Galera Replication

Clients

Transparent connections

Multi-master

Synchronous Replication

Galera Replication

DBMS

DBMS

DBMS
Pretty Good Scalability

- Sysbench
- OLTP mode
- Amazon EC2, large instances

MySQL 5.1.33

08/23/09 Codership @ OpenSQLCamp 2009
Contents

1. About Codership
2. Galera Overview
3. Advanced Features
4. Limitations
5. Benchmarks
6. Roadmap
About Me & Codership

- Me = Seppo Jaakola, CEO
- Friends = Alexey Yurchenko, Teemu Ollakka + hang around guys
- Fin-Rus community working from Finland
- Several years experience in building networking applications, distributed systems, clustering solutions, firewalls etc...
- We enjoy writing code, that's why we founded Codership
- Set Sails Oct 2007
Galera Replication

- Certification based replication model (based on academic research by F. Pedone et al)
- Multi-master synchronous replication  ➔ High Availability
- No middle-ware, connections directly to DBMS  ➔ Transparency
- Row level locking  ➔ Write scalability
Galera Replication

- Galera is implemented as software library
- Generic replication system to make a cluster from any transactional DBMS
- First implementation MySQL/Innodb cluster
  - More to come...
Galera Cluster

Clients

MySQl

wsrep

MySQl

wsrep

MySQl

wsrep

Transparent connections

Multi-master

Galera
True Multi master

➢ No restrictions for write access
➢ Galera sorts out write conflicts
➢ Client sees cluster node as standard DBMS server

G a l e r a  R e p l i c a t i o n

Client
Update table-A...
wsrep

Client
Update table-A...
wsrep

wsrep

wsrep
Synchronous Replication

Transaction is committed in all nodes => HA

Client

Commit

Transaction is committed in all nodes => HA
Certification Based Replication

Warning: Scientific content

- At commit time, a write set (ws) is extracted
- GCS assigns seqno for this ws and broadcasts to the cluster
- There can be a set of write sets in replication queue, which have not yet committed
- Certification test check conflicts against this set
- Ws contains all information to run the certification test deterministically in each cluster node
- Snapshot isolation
Certification Based Replication

MySQL
Query Processing
Commit Processing

write set population

WS extract

certification

commit
rollback

write set applier

certification

Total order broadcast
Replication

08/23/09
Codership @ OpenSQLCamp 2009
Generic Replication

First we take MySQL, then we take Postgres
wsrep API

- Defines a generic interface for DBMS and replication system
- Write set replication API for transactions
- DDL replication using TO isolation
- https://launchpad.net/wsrep
wsrep integration in MySQL

- Calls to wsrep provider:
  - ws populating,
  - replication...
- Handlers for various wsrep callbacks:
  - ws applying
  - DDL applying ...
- Changes in innodb code to provide prioritized transactions
- https://launchpad.net/codership-mysql
Write Set

- Contains all information needed for certifying and applying of transactions
- Data changes can be specified in several different replication levels:
  1. SQL statement
  2. Lex structures (AST) from parser
  3. RBR event
  4. Row (as binary image)
  5. Column value
## Write Set

- Sequence number of the trx
- Sequence number of the last committed trx, which affected the processing state
- All row changes are identified with keys
- SQL statements or RBR events

<table>
<thead>
<tr>
<th>Seqno</th>
<th>last_committed_seqno</th>
<th>Keys</th>
<th>Applying info</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Write Set

- By default write sets are flushed asap
- However, key information is stored in certification index
- Write sets can also be saved for future needs:
  - `wsrep_ws_persistency = ON/OFF`
  - For debugging purposes
  - For incremental state transfer
Recap

- ...learnt so far:
- Galera is true multi-master
- And synchronous replication
- Certification based replication model
- Provided by generic software library
- wsrep API defines the interface
- ...and then...
Advanced Replication Features

- Optimistic concurrency control
- Flow control
- Auto increment management
- Asymmetric lock granularity issue
- Parallel applying
- Retrying of aborted autocommit trxs
- etc…
Optimistic Concurrency Control

