

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

DATA INFRASTRUCTURE FOR THE CLOUD

Nicolas.travers@devinci.fr

NICOLAS TRAVERS

Full Professor of Computer Science at ESILV

Deputy director of the DVRC laboratory

Habilitated (HDR) in Computer Science from Sorbonne University in 2018. PhD thesis from Paris Saclay (UVSQ) in 2006.

Main topics: database management and optimization.
Specialization:

- Distributed databases,
- Graph databases,
- Information Retrieval
- Recommendation systems
- Ad hoc databases design in a transdisciplinary approach: Digital Tourism & Geography, Knowledge Management & History, IR & Musicology or Multimodal libraries.

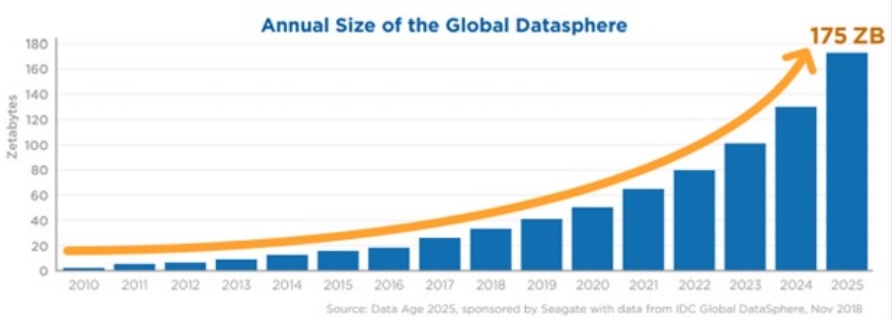


LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

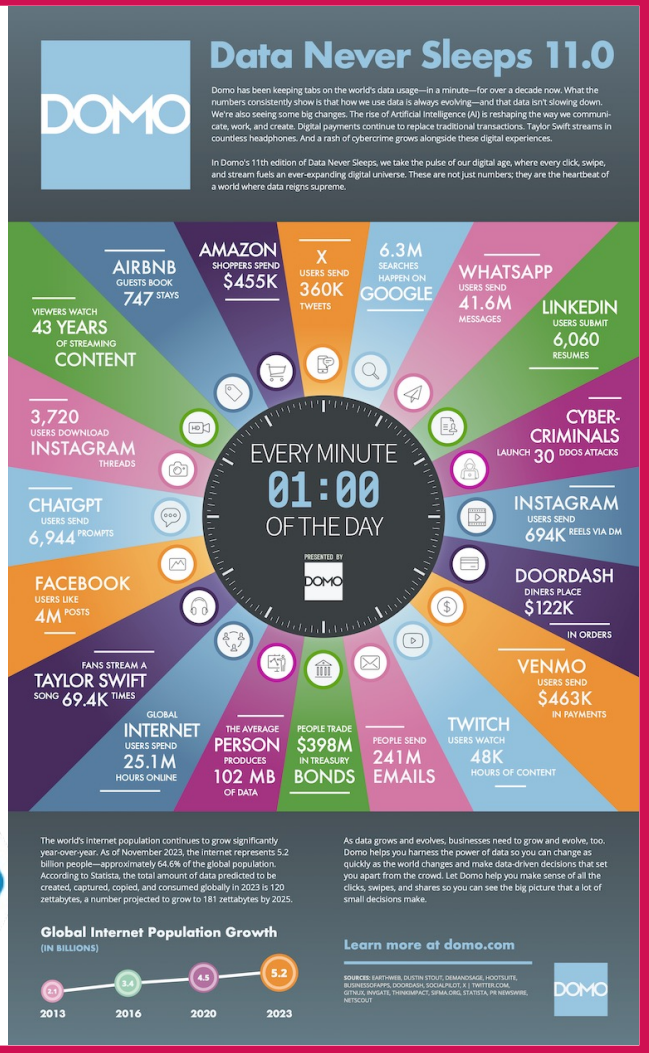
THE BIG DATA CONTEXT

Applications and Web platforms

- Exponential growth of the amount of Data x2 / 2 years
- Unprecedent management of this volume
- Need distribution: Computation & Data
- Clusters: Huge number of servers (Google, Amazon, Facebook, etc.)
  - DataCenter : ~5000 servers/data center
  - Google : ~1M de servers







LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

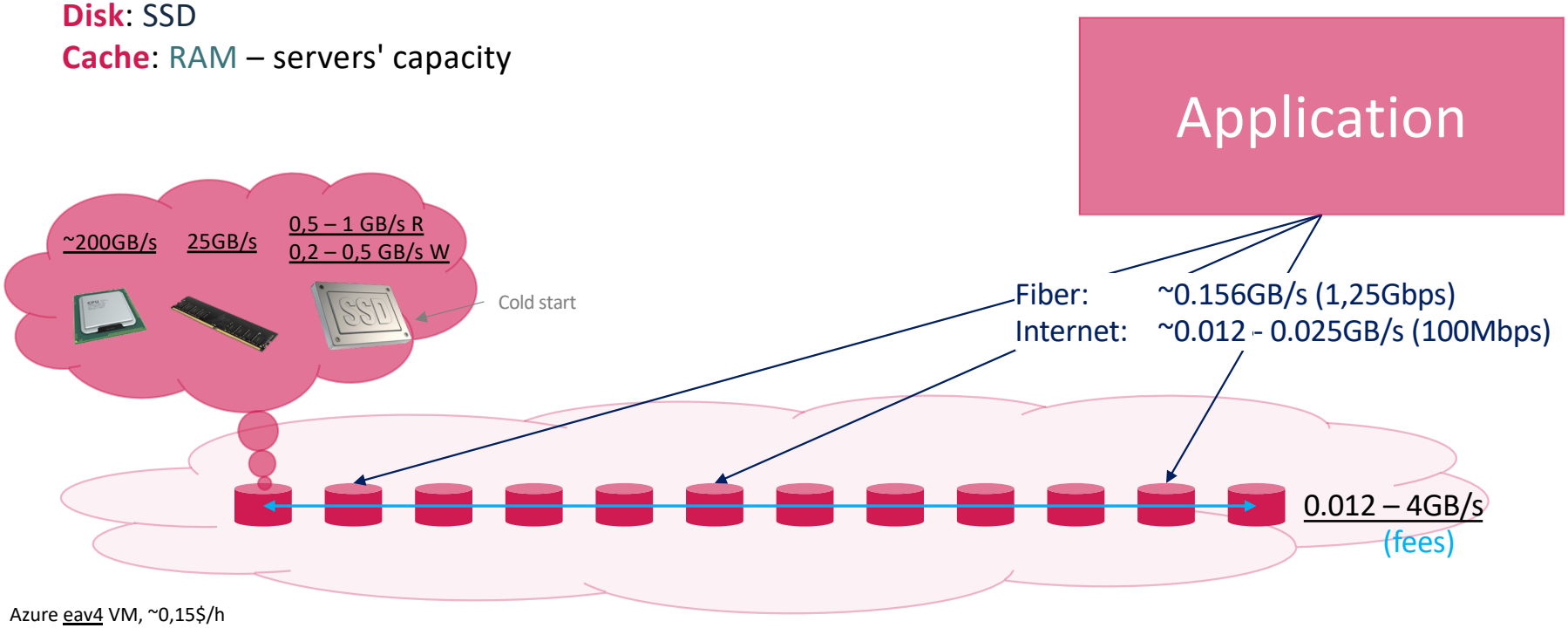
DBMS VS DISTRIBUTION

- **Pros DBMS**
  - Joins between tables
  - SQL: Rich query language
  - Constraints: integrity, data types, normal forms
- **Cons for Distribution**
  - How to partition/place data?
  - How make joins?
  - How to manage fault tolerance?

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

DISTRIBUTED ENVIRONMENT WHICH COST?

Network: inter communications + intra communications
Disk: SSD
Cache: RAM – servers' capacity



Azure eav4 VM, ~0,15$/h

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

WHAT IS NOSQL?

NoSQL : Not Only SQL

- New data storage/management approach
- Scales up the system (through distribution)
- Fault tolerant architecture (replication)
- Complex metadata management
  - Schema-Last approach

Do not substitute DBMS!!

