Fair Scheduler

What is the Fair Scheduler?

The ClustrixDB Fair Scheduler ensures that long-running queries do not monopolize CPU resources. It does this by prioritizing queries that return fewer rows ahead of queries that return a larger number of rows. This helps balance workloads across nodes and is particularly useful for clusters that mix long-running Online Analytical Processing (OLAP) and shorter-running Online Transaction Processing (OLTP).The Fair Scheduler will prioritize any query that reads 100 (by default) or fewer rows ahead of any running query that is processing more then 100 rows. The global variable gtm_schedule_til_batch_rows i s used to control the row threshold.

How can the Fair Scheduler Help?

Without the Fair Scheduler, long running queries can monopolize CPU resources and introduce unwanted latency for smaller queries. For example, if a short query is assigned to a cpu core that is already executing a long-running query, the shorter query would experience unexpected latency. The Fair Scheduler addresses that by balancing the resources between long-running and short-running queries. Higher query latency is more noticeable on a short-running query than a long-running query.

Global Variables

The following global variables control the Fair Scheduler. These variables are not available by session.

Name	Description	Default Value
gtm_schedule_til	Enable the Completely Fair Scheduler.	true
gtm_schedule_til_batch_rows	Rows to process before rescheduling.	100

Examining the Impact of the Fair Scheduler

While a query is running, you can see the impact of the Fair Scheduler in the system.transactions table After a query has completed, the query.log will include statistics about CPU waits.

query.log

The cpu_waits and cpu_waittime_ns statistics in the query.log show when a query was waiting for CPU resources, which may be due to being de-prioritized by the Fair Scheduler. Queries are logged to query.log once they have completed.

For example, you can see that the Fair Scheduler de-prioritized this query 45,752 times for a total of 21,975,562,278 nanoseconds or 21 seconds.

```
2017-03-01 16:49:01.530435 UTC ip-10-10-10-101 clxnode: INSTR SLOW SID:297145363 db=production user=production@10.10.10.254  ac=Y xid=58c4dbf59b223826 sql="UPDATE event_activities SET is_processed = true where batch_id = '1488386700'" [Ok: 91275 rows updated] time 9303.1 ms; reads: 91276; inserts: 0; deletes: 0; updates: 182550; counts: 91275; rows_read: 182550; forwards: 365101; broadcasts: 0; rows_output: 2; semaphore_matches: 0; fragment_executions: 365102; runtime_ns: 22744490156; cpu_waits: 45752; cpu_waittime_ns: 21975562278; bm_fixes: 592280; bm_loads: 0; bm_waittime_ns: 0; lockman_waits: 1; lockman_waittime_ms: 6844; trxstate_waits: 0; trxstate_waittime_ms: 0; wal_perm_waittime_ms: 0; bm_perm_waittime_ms: 0; sigmas: 0; sigma_fallbacks: 0; row_count: 91275; found_rows: -1; insert_id: 0; fanout: no; attempts: 1
```

system.transactions

The system.transactions table shows information for the same query while it is running:

```
sql> select xid, cpu, cpu_waits, cpu_waittime_ns from transactions where xid =
6420139111091474470;

+--------------------+------+-----------+-----------------+
| xid                | cpu  | cpu_waits | cpu_waittime_ns |
+--------------------+------+-----------+-----------------+
| 6420139111091474470 |    6 |         0 |               0 |
| 6420139111091474470 |    6 |     26447 |     45517956308 |
| 6420139111091474470 |    6 |         0 |               0 |
| 6420139111091474470 |    6 |         0 |               0 |
| 6420139111091474470 |    6 |         0 |               0 |
| 6420139111091474470 |    6 |         0 |               0 |
| 6420139111091474470 |    6 |         0 |               0 |
| 6420139111091474470 |    6 |         0 |               0 |
+--------------------+------+-----------+-----------------+
8 rows in set (0.00 sec)
```