- Transactions proceed independently in each cluster node assuming they can eventually commit
- Certification test tells if trxs committing from different nodes conflict
- Victim trx must abort in master node, and avoid committing in other nodes
- WS applying happens with high priority and this can further abort local trxs
- ER_DEADLOCK returned for cluster aborts
Optimistic Concurrency Control

- Hot spots are bad
  - e.g. DBT2 shows pretty high conflict rate
- Long lasting transactions are vulnerable
- Rollbacks eat performance, but rollback happens only in one node, all the rest nodes just avoid applying
- With problematic SQL load, cluster could be adjusted to have a smaller number of write capable nodes
Flow Control

- Synchronous operation requires that each node will process evenly.
- Without any flow control some node(s) would hijack all master activity => slave queues would grow indefinitely.
- Group communication actions for managing flow control.
- Limits for slave queue length, which trigger flow control.
Autoincrements

- Galera can manage automatically autoinc control variables (auto_increment_increment, auto_increment_offset)
- This happens optionally: wsrep_autoinc_control = ON/OFF
- Autoincrement control is triggered by cluster membership changes:
  - Increment = number of cluster nodes
  - Offset = node id
- Autoincrement control can yield best possible insert performance
Asymmetric Lock Granularity

- Write set defines modified rows by unique key values
- When applying the write set, DBMS may need to lock more “resources” (like InnoDB gap locks)
- Applying several write sets in parallel can lead to conflicts
Maximal Insert Performance
(theoretical thought content)

- Even faster inserts?
- Due to autoincrement management, inserts won't conflict
  - Transactions with only inserts (...and which get auto inc value) don't need to certify
  - We can/could skip certification test and process inserts even faster
retrying

- "Cluster can abort trxs at will"
- If autocommit trx was aborted due to cluster wide deadlock, it is safe to immediately retry the trx
- Option: wsrep_retryAutocommit = ON/OFF
Parallel Applying

- It is possible to launch several appliers
  - `wsrep_slaves = n`, option
- Appliers run in parallel until commit time
  But: no performance gain has been observed
    - RBR event applying is very fast compared to SQL query processing
    - Galera cluster scales even with 100% write rate
    - Flow control cuts some of parallelism
- (Implementation suffers from ALG issue)
Connecting

- Galera is true multi-master, clients can directly connect to any node
- Also connection pool can know the node addresses and balance connections to the cluster
- If single connection point is needed:
  - TCP load balancer is viable option (like Galera load balancer, glb)
  - Proxy component (MySQL Proxy)
Connecting

Client

DBMS

galera

Client

DBMS

DBMS

galera

Client

MySQL Proxy

DBMS

DBMS

galera
DB State Transfer

- To join new node in running cluster
- Copies DB state from an active node to the joiner
- The task is:
  - Donor must prepare a DB snapshot in known position of replication stream (seqno)
- DB State options (for innodb):
  - Mysqldump release 0.7
  - Xtrabackup release 0.8
  - LVM release 0.8
  - InnoDB HotBackup
DB State Transfer

Joiner

Donor

Active

Sst request

Galera Replication

seqno
SST in 0.7

joiner

SST request

Assign donor

mysqld

SST complete

Joined cluster

Synchronized with cluster

group

SST complete

Joined cluster

Synchronized with cluster

donor

SST request

mysqld

Stop at seqno and perform SST

SST (mysqldump)
Limitations
Twist in my Transparency

- Commit can return ER_DEADLOCK
- Implicit snapshot isolation
- SQL level limitations:
  - Locking session
  - Load data infile not supported (yet)
  - Lock functions
Lock Tables?

- No!
- Locking sessions require managing long term session to each node
  - Complicated and error prone
Load Data?

- “Load data infile” requires that each server has identical file in place
  - Not very practical in cluster
- “Load data local”, possible to implement but:
  - is not yet supported
Myisam support?

