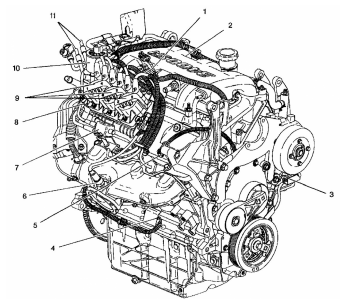
A distributed SQL query engine architecture for data analytics

Aleksei Ozeritskii

Yandex

2023

Contents



YQL (Yandex Query Language)

- A single entry point for various storage systems
  - YTSaurus, ClickHouse, YDB
- A library for query processing that can be embedded
  - YDB
  - Yandex Query (Yandex Cloud service)
  - YQL (internal Yandex service)
- YTSaurus data requests were executed by the Map/Reduce engine

Usage Example

```
select c_name, sum(o_totalprice) as totalprice from orders
join customer on o_custkey = c_custkey
join nation on n_nationkey = c_nationkey
where n_name = 'INDIA' group by c_name
order by totalprice desc limit 5
```

User Defined Functions

```
$f=Python3::f(@@
def f(x):
    """
    Callable<(String?)->String?>
    """
    return x.upper()
@@);

select name, $f(name) from hahn.`home/yql/tutorial/users`;
```

Cross Cluster Queries

```
select a.name, b.lastname, a.age, a.last_time_on_site
  from hahn.`home/yql/tutorial/users` as a
left join arnold.`home/yql/tutorial/lastnames` as b
  on a.name = b.name1
where a.age < 30
```

Join Clickhouse and YTSaurus

```
select yt_visits.age,
    count(DISTINCT yt_visits.name),
    count(DISTINCT ch_visits.UserID)
from hahn.`home/yql/tutorial/users` as yt_visits
    left join (
        select UserID,
            StartTime,
            Age
        from clickhousetutorial.`visits_all.visits_v1`
    ) as ch_visits ON ch_visits.Age = yt_visits.age
        group by yt_visits.age;
```

Map/Reduce: Pros And Cons

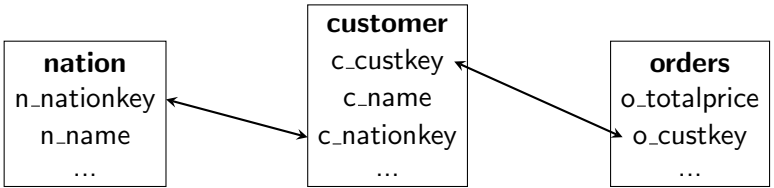
Pros

- Designed to handle large amounts of data (terabytes and petabytes of data)

Cons

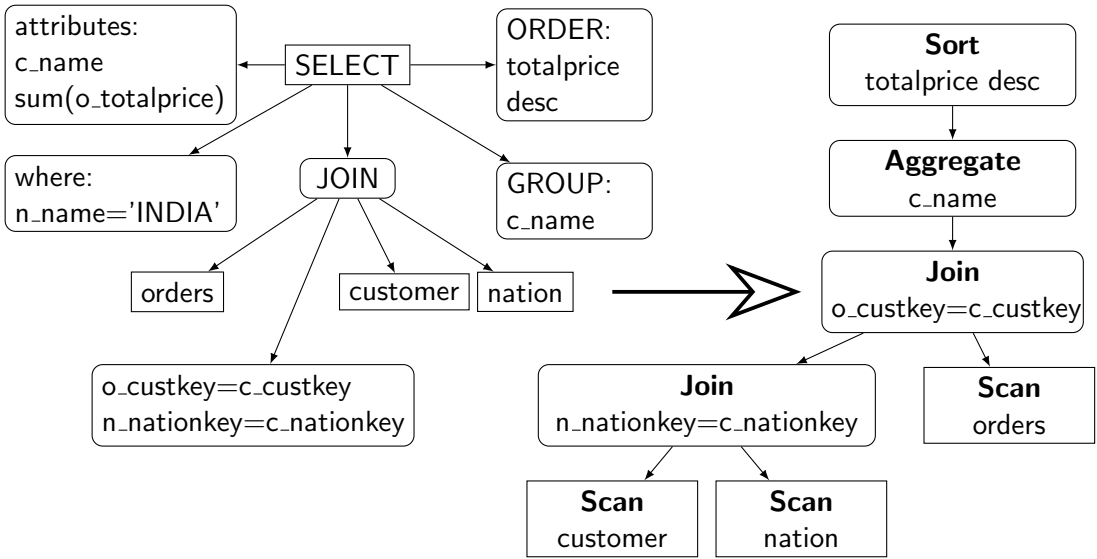
- A typical query execution pipeline includes many steps
- The result of each step is written to disk
- This pipeline becomes less efficient for relatively small amounts of data (100 GB)
- Only single cluster queries are supported

