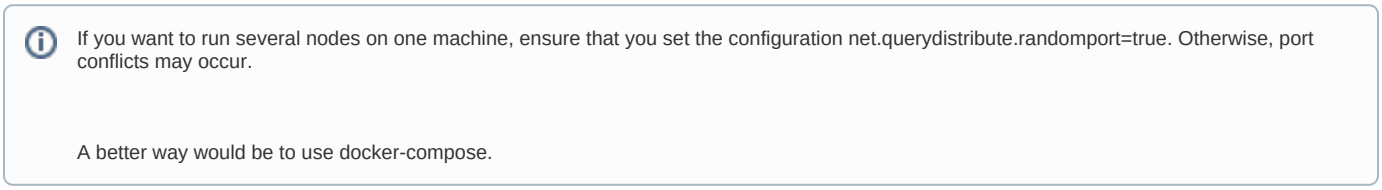
Distributing Queries

(currently work-in-progress)

If OdysseusNet is activated, the node has the possibility to distribute own queries to other remote OdysseusNodes. For this, the user has to specify that he wish to distribute:

```
#CONFIG DISTRIBUTE true
```

This statement in Odysseus Script indicates that the queries which follows should be distributed in the network.

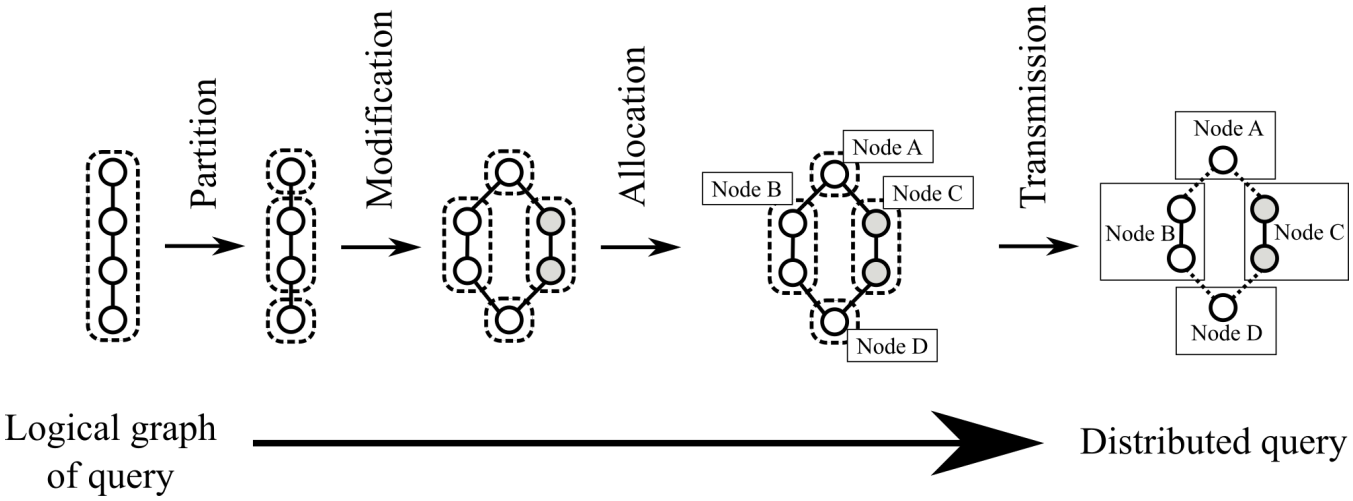


If you want to run several nodes on one machine, ensure that you set the configuration net.querydistribute.randomport=true. Otherwise, port conflicts may occur.

A better way would be to use docker-compose.

General process of query distribution

Query distribution in OdysseusNet is divided in three core phases: Partition, Modification and Allocation. Due to implementation reasons, two additional phases were integrated: Preprocess before Partition and Postprocess after Allocation. The picture below shows the three phases mentioned earlier (and the transmission phase).