NoSQL's target:

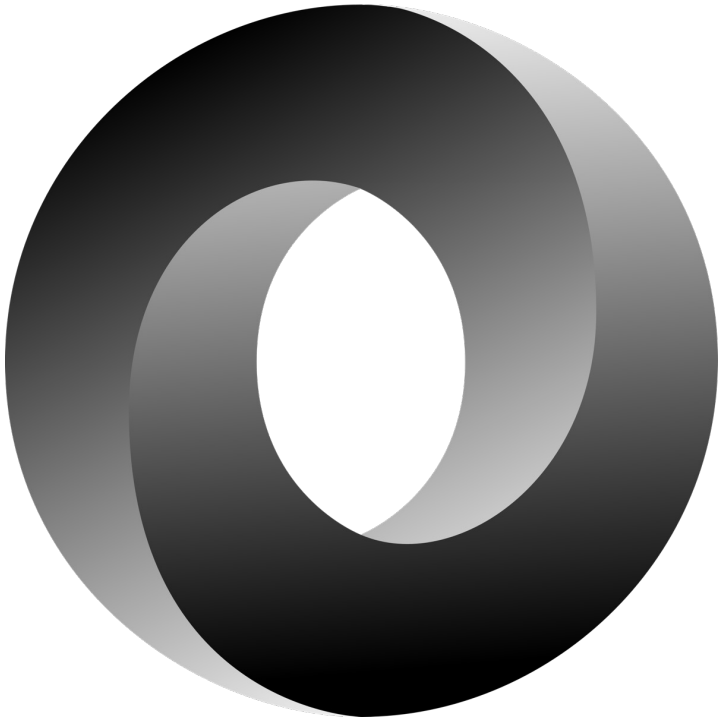
- Schema-first with SQL
- Very huge volume of data (PetaBytes)
- Very short response time
- Consistency is not mandatory



LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

NOSQL CHARACTERISTICS

- No more relations
  - No schema (hardly data types)
  - Dynamic schema
  - ➢ Collections
- Schema-Last
  - No more tuples (1st & *3rd Normal Forms)*
  - Nesting
  - Arrays
  - ➢ JSON documents
- Distribution
  - Parallelism (abstraction of Map/Reduce, not cloud computing)
- Data replication
  - Availability vs Consistency
  - No more transactions
  - ➢ Few writes, many reads



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

ACID VS BASE

Local DBMS: **ACID** transactions

- **A**tomicity: integral completion or none
- **C**onsistency: consistent at start and end
- **I**solation: no communication between them
- **D**urability: an operation cannot be reversed

