Midterm Report

RPQL DDB

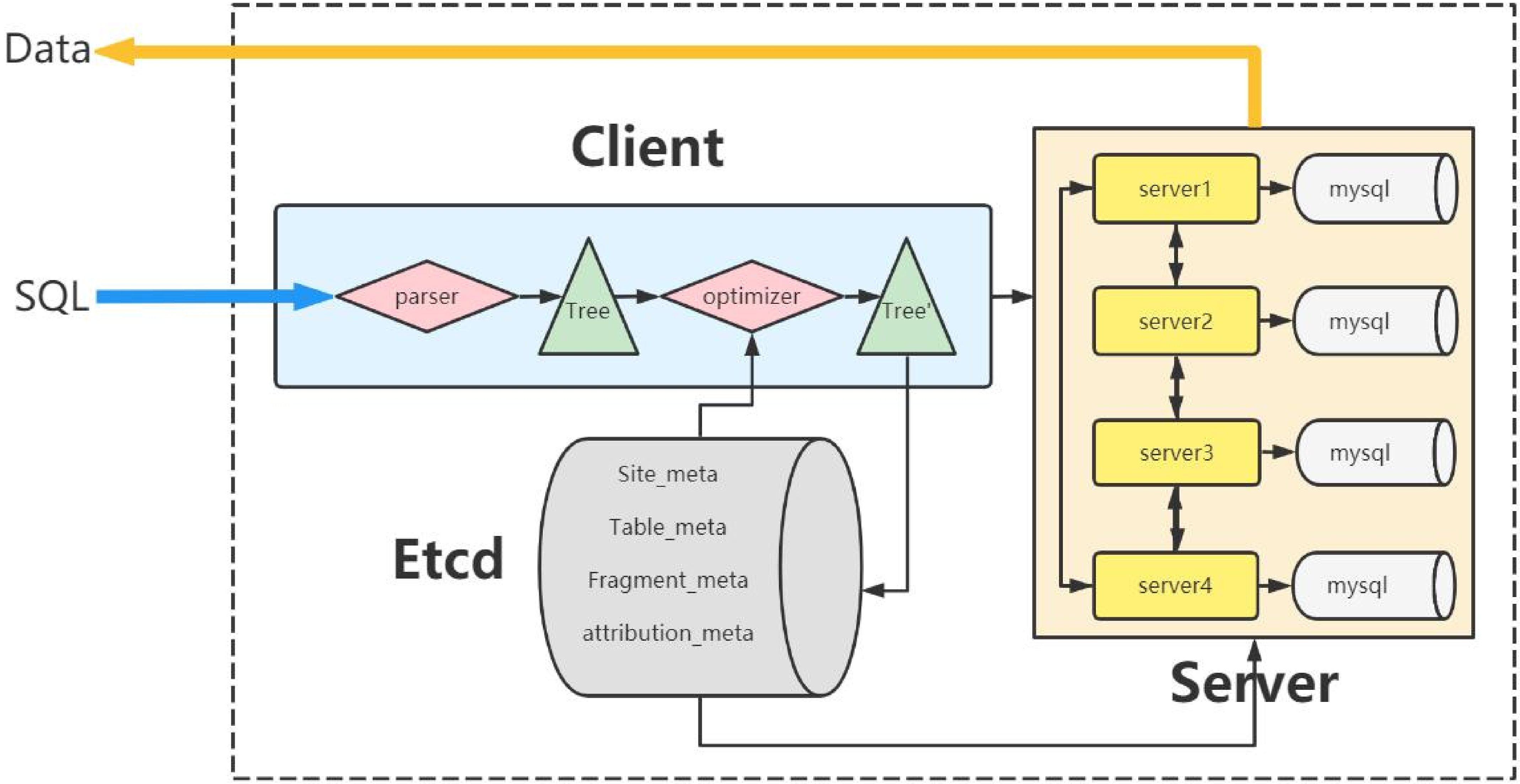
a Robust Performant SQL

薛钦亮 王芃 刘佳伟

2021/11/30

Work Division & Schedule

	10.01-11.23	11.23-11.30	12.01-12.07	12.08-12.14	12.15-12.21	12.22-12.28
薛钦亮		configure the environment	network and framework			
刘佳伟	learning the basic skills	learning SPJ and the excution	generating and excuting a tree	more data and tables	more complex sql and situation	optimazing the speed
王芃		learning optimazition of query tree	optimazing tree			



