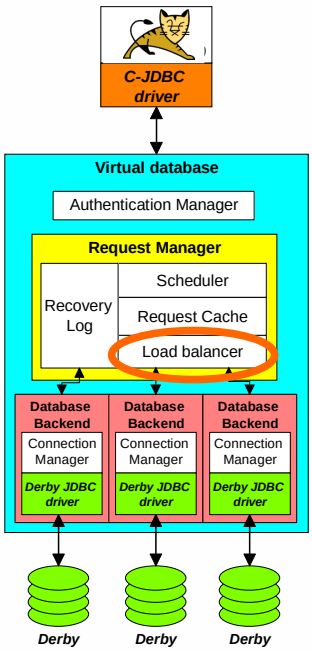
- 3 optional caches
  - tunable sizes
- parsing cache
  - parse request skeleton only once
  - INSERT INTO t VALUES (?,?,?)
- metadata cache
  - column metadata
  - fields of a request
- result cache
  - caches results from SQL requests
  - tunable consistency
  - fine grain invalidations
  - optimizations for findByPk requests

Load balancer 1/2



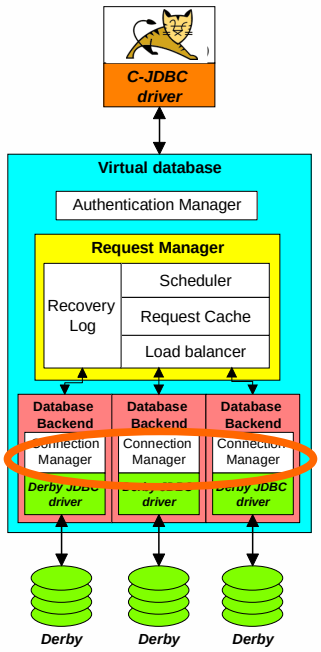
- RAIDb-0
  - query directed to the backend having the needed tables
- RAIDb-1
  - read executed by current thread
  - write executed in parallel by a dedicated thread per backend
  - result returned if one, majority or all commit
  - if one node fails but others succeed, failing node is disabled
- RAIDb-2
  - same as RAIDb-1 except that writes are sent only to nodes owning the updated table

Load balancer 2/2



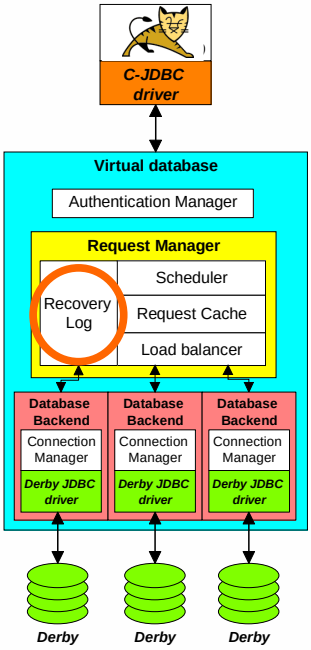
- Static load balancing policies
  - Round-Robin (RR)
  - Weighted Round-Robin (WRR)
- Least Pending Requests First (LPRF)
  - request sent to the node that has the shortest pending request queue
  - efficient even if backends do not have homogeneous performance

Connection Manager



- C-JDBC driver provides transparent connection pooling
- Connection pooling for a backend
  - no pooling
  - blocking pool
  - non-blocking pool
  - dynamic pool
- Connection pools defined on a per login basis
  - resource management per login
  - dedicated connections for admin

Recovery Log

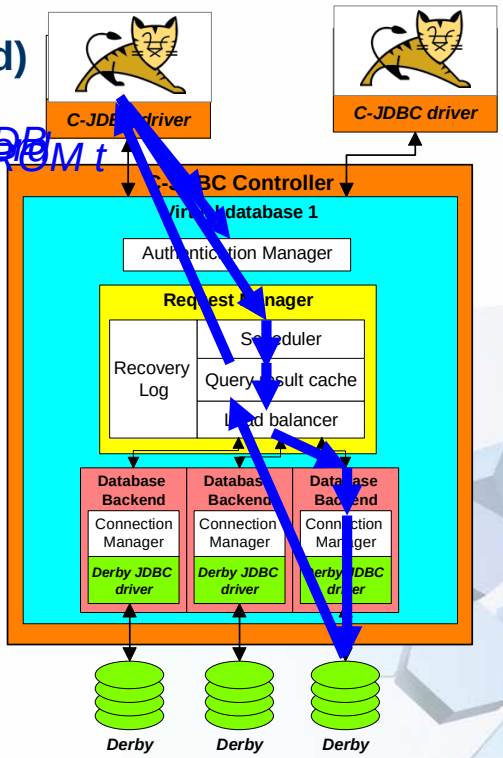


- Checkpoints are associated with database dumps
- Record all updates and transaction markers since a checkpoint
- Used to resynchronize a database from a checkpoint
- JDBCRecoveryLog
  - store log information in a database
  - can be re-injected in a C-JDBC cluster for fault tolerance

Functional overview (read)

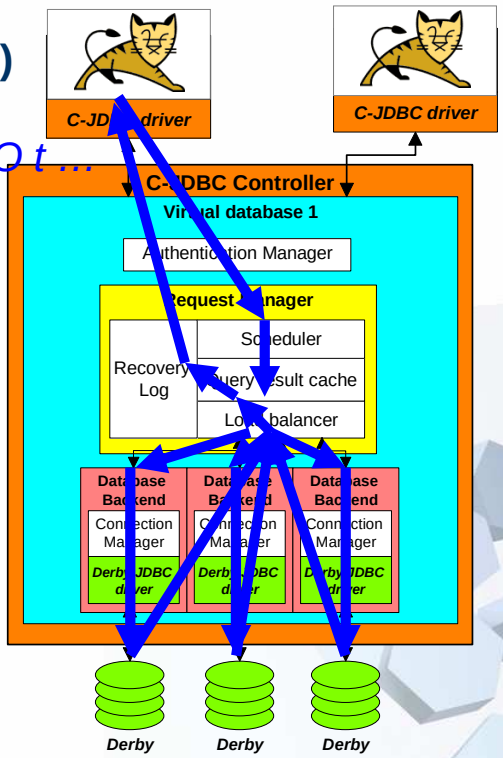
connect myDB

execute SELECT * FROM t



Functional overview (write)

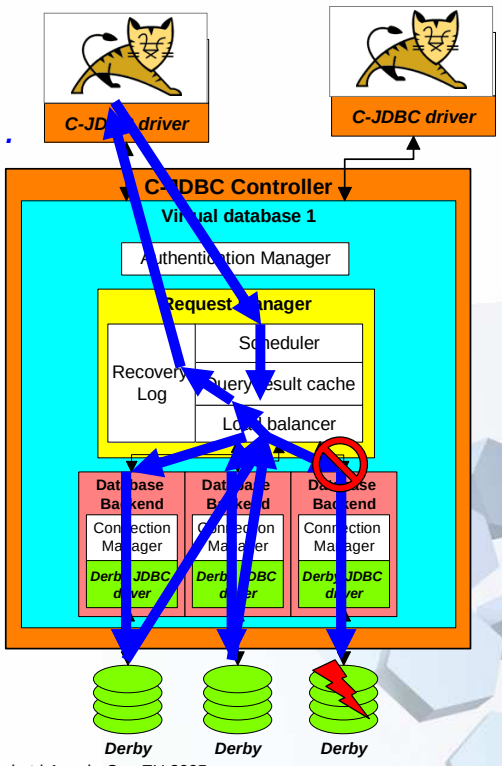
execute *INSERT INTO t …*



Failures

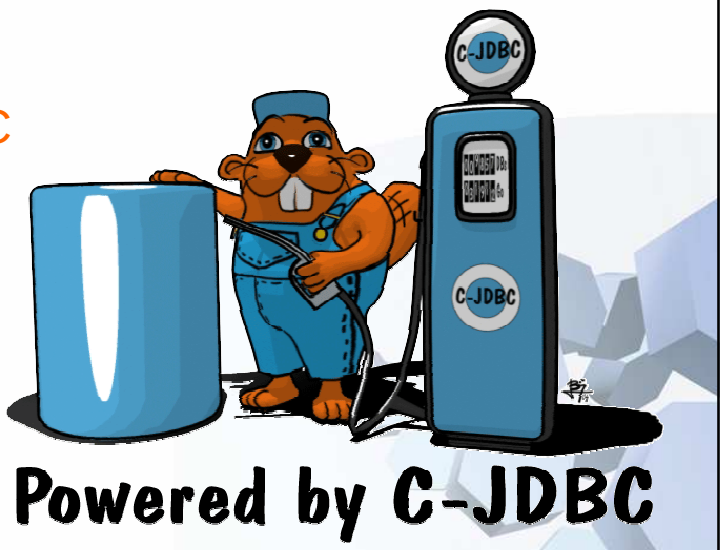
execute *INSERT INTO t …*

- No 2 phase-commit
  - parallel transactions
  - failed nodes are automatically disabled



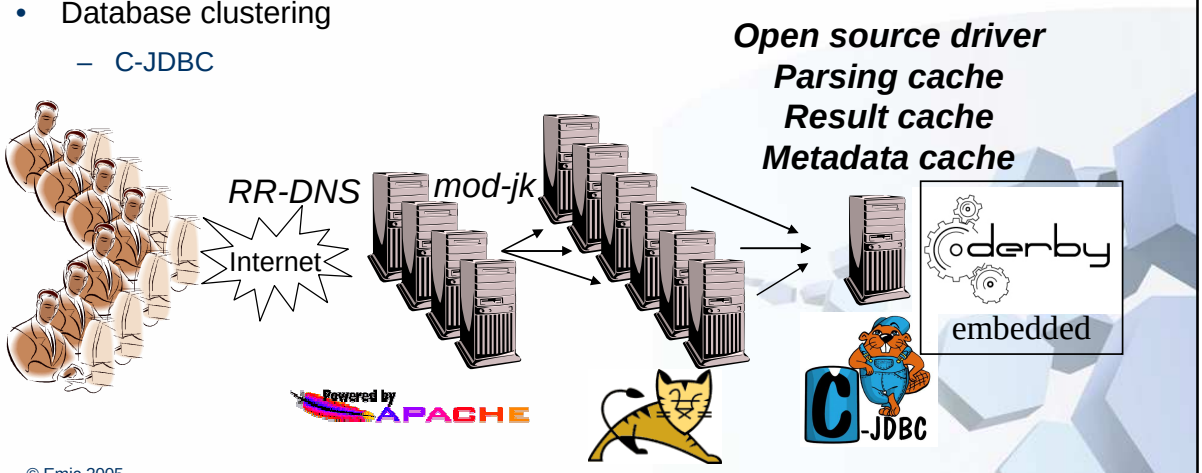
Outline

- RAIDb
- C-JDBC
- Derby and C-JDBC
- Scalability
- High availability



Highly available web site

- Apache clustering
  - L4 switch, RR-DNS, One-IP techniques, LVS, …
- Tomcat clustering
  - mod_jk (T4), mod_proxy/mod_rewrite (T5), session replication
- Database clustering
  - C-JDBC

