Region Server

View Manager

Client

Region Server

View Manager

View Manager

View Manager

Region Server

Region Server

View Table

Region Server

Base Table

1. Update Processing

RS 1

Base Table

WAL

VM 1

RS 2

View Table

WAL

Client

Commit Log

1. Client issues update
2. Update is written to WAL (later commited to base table)
3. Process on R1 gets notfied and checks hash ring for view manager
4. Process sends update to corresponding View Manager
5. View Manager processes update
   1. VM evaluates views
   2. VM analyses load
   3. VM queries view table
   4. VM queries base table
   5. VM calculates result
   6. VM inserts Update in View Table
   7. VM writes Update to commit log
6. Status Reports

RS 1

Base Table

WAL

VM 2

Commit Log

Master

VM 1

Commit Log

1. VM sends status report to RS
2. RS aggregates and sends status report to Master
3. Master 🡪 performs load balancing

3. Add Region Server

RS 2

Base Table

WAL

ZooKeeper

* RegionServer
  + RS 2
* ViewManager
  + VM1
  + VM2

Master

1. Region Server boots / Hashring-Process is started as Coprocessor
2. Region Server registers at Zookeeper
3. Master gets informed 🡪 Perform load balancing

4. Add View Manager

ZooKeeper

* RegionServer
  + RS 2
* ViewManager
  + VM1
  + VM2
  + VM3

Master

VM 3

Commit Log

1. VM registers at ZooKeeper
2. Master is informed about new VM 3
3. Master 🡪 Perform load balancing (VM)

5. Assign View Manager

RS 1

Base Table

WAL

ZooKeeper

* RegionServer
  + RS 2
* ViewManager
  + VM1
  + VM2
  + VM3

Master

VM 3

Commit Log

1. VM enters ASSIGNING Condition
   1. sends assignVM message to RS [RS == null && !shutdown]
2. RS assigns View Manager
   1. Adds Message Queue
   2. Starts sending Thread(Updates)
   3. adds VM to Hashring
   4. replies with VMassinged message
3. VM confirms assignment
   1. Sets Region Server to RS
   2. sends VMassigned message to master enters RUNNING
4. Master confirms assignment
   1. receives VMassigned message
   2. puts relation to zookeeper

6. Withdraw View Manager

RS 1

Base Table

WAL

ZooKeeper

* RegionServer
  + RS 2
* ViewManager
  + VM1
  + VM2
  + VM3

Master

VM 3

Commit Log

1. VM enters WITHDRAWING Condition
   1. VM sends withdrawVM message to RS
2. RS withdraws View Manager
   1. Removes VM from Hashring
   2. Stops sending Thread
   3. Removes Message Queue and reassigns Updates
   4. replies with VMwithdrawn message
3. VM confirms withdraw
   1. Processes remaining updates
   2. Sets Region Server to null
   3. Sends VMwithdrawn to master enters RUNNING
4. Master confirms withdraw
   1. receives VMwithdrawn message
   2. deletes relation from zookeeper

7. Reassign View Manager

ZooKeeper

* RegionServer
  + RS 2
* ViewManager
  + VM1
  + VM2
  + VM3

Master

VM 3

Commit Log

1. VM enters REASSIGNING Condition
   1. 🡪 Withdraw View Manager
   2. 🡪 Assign View Manager
2. VM sends VMreassigned to Master, enters RUNNING

8. Shutdown View Manager

ZooKeeper

* RegionServer
  + RS 2
* ViewManager
  + VM1
  + VM2
  + VM3

Master

VM 3

Commit Log

1. View Manager enters SHUTTING DOWN-Condition
2. 🡪 Withdraw View Manager
3. VM sends shutdownVM message to Master
4. Master
   1. Marks View Manager as deleted
   2. waits for Zookeeper node to expire
   3. 🡪 Perform load balancing

9. View Manager crash

RS 2

Base Table

WAL

ZooKeeper

* RegionServer
  + RS 2
* ViewManager

Master

VM 2

Commit Log

1. VM crashes
2. Master is notified via Zookeeper
3. Master collects Commit log and reads last entry
4. Master sends VMcrashed message with param lastEntry to RS
5. Region Servers replay Updates

10. Region Server crash

RS 1

Base Table

WAL

RS 2

Base Table

WAL

ZooKeeper

* RegionServer
  + RS 2
* ViewManager
  + VM1
  + VM2
  + VM3

Master

VM 3

Commit Log

VM 1

Commit Log

VM 2

Commit Log

1. Region Server 1 crashes
2. Master Server gets informed about crash of RS 1
3. Master Server reassigns Region to RS 2
4. Master Server reassgins VMs to RS 2
5. Master informs RS 2 about last commited updates