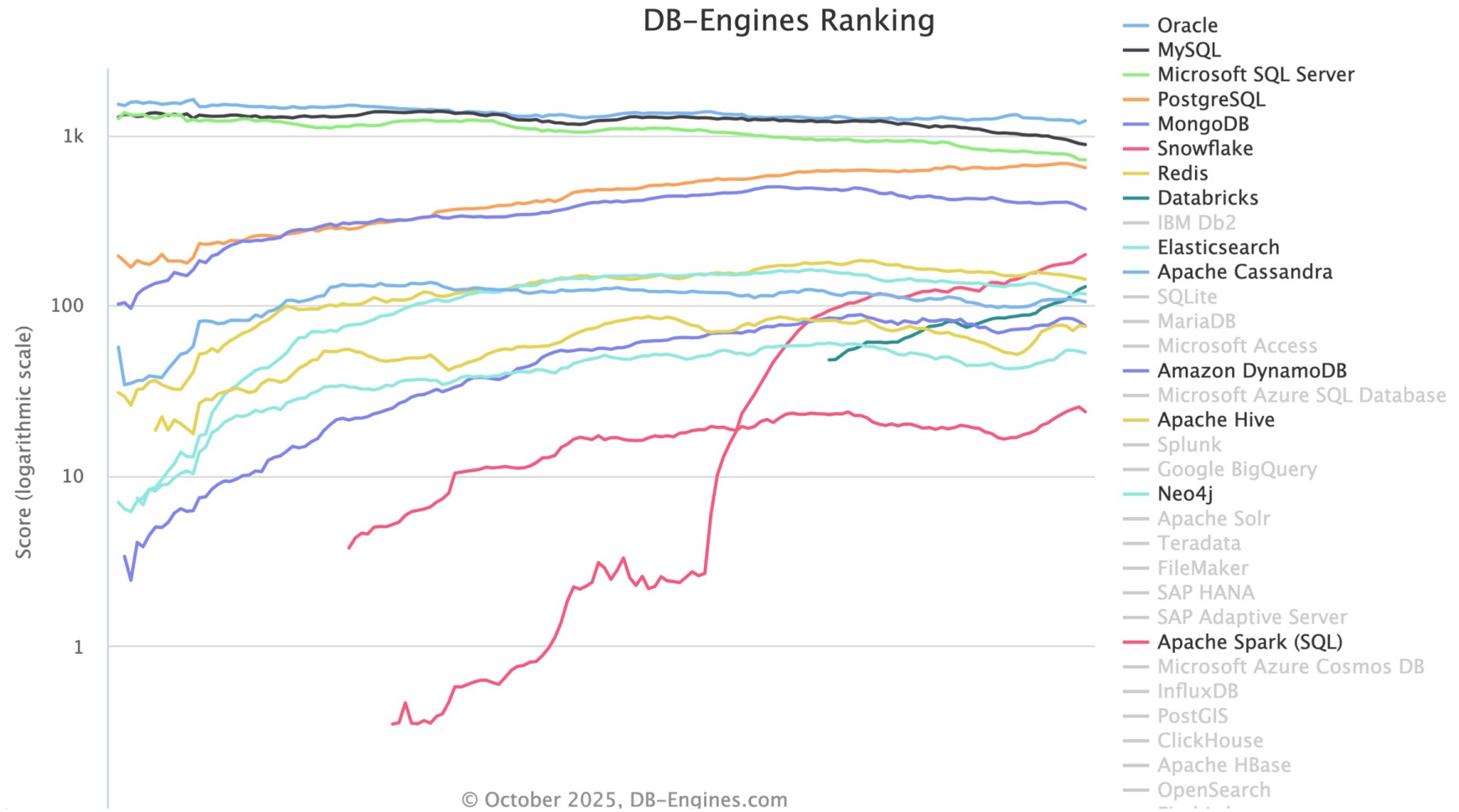


Distributed systems: **BASE**

- *Basically Available* :
  - Any request => An answer
  - Even in a changing state
- *Soft-state* :
  - Opposite to Durability.
  - System's state (servers or data) could change over time (without any update)
- *Eventually consistent* :
  - With time, data can be consistent
  - Updates have to be propagated

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

https://db-engines.com/en/ranking



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr



JOINS, DENORMALIZATION & SCHEMA

NOSQL VS JOINTURES

Distributed context incompatible with links between data collections

- Need to reduce join cost
- Cost model
- Multi-server join

Considering two data collections

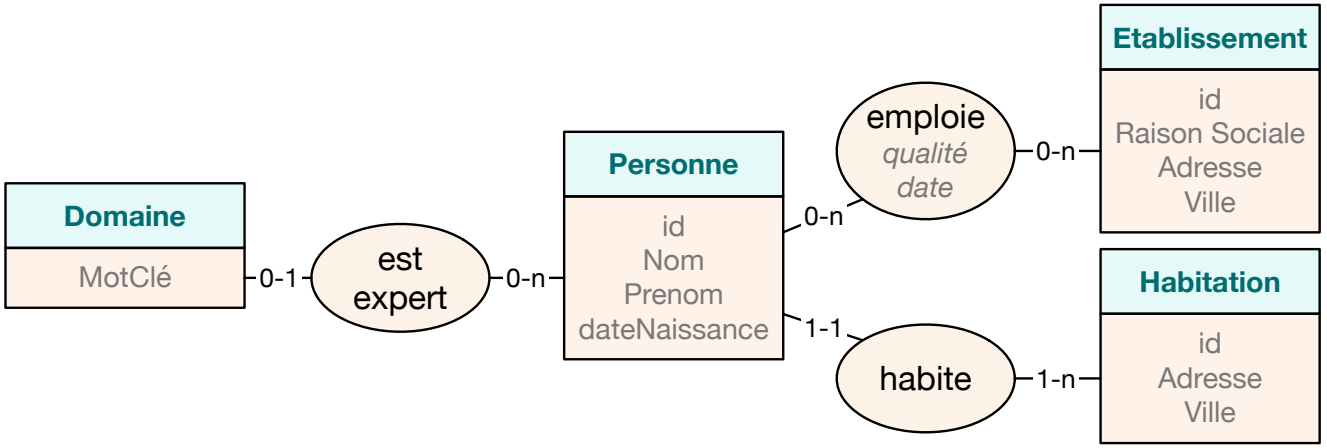
- Joins between two collections
  - On which server is the join performed?
  - Distribution of data to be joined?
- Network cost problem (more expensive than local)