Example




```
select c_name, sum(o_totalprice) as totalprice from orders
join customer on o_custkey = c_custkey
join nation on n_nationkey = c_nationkey
where n_name = 'INDIA' group by c_name
order by totalprice desc limit 5
```

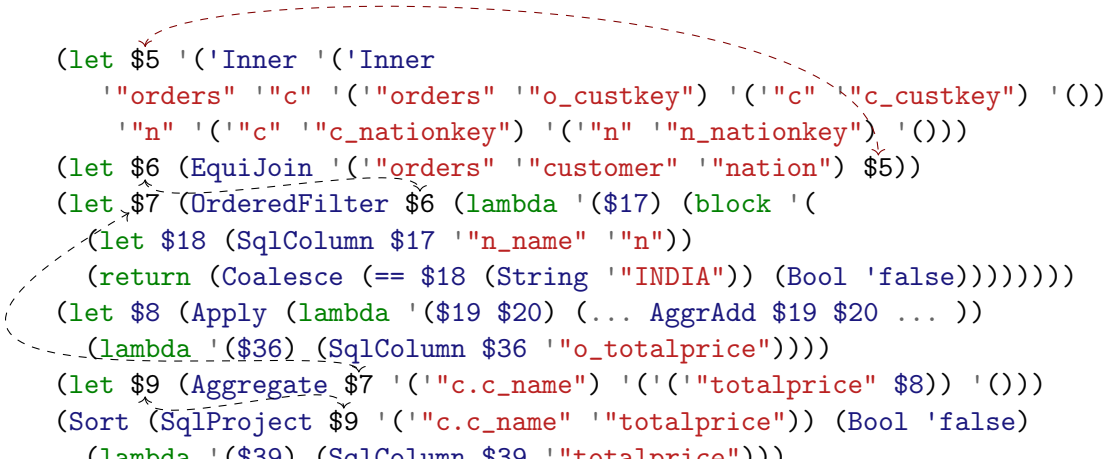
AST


attributes:
c_name
sum(o_totalprice)
SELECT
ORDER:
totalprice
desc
where:
n_name='INDIA'
JOIN
GROUP:
c_name
orders
customer
nation
o_custkey=c_custkey
n_nationkey=c_nationkey
Sort
totalprice desc
Aggregate
c_name
Join
o_custkey=c_custkey
Join
n_nationkey=c_nationkey
Scan
orders
Scan
customer
Scan
nation

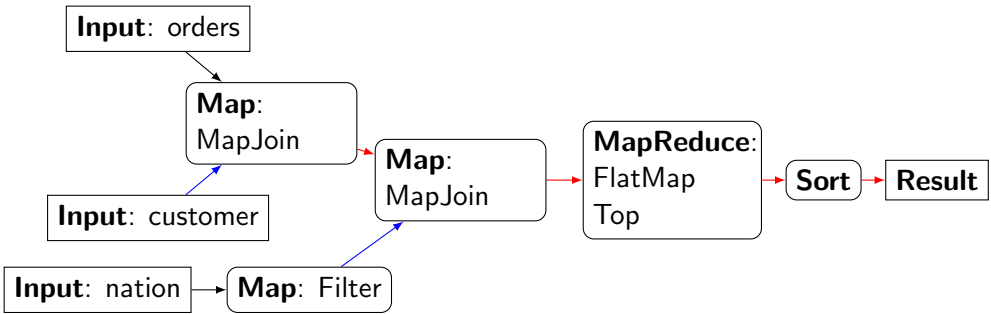
Relational Operators (s-expressions)

```
(let $5 '('Inner '('Inner
   '"orders" '"c" '('"orders" '"o_custkey") '('"c" '"c_custkey") '())
    '"n" '('"c" '"c_nationkey") '('"n" '"n_nationkey") '()))
(let $6 (EquiJoin '('"orders" '"customer" '"nation") $5))
(let $7 (OrderedFilter $6 (lambda '($17) (block '(
  (let $18 (SqlColumn $17 '"n_name" '"n"))
  (return (Coalesce (== $18 (String '"INDIA")) (Bool 'false))))))))
(let $8 (Apply (lambda '($19 $20) (... AggrAdd $19 $20 ... ))
  (lambda '($36) (SqlColumn $36 '"o_totalprice"))))
(let $9 (Aggregate $7 '('"c.c_name") '('('"totalprice" $8)) '()))
(Sort (SqlProject $9 '('"c.c_name" '"totalprice")) (Bool 'false)
  (lambda '($39) (SqlColumn $39 '"totalprice")))
```

