Control Parallelismn

Required Feature: Parallelization

You are able to use Odysseus Script to parallelize an created script automatically. To use this functionality Odysseus Script provides three keywords and a UI for benchmarking.

#PARALLELIZATION keyword

This keyword tells Odysseus, that the given query needs to be parallelized. There are two parameters that are mandatory and one optional parameter.

- **Parallelization-Type**: (mandatory) Inter-Operator or Intra-Operator. If Inter-Operator type is selected, the given query plan is modified. If Intra-Operator type is selected, different physical operators are used, that provides multithreading.

Additional Parameters if type is inter-operator:

- **Parallelization degree**: (mandatory) Defines the degree of parallelization that should be used. It is also possible to use the constant AUTO, to detect the available cores and use this value.
- **Buffer-size**: Defines the number of elements inside of the used buffers. There is also the possibility to use the constant AUTO, to use an default value.
- **Optimization: (optional)** With this parameter it is possible to enable or disable post optimization. Post optimization is done if two or more transformations are processed and it is possible to combine these transformations. Removes unneeded union and fragmentations. The optimization is only possible if fragment-operators have the same number of fragments and if the attributes are equal (only hashfragment). In addition to this it is not possible that there are stateful operators like unions or operators with stateful functions between both transformations. It is also not possible that there are splits of the datastream between both of them. These function is enabled by default. If you want to disable it, pass false as argument for this parameter.

Additional Parameters if type is intra-operator

- **Parallelization degree**: (mandatory) Defines the degree of parallelization that should be used. It is also possible to use the constant AUTO, to detect the available cores and use this value.
- **Buffer-size**: Defines the number of elements inside of the parallel operators. There is also the possibility to use the constant AUTO, to use an default value.

The following example shows the usage of this keyword. This example uses the inter-operator parallelization with an degree of 4 and an automatic buffersize.

```
#PARSER PQL
#PARALLELIZATION INTER_OPERATOR 4 AUTO true
/// other possible definition of parameters for this keyword
#PARALLELIZATION (type=INTER_OPERATOR) (degree=4) (buffersize=AUTO) (optimization=true)
#RUNQUERY
windowBid = TIMEWINDOW({SIZE = [1, 'MINUTES'],
                advance = [1, 'SECONDS']
                }, bid)

windowAuction = TIMEWINDOW({SIZE = [10, 'MINUTES'],
                advance = [1, 'SECONDS']
                }, auction)

join = JOIN({PREDICATE = 'bid.bidder == auction.id'}, windowBid, windowAuction)
```

ⓘ   If this keyword is used, every operator of the query, which has an compatible parallelization strategy is transformed. If only one or subset of operators should parallelized, the following keyword need to be used in addition.

#INTEROPERATOR keyword

The #INTEROPERATOR keyword is an addition to the #PARALLELIZATION keyword for inter-operator parallelization. With this keyword it is possible to select one or more operators, which should be parallelized. There is also the possibility to configure the parallelization for each operator. This keyword provides following parameters:

- **Operator-Ids (mandatory):** one or more operatorIds for operators the need to be parallelized. If more than one id is defined the ids need to be separated by commas (avoid blanks between ids and commas).
- **Parallelization degree:** (mandatory) Defines the degree of parallelization that should be used. It is also possible to use the constant AUTO to detect the available cores and use this value, or GLOBAL to use the value defined in the #PARALLELIZATION keyword.
- **Buffer-size:** (mandatory) Defines the number of elements inside of the used buffers. There is also the possibility to use the constant AUTO to use an default value, or GLOBAL to use the value defined in the #PARALLELIZATION keyword.
- **Parallelization strategy** (optional): the parallelization strategy that should be used for the given operatorIds. Otherwise the preferred strategy is automatically detected.
- **Fragmentation type** (optional): the fragmentation type that should be used inside of the strategy. Otherwise the preferred type is automatically selected.
- **Use parallel Operators** (optional): if this option is selected, parallel operators are used instead of normal sequential operators. If the given operator does not support parallel execution, this option is ignored. Default value is false.