➢ *Denormalizing schema for join reduction*

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

SCHEMA DENORMALIZATION

- Which centrale entity to choose?
- How to merge?
- Which impact?



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

DENORMALIZATION: METHODOLOGY

- Query-Driven [Chebotko15]
- Document-oriented NoSQL modelling [Mason15]
- Model-Driven-Architecture [Abdelhedi et al. 17, Mali et al. 20, Mali et al. 22]
- Goal:
  - Avoid costly joins
  - Combine barely updated data and/or independent data

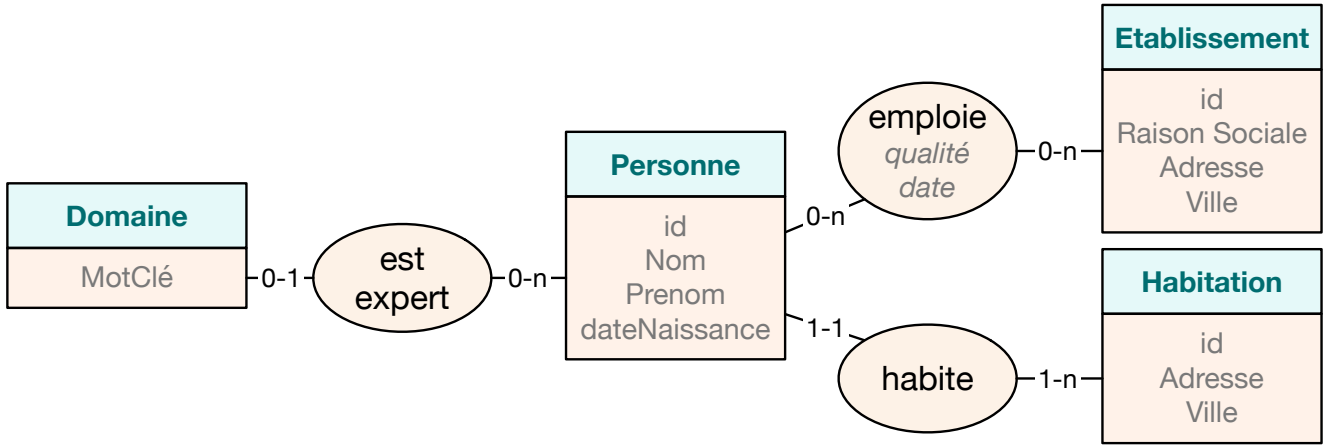
Consistency issues!

1 – [Chebotko15] https://pdfs.semanticscholar.org/22c6/%20740341ef13d3c5ee52044a4fbaad911f7322.pdf
2 – [Mason15] http://proceedings.informingscience.org/InSITE2015/InSITE15p259-268Mason1569.pdf
3 – [Abdelhedi et al. 17] https://cedric.cnam.fr/index.php/publis/article/AAA17
4 – [Mali et al. 20] – https://link.springer.com/content/pdf/10.1007%2F978-3-030-59003-1_9.pdf
5 – [Mali et al. 22] - https://link.springer.com/chapter/10.1007/978-3-031-05760-1_31