Relational Operators (s-expressions)

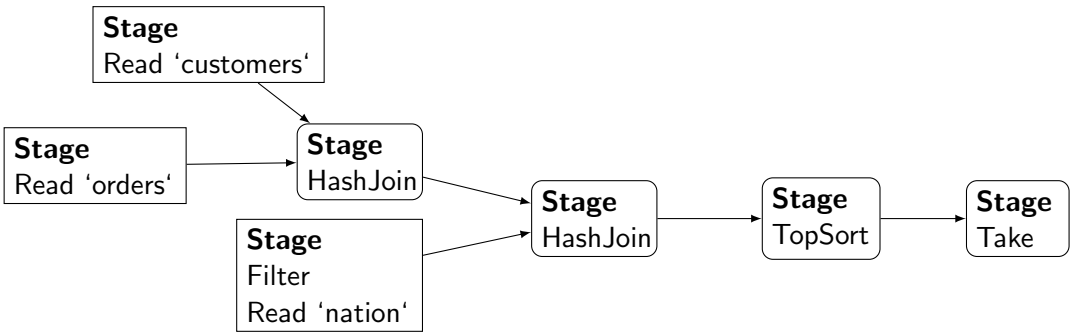


Map/Reduce Plan



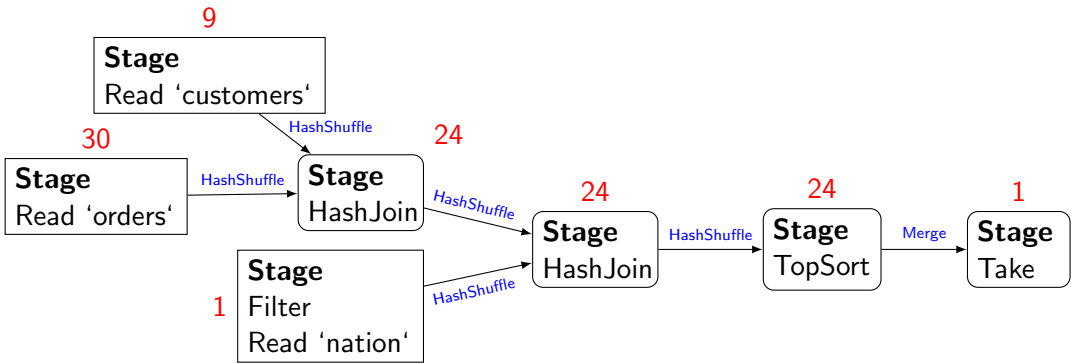
- Materializations: 5
- Dictionary inputs for MapJoin: ‘customer‘, filtered ‘nation‘
- Execution Time: 13:17