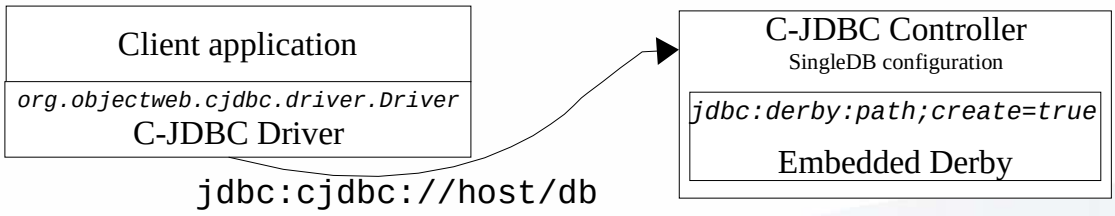


Result cache

- Cache contains a list of SQL→ResultSet
- Policy defined by queryPattern→Policy
- 3 policies
  - EagerCaching: variable granularities for invalidations
  - RelaxedCaching: invalidations based on timeout
  - NoCaching: never cached

RUBiS bidding mix with 450 clients	No cache	Coherent cache	Relaxed cache
Throughput (rq/min)	3892	4184	4215
Avg response time	801 ms	284 ms	134 ms
Database CPU load	100%	85%	20%
C-JDBC CPU load	-	15%	7%

Configuring C-JDBC as a Derby driver (1/3)



- copy `c-jdbc-driver.jar` in client application classpath

```
Class.forName("org.objectweb.cjdbc.driver.Driver");
Connection c = DriverManager.getConnection(
  "jdbc:cjdbc://host/db", "login", "password");
```

- copy `derby.jar` in `$CJDBC_HOME/drivers`

Configuring C-JDBC as a Derby driver (2/3)

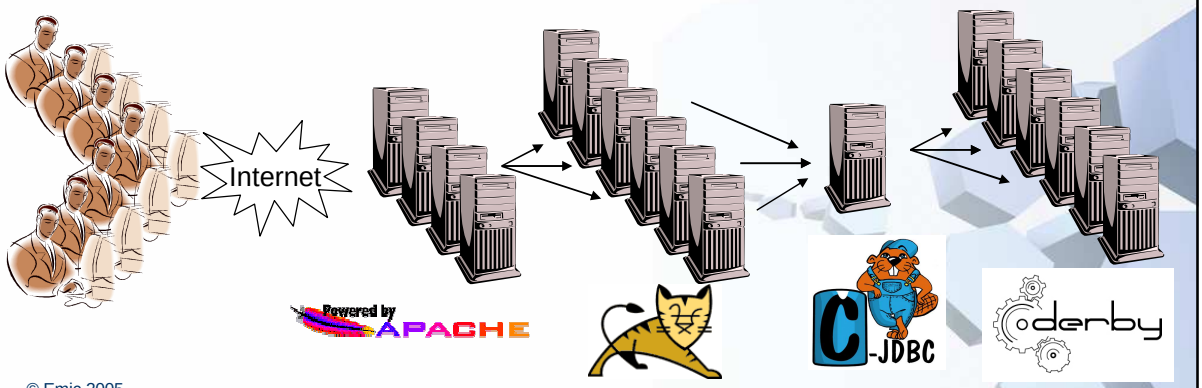
```
<?xml version="1.0" encoding="UTF8"?>
<!DOCTYPE C-JDBC PUBLIC "-//ObjectWeb//DTD C-JDBC 1.1//EN" …>
<C-JDBC>
    <VirtualDatabase name="xpetstore">
      <AuthenticationManager>
        <Admin> <User username="admin" password=""/> </Admin>
        <VirtualUsers> <VirtualLogin vLogin="user" vPassword="x"/> </VirtualUsers>
    </AuthenticationManager>

    <DatabaseBackend name="derby"
            driver="org.apache.derby.jdbc.EmbeddedDriver"
            url="jdbc:derby:c:/xpetstore;create=true"
            connectionTestStatement="values 1">
      <ConnectionManager vLogin="user" rLogin="APP" rPassword="APP">
        <VariablePoolConnectionManager initPoolSize="1" maxPoolSize="50"/>
      </ConnectionManager>
    </DatabaseBackend>
```

Configuring C-JDBC as a Derby driver (3/3)

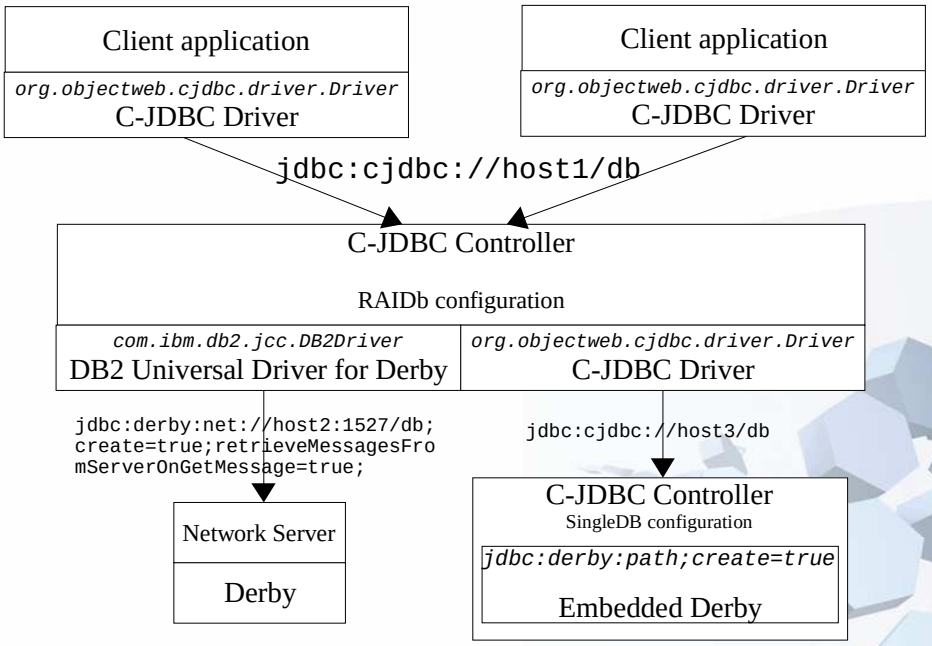
```
    <RequestManager>
            <RequestScheduler>
              <SingleDBScheduler level="passThrough"/>
            </RequestScheduler>

            <RequestCache>
              <MetadataCache/>
              <ParsingCache/>
              <ResultCache granularity="table"/>
            </RequestCache>

            <LoadBalancer>
              <SingleDB/>
            </LoadBalancer>
    </RequestManager>
  </VirtualDatabase>
</C-JDBC>
```

Highly available web site

- Multiple databases
  - choosing RAIDb level
  - recovery log for
    - adding nodes dynamically
    - recovering from failures



Derby clustering with C-JDBC



Configuring C-JDBC with Derby Network server

- Virtual database configuration file

```
<DatabaseBackend name="derby1"
 driver="com.ibm.db2.jcc.DB2Driver"
 url=
   "jdbc:derby:net://localhost:1527/xpetstore;create=true;retr
   ieveMessagesFromServerOnGetMessage=true;"
 connectionTestStatement="values 1"> <ConnectionManager …/>
</DatabaseBackend>
```

Configuring C-JDBC with Derby/C-JDBC

- Virtual database configuration file

```
<DatabaseBackend name="derby2"
 driver="org.objectweb.cjdbc.driver.Driver"
 url="jdbc:cjdbc://host/xpetstore"
 connectionTestStatement="values 1"> <ConnectionManager …/>
</DatabaseBackend>
```

Configuring C-JDBC Clustering with Derby (1/2)

```
<RequestManager>
  <RequestScheduler>
    <RAIDb-1Scheduler level="passThrough"/>
  </RequestScheduler>

  <RequestCache>
    <MetadataCache/>
    <ParsingCache/>
    <ResultCache granularity="table" />
  </RequestCache>

  <LoadBalancer>
    <RAIDb-1>
      <RAIDb-1-LeastPendingRequestFirst/>
    </RAIDb-1>
  </LoadBalancer>
```

Configuring C-JDBC Clustering with Derby (2/2)

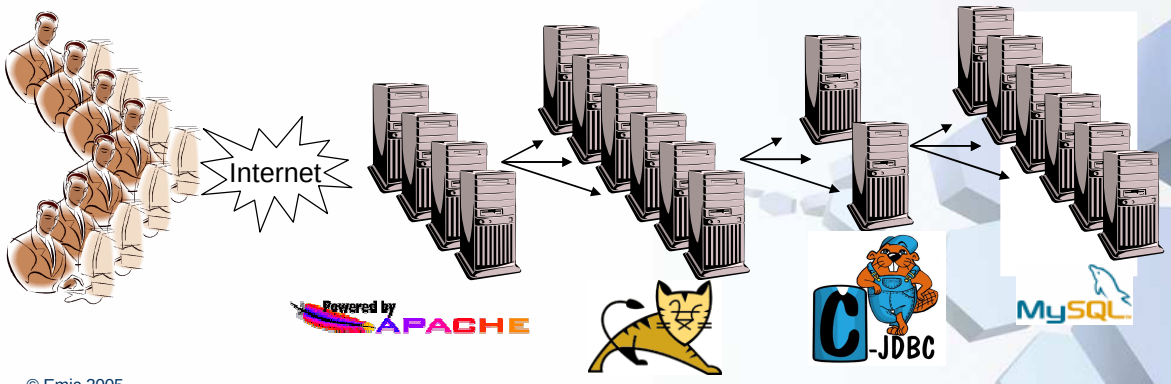
```
    <RecoveryLog>
      <JDBCRecoveryLog
            driver="com.ibm.db2.jcc.DB2Driver“
            url="jdbc:derby:net://localhost:1529/xpetstore;
            create=true;retrieveMessagesFromServerOnGetMessage=true;"
            login="APP" password="APP">
        <RecoveryLogTable tableName="RECOVERY"
            idColumnType="BIGINT NOT NULL" sqlColumnName="sqlStmt“
            sqlColumnType="VARCHAR(8192) NOT NULL“
            extraStatementDefinition=",PRIMARY KEY (id)"/>
        <CheckpointTable tableName="CHECKPOINT"/>
        <BackendTable tableName="BACKENDTABLE"/>
        <DumpaTable tableName=“DUMPTABLE”/>
      </JDBCRecoveryLog>
    </RecoveryLog>
  </RequestManager>
</VirtualDatabase>
```

