# Hermes Walkthrough

Database Systems

DataLab, CS, NTHU

Spring, 2021

# Goal

- This walkthrough will bridge the gap between the paper and the code

# Outline

- Prior Knowledge
- Hermes Architecture
- Client’s Job Flow & Code
- Server’s Job Flow & Code
- Keep in Mind

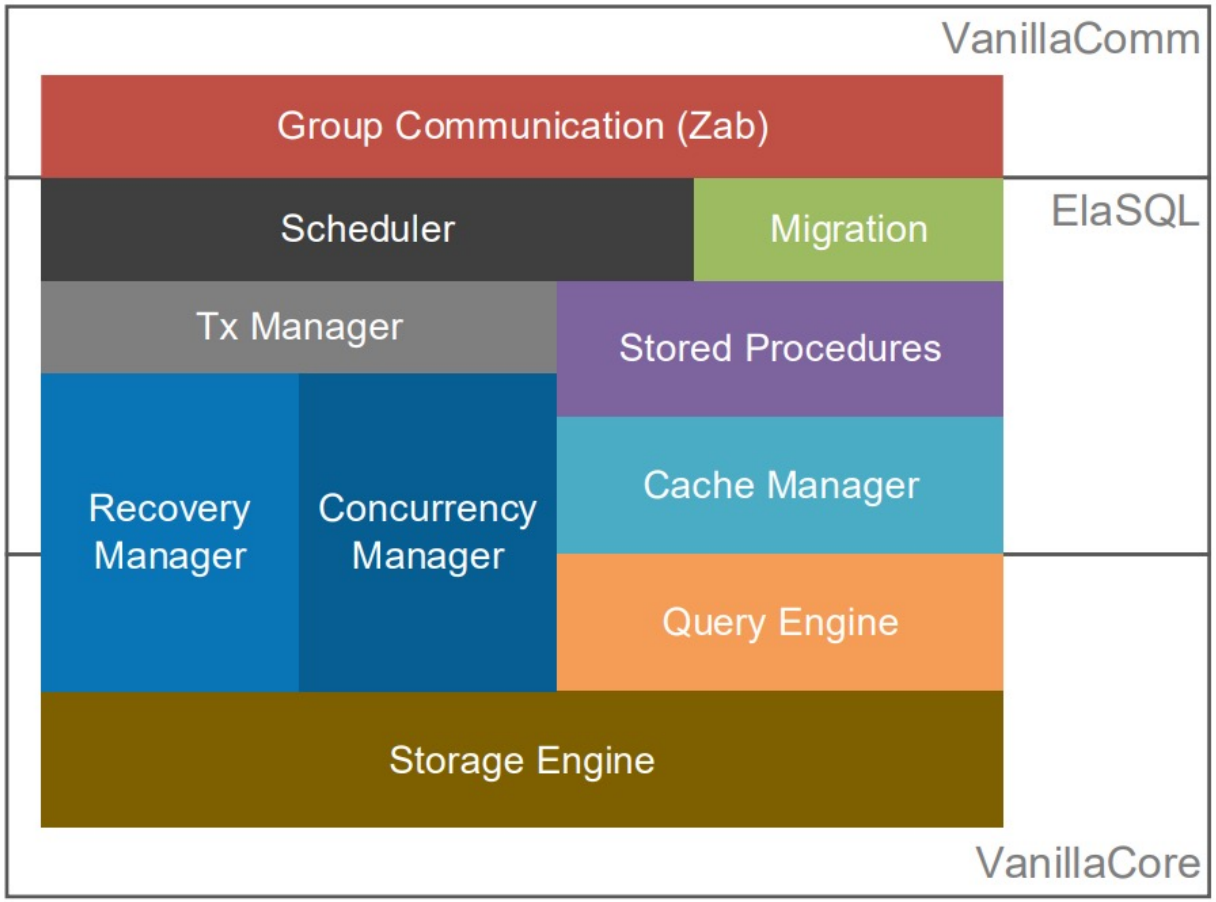
# Outline

- Prior Knowledge
- Hermes Architecture
- Client’s Job Flow & Code
- Server’s Job Flow & Code
- Keep in Mind

# Prior Knowledge

- In Elasql, Hermes reuse much code based on another system called *T-part (SIGMOD'16)*
- Without the knowledge of T-part, we could still know what Hermes does