Distrubuted Query (DQ) Plan



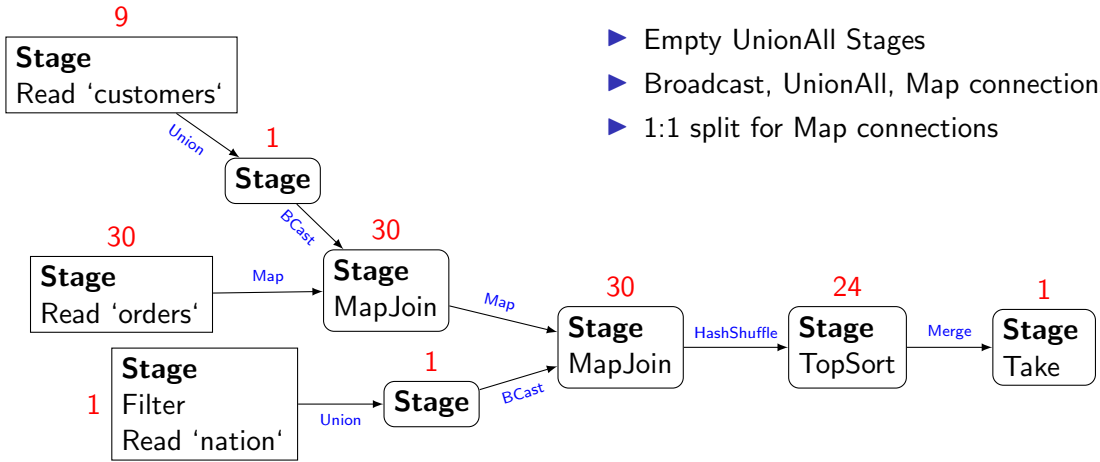
- Execution Time: 0:36

Distrubuted Query (DQ) Plan (Tasks and Connections)



- Split the stages into tasks (Read ‘orders‘ in 30 tasks)
- Connect stages by connections (HashShuffle, Merge)

Distrubuted Query (DQ) Plan (Tasks and Connections)

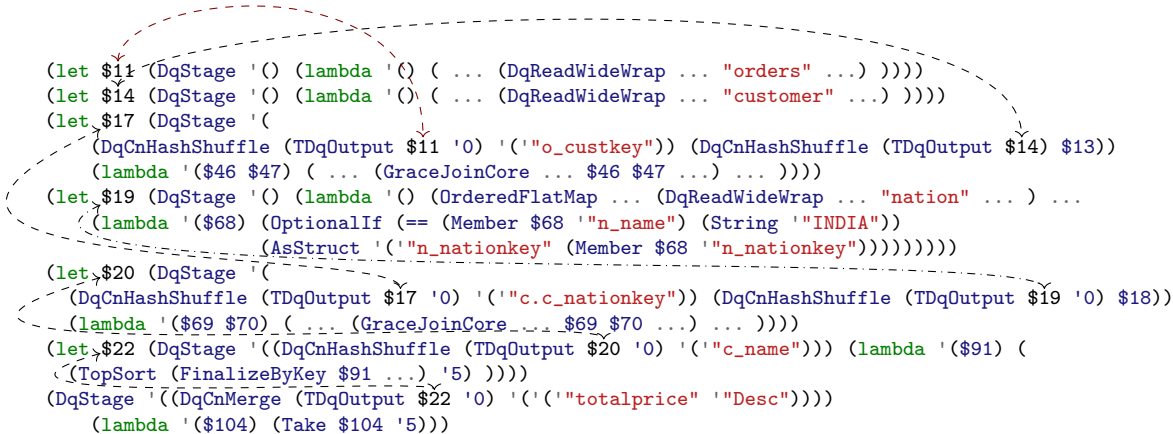


- Empty UnionAll Stages
- Broadcast, UnionAll, Map connections
- 1:1 split for Map connections