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

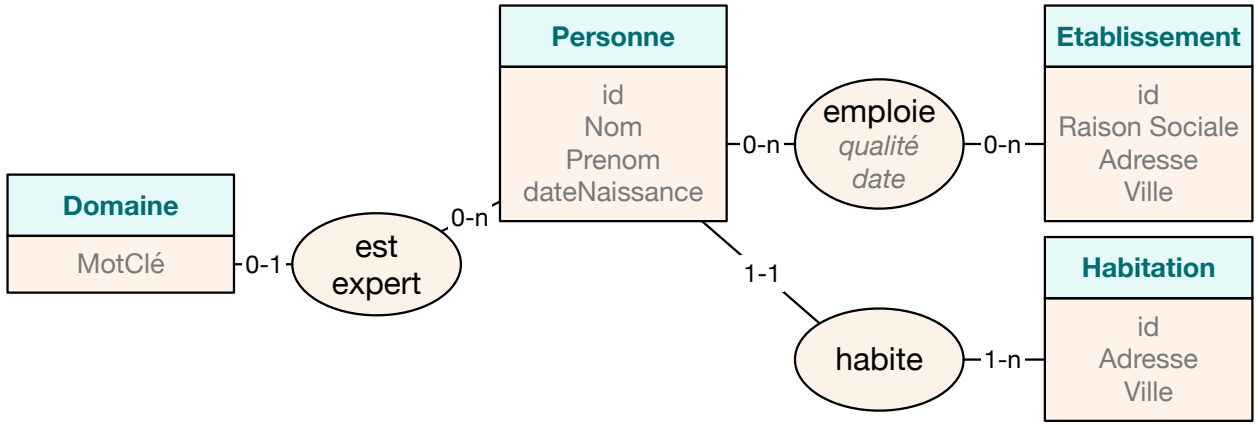
DENORMALIZATION: OPERATIONS

- Merge (A{B} & B[{A}])
- Split (A_1 & A_2)
- Materialization (aggregates)
- Surcharge (duplicates)