Controller replication

<VirtualDatabase name="myDB">

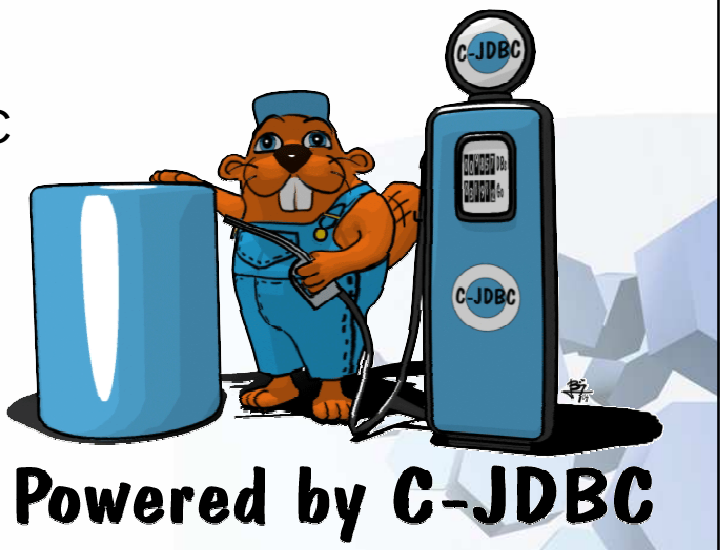
<Distribution/>

…



Outline

- RAIDb
- C-JDBC
- Derby and C-JDBC
- Scalability
- High availability

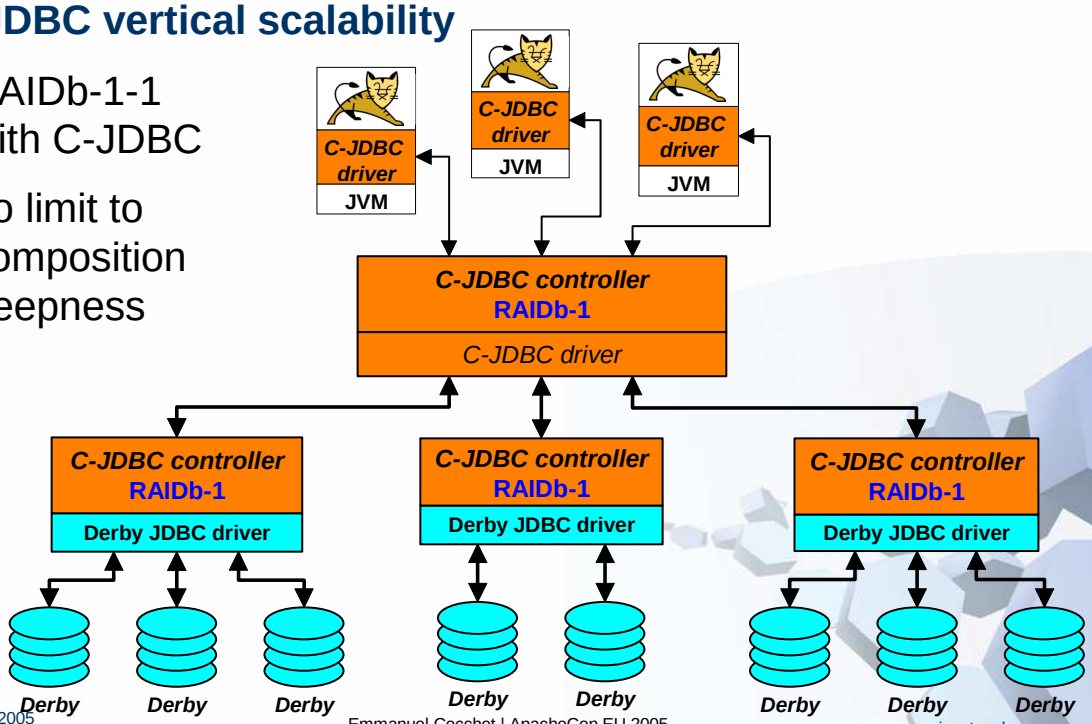


C-JDBC vertical scalability

- allows nested RAIDb levels
- allows tree architecture for scalable write broadcast
- necessary with large number of backends
- C-JDBC driver re-injected in C-JDBC controller

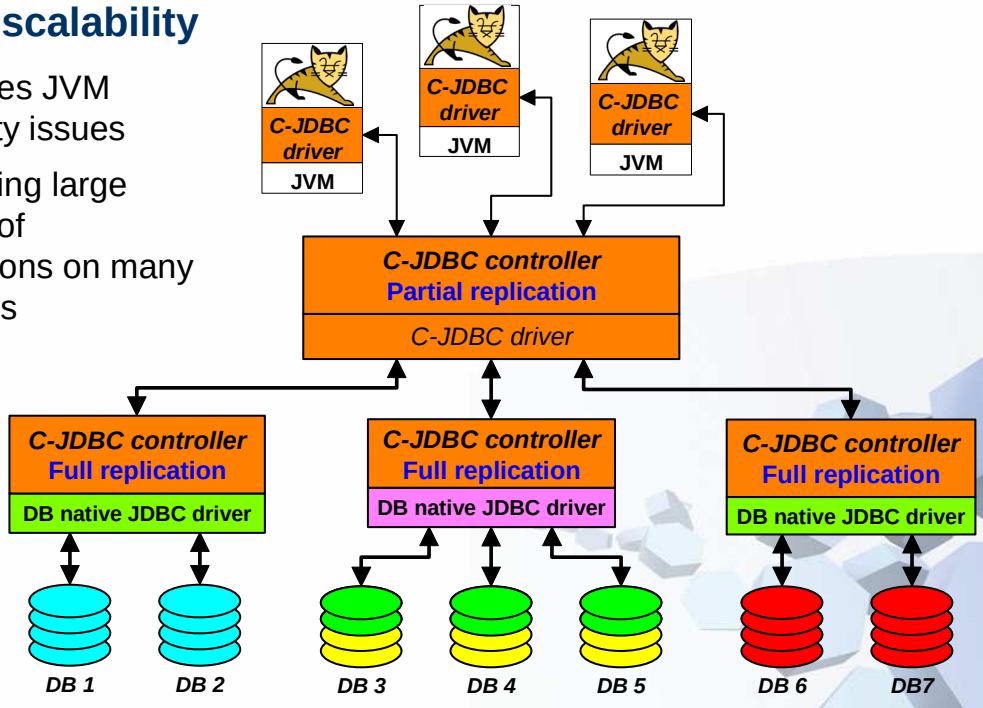
C-JDBC vertical scalability

- RAIDb-1-1 with C-JDBC
- no limit to composition deepness



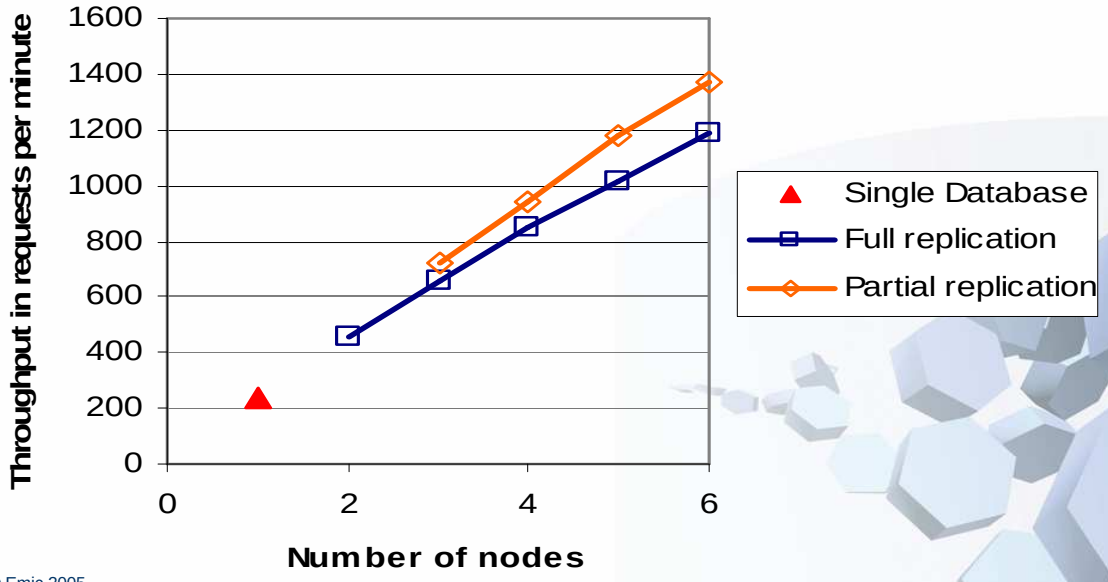
Vertical scalability

- Addresses JVM scalability issues
- Distributing large number of connections on many backends



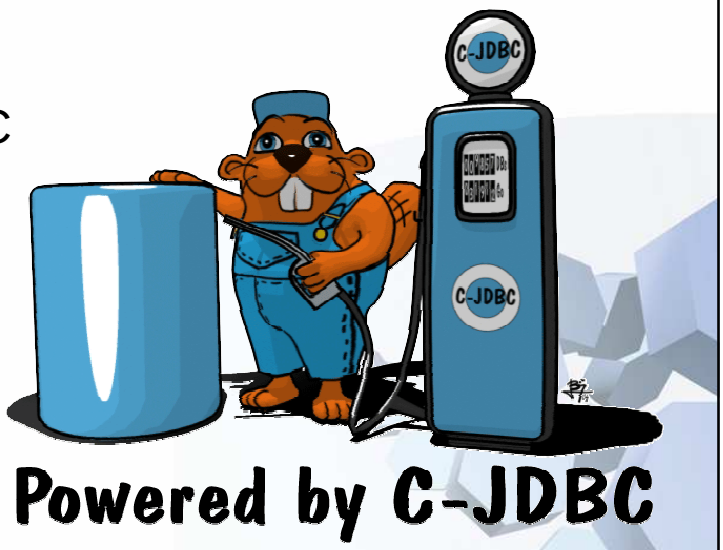
TPC-W benchmark (Amazon.com)