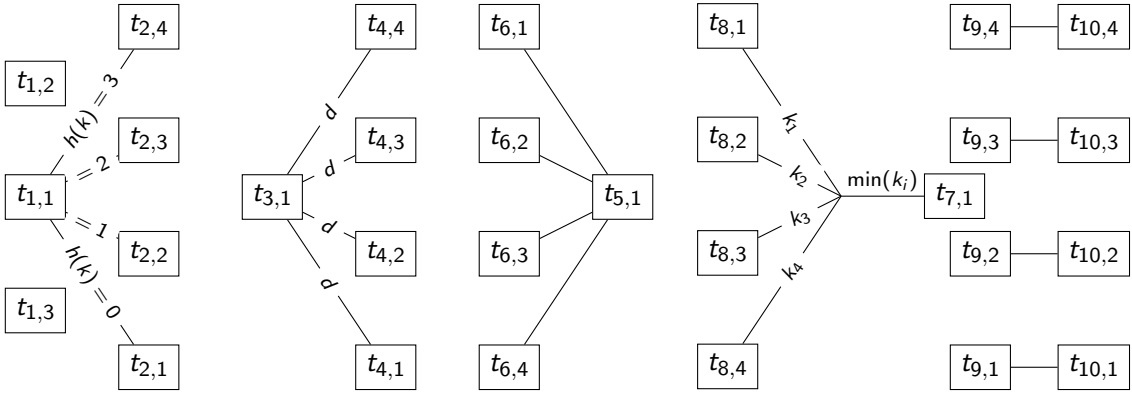
DQ Plan (s-expressions)

```
(let $11 (DqStage '() (lambda '() ( ... (DqReadWideWrap ... "orders" ...) ))))
(let $14 (DqStage '() (lambda '() ( ... (DqReadWideWrap ... "customer" ...) ))))
(let $17 (DqStage '(
    (DqCnHashShuffle (TDqOutput $11 '0) '('"o_custkey")) (DqCnHashShuffle (TDqOutput $14) $13))
    (lambda '($46 $47) ( ... (GraceJoinCore ... $46 $47 ...) ... ))))
(let $19 (DqStage '() (lambda '() (OrderedFlatMap ... (DqReadWideWrap ... "nation" ... ) ...
    (lambda '($68) (OptionalIf (== (Member $68 '"n_name") (String '"INDIA"))
                   (AsStruct '('"n_nationkey" (Member $68 '"n_nationkey"))))))))
(let $20 (DqStage '(
  (DqCnHashShuffle (TDqOutput $17 '0) '('"c.c_nationkey")) (DqCnHashShuffle (TDqOutput $19 '0) $18))
  (lambda '($69 $70) ( ... (GraceJoinCore ... $69 $70 ...) ... ))))
(let $22 (DqStage '((DqCnHashShuffle (TDqOutput $20 '0) '('"c_name"))) (lambda '($91) (
  (TopSort (FinalizeByKey $91 ...) '5) ))))
(DqStage '((DqCnMerge (TDqOutput $22 '0) '('('"totalprice" '"Desc"))))
    (lambda '($104) (Take $104 '5)))
```

DQ Plan (s-expressions)



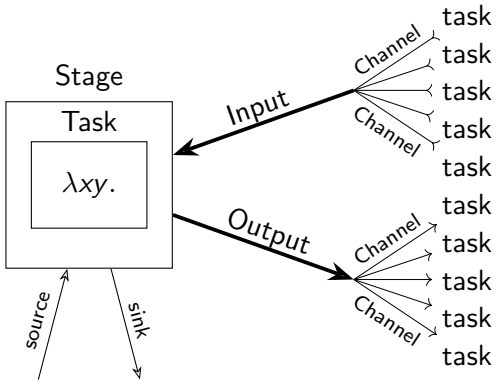
Connection Types



Hash Shuffle | Broadcast | UnionAll | Merge | Map

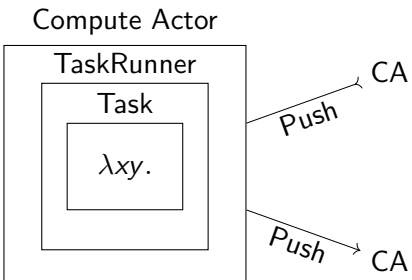
$t_{i,j}$ - Task, i - Stage Id, j - Task Id

Task, Inputs, Outputs, Sources, Sinks, Channels, Connections



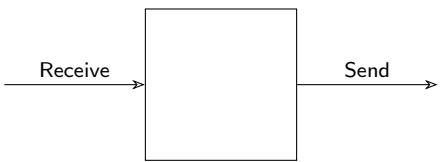
- Task: 0..N Inputs, 0..N Outputs
- Inputs: UnionAll, Merge
- Outputs: Hash Shuffle, Broadcast, Map
- Channel: task to task
- Source: read from (YDB, S3, CH, ...)
- Sink: write to (YDB, S3, ...)