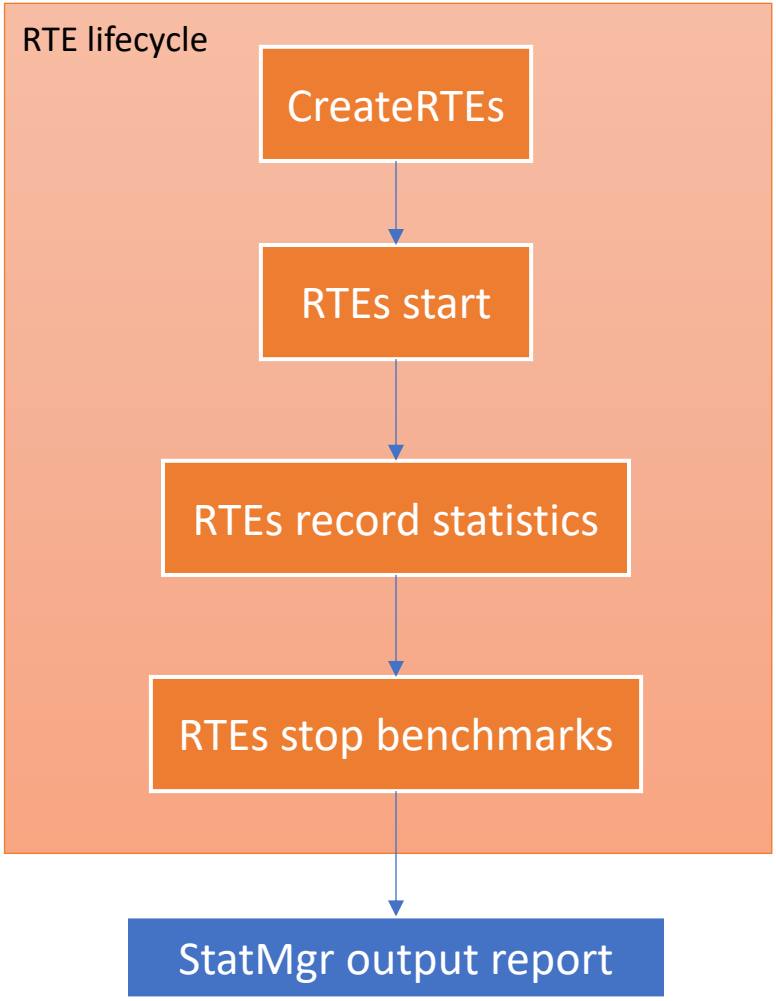
# Hermes Architecture



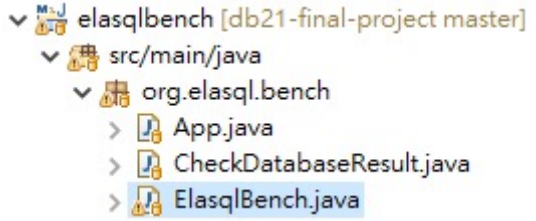
# Outline

- Prior Knowledge
- Hermes Architecture
- Client’s Job Flow & Code
- Server’s Job Flow & Code
- Keep in Mind

# Client's Job Flow



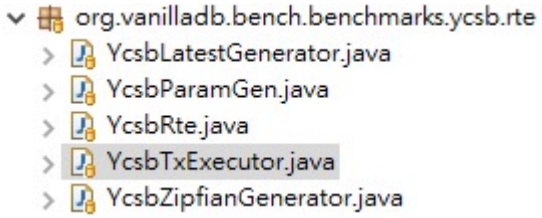
# Create RTE




```
61    public void benchmark() {
62        try {
63            if (logger.isLoggable(Level.INFO))
64                logger.info("checking the database on the server...");
65
66            SutConnection conn = getConnection();
67 //         boolean result = checkDatabase(conn);
68 //
69 //         if (!result) {
70 //             if (logger.isLoggable(Level.SEVERE))
71 //                 logger.severe("the database is not ready, please load the database again.");
72 //             return;
73 //         }
74
75            if (logger.isLoggable(Level.INFO))
76                logger.info("database check passed.");
77
78            if (logger.isLoggable(Level.INFO))
79                logger.info("creating " + BenchmarkerParameters.NUM_RTES + " emulators...");
80
81            int rteCount = benchmarker.getNumOfRTEs();
82            RemoteTerminalEmulator<?>[] emulators = new RemoteTerminalEmulator[rteCount];
83            emulators[0] = benchmarker.createRte(conn, statMgr); // Reuse the connection
84            for (int i = 1; i < emulators.length; i++)
85                emulators[i] = benchmarker.createRte(getConnection(), statMgr);
86
```