- Nearly linear speedups with the shopping mix



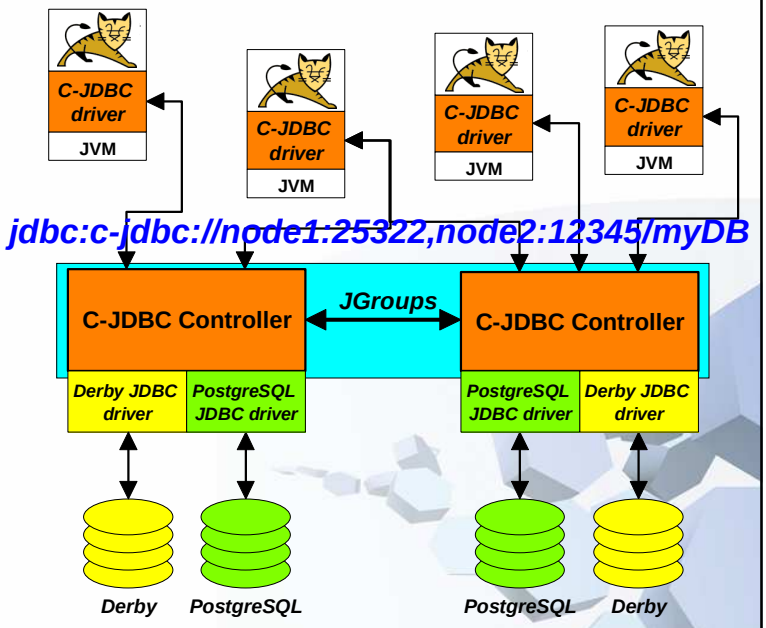
Outline

- RAIDb
- C-JDBC
- Derby and C-JDBC
- Scalability
- High availability



Controller replication

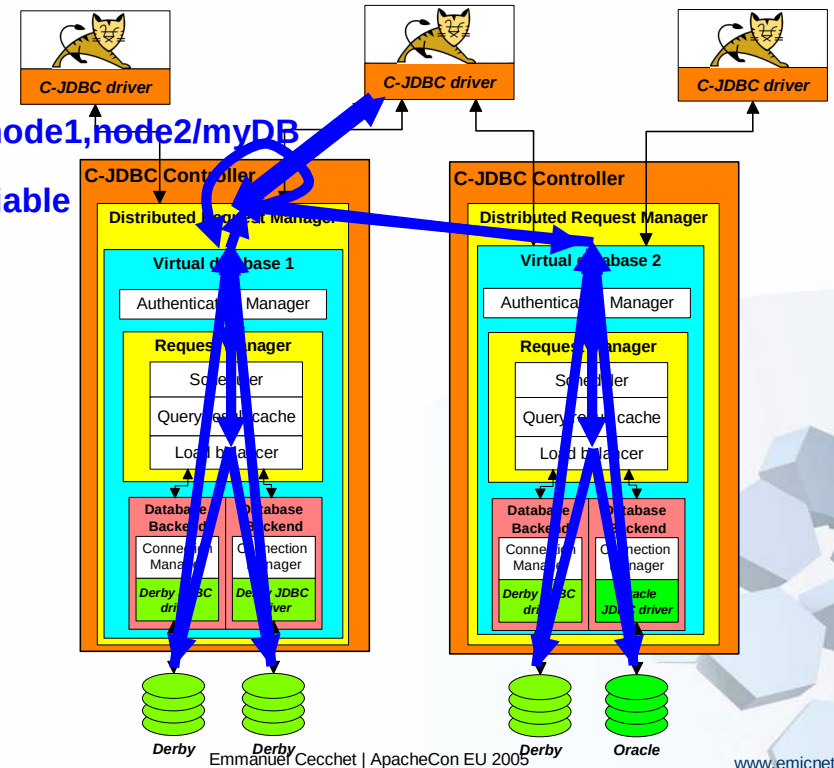
- Prevent the controller from being a single point of failure
- Group communication for controller synchronization
- C-JDBC driver supports multiple controllers with automatic failover



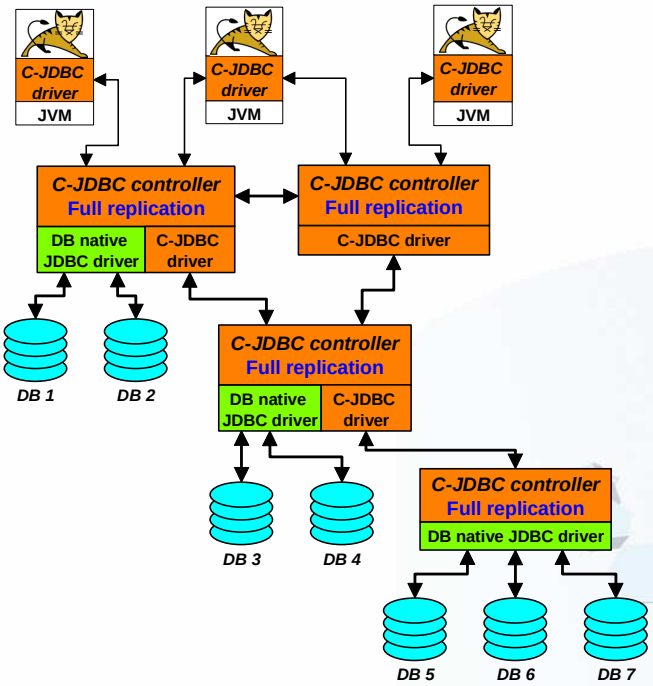
Controller replication

jdbc:cjdbc://node1,node2/myDB

Total order reliable multicast

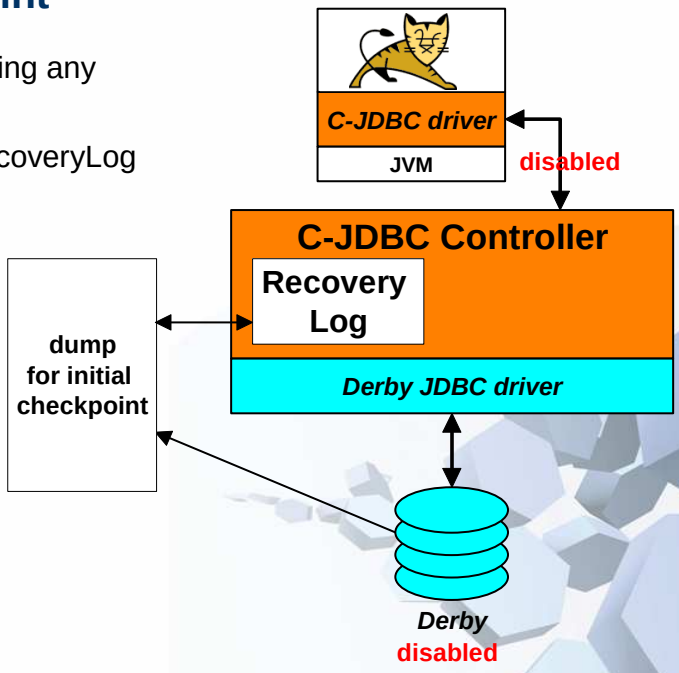


Mixing horizontal & vertical scalability



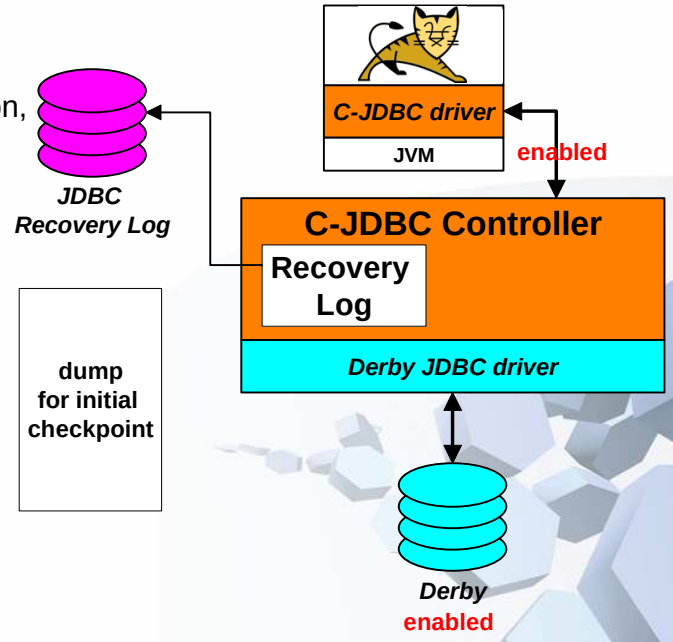
Building initial checkpoint

- Dump initial Derby database using any tools (tar, zip, …)
- Initial checkpoint inserted in RecoveryLog



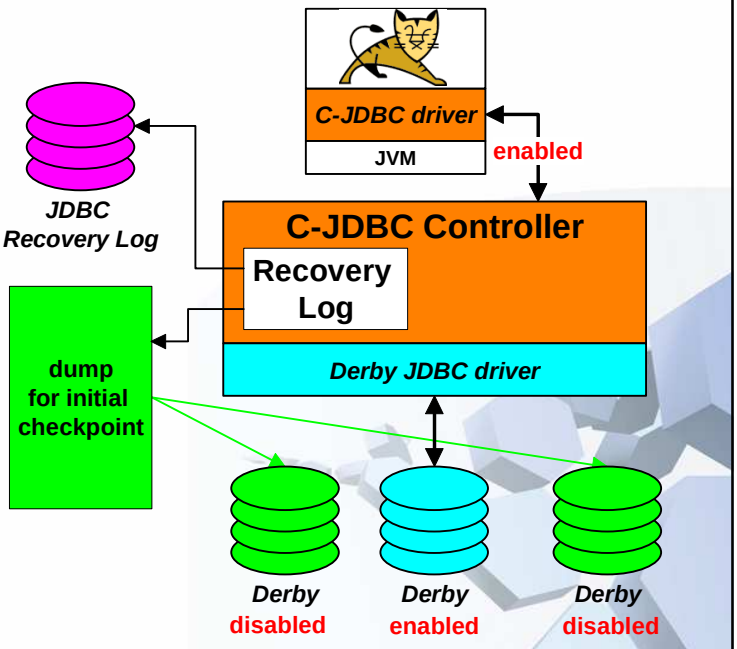
Logging

- Backend is enabled
- All database updates are logged (SQL statement, user, transaction, …)



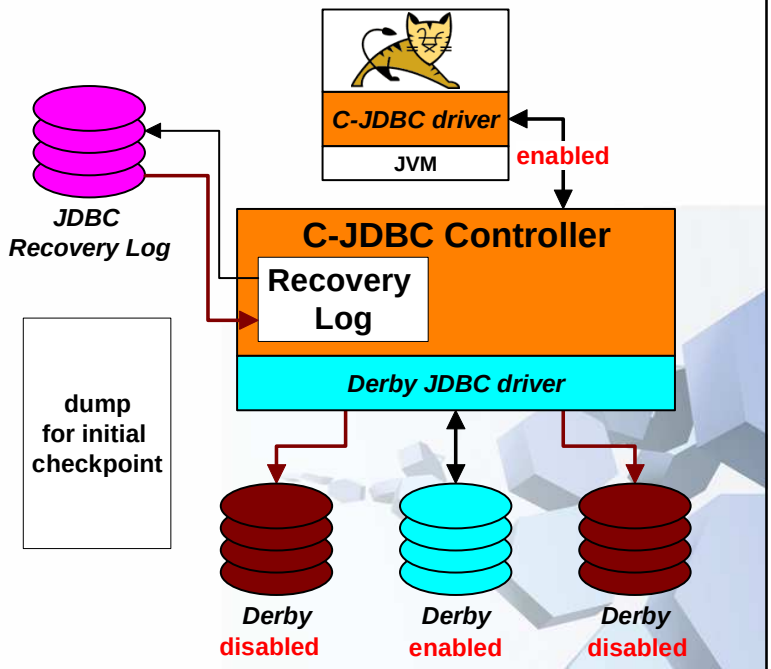
Adding new backends 1/3

- Add new backends while system online
- Restore dump corresponding to initial checkpoint