Compute Actor (CA), TaskRunner



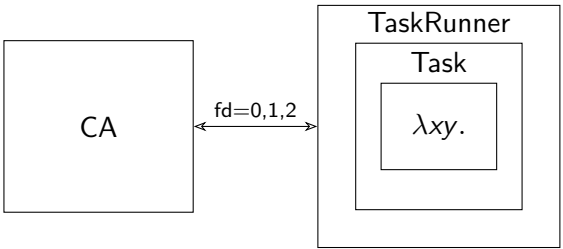
- CA pushes data to other CA
- CA runs TaskRunner on new data
- CA gets data from TaskRunner

Actors



- Inspired by Akka, Erlang
- Processes events in 1 thread
- `Send(ActorId, Event)`
- `Receive(Event(ActorId, Data))`
- `Become(NewReceiveHandler)`
- `Register(new Actor) -> ActorId`

CA and TaskRunner isolation



- `pipe`
- `fork/exec`
- `dup2`
- `stdin,stdout,stderr`
- run process in container

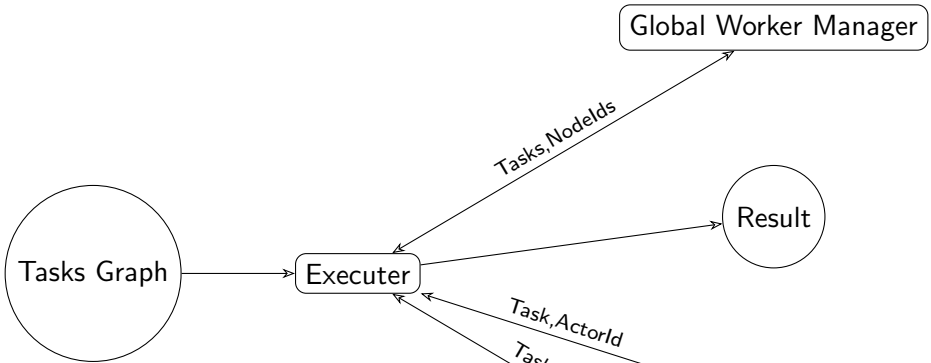
User Code Isolation

```
$f=Python3::f(@@
def f(x):
    """
    Callable<(Int32)->Int32>
    """
    import ctypes
    print(ctypes
       .cast(1, ctypes.POINTER(ctypes.c_int))
       .contents)
    return 0
@@);

select $f(0);
```

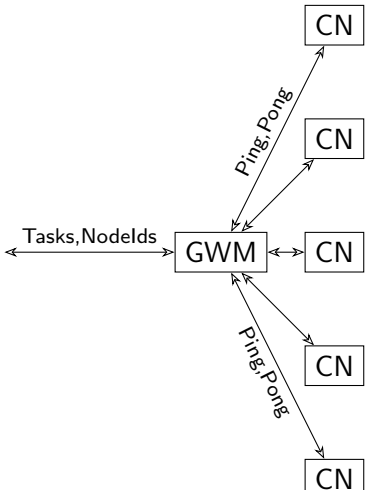
```
Container killed by signal: 11 (Segmentation fault)
?? at .../b4382c8e-78fcb74c-519140b6-33:0:0
Simple_repr at .../_ctypes.c:4979:12
PyObject_Str at .../object.c:492:11
PyFile_WriteObject at .../fileobject.c:129:17
builtin_print at .../bltinmodule.c:2039:15
cfunction_vectorcall... at .../methodobject.c:443:24
PyObject_Vectorcall at .../pycore_call.h:92:11
_PyEval_EvalFrameDefault at .../ceval.c:0:0
...
```

Executer



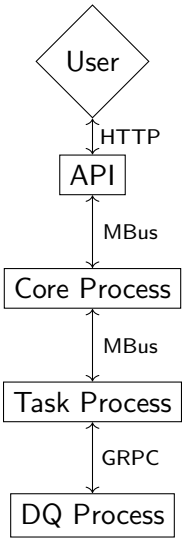
- Local Worker Manager
  - Runs on each compute node (CN)
  - Has known ActorId
  - Responsible for CA lifetime

Global Worker Manager (Scheduler, GWM)