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

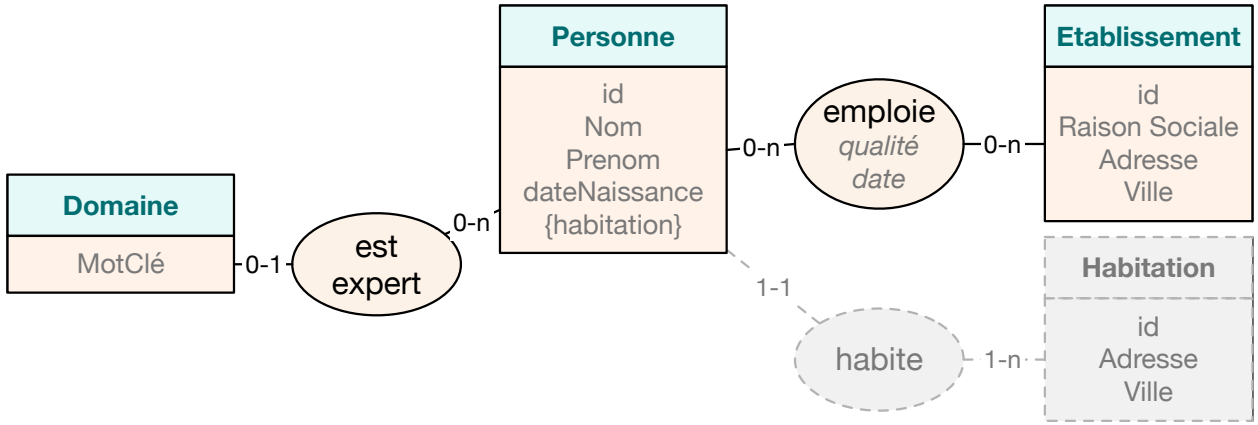
DENORMALIZATION: MODELISATION STEPS



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

DENORMALIZATION: MODELISATION STEPS

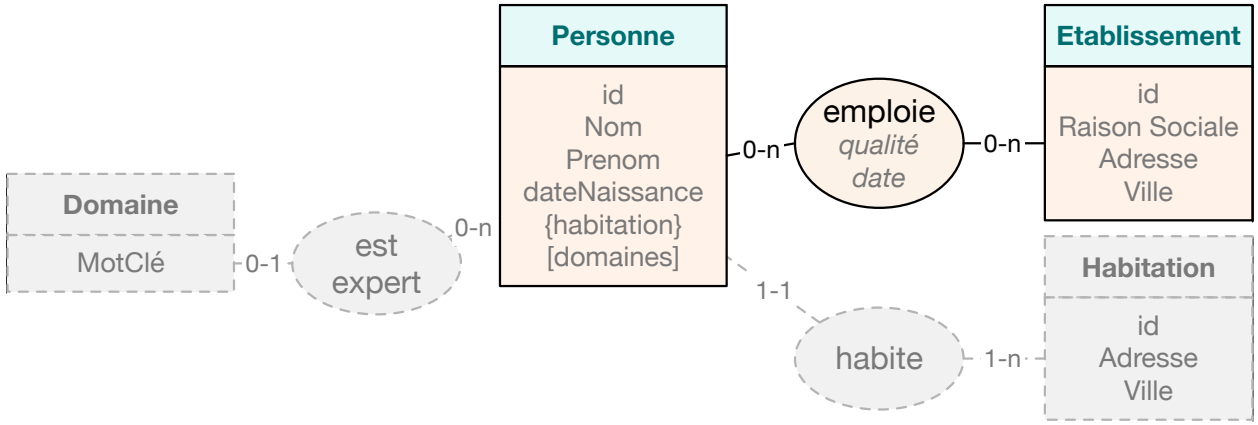
- Merges:
    1. Frequent queries *(e.g., join between Personne & Habitation)*



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

DENORMALIZATION: MODELISATION STEPS

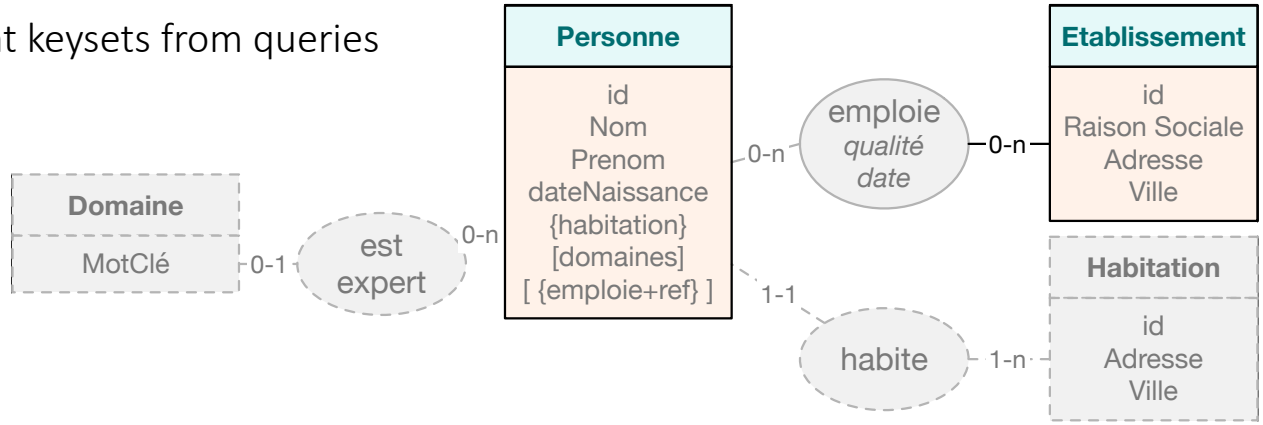
- Merges:
  1. Frequent queries *(e.g., join between Personne & Habitation)*
  2. Independent data *(e.g., Domain in Personne)*



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

DENORMALIZATION: MODELISATION STEPS

- Merges:
  1. Frequent queries *(e.g., join between Personne & Habitation)*
  2. Independent data *(e.g., Domain in Personne)*
  3. For 0-n cardinalities on both sides:
     1. Nesting the foreign key *(e.g., employer id in the list)*
- *Splits:*
  1. Independent keysets from queries



