Scheduling

When you have only one query with one source, scheduling is not a thing to think about, in other cases some options can be done by configuring the scheduler.

Scheduling in Odysseus has two parts:

1. Choose the query to schedule. This is done by the **scheduler**
2. Scheduler the plan of the query. This is done with the **scheduling strategy**

In ODYSSEUS_HOME a file called scheduling.conf defines the basic scheduling mechanisms:

defaultScheduler=<SCHEDULER>

defaultStrat=<SCHEDULING STRATEGY>

The scheduling can be changed via Odysseus Script (Other Commands).

```
#SCHEDULER "Single Thread Scheduler RR" "Round Robin"
```

Remark: This will not update scheduling.conf file.

Schedulers:

- **ST Scheduler RR MS Limit Thread**: This scheduler schedules different sources in multiple thread. Each source has the same priority. You can configure (`Scheduler.Simplethreaded.SourcesPerThread`) how many sources should be scheduled by one thread. If more sources need to be scheduled a new thread is spawned.
- **Single Thread Scheduler RR Multi Source**: This scheduler schedules different sources in multiple thread. Here you can define how many threads should be used to schedule all sources (`Scheduler.Simplethreaded.SourceThreadCount`), independed of the number of sources
- Single Thread Scheduler RR: This scheduler schedules each sources in a different thread. No further configuration is necessary.

Remark: The configuration is done in Odysseus.conf. It is also possible to use Odysseus Script (#ODYSSEUS_PARAM see Other Commands).

In the following example the "ST Scheduler RR MS Limit Thread" scheduler is used with the "Round Robin" scheduling strategy. The following parameter states, that there should be one source per thread, i.e. each source in a query gets its own thread and will run independently.

```
#SCHEDULER "ST Scheduler RR MS Limit Thread" "Round Robin"
#ODYSSEUS_PARAM Scheduler.Simplethreaded.SourcesPerThread 1
```

The following configurations can be done in the Odysseus.conf:

- scheduler_simpleThreadScheduler_executorThreadsCount: How many theads should be used in parallel. If set to -1 (or not set) the number of processor cores is used. Save value = 1
- Scheduler.Simplethreaded.SourceThreadCount: How many threads should be used for all sources, independed of the number of sources
- Scheduler.Simplethreaded.SourcesPerThread (for ST Scheduler RR MS Limit Thread): How many sources should be access by one thread, if there are more sources, then a new scheduler is used

Other, non core schedulers could provide further parameters.

There are some features, that provide further schedulers

- **Simple Dynamic Priority Scheduler**: TBD
- **Static Priority Plan Scheduler**: TBD
- **SLA Scheduler**: TBD

Scheduling Strategy

By the scheduling strategy, the query is separated into parts, that could be scheduled independently. Only if a query contains at least one Buffer operator, the scheduling strategy is relevant. In other cases, this strategy does not make any difference.

Remark: If there are no multiple threads, then the processing is done from a source until a sink. If an operator has more than one outputs, the order is from left to the right, but always until a sink. So if one output port is connected to a query part that has a large number of following operators (or an operator that has a high latency) this will block the execution of the other output port until each part after the port is exeucuted. You could add Buffer operators after each output to allow parallel execution of each output line.

List of currently available Scheduling Strategies (Availablity depends on contained bundles):

- Aurora Min Latency: This strategy tries to process each element as fast as possible.
- Aurora Min Cost: This strategy tries to process each element with lowest costs.

See http://cs.brown.edu/research/aurora/vldb03_scheduler.pdf for deeper information about Aurora based schedulings

- Chain: The strategy tries to clear waiting queues as fast as possible, i.e. process a path in the query, that will consume fewer memory.
- Chain (Iter): The same, but implemented in another way.

See http://i.stanford.edu/~dilys/papers/vldb132.pdf for deeper information about Chain Scheduling

- Biggest Queue: Simple strategy that always schedules the queue with the largest numbers of inputs. Remark: This strategy has no notion of fairness or starvation, i.e. some queues could never be processed with this strategy.
- Round Robin: In this strategy, all buffers are scheduled round robin.
- Round Robin (Iter): Same as above, but other implementation

You could also bypass all scheduling strategies, even with buffers, when using a threaded buffer (see Buffer operator) and let the virtual machine/operating system do the thread handling. In many cases this is faster than using a strategy.