The following code example shows the usage of this keyword. Only the aggregation is parallelized, because only this id is defined. The global parallelization degree is overwritten with the value of 2. With the constant GLOBAL the value for the buffersize is used from the global definition. In addition to this parameters, also the parallelization strategy is defined manually. In this case the AggregateMultithreadedTransformationStrategy is used. Note that the strategy need to be fit to the operator type defined with the id. In addition the strategy need to be compatible for the operator. In some cases it is not possible to use the selected strategy, e.g. an grouping inside the aggregation is needed. See the list below, for more informations. The last parameter in this example is the optional selection of an fragmentation type. Note that not every strategy supports all fragmentation types. See the list below for all possible combinations.

```
#PARSER PQL
#PARALLELIZATION (type=INTER_OPERATOR) (degree=4) (buffersize=AUTO) (optimization=true)
#INTEROPERATOR aggregateId 2 GLOBAL NonGroupedAggregateTransformationStrategy ShuffleFragmentAO
/// other possible definition of parameters for this keyword
#INTEROPERATOR (id=aggregateId) (degree=2) (buffersize=GLOBAL)
(strategy=NonGroupedAggregateTransformationStrategy) (fragment=ShuffleFragmentAO) (useParallelOp=true)
#RUNQUERY

windowBid = TIMEWINDOW({SIZE = [1, 'MINUTES'],
                  advance = [1, 'SECONDS']
                  }, bid)

windowAuction = TIMEWINDOW({SIZE = [10, 'MINUTES'],
                  advance = [1, 'SECONDS']
                  }, auction)

join = JOIN({ID = 'joinId', PREDICATE = 'bid.bidder == auction.id'}, windowBid, windowAuction)

sum_price_bidder = AGGREGATE({ID = 'aggregateId',
                              aggregations = [
                                ['SUM', 'price', 'sum_price_bidder']
                              ]
                            },
                            join
                          )
```

Definition of Endpoints for parallelization

If the #INTEROPERATOR-keyword is used, it is also possible to define a start and enpoint for parallelization. The definition of the start and end point of parallelization is possible with a tuple or triple inside of the id-parameter. The following example shows how to use this feature.

```
#PARSER PQL
#PARALLELIZATION (type=INTER_OPERATOR) (degree=4) (buffersize=AUTO) (optimization=true)
#INTEROPERATOR (id=(aggregateId:selectId)) (degree=4) (buffersize=GLOBAL)
(strategy=GroupedAggregateTransformationStrategy) (fragment=HashFragmentAO)
/// assoure no semantic changes
#INTEROPERATOR (id=(aggregateId:selectId:true)) (degree=4) (buffersize=GLOBAL)
(strategy=GroupedAggregateTransformationStrategy) (fragment=HashFragmentAO)
#RUNQUERY

windowBid = TIMEWINDOW({SIZE = [1, 'MINUTES'],
                  advance = [1, 'SECONDS']
                  }, bid)

sum_price_bidder = AGGREGATE({ID = 'aggregateId',
                              aggregations = [
                                ['SUM', 'price', 'sum_price_bidder']
                              ],
                            GROUP_BY = ['bidder'],
                              FASTGROUPING =
true
                            },
                            windowBid
                        )

selectBidder = SELECT({ID = 'selectId',PREDICATE = 'bid.bidder > 1'}, sum_price_bidder)
```

In this example the aggregate operator is parallelized, but it is also needed that the following selection is also parallelized. So to do this at first the id of the aggregation is set, after that, seperated with a double quote, the second id of the selection is definied as the endpoint of parallelization. There is the possibility, that the parallelization changes the semantic of the query. If you want to avoid this append :true to the pair of start and end id. By default this option is set to false. If it is enabled, a possible semantic change results in an exception.