- Can be supported, if myisam writes will go through one selected node
  - Myisam support will fall back to master slave replication (synchronous)
- Galera can reject myisam writes in any other node
Large Write Sets

TODO list

• If transaction modifies a lot of rows, the write set size can grow too large to handle
  ➔ Escalate to page/table level locking
  ➔ Escalate to SQL replication

• Long running transactions
  ➔ Limit the number of write capable nodes
Benchmarking
Benchmarking

- Tested with several benchmarks
  - Sysbench, dbt2, DOTS, osdb, jmeter, sqlgen...
- Benchmarks testing with 'physical hardware' and with Amazon EC2 small and large instances
- Currently tests only up to 5 cluster nodes
- In general, shows good scalability even with write intensive work loads
Sysbench Benchmarks

- Sysbench oltp mode test
- EC2 Large instances

<table>
<thead>
<tr>
<th>nodes</th>
<th>users</th>
<th>trx/s</th>
<th>deadlks</th>
<th>95%lat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>385</td>
<td>0</td>
<td>0.092</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>761</td>
<td>2.54</td>
<td>0.100</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>900</td>
<td>3.42</td>
<td>0.103</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>1034</td>
<td>4.54</td>
<td>0.120</td>
</tr>
</tbody>
</table>

official 5.1.33 binary:
<table>
<thead>
<tr>
<th>nodes</th>
<th>users</th>
<th>trx/s</th>
<th>deadlks</th>
<th>95%lat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>451</td>
<td>0</td>
<td>0.079</td>
</tr>
</tbody>
</table>
Dbt2 Benchmark

- EC2 large instances
- Dbt2 benchmark
- 60 warehouses

<table>
<thead>
<tr>
<th>Conns</th>
<th>NOTPM</th>
<th>Rollbacks(%)</th>
<th>TRX duration(sec)</th>
<th>Dump load(min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain 5.1.30:</td>
<td>20</td>
<td>~7220</td>
<td>1</td>
<td>2.27</td>
</tr>
<tr>
<td>1 node</td>
<td>12</td>
<td>~7420</td>
<td>1</td>
<td>2.17</td>
</tr>
<tr>
<td>2 nodes</td>
<td>24</td>
<td>~9630</td>
<td>3</td>
<td>1.63</td>
</tr>
<tr>
<td>3 nodes</td>
<td>36</td>
<td>~10555</td>
<td>4</td>
<td>1.41</td>
</tr>
<tr>
<td>4 nodes</td>
<td>48</td>
<td>~10753</td>
<td>5</td>
<td>1.32</td>
</tr>
</tbody>
</table>
Drupal Scale-Out

- Proof of concept
- Each Drupal node has local MySQL
- All nodes identical
- ~10% of CPU for MySQL
- glb load balancer
The Test

• Jmeter thread groups:
  – Posters write new pages
  – Commenters read pages and add comments
  – Browsers read pages and comments
• Threads created in proportion: 4/12/24
• Write intensive work load
Issues

• Drupal Cache must be off
• 'files' directory contents
  → use network file system
  → store all files in DB
• Concurrent login failures
• Autoinc insert bug:
  http://drupal.org/node/282555
  – Sorted out by a workaround to retry failed autoinc insert
Results

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Users</th>
<th>Throughput (req/min)</th>
<th>Latency (ms, median)</th>
<th>Latency (ms, average)</th>
<th>Errors (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>180</td>
<td>724</td>
<td>1203</td>
<td>1827</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>360</td>
<td>1436</td>
<td>1190</td>
<td>1829</td>
<td>0.03</td>
</tr>
<tr>
<td>3</td>
<td>540</td>
<td>2091</td>
<td>1280</td>
<td>2150</td>
<td>0.06</td>
</tr>
<tr>
<td>4</td>
<td>720</td>
<td>2717</td>
<td>1214</td>
<td>2330</td>
<td>0.12</td>
</tr>
</tbody>
</table>

