Sequoia is an open source software (Apache v2 License) hosted by Continuent.org. Sequoia is the continuation of the C-JDBC project hosted by the ObjectWeb consortium (http://c-jdbc.objectweb.org).
Motivations

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

- Clients connect to the application server
- Application server builds web pages with data coming from the database
- Application server clustering solves application server failure
- Database outage causes overall system outage
Disk replication/clustering

- Eliminates the single point of failure (SPOF) on the disk
- Disk failure does not cause database outage
- Database outage problem still not solved
Master/slave replication

- Lazy replication at the disk or database level
- No scalability
- Data lost at failure time
- System outage during failover to slave
- Failover can take several hours
Scaling the database tier – Master-slave replication

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

MySQL replication uses a master/slave replication scheme. Writes must be directed to the master node and updates are lazily replicated to slave nodes.
Scaling the database tier – Atomic broadcast

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

Queries are sent atomically in the same total order to all replicas. Optimizations may include executing the write query at one node and propagate only the modified rows to the other nodes. This requires heavy modifications of the database engine and makes locking management very tedious.
Scaling the database tier – SMP

- Cons
  - Cost
  - Scalability limit

This is the most common solution in commercial setups: large SMP machine and prohibitive DB license costs
Database clustering with shared disk

- Multiple database instances share the same disk
- Disk can be replicated to prevent SPOF on disk
- No dynamic load balancing
- Database failure not transparent to users (partial outage)
- Failed instance leaves incorrect data in the disks
- Manual failover + manual cleanup needed
Sequoia competitive advantages

• High availability with fully transparent failover
• Scalability with dynamic load balancing
• Commodity hardware
• No modification to existing client application
• Database vendor independent
• Heterogeneity support
• Platform independent
Outline

• RAIDb
• Sequoia
• Geronimo/Sequoia/Derby
• 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 is to databases what RAID is to disks.
RAIDb levels

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

SQL requests

RAIDb controller

- table 1
- table 2 & 3
- table ...
- table n-1
- table n
**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

![Diagram showing RAIDb controller and SQL requests](diagram.png)
RAIDb levels composition

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

- RAIDb
- Sequoia
- Geronimo/Sequoia/Derby
- Scalability
- High availability
Sequoia overview

- Middleware implementing RAIDb
  - 100% Java implementation
  - open source (Apache v2 License)
- Two components
  - generic JDBC driver (Sequoia driver)
  - Sequoia Controller
- Read-one, Write all approach
  - provides eager (strong) consistency
- Supports heterogeneous databases
Sequoia can be seen as a JDBC proxy between the application and the database native driver.
Sequoia is the continuation of the C-JDBC project (http://c-jdbc.objectweb.org). This slide shows the main differences between Sequoia and C-JDBC.

<table>
<thead>
<tr>
<th></th>
<th>C-JDBC</th>
<th>Sequoia</th>
</tr>
</thead>
<tbody>
<tr>
<td>License</td>
<td>LGPL</td>
<td>APL v2</td>
</tr>
<tr>
<td>Bug tracking</td>
<td>GForge</td>
<td>JIRA</td>
</tr>
<tr>
<td>Hosting</td>
<td>ObjectWeb</td>
<td>Continuent.org</td>
</tr>
<tr>
<td>Write scalability</td>
<td>Single threaded</td>
<td>Multi threaded</td>
</tr>
<tr>
<td>Group communication</td>
<td>Tribe/JGroups</td>
<td>Hedera/JGroups</td>
</tr>
<tr>
<td>C/C++ support</td>
<td>-</td>
<td>Carob</td>
</tr>
<tr>
<td>Mgt console</td>
<td>Text</td>
<td>Text + Oak Eclipse plugin</td>
</tr>
</tbody>
</table>
The Sequoia controller is a Java application. It can be co-located with the driver or the database or run on a separate machine.
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
- Backends can be transferred between controllers
- 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
• Manages concurrency control
• Assigns unique id to requests and transactions
• Ensure serializable transactional order
• Relies on database (row-level) locking
Request cache

- 3 optional caches
  - tunable sizes
- parsing cache
  - parse request skeleton only once
  - \texttt{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
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

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

Eager caching accepts database, table, column and column unique granularities. Basically, every request is parsed and C-JDBC checks which tables or rows are needed to compute the result. When a write queries is executed, it invalidates all results that required the data that has been invalidated. The finer grain invalidation requires more extensive parsing (more cpu) but have better hit ratio.