Strategies and supported fragmentation types

Logical operator	Parallelization strategies	Description	Supported fragmentation types	Allows definition of endpoint
JoinAO	JoinTransformationStrategy	Uses an Hash-Fragmentation for both input streams. The fragmentation attributes are gathered from the join attributes. Note that only equals-predicates (which are concatenated with &&) are supported. The fragmented datastream is merged with an UNION Operator.	**HashFragmentAO**	✅
AggregateAO	NonGroupedAggregateTransformationStrategy	Uses an RoundRobin or Shuffle Fragmentation for splitting the input datastream. This strategy works with partial aggregates and merges the datastream both with an union operator and an additional aggregate operator for merging the partial aggregates. This strategy works with and without grouping. Only aggregations with one input attribute are supported.	**RoundRobinFragmentAO** ShuffleFragmentAO	❌
	GroupedAggregateTransformationStrategy	Uses a Hash-Fragmentation for the input stream. The fragmentation attributes are gathered from the grouping attributes. So this strategy only works if the aggregate operator has an grouping. The fragmented datastream is merged with an UNION Operator.	**HashFragmentAO**	✅

ⓘ Bold fragmentation types shows the preferred type if nothing is defined.

#INTRAOPERATOR keyword

The #INTRAOPERATOR keyword is an addition to the #PARALLELIZATION keyword for intra-operator parallelization. With this keyword it is possible to select one or more operators, which should be parallelized. There is also the possibility to configure the parallelization for each operator. This keyword provides following parameter

- Operator-Ids (mandatory): one or more operatorIds for operators the need to be parallelized. If more than one id is defined the ids need to be separated by commas (avoid blanks between ids and commas).
- Parallelization degree: (mandatory) Defines the degree of parallelization that should be used. It is also possible to use the constant AUTO to detect the available cores and use this value.

- Buffer-size: (mandatory) Defines the number of elements inside of the used buffers. There is also the possibility to use the constant AUTO to use an default value.

If this keyword is used, in transformations a parallel physical operator is used. Note that this is only possible if the operator supports such a parallel physical operator. The following code, shows the usage of this keyword:

```
#PARSER PQL
#PARALLELIZATION (degree=8) (buffersize=10000) (type=INTRA_OPERATOR)
#INTRAOPERATOR (buffersize=10000) (id=joinId) (degree=8)
#RUNQUERY
windowBid = TIMEWINDOW({SIZE = [1, 'MINUTES'],
                   advance = [1, 'SECONDS']
                   }, bid)

windowAuction = TIMEWINDOW({SIZE = [10, 'MINUTES'],
                   advance = [1, 'SECONDS']
                   }, auction)

join = JOIN({id='joinId',PREDICATE = 'bid.auction == auction.id'}, windowBid, windowAuction)
```

ⓘ **Note**

It is possible to use both inter and intra operator parallelization at the same time. The following example shows how this works:

```
#PARSER PQL
#PARALLELIZATION (degree=8) (buffersize=10000) (type=INTRA_OPERATOR)
#INTRAOPERATOR (buffersize=10000) (id=aggregateId) (degree=8)

#PARALLELIZATION (type=INTER_OPERATOR) (degree=4) (buffersize=AUTO) (optimization=true)
#INTEROPERATOR (id=(joinId:selectId)) (degree=2) (buffersize=10000000)
(strategy=JoinTransformationStrategy) (fragment=HashFragmentAO)
#RUNQUERY
....
```

If you want to use inter and intra operator parallelization both for the same operator use the option useParallelOp=true

```
#PARSER PQL
#PARALLELIZATION (type=INTER_OPERATOR) (degree=4) (buffersize=AUTO) (optimization=true)
#INTEROPERATOR (id=aggregateId) (degree=2) (buffersize=GLOBAL)
(strategy=NonGroupedAggregateTransformationStrategy) (fragment=ShuffleFragmentAO) (useParallelOp=true)
#RUNQUERY
```