Benchmarker would be ElasqlYcsbBenchmark

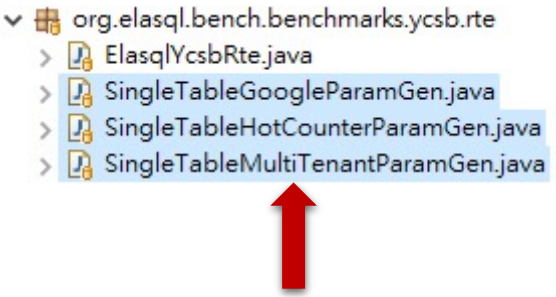
# RTE Starts & Execute Tx




```
17  public TxnResultSet execute(SutConnection conn) {
18      try {
19          // generate parameters
20          Object[] params = pg.generateParameter();
21
22          // send txn request and start measure txn response time
23          long txnRT = System.nanoTime();
24
25          SutResultSet result = executeTxn(conn, params);
26
27          // measure txn response time
28          long txnEndTime = System.nanoTime();
29          txnRT = txnEndTime - txnRT;
30
31          // display output
32          if (TransactionExecutor.DISPLAY_RESULT)
33              System.out.println(pg.getTxnType() + " " + result.outputMsg());
34
35          return new TxnResultSet(pg.getTxnType(), txnRT, txnEndTime,
36                  result.isCommitted(), result.outputMsg());
37      } catch (Exception e) {
38          e.printStackTrace();
39          throw new RuntimeException(e.getMessage());
40      }
41  }
```

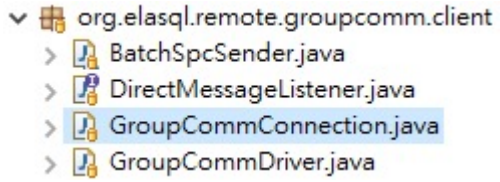
Step Into

Parameters are generated depends on the workload.




If you want to see the workloads, please have a look at xxxParamGen.java

# RTE callStoreProc




```
57    public ElasqlSpResultSet callStoredProc(int connId, int pid, Object... pars) {
58        // Check if there is a queue for it
59        BlockingQueue<ClientResponse> respQueue = rteToRespQueue.get(connId);
60        if (respQueue == null) {
61            respQueue = new LinkedBlockingQueue<ClientResponse>();
62            rteToRespQueue.put(connId, respQueue);
63        }
64
65        batchSender.callStoredProc(connId, pid, pars);
66
67        // Wait for the response
68        try {
69            ClientResponse cr = respQueue.take();
70            Long lastTxNumObj = rteToLastTxNum.get(connId);
71            long lastTxNum = -1;
72            if (lastTxNumObj != null)
73                lastTxNum = lastTxNumObj;
74
75            while (lastTxNum >= cr.getTxNum())
76                cr = respQueue.take();
77
78            // Record the tx number of the response
79            rteToLastTxNum.put(connId, cr.getTxNum());
80
81            return cr.getResultSet();
82        } catch (InterruptedException e) {
83            e.printStackTrace();
84            throw new RuntimeException("Something wrong");
85        }
86    }
```