At the beginning, Odysseus translates the query in a logical graph. This logical graph is preprocessed and then partitioned. The result of the Partition phase is a collection of query parts, which are individually modified in the Modification phase. However, the Modification-phase is optional an can be omitted. After that, the Allocation phase decides, which query part should be executed on which OdysseusNode. Finally, the query parts are transmitted to the selected nodes. Preprocess, Partition, Modification, Allocation and Postprocess provide multiple strategies the user can select from (individually for each query, if needed). Additionally, some strategies can be parameterised . The user has to use specific OdysseusScript keywords. One example for query distribution is as follows:

```
/// Indication that we want to distribute queries
#CONFIG DISTRIBUTE true

#NODE_PREPROCESS SOURCE                                    /// Select Source strategy for pre-processing
#NODE_PARTITION OPERATORCLOUD                         /// Select OperatorCloud strategy for Partition phase
#NODE_MODIFICATION REPLICATION 2                /// Select Replication strategy for Modification phase (with
replication degree 2)
#NODE_ALLOCATION USER                                          /// Select User-strategy for Allocation phase
#NODE_POSTPROCESSOR LOCALSOURCES                /// Select multiple strategies for post-processing
#NODE_POSTPROCESSOR LOCALSINKS
#NODE_POSTPROCESSOR MERGE

/// "Typical" query definition here
#METADATA TimeInterval
#PARSER PQL
#ADDQUERY
<Query>
```

The strategies are explained in the next sections. Partition and Allocation allow only one selected strategy. The other phases can be used multiple times per query. If the user had not specified strategies for some phases, OdysseusNet will use specific strategies as default (configurable):

Phase	Default strategy
Preprocess	`<none>`
Partition	`querycloud`
Modification	`<none>`
Allocation	`querycount`
Postprocess	`merge`

The Transmission phase cannot be altered.

Preprocess

Strategy name	Description
`source`	StreamAOs are replaced with their logical operators.

Partition

Strategy name	Description
`querycloud`	The entire query is one query part (no partitioning).
`operatorcloud`	Each logical operator is its own query part (max. partitioning)

Modification

Strategy name	Description
`replication <degree>`	Query parts are replicated (with replicationdegree `degree`), executed multiple times. See Replication.

Allocation

Strategy name	Description
`direct`	All query parts are assigned to the specified OdysseusNode. `#NODE_ALLOCATION DIRECT MyNode1`

`querycount`	Query parts are assigned to nodes with the least count of queries.
`roundrobin`	Query parts are assigned in order.
`user`	Query parts are assigned to user-specified nodes. For this, each logical operator has a `DESTINATION`-Parameter in PQL. Two operators which have the same `DESTINATION`-value are assigned to the same node.

Postprocess

Strategy name	Description
`localsink`	Logical sink operators are staying local (on the distributing node) overriding allocation. Useful if the user wants to have the last operator to check the data stream results.
`localsource`	Logical source operators are staying local (on the distribution node) overriding allocation. Useful if the user do not want to share its source operators to other nodes.
`merge`	Two adjacent query parts which are assigned to the same node are merged to omit unneeded network-transmission operators.
discardedreplicates	Post processor to insert a sender for each {@link ReplicationMergeAO}. The sender will be inserted for the output port 1 that is not used normally. All discarded replicates are sent to port 1. The sender writes the data in a CSV file (one file per merger). The only argument for this post processor is the path to the CSV files. The names of the files are determined by the {@link ReplicationMergeAO} (name and hashCode). Used sender settings: - transport handler "file" - data hgandler "tuple" - wrapper "GenericPush" - protocol handler "csv" - "createDir" "true" - "filename", path_from_user + "/" + merger.getName() + merger.hashCode() + ".csv" - "append", "true"
CalcLatency	Inserts a CalcLatencyAO before every real sink. Note that the ILatency metadata will be added to the sources automatically. This postprocessor needs the feature "Calculation Postprocessor Feature".
CalcDatarate	Inserts a DatarateAO after every source. Note that the IDatarate metadata will be added to the sources automatically. This postprocessor needs the feature "Calculation Postprocessor Feature".

RemoteQuery

A simple way to distribute whole queries (inklusing Odysseus Script parts) to other nodes can be done with the #REMOTEQUERY command:

```
#REMOTEQUERY (name=worker_1)
#PARSER PQL
#RUNQUERY
timer = TIMER({PERIOD = 1000, SOURCE = 'source'})
map = MAP({EXPRESSIONS = ['toString("marco")'], KEEPINPUT = true}, timer)
#REMOTEQUERY (name=worker_2)
#PARSER PQL
#RUNQUERY
timer = TIMER({PERIOD = 1000, SOURCE = 'source'})
map = MAP({EXPRESSIONS = ['toString("marco")'], KEEPINPUT = true}, timer)
```

Here, anything between two #REMOTEQUERY commands (or the end of the document) are copied and send as whole to the node to be processed there. Remark: This this different than "direct" from above as the query is not translated locally. By this, you could have e.g. multiple master nodes that will get the whole queries from another (super master) node.