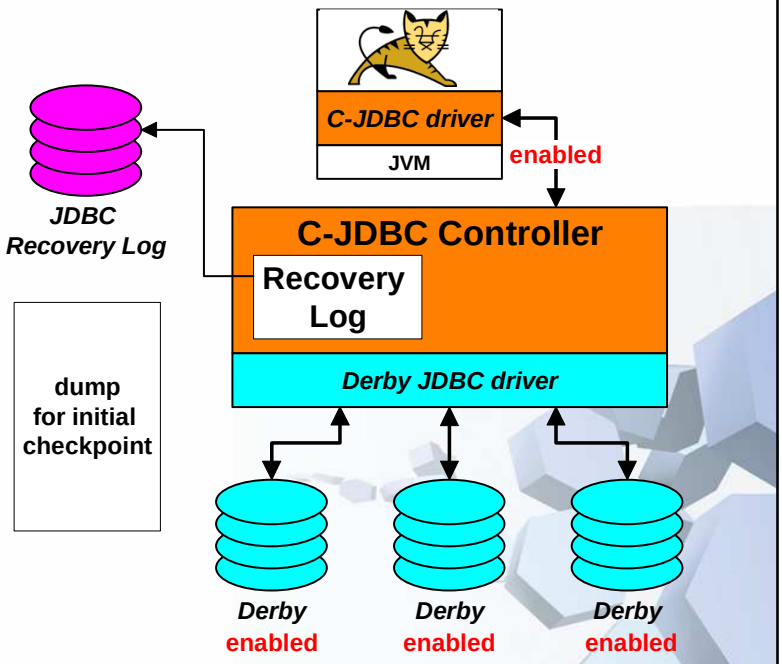
Adding new backends 2/3

- Replay updates from the log



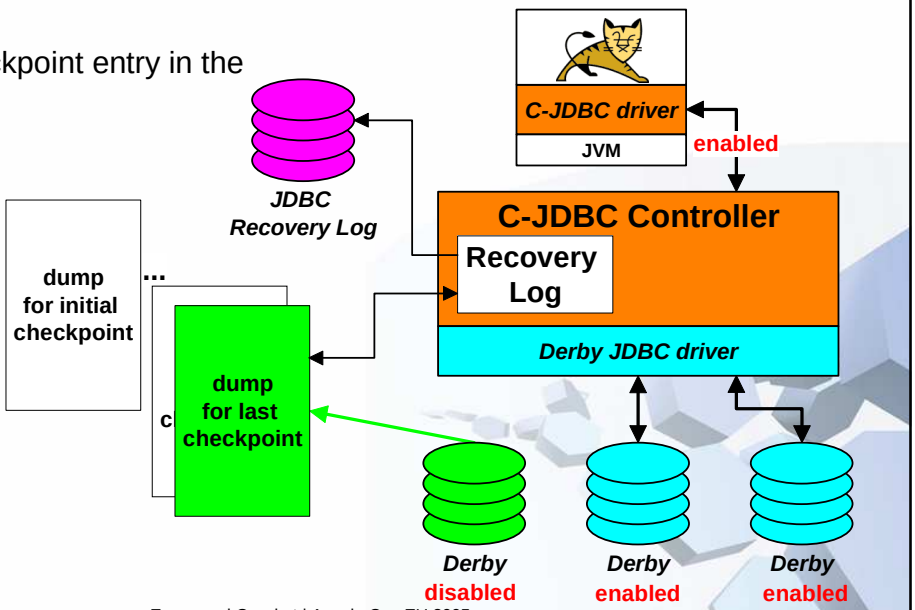
Adding new backends 3/3

- Enable backends when done



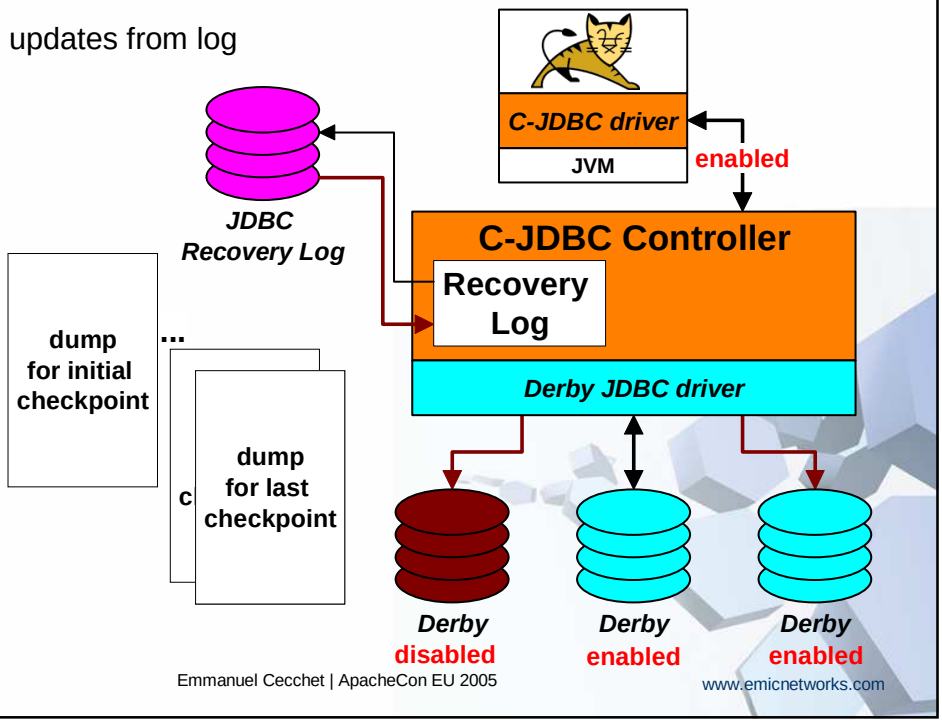
Making new checkpoints (1/3)

- Disable one backend to have a coherent snapshot
- Mark the new checkpoint entry in the log
- Dump with tar/zip



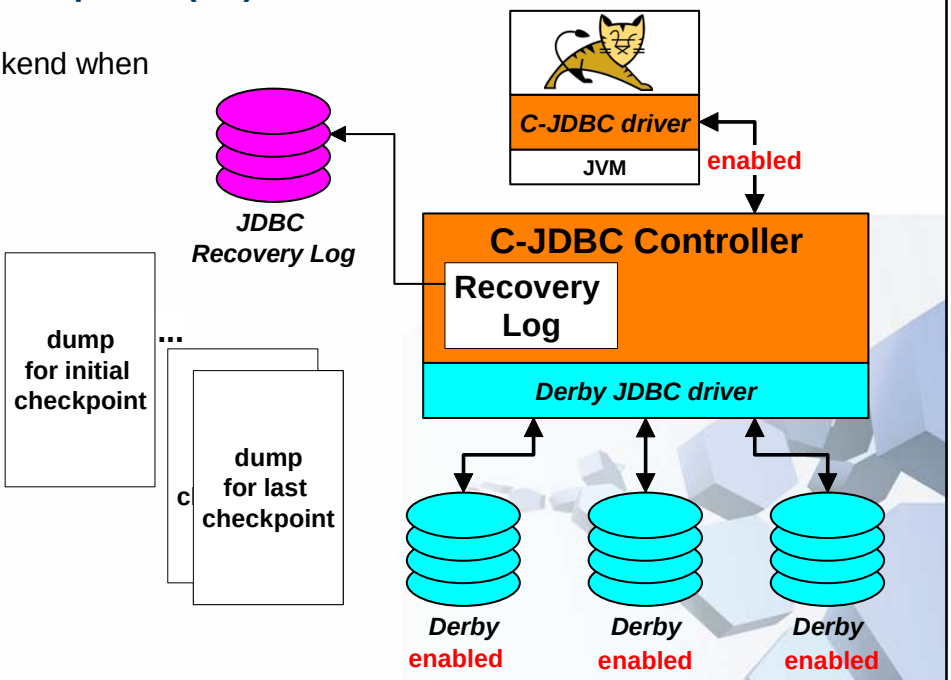
Making new checkpoints (2/3)

- Replay missing updates from log



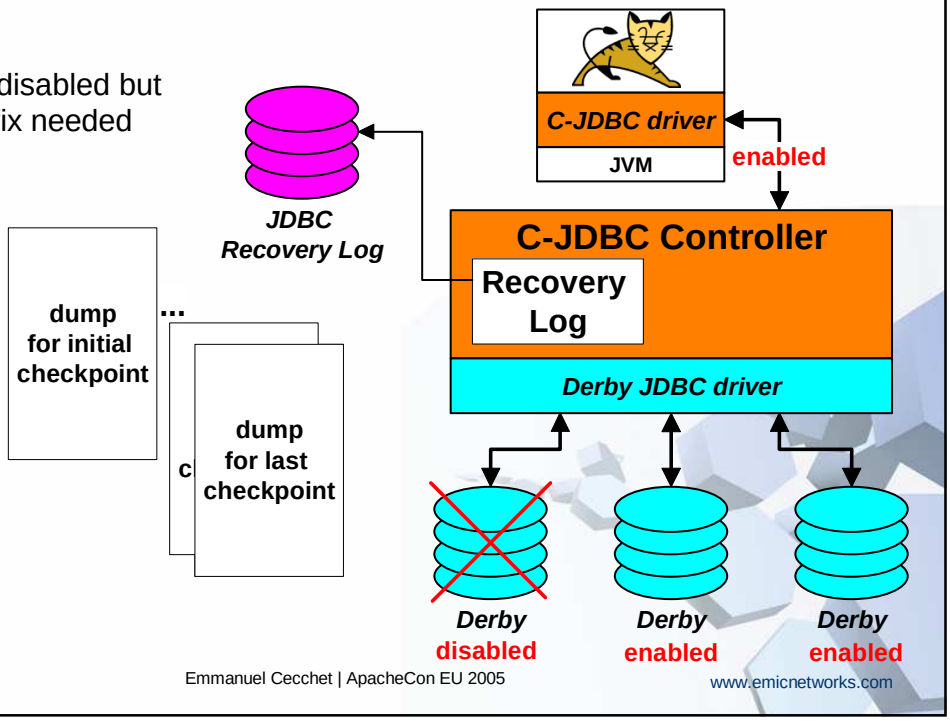
Making new checkpoints (3/3)