11. Perform load balancing

EventProcessor

VM 2

Load Balancer

Component Controller

Message Service

Recovery Manager

Zookeeper Service

VM 1

Dispatching

Send Commands / Await Response

Calculate Load   
Balancing

1. Master calculates Assignment plan
2. RS assigns View Manager
3. VM sends VMassigned message to master

Scenario 1: Add View Manager

RS 1

VM 3

VM 2

VM 1

RS 2

1. Search for Region Servers without VMs
   1. If RS without VM exist 🡪 add VM to it
   2. If RS without VM don’t exist 🡪 calculate load of RS, add Vm to heaviest loaded RS

Scenario 2: Add Region Server

RS 1

VM 2

VM 1

RS 2

1. Add Region Server
2. Search for RS with >= 2 VMs
   1. If RS with 2 VM exist 🡪 calculate load, reassign Vm from least loaded RS to new RS
   2. If RS with 2 VM don‘t exist 🡪 do nothing

Scenario 3 Remove/Crash View Manager

RS 1

VM 2

VM 1

RS 2

VM 3

1. Remove View Manager
2. Add Region Server

Scenario 4: Remove/Crash Region Server

RS 1

VM 2

VM 1

RS 2

VM 3

1. Remove Region Server
2. Loop through remaing VMs / Add View Manager

12. Perform recovery

EventProcessor

VM 2

Load Balancer

Component Controller

Message Service

Recovery Manager

Zookeeper Service

VM 1

Dispatching

Send Commands / Await Response

Calculate Load   
Balancing

Commit Log

Read CommitLog/  
Inform RS

13. WAL Reader

1. Master calculates Assignment plan
2. RS assigns View Manager
3. VM sends VMassigned message to master

Base Table

HDFS

WAL Reader

logs

HLogfile 1

HLogfile 2

oldlogs

HLogfile 3

HLogfile 4

Components

Region Server

RS 2

Base Table

WAL

Interface

1. Ingoing
   1. Put, Get, Delete
   2. addViewManager
   3. removeViewManager
   4. replayWriteAheadLog
2. Outgoing
   1. processUpdate

Master

Master

Interface

1. Ingoing
   1. callRegionServerRegistered
   2. callViewManagerRegistered
   3. callRegionServerRemoved
   4. callViewManagerRemoved
   5. callRegionServerCrashed
   6. callViewManagerCrashed
   7. callViewManagerStatusReport
2. Outgoing
   1. addViewManager
   2. assignViewManager
   3. removeViewManager
   4. deleteViewManager
   5. requestCommitLog
   6. replayWriteAheadLog

View Manager

View Manager

Commit Log

Update Processor

Interface

1. Ingoing
   1. callProcessUpdate
   2. callLastCommitedUpdate
2. Outgoing
   1. register
   2. updateViewTable
   3. sendStatusReport
   4. query Region Servers Put, Get Delete

Region Server

RS 2

Base Table

WAL

VM 1

VM 2

VM 3

Update Distributor

Client

Client

Client

HBasePort

Update Assigner

RSManager

ServerPort

Master

WAL Reader

RS-Interface

1. Ingoing
   1. Put, Get, Delete
   2. addViewManager
   3. removeViewManager
   4. replayWriteAheadLog
2. Outgoing
   1. processUpdate

RS - Messages

1. sendUpdate <baseTable, regionServer, seqNo, key, values>

RS - Components

1. Base table
2. WAL
3. Update Assigner(numVNodes)
   1. addViewManager
   2. removeViewManager
   3. assignUpdate
4. RSManager
   1. Controlling Interface
   2. replayLog -> openWAL
5. UpdateDistributor
   1. queueUpdate

View Manager

RS 1

VM 3

Commit Log

Server

VMManager

Server

Update Proc

Get View Record

Query BaseTable

Calculate Result

Update View

Client

Master

Pre Proc

Get Views

Analyze Load

Zookeeper

VM-Interface

1. Ingoing
   1. callProcessUpdate
   2. callLastCommitedUpdate
2. Outgoing
   1. register
   2. updateViewTable
   3. sendStatusReport
   4. query Region Servers Put, Get Delete

VM- Messages

1. sendStatusReport<viewmanager, vmname, operation, status<q1:amount;q2:amount;q3:amount>>
2. requestViewRecord get<key>
3. queryBaseTable get<key>
4. updateViewTable put<key, value>