RPQL DDB Framework

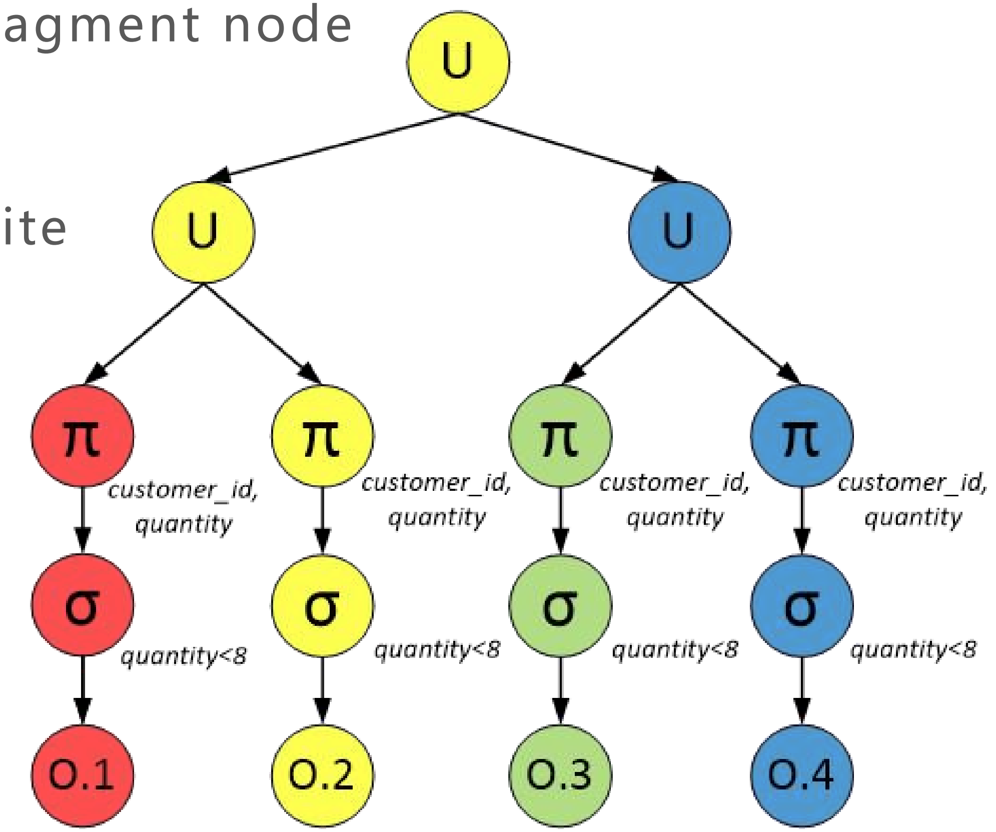
System Environment

- 3× Ubuntu 20.04 LTS in Windows 10 wsl
- python 3.8.0
- mysql 8.0.27
- ectd 3.5.1
- python package
  - grpcio == 1.42.0
  - grpcio-tools == 1.42.0
  - protobuf == 3.19.1
  - etcd3 == 0.12.0
  - PyMySQL == 1.0.2
  - sqlparse == 0.4.2

sqlparse

- parse the SQL to AST, then build query tree based on AST

- TreeNode: contain Union, Join, Project, Select, Fragment node
- The Union, Join node has two children
- Every node need to specify execution on which site