- Re-enable backend when done



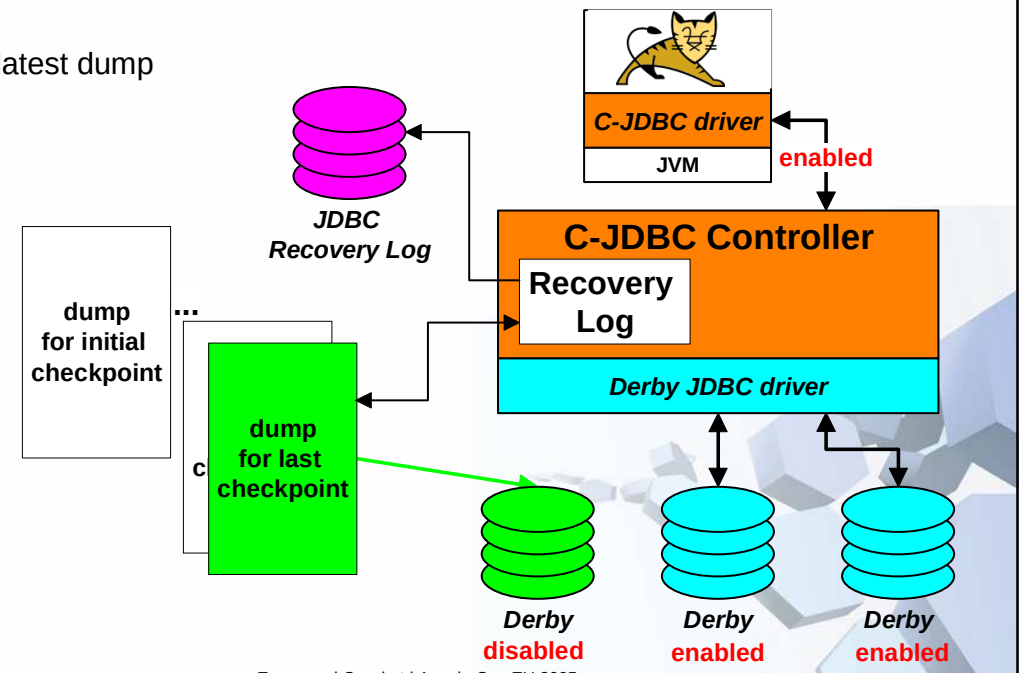
Handling failures

- A node fails!
- Automatically disabled but administrator fix needed



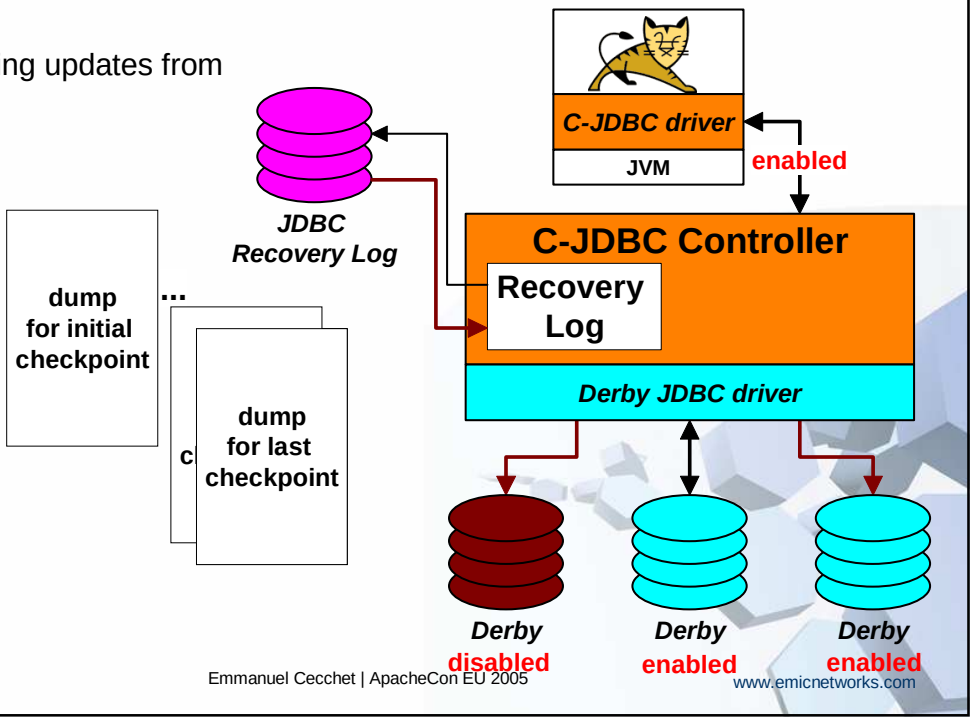
Recovery 1/3

- Restore latest dump



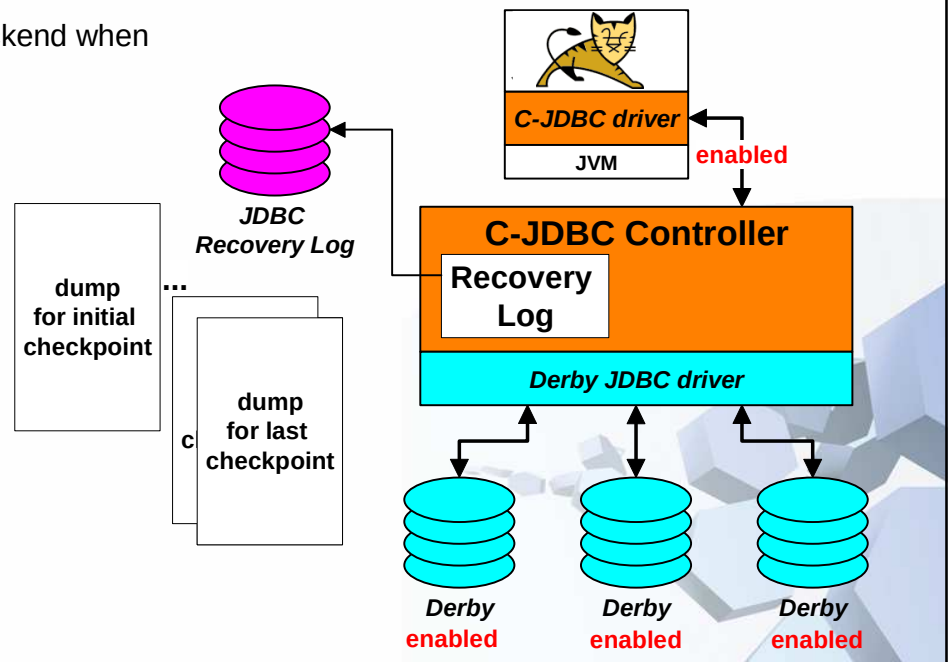
Recovery 2/3

- Replay missing updates from log



Recovery 3/3

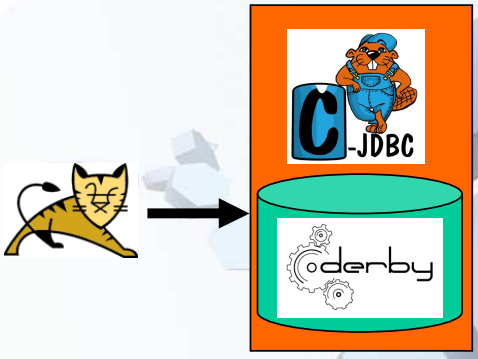
- Re-enable backend when done



Demo – xPetstore/Derby

- http://xpetstore.sourceforge.net/
  - open source implementation of Petstore
  - servlet version
- C-JDBC used as a driver for Derby remote access

xPetstore Servlet



Demo 2 – xPetstore/C-JDBC/Derby

xpetstore Servlet

Recovery log

embedded

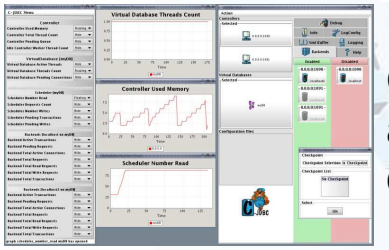
backend1

RAIDb-1

embedded

backend2

JMX



C-JDBC administration console

C-JDBC today

- Web site
  - ~200.000 hits/month
  - ~ 44.000 downloads
- Community
  - 27 committers both industrial & academics
  - c-jdbc@objectweb.org: ~300 subscribers, 200-300 msgs/month
  - translation in japanese, italian, german, chinese, turkish, french
- RPM on JPackage.org

Current limitations

- JDBC only
- Distributed joins
- Out parameters for stored procedures
- Some JDBC 3.0 extensions
- XA support through XAPool only
- network partition/reconciliation not supported

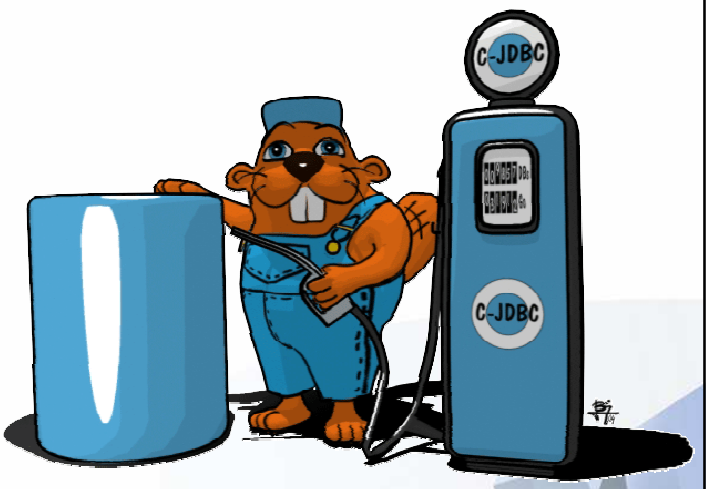
Conclusion

- RAIDb
  - RAID-like scheme for databases
- C-JDBC
  - open source middleware for database replication
  - performance scalability
  - high availability
- Derby & C-JDBC
  - open source driver for Derby
  - high availability solution for Derby

emic
Mission Critical Open Source

Q&A

Thanks to all users and contributors ...