Relaxed caching can further enhance the cache performance by relaxing the consistency of queries that can accept a given staleness (for example a page whose content is only refreshed every minute).
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

RAIDb-0 is for partitioning (tables distributed on the different backends).
RAIDb-1 is for full mirroring.
RAIDb-2 is for partial replication.
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
• Sequoia 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
- Checkpoints are associated with database dumps
- Record all updates and transaction markers since a checkpoint
- Used to resynchronize a database from a checkpoint
- Store log information in a database
- Can be re-injected in a Sequoia cluster for fault tolerance
Functional overview

- Connection establishment
- Read request
- Write request
- Failure handling
- Recovery
- Summary
Connection establishment

- Sequoia JDBC URL
  - `jdbc:sequoia://controller1[:port1],controller2[:port2]/vdbname`
- Driver connection to a controller according to a policy defined in the URL (random, round-robin, ordered, …)
- Controller connections to a physical database use a connection pool defined per backend
- All queries on a driver connection go to the same controller but controller load balance queries on the underlying backends
The connection is first authenticated by the authentication manager. Then SELECT requests are scheduled, checked again the cache (a cache hit immediately returns) and then given to the load balancer that chooses a backend to execute the query. Upon completion, the cache is updated and the result is returned to the client.
Write request

- Query broadcast using total order between controllers
- Backends execute query in parallel (possibly asynchronously)
- Automatically disable backends that fail if others succeed
- Update recovery log (transactional log) asynchronously
- Non-conflicting transactions execute in parallel
  - table-based locking used as a hint for detecting conflicts
  - conflicting queries execute in a single-threaded queue
  - stored procedures execute one at a time
The distributed request manager wraps the regular (centralized) virtual database by sending queries in the same total order at all controllers. Queries are then execute locally as if we were in a centralized case. Upon completion, an acknowledgement message is sent to the initiator of the request.
Sequoia failure handling (1/2)

• Failure of a connection to a controller
  – transparently reconnected to another controller (according to user defined policy)
  – failure inside a transaction restores the transactional context upon reconnection

• backend failure
  – backend removed from load balancer (no noticeable failover time)
  – failure during read: retry on another backend
  – failure during write: ignored if successful on other backends
Sequoia failure handling (2/2)

- controller failure
  - database backends attached to dead controller are lost
  - clients are transparently reconnected to another controller (according to user defined policy)
  - failure during read: transparently retried (even within a transaction)
  - failure during write: exception thrown to the application telling that the write might have been executed but we cannot be sure
If a node fails to execute an update query but other backends succeed, the failing backend is automatically disabled.
Outline

- RAIDb
- Sequoia
- Geronimo/Sequoia/Derby
- Scalability
- High availability
Even with a single database backend, C-JDBC can significantly improve performance with its various caches that prevent requests from going to the database. This is well-suite for eCommerce applications that have read-mostly workloads.
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

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

Eager caching accepts database, table, column and column unique granularities. Basically, every request is parsed and C-JDBC checks which tables or rows are needed to compute the result. When a write query is executed, it invalidates all results that required the data that has been invalidated. The finer grain invalidation requires more extensive parsing (more CPU) but have better hit ratio.

Relaxed caching can further enhance the cache performance by relaxing the consistency of queries that can accept a given staleness (for example a page whose content is only refreshed every minute).
Highly available web applications

- choosing RAIDb level
- recovery log for
  - adding nodes dynamically
  - recovering from failures
Geronimo configuration

- Copy sequoia-driver.jar in sequoia/jars/sequoia-driver.jar

```xml
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector" version="1.5"
  configId="MyDatabase" parentId="org/apache/geronimo/Server">
  <dependency>
    <uri>
      sequoia/jars/sequoia-driver.jar
    </uri>
  </dependency>
  <resourceadapter>
    <name>PostgreSQLDataSource</name>
    <config-property-setting name="UserName">dbuser</config-property-setting>
    <config-property-setting name="Password">dbpw</config-property-setting>
    <config-property-setting name="Driver">org.continuent.sequoia.Driver</config-property-setting>
    <config-property-setting name="ConnectionURL">jdbc:sequoia://c1,c2/mydb</config-property-setting>
  </resourceadapter>
</connector>
```
Configuring Sequoia with Derby Network server

- Virtual database configuration file

```xml
<VirtualDatabase name="myDB">
  <Distribution>
    <MessageTimeouts/>
  </Distribution>
  ...
  <DatabaseBackend name="derby1"
       driver="org.apache.derby.jdbc.ClientDriver"
       url="jdbc:derby:net://localhost:1527/xpetstore;create=true;retrieveMessagesFromServerOnGetMessage=true;"
       connectionTestStatement="values 1">
  </DatabaseBackend>
</VirtualDatabase>
```
RAIDb-1 scheduler and load balancer ensures proper database replication.
Configuring Sequoia Clustering with Derby (2/2)