etcd

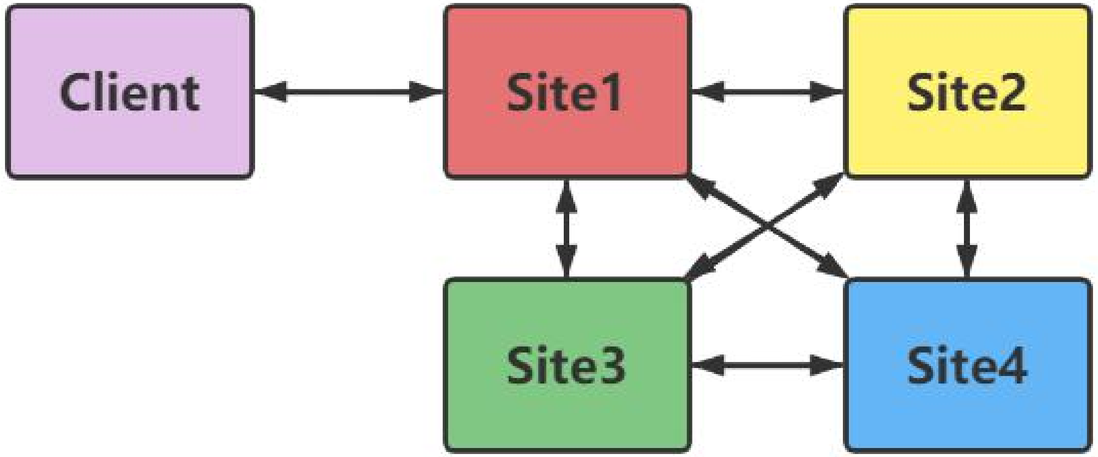
- start the etcd servce
- etcd used to store site information, table schema, attribution schema, and fragment information.
- query optimizer based on the query tree

```python
class table_meta:
    def __init__(self, name, field_meta_list, fragment_meta_list):
        self.name = name
        self.field_meta_list = field_meta_list
        self.fragment_meta_list = fragment_meta_list


class field_meta:
    def __init__(self, name, attrtype, iskey=False, nullable=True):
        self.name = name
        self.attrtype = attrtype
        self.iskey = iskey
        self.nullable = nullable


class fragment_meta:
    def __init__(self, ip, port, db):
        self.ip = ip
        self.port = port
        self.db = db
```


```json
{
    "SITES": [
        {
            "IP": "10.46.234.251",
            "PORT": "8883",
            "DB": "db1"
        },
        {
            "IP": "10.46.234.251",
            "PORT": "8885",
            "DB": "db2"
        },
        {
            "IP": "10.46.65.113",
            "PORT": "8885",
            "DB": "db1"
        }
    ]
}
```

grpc

- server started based grpcServer
- define the type of request data and response data
- client connect the server using net_pb2_grpc.NetServiceStub



the black line represent the connection of GRPC

run

- start etcd and init etcd with sites information
- start server and connect each other through grpc
- start client and choose a server to connect randomly
- excute the sql

```
(rpql) root@LAPTOP-V3K2ANAE:~/DDB_RPQL/client# python clientmaster.py
> select * from Publisher limit 10;
[('10.46.234.251', '8883', 'db1'), ('10.46.234.251', '8885', 'db2')]
| id     | name              | nation  |
| 100001 | Publisher #100001 | USA    |
| 100002 | Publisher #100002 | USA    |
| 100003 | Publisher #100003 | USA    |
| 100006 | Publisher #100006 | USA    |
| 100010 | Publisher #100010 | USA    |
| 100011 | Publisher #100011 | USA    |
| 100014 | Publisher #100014 | USA    |
| 100016 | Publisher #100016 | USA    |
| 100017 | Publisher #100017 | USA    |
| 100018 | Publisher #100018 | USA    |
```

Query Optimization

After parsing, we get all:

- (final) needed attributes
- relative tables
- select conditions
- join conditions

Steps:

1. For the relative vertical fragments, prune the attributes which will not be used in the query.
2. For the rest vertical fragments, if there are selections on the attributes, first select, then join the fragments.
3. For the relative horizontal fragments, prune the ones which conflict with the select conditions.
4. For the rest horizontal fragments, project the attributes which will be used (keys for join and final select columns)
5. If the ranges of two attributes for join in two fragments are conflicting, it can be eliminated.
6. Using Distributed INGRES QOA algorithm to determine the join processing sites.
7. Union the results of all sites.