The RTE will collect a batch of SP, and send them to Sequencer

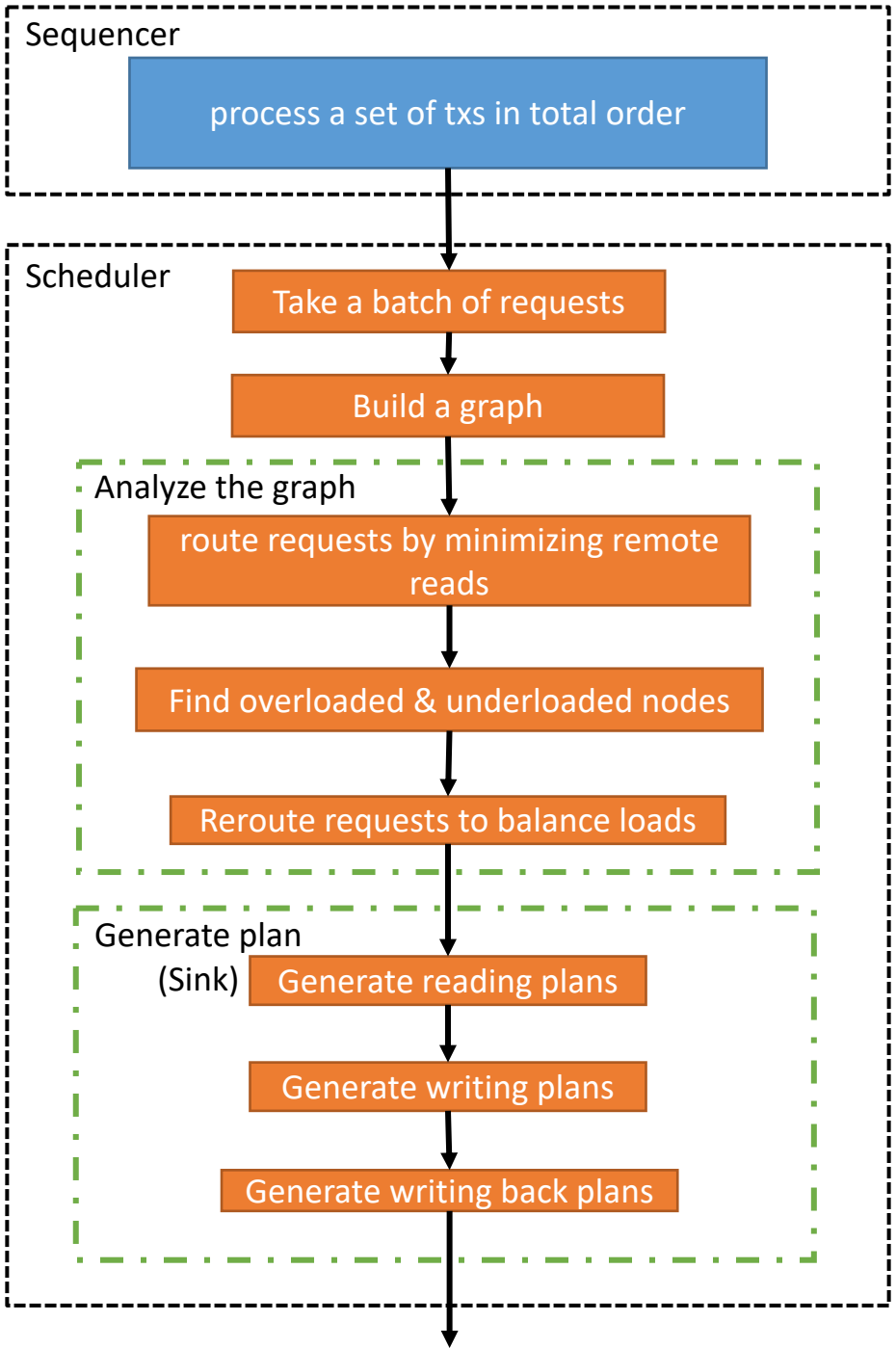
The rest of client's job is quite simple and similar to as2.

You could see ElasqlBench.benchmark() for more details.