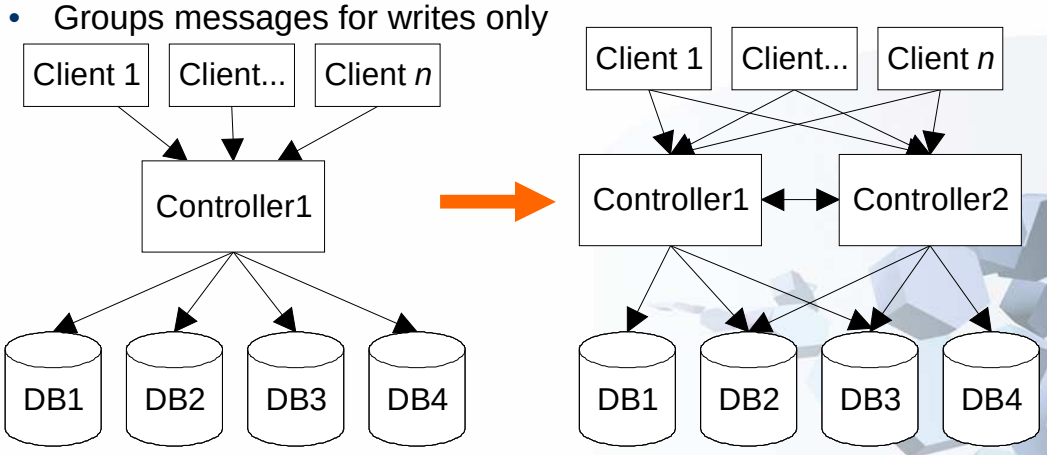
http://c-jdbc.objectweb.org

Bonus slides

HORIZONTAL SCALABILITY

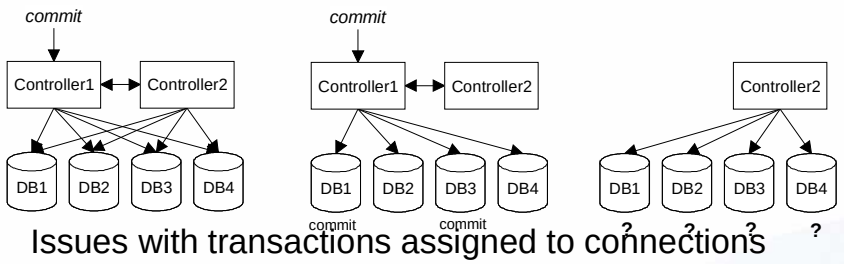
Horizontal scalability

- JGroups for controller synchronization
- Groups messages for writes only

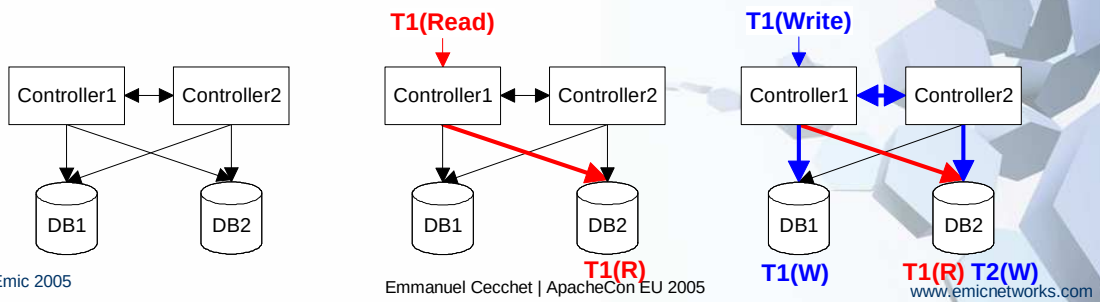


Horizontal scalability

- Centralized write approach issues

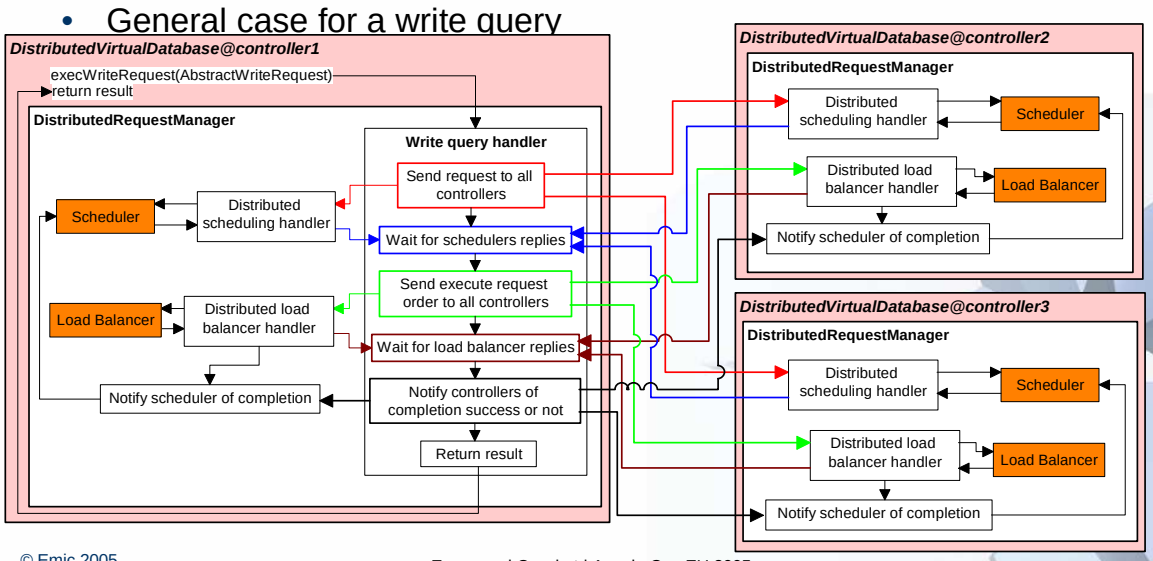


- Issues with transactions assigned to connections



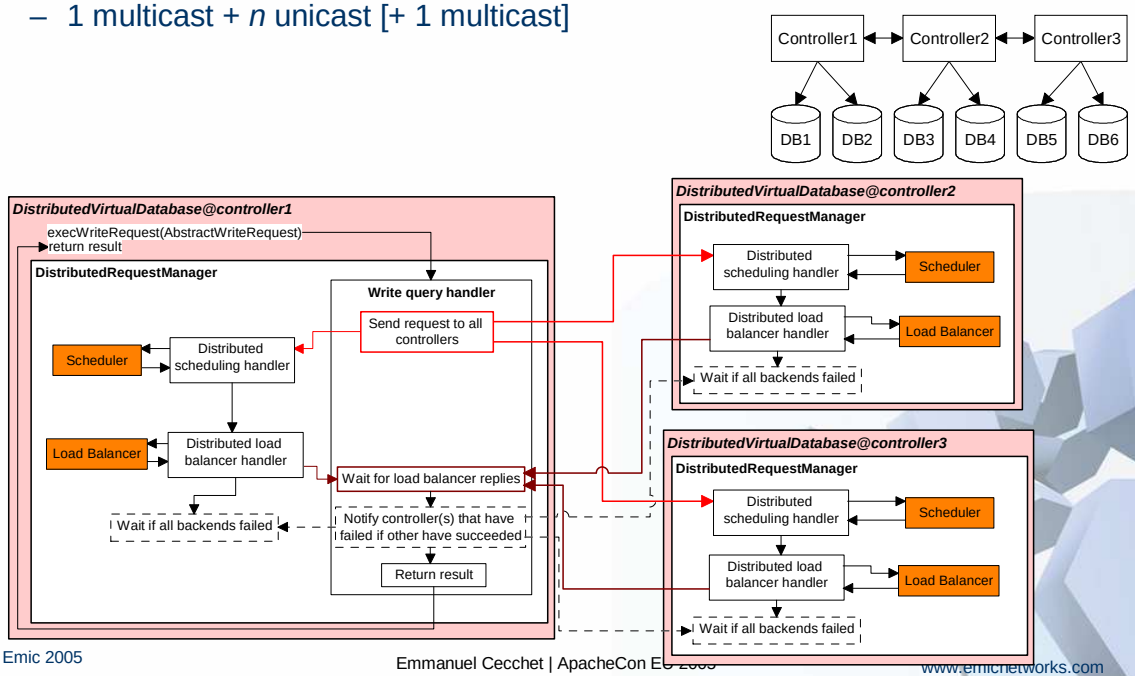
Horizontal scalability

- General case for a write query



Horizontal scalability

- Solution: No backend sharing
  - 1 multicast + *n* unicast [+ 1 multicast]

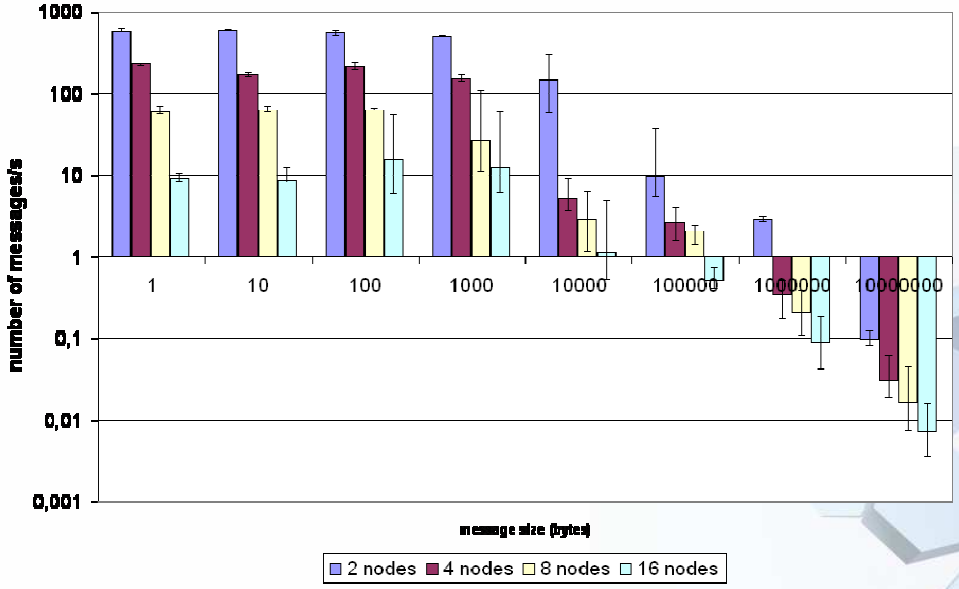


Horizontal scalability

- Issues with JGroups
  - resources needed by a channel
  - instability of throughput with UDP
  - performance scalability
- TCP better than UDP but
  - unable to disable reliability on top of TCP
  - unable to disable garbage collection
  - ordering implementation is sub-optimal
- Need for a new group communication layer optimized for cluster