Example

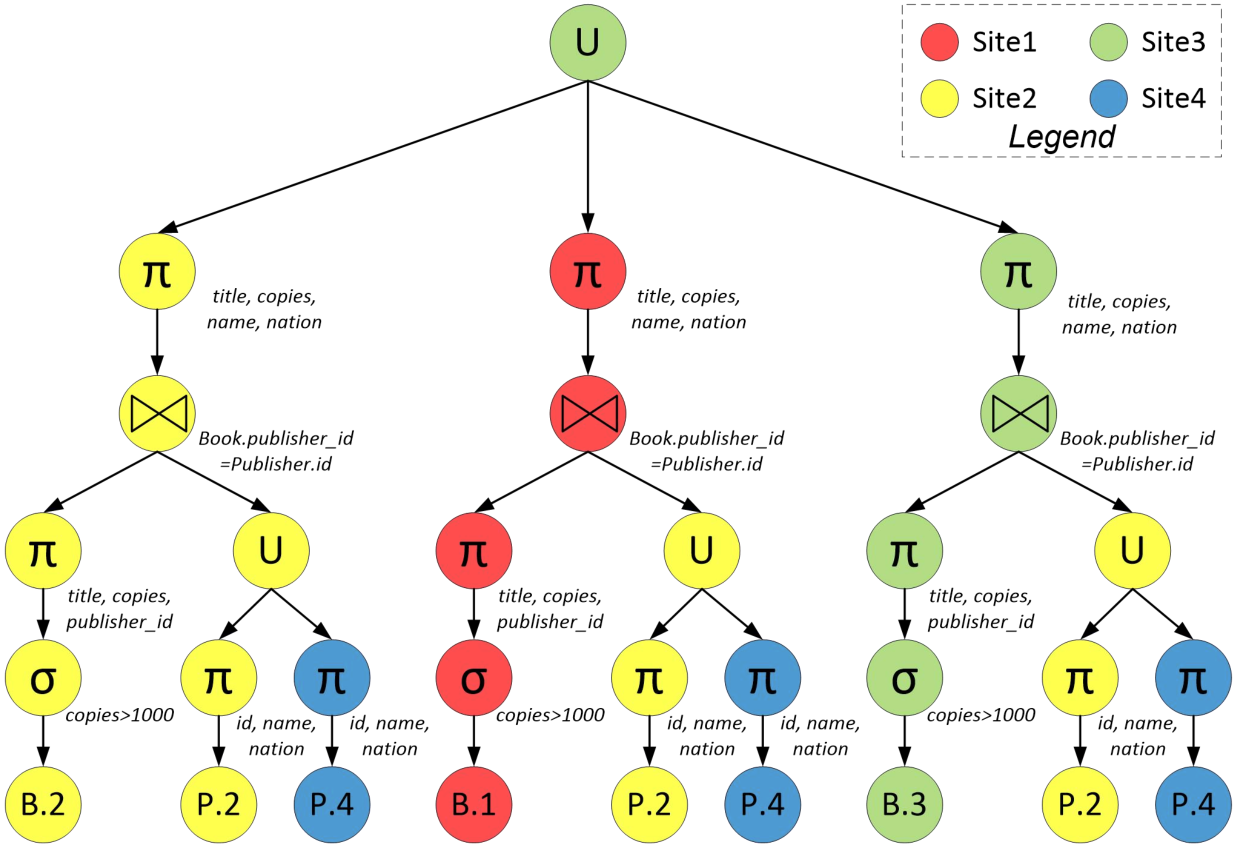
select Book.title,Book.copies,Publisher.name,Publisher.nation

from Book,Publisher

where Book.publisher_id=Publisher.id

and Book.copies > 1000

and Publisher.nation='USA'



Structures

Tree

nodes: List of Node
root: int //index of the root node

Node

index: int
type: string //fragment select project union join
parent: int
children: list of int
tables: list of string
site: int
size: int //num*size per record
fragment_id: int
select_condition: list of Selection
projection: list of Attribute
join: list of Join
union: list of string

Fragment

index: int
table: string
hor_or_ver: int
hori_condi: list of Selection
veri_condi: list of Attribute
......