<RecoveryLog>
  <JDBCRecoveryLog>
    driver="org.apache.derby.jdbc.ClientDriver"
    url="jdbc:derby:net://localhost:1529/xpetstore;create=true;retrieveMessagesFromServerOnGetMessage=true;"
    login="APP" password="APP">
  </JDBCRecoveryLog>
  <RecoveryLogTable tableName="RECOVERY"
    idColumnType="BIGINT NOT NULL" sqlColumn="sqlStmt"
    sqlColumnType="VARCHAR(8192) NOT NULL"
    extraStatementDefinition="", PRIMARY KEY (id)="/>
  <CheckpointTable tableName="CHECKPOINT"></>
  <BackendTable tableName="BACKENDTABLE"></>
  <DumpTable tableName="DUMPTABLE"></>
</RecoveryLog>
</RequestManager>
</VirtualDatabase>

Emmanuel Cecchet | ApacheCon US 2005

Note that it is also possible to use an embedded Derby for the recovery log.
Outline

- RAIDb
- C-JDBC
- Geronimo/Sequoia/Derby
- Scalability
- High availability
Sequoia vertical scalability

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

Vertical Scalability

Mixed Horizontal and Vertical Scalability

© Continuent 2005

Emmanuel Cecchet | ApacheCon US 2005

www.continuent.org
TPC-W benchmark (Amazon.com)

- Nearly linear speedups with the shopping mix

TPC-W is a database benchmark that models an online bookstore like amazon.com. More information can be found at http://www.tpc.org
Outline

- RAIDb
- C-JDBC
- Geronimo/Sequoia/Derby
- Scalability
- High availability
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, ...)

```
<table>
<thead>
<tr>
<th>JVM</th>
<th>C-JDBC driver</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>Derby JDBC driver</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>Recovery Log</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>JDBC Recovery Log</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>dump for initial checkpoint</th>
</tr>
</thead>
</table>
```

Emmanuel Cecchet | ApacheCon US 2005

© Continuent 2005
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

```
<table>
<thead>
<tr>
<th>Adding new backends 3/3</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Enable backends when done</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Derived JDBC driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derby enabled</td>
</tr>
<tr>
<td>Derby enabled</td>
</tr>
<tr>
<td>Derby enabled</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>JDBC Recovery Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>dump for initial checkpoint</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>C-JDBC Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recovery Log</td>
</tr>
<tr>
<td>Derby JDBC driver</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>JVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th><a href="http://www.continuent.org">www.continuent.org</a> © Continuent 2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emmanuel Cecchet</td>
</tr>
<tr>
<td><a href="http://www.continuent.org">www.continuent.org</a></td>
</tr>
</tbody>
</table>
```
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

- Dump for initial checkpoint
- Dump for last checkpoint
- Dump for last checkpoint
- Dump for initial checkpoint

- derby enabled
- derby enabled
- derby enabled
- derby enabled

- recovery log enabled
- recovery log enabled
- recovery log enabled
- recovery log enabled

- C-JDBC Controller
- JVM
- C-JDBC driver
- derby enabled
Handling failures

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

dump for initial checkpoint

dump for last checkpoint

JDBC Recovery Log

C-JDBC Controller

C-JDBC driver

JVM

enabled

JDBC

Derby

enabled

disabled

Recovery Log

Derby JDBC driver

JOnAS, WebLogic, JBoss, WebSphere, ...

dump for last checkpoint

dump for initial checkpoint
Recovery 1/3

- Restore latest dump

-现象图展示了一个系统的恢复过程。图中包含一个C-JDBC Controller，它连接到JDBC Recovery Log和Derby JDBC driver。Derby JDBC driver有三个状态：disabled、enabled和enabled。图中还显示了dump for initial checkpoint和dump for last checkpoint。
Recovery 2/3

- Replay missing updates from log

- Dump for initial checkpoint
- Dump for last checkpoint

C-JDBC Controller
Recovery Log

JDBC Recovery Log

Derby JDBC driver

Derby enabled

Derby disabled

Derby enabled

C-JDBC driver

JVM

enabled
Recovery 3/3

- Re-enable backend when done
Current limitations

- 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

• Sequoia
  – open source middleware for database replication
  – performance scalability
  – high availability

• Geronimo/Sequoia/Derby
  – End-to-end open source highly available web applications
Q&A

Thanks to all users and contributors …

http://sequoia.continuent.org