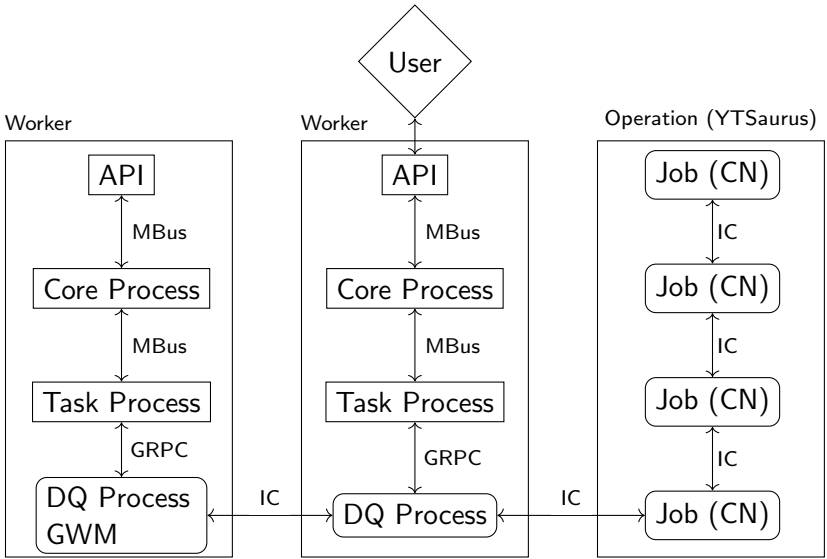
- Only 1 GWM per cluster
- CN uses YTSaurus to get GWM address
- CN reports via pings
  - CPU usage
  - CA count
  - Other resources (user files, UDFs)
- GWM can send commands via pongs
  - Download file (UDF, user file)
  - Stop command

Operation Start

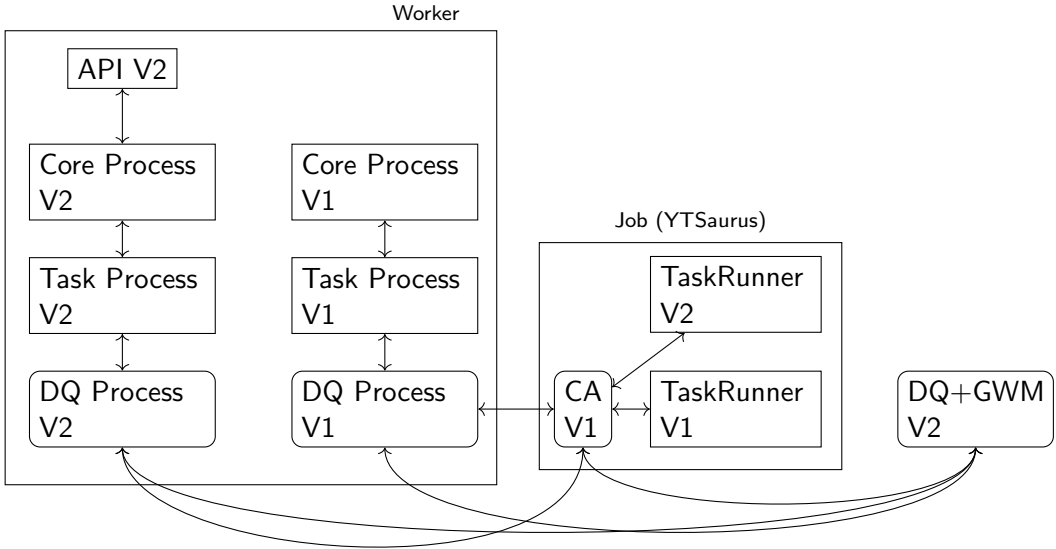


- API
  - Operation (Start, Stop, Status)
  - Named Queries
  - History (View, Search)
  - ACL
- Core Process
  - Starts Task Process per operation
  - Communicates with API
- Task Process
  - AST transform and optimize
- DQ Process
  - Starts Executer actor per operation

High Level Architecture



New version deploy



Plans

- Vectorization (in progress)
- Cost based optimizer (in early progress)
- Disk spilling (in early progress)
- Reoptimization

Thank You

YDB

YTSaurus

Actors

DQ

aozeritsky@ydb.tech