VM - Components

1. Update Processor
   1. getViewRecord
   2. queryBaseTable
   3. calculateResult
   4. updateView
   5. CommitLog
2. VM-Manager
   1. register
   2. sendStatusReport 🡪 Thread
   3. retrieveLastCommitedUpdate

Master

RS 2

VM 3

Master

Load Balancer

Component Controller

Server

Client

Recovery Manager

VM 1

Event Processor

Zookeeper Client

Master-Interface

1. Ingoing
   1. callRegionServerRegistered
   2. callViewManagerRegistered
   3. callRegionServerRemoved
   4. callViewManagerRemoved
   5. callRegionServerCrashed
   6. callViewManagerCrashed
   7. callViewManagerStatusReport
2. Outgoing
   1. assignViewManager
   2. removeViewManager
   3. requestCommitLog
   4. replayWriteAheadLog
   5. getLastUpdate

Master- Messages

1. addViewManager <master, add, vmname>
2. removeViewManager <master, remove, vmname>
3. replayWAL<master, vmname, seqNo>
4. getLastUpdate<master, rsName>

Master - Components

1. Component-Controller
   1. assignViewManager
   2. removeViewManager
   3. replayWAL(ViewManager)
2. LoadBalancer
   1. invokeLoadBalancing
      1. VM added/removed / crashed
      2. RS added / removed /crashed
      3. VM overloaded
3. RecoveryManager
   1. handleViewManagerCrash
   2. handleRegionServerCrash

Non-idempodent view updates(SUM-View)

VM 1

RS 2

View Table

WAL

I

1. VM 1 receives Insert( k1,x1,100)
2. VM 1 commits Insert( k1,x1,100) 🡪
3. VM 1 crashes
4. VM 1 comes up again commits Insert( k1,x1,100) a second time
5. lastValue = 200 should be 100

no convergence because the same update was commited twice

Update interference[1](MIN-View)

RS 1

Base Table

WAL

VM 1

RS 2

View Table

WAL

D

I

U

1. RS 1 Delete( k1,x1,400)
2. VM 1 asks for new Minimum
3. RS 1 Insert (k1,x1,200)
4. RS 1 Update (k1,x1,500)
5. VM 1 receives answer from 2. 🡪 new minimum = (x1,500)
6. VM 1 commits Insert(k1,x1,200) 🡪 new minimum = (x1,200)
7. VM 1 commits Update(k1,x1,500) 🡪 new minimum (x1,200)

only weak consistency because of local/global Update Interference

Update interference[2](JOIN-View)

RS 1

Base Table

WAL

VM 1

RS 3

View Table

WAL

I

RS 2

Base Table

WAL

I

VM 2

1. RS 1 InsertT1( k1,x1,400)
2. VM 1 InsertT1 / VM 1 asks R2 for T1 join T2
3. RS 2 InsertT2(j1,x1,200)
4. VM 2 InsertT2 / VM 2 asks R1 for T2 join T1
5. VM 1 receives answer from 2. 🡪 Insert (x1, 400, 200)
6. VM 2 receives answer from 4. 🡪 Insert (x1, 400, 200)

Double join entries because of Update interference 🡪

Concurrent Updates[1] (SUM-View)

VM 1

VM 2

RS 2

View Table

WAL

I

I

* 1. VM 1 receives Insert( k1,x1,200)
  2. VM 2 receives Insert(k3,x1,100)
  3. VM 1 asks RS 2 for value of x1 🡪 gets (x1, 50)
  4. VM 2 asks RS 2 for value of x1 🡪 gets (x1, 50)
  5. VM 1 calculates 50+200 and commits 250
  6. VM 2 calculates 50+100 and commits 150

1. last Value = 150 should be 350🡪

no convergence because two update were commited on the same value

Concurrent Updates[1] (MIN-View)

RS 1

Base Table

WAL

VM 1

RS 3

View Table

WAL

I

RS 2

Base Table

WAL

I

VM 2

1. RS 1 Insert( k1,x1,400)
2. RS 2 Insert (k5,x1,200)
3. VM 2 commits Insert(k5,x1,200) 🡪 new minimum = (x1,200)
4. VM 1 commits Insert(k1,x1,400) 🡪 new minimum = (x1,400)

no convergence because of wrong commit order

Concurrent Updates / Wrong commit order[1](MIN-View)