• http://www.codership.com/content/scaling-drupal-stack-galera-part-2-mystery-failed-login
The Way to Do It

Large Drupal farm

Small Galera cluster

Drupal

MySQL

Galera

MySQL

Galera

MySQL

Galera

memcached

glb load balancer
Sqlgen r/w: 50/50

MySQL 5.1.28

50 tables / 1000 rows

<table>
<thead>
<tr>
<th>R/W</th>
<th>SELECTs</th>
<th>UPDATEs</th>
<th>TRANSACT</th>
<th>abort%</th>
<th>CONNECTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain</td>
<td>1.0</td>
<td>1458.06</td>
<td>1461.39</td>
<td>583.75</td>
<td>0.0</td>
</tr>
<tr>
<td>1 node</td>
<td>1.0</td>
<td>1104.51</td>
<td>1104.43</td>
<td>441.24</td>
<td>0.0</td>
</tr>
<tr>
<td>2 nodes</td>
<td>1.0</td>
<td>1796.16</td>
<td>1796.39</td>
<td>718.60</td>
<td>0.0</td>
</tr>
<tr>
<td>3 nodes</td>
<td>1.0</td>
<td>2297.58</td>
<td>2298.43</td>
<td>919.54</td>
<td>0.1</td>
</tr>
<tr>
<td>4 nodes</td>
<td>1.0</td>
<td>2698.50</td>
<td>2696.27</td>
<td>1077.75</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Sqlgen r/w: 0/100

<table>
<thead>
<tr>
<th>R/W</th>
<th>SELECTs</th>
<th>UPDATEs</th>
<th>TRANSACT</th>
<th>abort%</th>
<th>CONNECTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain</td>
<td>0.0</td>
<td>2841.50</td>
<td>568.09</td>
<td>0.0</td>
<td>37.94</td>
</tr>
<tr>
<td>1 node</td>
<td>0.0</td>
<td>2002.69</td>
<td>400.27</td>
<td>0.0</td>
<td>26.64</td>
</tr>
<tr>
<td>2 nodes</td>
<td>0.0</td>
<td>3077.22</td>
<td>614.61</td>
<td>0.1</td>
<td>41.08</td>
</tr>
<tr>
<td>3 nodes</td>
<td>0.0</td>
<td>3773.47</td>
<td>752.58</td>
<td>0.3</td>
<td>50.14</td>
</tr>
<tr>
<td>4 nodes</td>
<td>0.0</td>
<td>4253.30</td>
<td>846.76</td>
<td>0.5</td>
<td>56.51</td>
</tr>
</tbody>
</table>

MySQL 5.1.28
Project Status
State of the Code

- Public releases since Jan 2009
- Current release 0.6.2
  - Good for benchmarking & evaluating
  - Lacks “node join” feature
  - Only vsbes group communication
- Planned releases 0.7, 0.8, 09 and 1.0
- Release 0.7 is fully open source and is targeted for Sep 2009
Roadmap

- **Release 0.7 - Stability Milestone**
  - Node join capability
  - Open Source
  - Fault tolerant GCS
- **Release 0.8 – Optimization Milestone**
  - Incremental State Transfer
  - Xtrabackup, LVM
- **Release 0.9 - Security Milestone**
  - TLS tunneling
- **Release 1.0 - Management Milestone**
  - Management tools
Reaching Out

- wsrep integration code should be maintained by DBMS provider
- MariaDB integration to happen in near future
- Drizzle replication framework is shaping up
- PostgreSQL support
  - We need a partner for this
  - Postgres-R by Markus Wanner
- Any other transactional engine?
  - Check out wsrep API
  - Snapshot isolation
  - Prioritized transactions
Summary

- Certification based replication turns out effective
  - High Availability
  - Transparency
  - Good scalability even with high write rates
- wsrep API is “not too hard” to implement
- Any (transactional) DBMS can leverage this replication possibility
Get in Touch!

- Downloads available: http://www.codership.com
- R&D consulting services
- Evaluation support
- info@codership.com