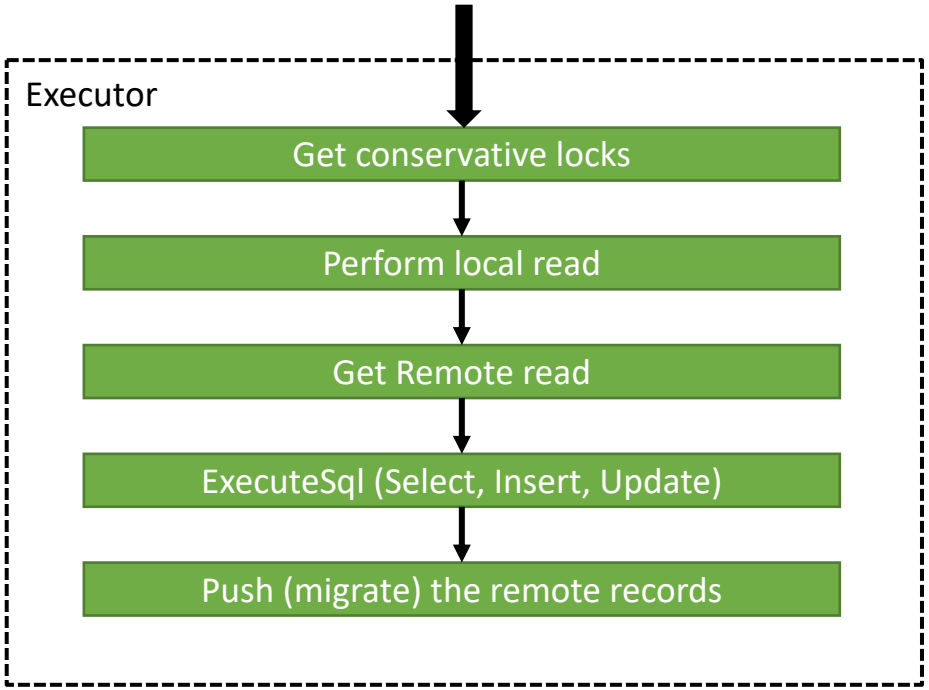
# Outline

- Prior Knowledge
- Hermes Architecture
- Client’s Job Flow & Code
- Server’s Job Flow & Code
- Keep in Mind

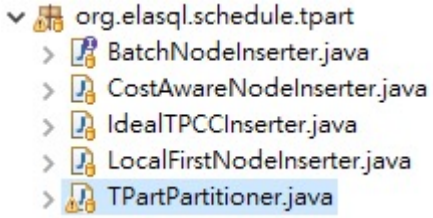
# Server’s Job Flow




Sink is a procedure that converts a graph to an execution plan


Executor
Get conservative locks
Perform local read
Get Remote read
ExecuteSql (Select, Insert, Update)
Push (migrate) the remote records

# Hermes Scheduler



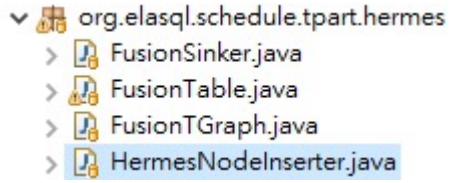

Hermes' scheduler leverages the code of TpartPartitioner

```
81      public void run() {
82          List<TPartStoredProcedureTask> batchedTasks = new LinkedList<TPartStoredProcedureTask>();
83
84          while (true) {
85              try {
86                  // blocked if the queue is empty
87                  StoredProcedureCall call = spcQueue.take();
88                  TPartStoredProcedureTask task = createStoredProcedureTask(call);
89
90                  // schedules the utility procedures directly without T-Part
91                  // module
92                  if (task.getProcedureType() == ProcedureType.UTILITY) {
93  //                  VanillaDb.taskMgr().runTask(task);
94                      continue;
95                  }
96
97                  // TODO: Uncomment this when the migration module is migrated
98  //                if (task.getProcedureType() == ProcedureType.MIGRATION) {
99  //                    // Process and dispatch it immediately
100 //                    processMigrationTx(task);
101 //                    continue;
102 //                }
103
104                 if (task.getProcedureType() == ProcedureType.NORMAL) {
105                     batchedTasks.add(task);
106                 }
107
108                 // sink current t-graph if # pending tx exceeds threshold
109                 if ((batchingEnabled && batchedTasks.size() >= NUM_TASK_PER_SINK)
110                         || !batchingEnabled) {
111                     processBatch(batchedTasks);
112                     batchedTasks.clear();
113                 }
114
115             } catch (InterruptedException ex) {
116                 if (logger.isLoggable(Level.SEVERE))
117                     logger.severe("fail to dequeue task");
```