RS 1

Base Table

WAL

VM 1

RS 3

View Table

WAL

D

RS 2

Base Table

WAL

I

VM 2

1. RS 1 Delete( k1,x1,400)
2. VM 1 sees that current minimum is deleted/VM 1 asks for new minimum
3. VM 1 receives answer from 2. 🡪 new minimum = (x1, 300)
4. RS 2 Insert (k5,x1,200)
5. VM 2 commits Insert(k5,x1,200) 🡪 new minimum = (x1,200)
6. VM 1 commits Delete(k1,x1,400) 🡪 new minimum = (x1,300)

no convergence because of wrong commit order

Concurrent Updates / Wrong commit order [2](JOIN-View)

RS 1

Base Table

WAL

VM 1

RS 3

View Table

WAL

I

RS 2

Base Table

WAL

D

VM 2

1. RS 1 InsertT1( k1,x1,400)
2. VM 1 InsertT1 / VM 1 asks R2 for T1 join T2
3. VM 1 receives answer from 2. 🡪 Insert (x1, 400,200)
4. RS 2 DeleteT2(j1,x1,200)
5. VM 2 commits DeleteT2 (j1,x1,200) 🡪Delete(x1,400,200)
6. VM 1 commits InsertT1 🡪 Insert (x1, 400, 200)

Existing join entry where no entry should be

because of Wrong commit order

Wrong update order / Wrong commit order(Selection View <300)[1]

VM 1

VM 2

RS 2

View Table

WAL

U

U

* 1. VM 1 receives Update( k1,100)
  2. VM 2 receives Update(k1,200)
  3. VM 2 commits k1,200
  4. VM 1 commits k1,100
  5. last Value = 100 should be 200

no convergence because **timeline of key was violated**

Wrong update order / Wrong commit order(Join View)[2]

The

RS 3

View Table

WAL

I

RS 2

Base Table

WAL

D

VM 2

1. RS 1 InsertT1( k1,x1,400)
2. VM 1 InsertT1 / VM 1 asks R2 for T1 join T2
3. VM 1 receives answer from 2. 🡪 Insert (x1, 400,200)
4. RS 2 DeleteT2(j1,x1,200)
5. VM 2 commits DeleteT2 (j1,x1,200) 🡪Delete(x1,400,200)
6. VM 1 commits InsertT1 🡪 Insert (x1, 400, 200)
7. last Value = 100 should be 200

no convergence because **timeline of key was violated**

Problems

Multiple Updates(non-idempotent )

Request VR

Commit U1

Request VR

U1

Calculate new VR

Calculate new VR

Commit U1

Update Interference

Request VR

U1

Query to RS

Commit U1

U2

Calculate new VR

Receive VR

Receive A+U2

Concurrent updates

Request VR

U2

Commit U2

Request VR

U1

Receive VR

Calculate new VR

Calculate new VR

Concurrent Updates / Wrong commit order

Request VR

U2

Commit U2

Request VR

U1

Query to RS

Commit U1

Receive VR

Calculate new VR

Calculate new VR

Wrong commit order

Request VR

ReceiveU2

Commit U2

Request VR

Receive VR

Calculate new VR

Calculate new VR

U2

U1

Commit U1

Solutions

Standard Update Complete[1] 🡪 Test And Set / Global Update

RS 1

Base Table

WAL

VM 1

RS 2

View Table

WAL

Client

1. Client issues update
2. Update is written to WAL then commited to base table
3. Process on R1 gets notfied and checks hash ring for view manager
4. Process sends update to corresponding View Manager
5. View Manager asks RS 2 for ViewRecord + Signature  
   RS 2 delivers ViewRecord + Signature
6. View Manager checks with Signature if Update has already been made
7. View Manager sends query to all RS with BT  
   View Manager receives answer
8. View Manager computes delta and new Signature
9. View Manager commits new value with new Signature with a TestAndSet-Method on the old Signature

Prevent Multiple Updates

Request VR

Commit U1

Request VR

U1

Calculate new VR

Calculate new VR

Reject U1

* Check Signature of View Record if Update has already been made
* Updates that have been made can be found in the signature
* Region Server and SeqNo deliver a global ID for Updates
* SeqNo is a consecutive number in the Write Ahead Log

Example: Update (k4,x1,15) from RS 3, SeqNo 3 is repeated 🡪 View Manager checks View Record and throws Update away

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| View Table |  | Base Table Updates |  |  |  |
| AggregationKey | Sum | Region Server | SeqNo | View Manager | Timestamp |
| x1 | 45 | RS 1 | 4 | VM 1 | 10.01 Uhr |
|  |  | RS 2 | 2 | VM 1 | 10.02 Uhr |
|  |  | RS 1 | 5 | VM 1 | 10.05 Uhr |
|  |  | RS 3 | 1 | VM 2 | 10.07 Uhr |