Horizontal scalability

- JGroups performance on UDP/FastEthernet

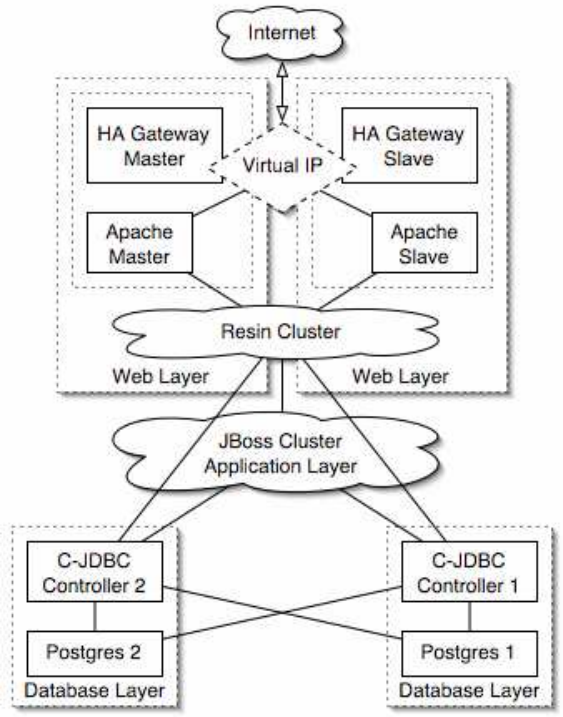


emic
Mission Critical Open Source

USE CASES

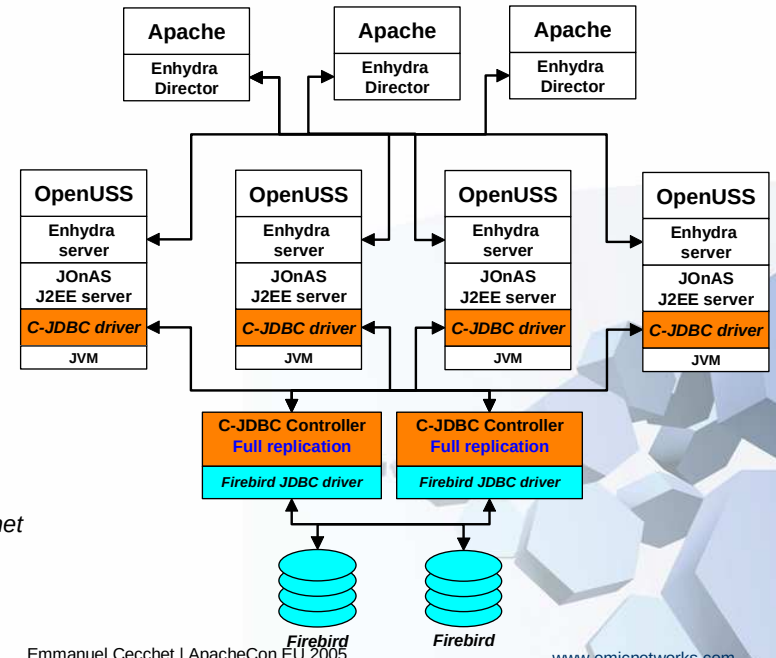
Budget High Availability

- High availability infrastructure "on a budget"
- Typical eCommerce setup
- *http://www.budget-ha.com*



OpenUSS: University Support System

- eLearning
- High availability
- Portability
  - Linux, HP-UX, Windows
  - InterBase, Firebird, PostgreSQL, HypersonicSQL



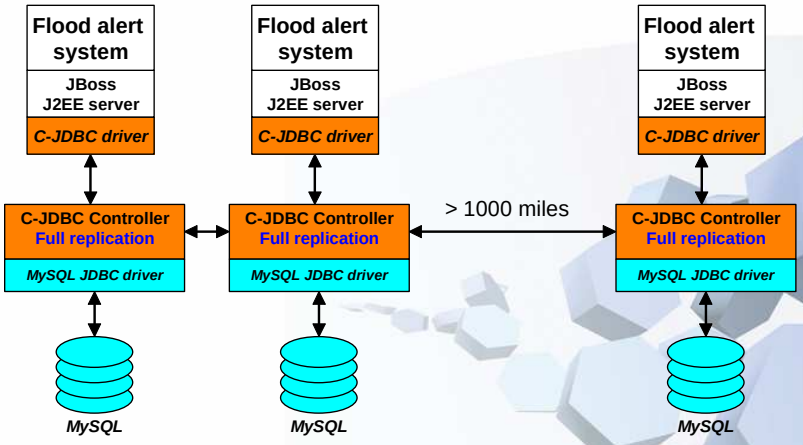
- *http://openuss.sourceforge.net*

.Flood alert system

Disaster recovery

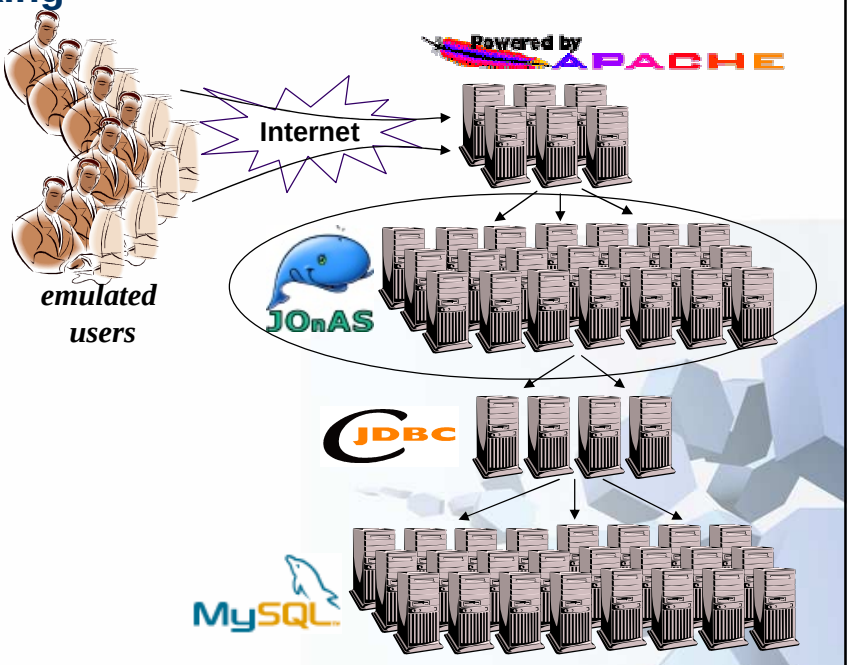
- Independent nodes synchronized with C-JDBC
- VPN for security issues

- *http://floodalert.org*



J2EE benchmarking

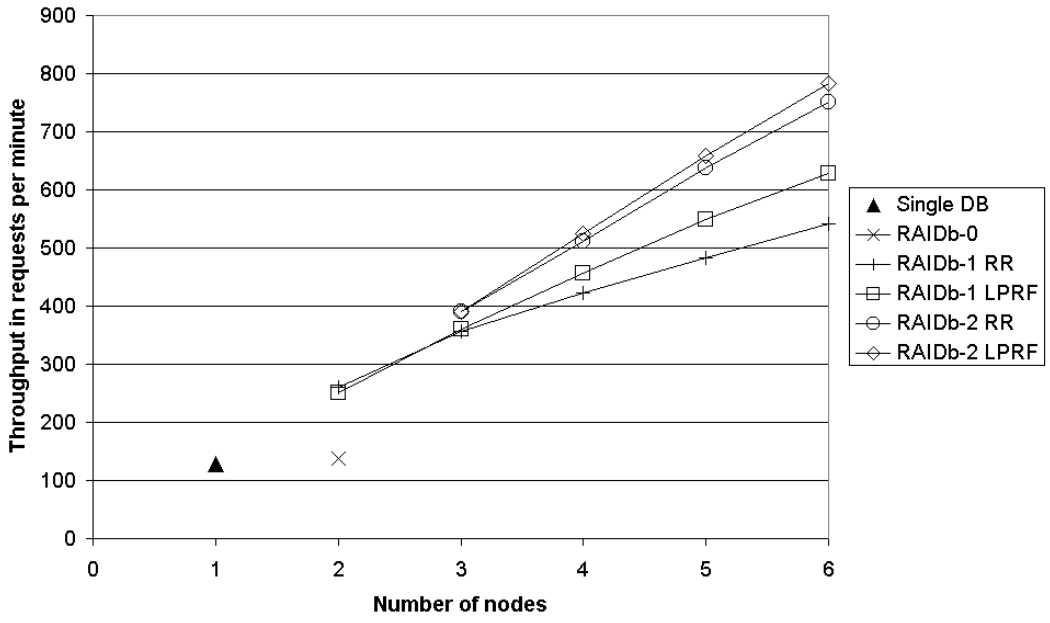
- Large scale J2EE clusters
- *http://jmob.objectweb.org*



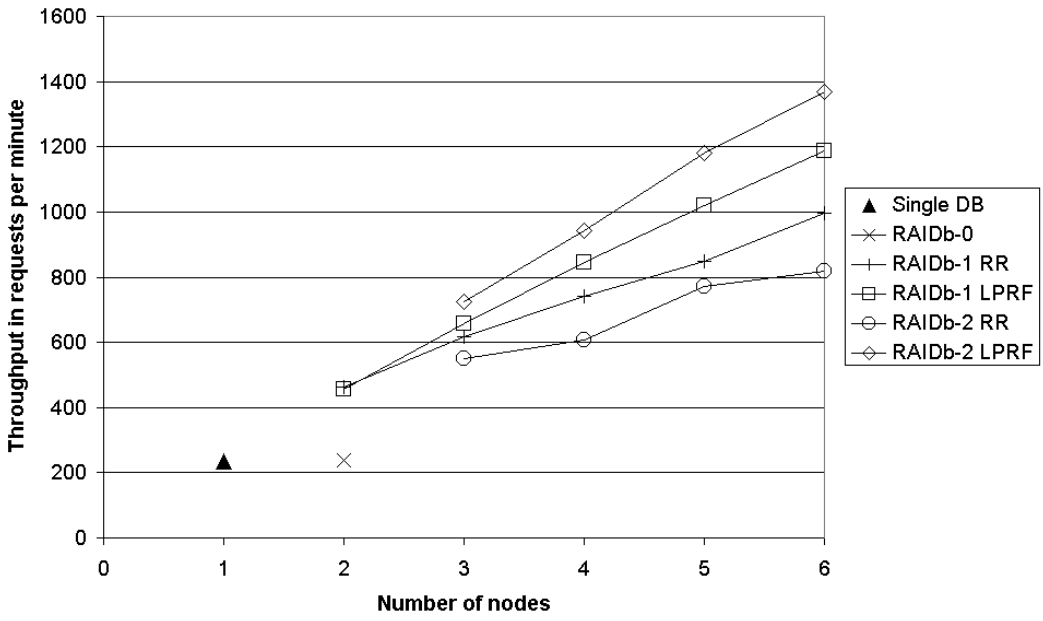
PERFORMANCE

emic
Mission Critical Open Source

.TPC-W Browsing mix performance



.TPC-W Shopping mix performance



.TPC-W Ordering mix performance