1. Take a batch of requests

2. Build a graph

# Analyze the graph

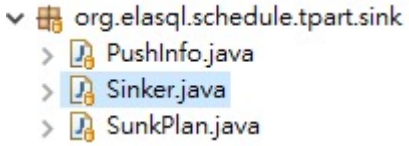



```
36      public void insertBatch(TGraph graph, List<TPartStoredProcedureTask> tasks) {
37          // Step 0: Reset statistics
38          resetStatistics();
39
40          // Step 1: Insert nodes to the graph
41          for (TPartStoredProcedureTask task : tasks) {
42              insertAccordingRemoteEdges(graph, task);
43          }
44
45          // Step 2: Find overloaded machines
46          overloadedThreshold = (int) Math.ceil(
47                  ((double) tasks.size() / partMgr.getCurrentNumOfParts()) * (IMBALANCED_TOLERANCE + 1));
48          if (overloadedThreshold < 1) {
49              overloadedThreshold = 1;
50          }
51          List<TxNode> candidateTxNodes = findTxNodesOnOverloadedParts(graph, tasks.size());
52
53  //      System.out.println(String.format("Overloaded threshold is %d (batch size: %d)", overloadedThresh
54  //      System.out.println(String.format("Overloaded machines: %s, loads: %s", overloadedParts.toString(
55
56          // Step 3: Move tx nodes from overloaded machines to underloaded machines
57          int increaseTolerence = 1;
58          while (!overloadedParts.isEmpty()) {
59  //          System.out.println(String.format("Overloaded machines: %s, loads: %s, increaseTolerence: %d'
60              candidateTxNodes = rerouteTxNodesToUnderloadedParts(candidateTxNodes, increaseTolerence);
61              increaseTolerence++;
62
63              if (increaseTolerence > 100)
64                  throw new RuntimeException("Something wrong");
65          }
66
67  //      System.out.println(String.format("Final loads: %s", Arrays.toString(loadPerPart)));
68      }
69
```

3. Minimize remote reads

4. Find overloaded machines

5. Balance the loading of each node




# Generate plans

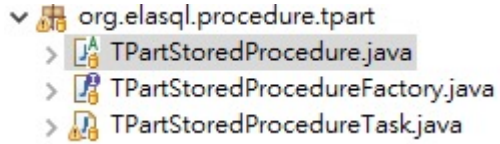
```
50    protected List<TPartStoredProcedureTask> createSunkPlan(TGraph graph) {
51        List<TPartStoredProcedureTask> localTasks = new LinkedList<TPartStoredProcedureTask>();
52
53        // Build a local execution plan for each transaction node
54        for (TxNode node : graph.getTxNodes()) {
55            // Debug
56 //           System.out.println(String.format("Node %d: %s (writeback: %d)", node.getTxNum(),
57 //                   node.getTask().getProcedure().getClass().getSimpleName(), node.getWriteBackE
58
59            // Check if this node is the master node
60            boolean isHereMaster = (node.getPartId() == myId);
61            SunkPlan plan = new SunkPlan(sinkProcessId, isHereMaster);
62
63            // Generate reading plans
64            generateReadingPlans(plan, node);
65
66            // Generate writing plans
67            generateWritingPlans(plan, node);
68
69            // Generate write back (to sinks) plans
70            generateWritingBackPlans(plan, node);
71
72            // Decide if the local node should execute this plan
73            if (plan.shouldExecuteHere()) {
74                // Debug
75 //               System.out.println(String.format("Tx.%d Plan: %s", node.getTxNum(), plan));
76
77                node.getTask().decideExceutionPlan(plan);
78                localTasks.add(node.getTask());
79            }
80        }
81
82        return localTasks;
83    }
```

6. Generate reading plans

7. Generate writing plans

8. Generate writing back plans

If you press F3, you will step into a function that belongs to T-part.
Please check out FusionSinker.java for the Hermes version.

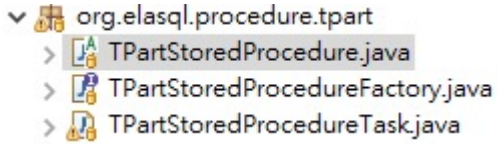



# Execute

```
112     public SpResultSet execute() {
113         try {
114 //          Timer.getLocalTimer().startComponentTimer("Get locks");
115             getConservativeLocks();
116 //          Timer.getLocalTimer().stopComponentTimer("Get locks");
117 
118             executeTransactionLogic();
119 
120             tx.commit();
121             isCommitted = true;
122         } catch (Exception e) {
123             e.printStackTrace();
124             System.out.println("Tx." + txNum + "'s plan: " + plan);
125             tx.rollback();
126         }
127         return new SpResultSet(
128             isCommitted,
129             paramHelper.getResultSetSchema(),
130             paramHelper.newResultSetRecord()
131         );
132     }
```