Selection

attribute: Attribute
operation: int // > = <
num_value: float
str_value: string

Join

leftattr: Attribute
rightattr: Attribute

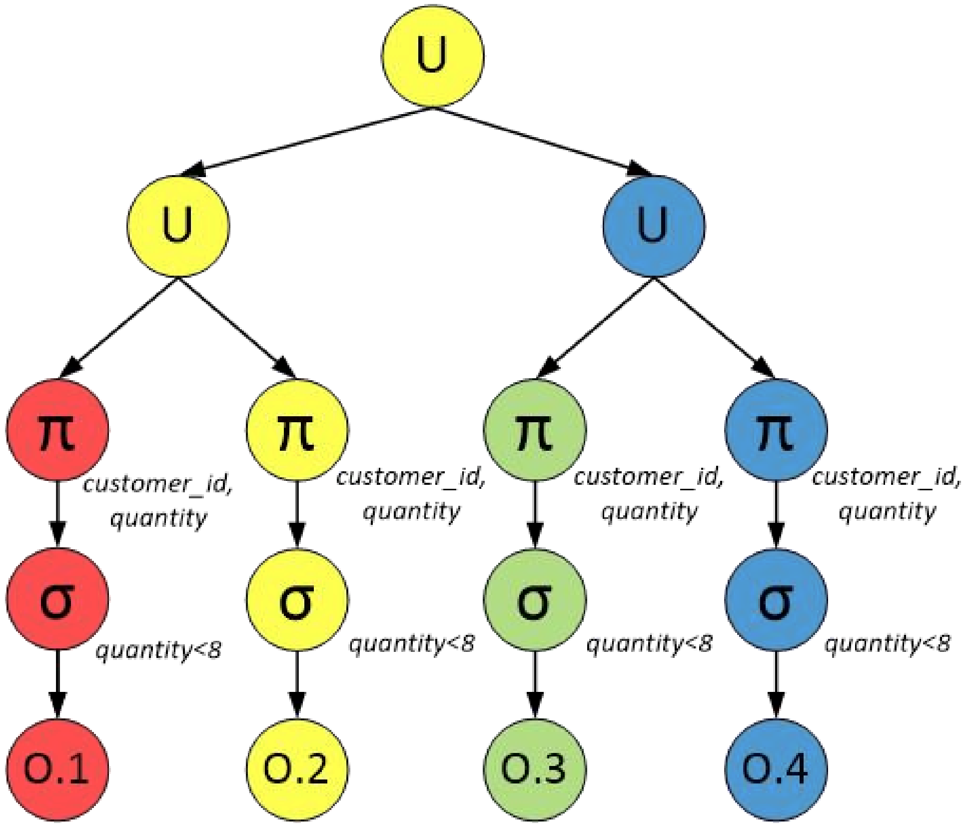
Attribute

table: string
attr: string

SQL Execute – DFS

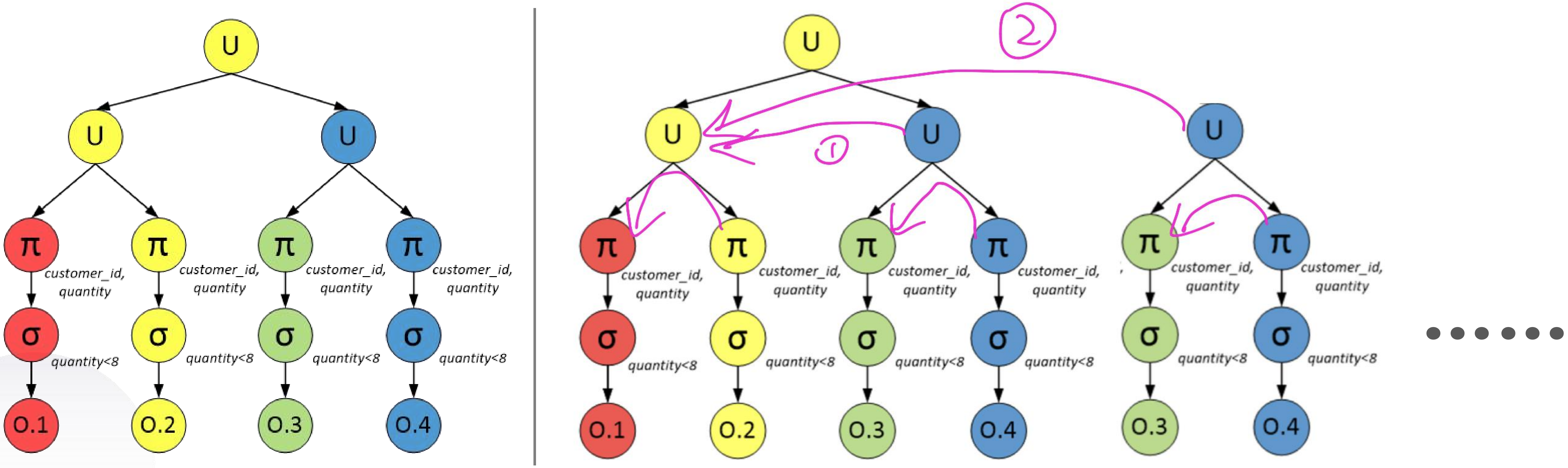
- DFS SQL Parse Tree
  - sent SQL order when SEARCHING
  - get SQL result when BACKTRACKING

- Node in the Tree contains:
  - Type (fragment, select, project, join, union)
  - Parent & Kids
  - MY site no. & size
  - attributes about SPJ



SQL Execute – DFS

- Union
  - 2 kids: transport Data on RIGHT kid to LEFT kid
  - more kids: transport each kid to the first kid successively



SQL Execute – grpc

- Use a CLASS enclosing the message between servers by grpc
- Every write have a REPLY

SQL Execute – parallel optimization

- Reality
  - MORE than 2 servers
  - MORE than 2 kids of each node in SQL Parse Tree
  - Far More than 2 cores of Processor
- Parallel
  - Every Search in DFS start a new THREAD
  - Use Thread Pool