Parallelization Benchmarker

The parallelization component in Odysseus provides a UI, which allows the comparison of different parallelization configurations. To start this component, click on the bar icon in the top menu bar in Odysseus (please select the query (editor) which you want to parallelize before). The benchmarker starts initializing the existing query. You do not need to remove parallelization keywords before (these are ignored in benchmarker).
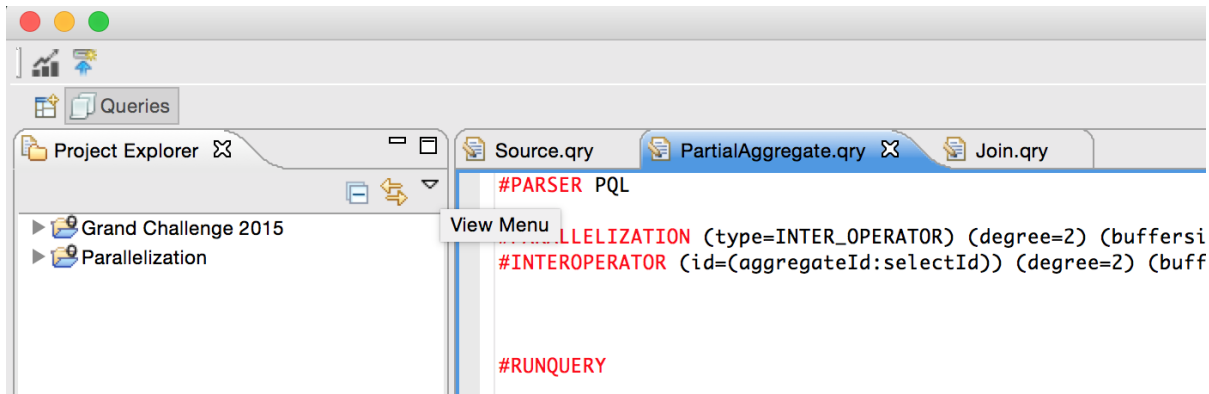
After Initialization is done, you need to configure the benchmarker. Note that multiple degrees or the selection of many strategies leads to many executions and a longer time for doing the analysis. The analysis counts a given umber of elements and gets the execution time of this. The configuration is splitted in three parts:

Global configuration:

- number of elements to count
- maximum time for each analysis (to avoid endless running)
- number of executions for every configuration (this is the number each configuration is executed every time)

Inter-Operator parallelization

- the degrees which should be tested (comma seperated)
- the buffersize to use in buffers
- select if you want to use threaded buffers or buffers which are controlled via sheduler
- select if you want to allow post optimization
- select if you want to use parallel operators (if the exists)
- select strategies and fragmentations you want to test (it is also possible to select the end operator id and custom degrees)

Intra-Operator parallelization

- the degrees which should be tested (comma seperated)
- insert operator ids if you want to test only a subset of operators (comma seperated)
- the buffersize to use in parallel operators

If configuration is done, hit the "Start Analysis" button and the benchmarker calculates all needed executions and starts them one after the other. This may take some time. If the benchmarking is done, the result is shown and there is the possibility to copy the resulting Odysseus script keywords in your query.

Analyse parallelization of current query

Finished !

Analysis started...
Executing analysis 1 of 2: Inter-Operator Parallelization: Global degree: 4, Post optimization allowed: true, Use threaded buffer: true, (OperatorId: aggregateId, degree: 4, Strategy: NonGroupedAggregateTransformationStrategy, Fragmentation: RoundRobinFragmentAO) , Use threaded operators: false
Done. 20000 elements in an average time of 3562 ms processed. (2 executions)

Executing analysis 2 of 2: Inter-Operator Parallelization: Global degree: 8, Post optimization allowed: true, Use threaded buffer: true, (OperatorId: aggregateId, degree: 8, Strategy: NonGroupedAggregateTransformationStrategy, Fragmentation: RoundRobinFragmentAO) , Use threaded operators: false
Done. 20000 elements in an average time of 3552 ms processed. (2 executions)

Parallelization Benchmark anaylsis complete

Result of parallelization benchmarker. Put this Snippet in your script.

#PARALLELIZATION (buffersize=10000000) (optimization=true) (type=INTER_OPERATOR) (degree=8) (threadedbuffer=true)
#INTEROPERATOR (strategy=NonGroupedAggregateTransformationStrategy) (useParallelOp=false) (fragment=RoundRobinFragmentAO) (degree=8) (id=aggregateId) (buffersize=10000000)

Done   Copy to Clipboard