Prevent concurrent updates

Calculate new VR

Request VR

Request VR

U2

Fail U2

Request VR

U1

Receive VR

Calculate new VR

Calculate new VR

Receive VR

Commit U2

* Use Signature to recognize if View Record has been modified in the mean time
* Set View Values with TestAndSet-Method

Example: Insert(k4,x1,30) from RS 1 , SeqNo 6 🡪 View Manager gets View Record Value of 45 and calculates new Value of 75. Insert into view fails because the Signature has changed after commit of RS 3, SeqNo 2

Get View Record Value:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| View Table |  | Base Table Updates |  |  |  |
| AggregationKey | Sum | Region Server | SeqNo | View Manager | Timestamp |
| x1 | 45 | RS 1 | 4 | VM 1 | 10.01 Uhr |
|  |  | RS 2 | 2 | VM 1 | 10.02 Uhr |
|  |  | RS 1 | 5 | VM 1 | 10.05 Uhr |
|  |  | RS 3 | 1 | VM 2 | 10.07 Uhr |
| Test And Set fails  View Table |  | Base Table Updates |  |  |  |
| AggregationKey | Sum | Region Server | SeqNo | View Manager | Timestamp |
| x1 | 60 | RS 1 | 4 | VM 1 | 10.01 Uhr |
|  |  | RS 2 | 2 | VM 1 | 10.02 Uhr |
|  |  | RS 1 | 5 | VM 1 | 10.05 Uhr |
|  |  | RS 3 | 1 | VM 2 | 10.07 Uhr |
|  |  | RS 3 | 2 | VM 2 | 10.10 Uhr |

Prevent Update Interference

Request VR

U1

Version-Query to RS

Commit U1

U2

Calculate new VR

Receive VR-Sig

Receive A

* Use Versioning-Feature of HBase
* Take Signature of View Record to formulate query

Example: Insert(k4,x1,200) from RS 1 , SeqNo 6 🡪 View Manager takes Signature of View Record and uses the timestamp of the very last Update from the corresponding RS for the query.

Query RS 2 with Insert(k4,x1,200) join RS 2 version <= 10.02 Uhr  
 Query RS 3 with Insert(k4,x1,200) join RS 3 version <= 10.07 Uhr

Then insert join deltas. If signature has been modfied by a delete for example then 🡪 concurrent updates

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| View Table |  | Base Table Updates |  |  |  |
| AggregationKey | Sum | Region Server | SeqNo | View Manager | Timestamp |
| x1 | 45 | RS 1 | 5 | VM 1 | 10.01 Uhr |
|  |  | RS 2 | 5 | VM 1 | 10.02 Uhr |
|  |  | RS 1 | 4 | VM 1 | 10.05 Uhr |
|  |  | RS 3 | 3 | VM 2 | 10.07 Uhr |

Prevent wrong commit order

* Use flow control of view Managers
* Take Signature of View Record to recognize wrong timeline

Problems with TestAndSet

* Overlapping transactions force recalculation
* A lot of unecessary queries are executed
* Global queries may starve if there are a lot of concurrent local queries

Request VR

U1

Version-Query to RS

Commit U3

Calculate new VR

Request VR

Calculate new VR

Request VR

Version-Query to RS

Calculate new VR

U2

U3

Receive VR

Fail U2

Fail U1

Request VR

U1

Version-Query to RS

Calculate new VR

Request VR

Version-Query to RS

Calculate new VR

U2

Fail U2

Commit U1

Request VR

Version-Query to RS

Calculate new VR

U2

Fail U2

Standard Update Revisted 🡪 Parallizing Update Computation/ Global Update

RS 1

Base Table

WAL

VM 1

RS 2

View Table

WAL

Client

1. Client issues update
2. Update is written to WAL then commited to base table
3. Process on R1 gets notfied and checks hash ring for view manager
4. Process sends update to corresponding View Manager
   1. View Manager asks RS 2 for ViewRecord + Signature
   2. RS 2 delivers ViewRecord + Signature
5. View Manager checks with Signature if Update has already been made
   1. View Manager sends query to all RS with BT
   2. View Manager receives answer
6. View Manager computes delta and new Signature
7. View Manager commits new value with new Signature with a TestAndSet-Method on the old Signature

**Column identifier**

K1

ViewTableContent

**(rs1,123)**

**(rs1,124)**

Signatures

**Rowkey**

**Column value**

aggregationKey

X10

100

aggregationValue

**(rs2,55)**

**Column family**

rs1

rs1

rs1