9. Acquire conservative locks

Let's step into this function (next slide)




# Execute cont.

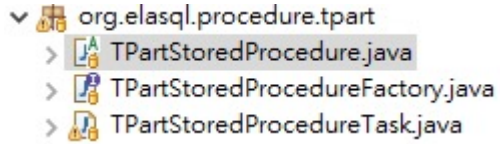
```
193    private void executeTransactionLogic() {
194        int sinkId = plan.sinkProcessId();
195 //     Timer timer = Timer.getLocalTimer();
196
197        if (plan.isHereMaster()) {
198            Map<PrimaryKey, CachedRecord> readings = new HashMap<PrimaryKey, CachedRecord>();
199            // Read the records from the local sink
200 //         timer.startComponentTimer("Read from sink");
201            for (PrimaryKey k : plan.getSinkReadingInfo()) {
202                readings.put(k, cache.readFromSink(k));
203            }
204 //         timer.stopComponentTimer("Read from sink");
205
206            // Read all needed records
207 //         timer.startComponentTimer("Read from cache");
208            for (PrimaryKey k : plan.getReadSet()) {
209                if (!readings.containsKey(k)) {
210                    long srcTxNum = plan.getReadSrcTxNum(k);
211                    readings.put(k, cache.read(k, srcTxNum));
212                    cachedEntrySet.add(new CachedEntryKey(k, srcTxNum, txNum));
213                }
214            }
215 //         timer.stopComponentTimer("Read from cache");
216
217            // Execute the SQLs defined by users
218 //         timer.startComponentTimer("Execute SQL");
219            executeSql(readings);
220 //         timer.stopComponentTimer("Execute SQL");
```

10. Perform local read

11. Perform remote read

12. ExecuteSql (Select, Insert, Update)

# Execute cont.




```
224         Map<Integer, Set<PushInfo>> pi = plan.getPushingInfo();
225         if (pi != null) {
226             // read from local storage and send to remote site
227             for (Entry<Integer, Set<PushInfo>> entry : pi.entrySet()) {
228                 int targetServerId = entry.getKey();
229 
230                 // Construct a tuple set
231                 TupleSet rs = new TupleSet(sinkId);
232                 for (PushInfo pushInfo : entry.getValue()) {
233                     CachedRecord rec = cache.read(pushInfo.getRecord(), txNum);
234                     cachedEntrySet.add(new CachedEntryKey(pushInfo.getRecord(), txNum, pushInfo.getDestTxNum()));
235                     rs.addTuple(pushInfo.getRecord(), txNum, pushInfo.getDestTxNum(), rec);
236                 }
237 
238                 // Push to the remote
239                 Elasql.connectionMgr().pushTupleSet(targetServerId, rs);
240             }
241         }
242         timer.stopComponentTimer("Push");
```

13. Push (migrate) the remote records

# Outline

- Prior Knowledge
- Hermes Architecture
- Client’s Job Flow & Code
- Server’s Job Flow & Code
- Keep in Mind

# Hermes vs T-part


Check out Elasql.initTpartScheduler to see the key differences between Hermes & T-part

```
241    public static Scheduler initTPartScheduler(TPartStoredProcedureFactory factory) {
242        TGraph graph;
243        BatchNodeInserter inserter;
244        Sinker sinker;
245        FusionTable table;
246        boolean isBatching = true;
247
248        switch (SERVICE_TYPE) {
249        case TPART:
250            graph = new TGraph();
251            inserter = new CostAwareNodeInserter();
252            sinker = new Sinker();
253            isBatching = true;
254            break;
255        case HERMES:
256            table = new FusionTable();
257            graph = new FusionTGraph(table);
258            inserter = new HermesNodeInserter();
259            sinker = new FusionSinker(table);
260            isBatching = true;
261            break;
```

# Code Tracing

- You are supposed to know the Hermes job's flow before tracing the code
- Make good use of Eclipse to help code tracing
- Please ignore every TODO comments in the code

# Implementation

- Please make sure that your routing algorithm is a deterministic algorithm, or you may get a clog system