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

DENORMALIZATION: MERGE

- Nb merges, Fubini number =

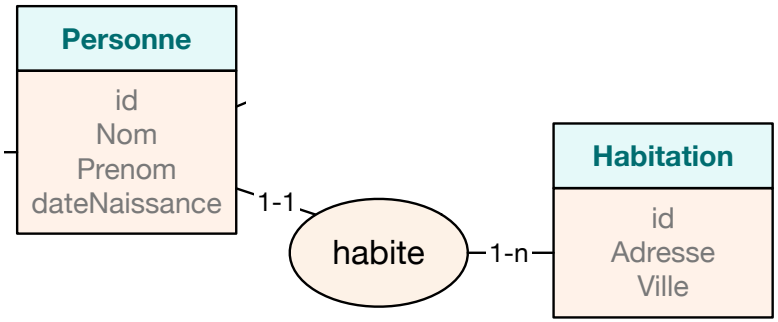
$$Fn_{|\mathcal{L}|} = \sum_{k=0}^{|\mathcal{L}|} k! \times \binom{|\mathcal{L}|}{k}$$

⇒ *Document/Wide-columns Oriented*

P{H}: {"idP":1, "habitation”: {"idH":10,…}}
{"idP":11, "habitation”: {"idH":10,…}}

H[{P}]: {"idH":10, "habitants":[
{"idP":1, …},
{"idP":11, …},
…]}

[Mali et al. 22]



LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

DENORMALIZATION: SPLIT

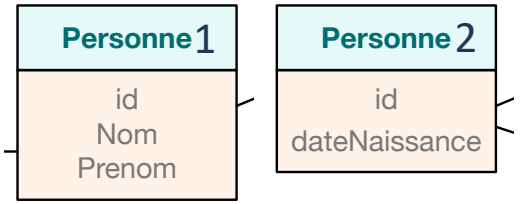
- Nb splits, Bell number:

$$B_{|keys(r)|} = \sum_{k=0}^{|keys(r)|} \binom{|keys(r)|}{k} \times B_{|keys(r)|-1}$$

=> Columnar oriented (eventually wide-column)

P1: {"idP":1, "Nom":"Skywalker", "Prenom":"Luke"}

P2: {"idP":1, "dateNaiss":"19 av. BY"}



[Belbachir et al. 21]
* Dependent on nb keys (without Primary Key)

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

DENORMALIZATION: COMBINAISON

- Nb merges: $$Fn_{|\mathcal{L}|} = \sum_{k=0}^{|\mathcal{L}|} k! \times \binom{|\mathcal{L}|}{k}$$
- Nb splits: $$B_{|keys(r)|} = \sum_{k=0}^{|keys(r)|} \binom{|keys(r)|}{k} \times B_{|keys(r)|-1}$$
- Nb denormalizations: $$|\mathcal{M}| = (Fn_{|\mathcal{R}|} \times d) \times \prod_{k=1}^{|\mathcal{R}|} B_{|keys(r_k)|}$$

possible splits among rows

possible merges per split DM

Hierarchy depth impacting number of possible merges vs splits

$$d = \sum_{k=1}^{|\mathcal{R}|} keys(r_k) - |\mathcal{R}| + 1$$

[Mali et al. 24, Mali et al. 25]

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

IMPACT ON DATABASE SIZE

Data types:

- Numbers: 4B
- VARCHAR : ~8B -> 200B
- Key/value pairs: 12B
- Size ~ $\sum_{k=0}^{n}(size_{kv} + size_{k})$

Vary with merges and splits

For each collection (C)

- $vol_{DB} = \sum_{c=1}^{C} size_{doc}(c) * \#doc(c)$

Cluster dimensioning:

- $\#servers = max\left(\left\lceil \frac{vol_{DB} * 3}{capacity_{storage}} \right\rceil, \left\lceil \frac{vol_{DB}}{capacity_{RAM}} \right\rceil\right) * 2$

For database evolution purpose

Monthly cost:

- $price(DB) = price(s) * \#servers + externalFees * \sum_{q=1}^{Q} vol_{external\ network}(q) * freq(q)$

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

DENORMALIZATION: CONCLUSION

- Complex to determine a good denormalization
  - Tradeof between efficiency and size
  - Materialization / Surcharge
  - Beware of updates – inconsistency (*no normal forms*)

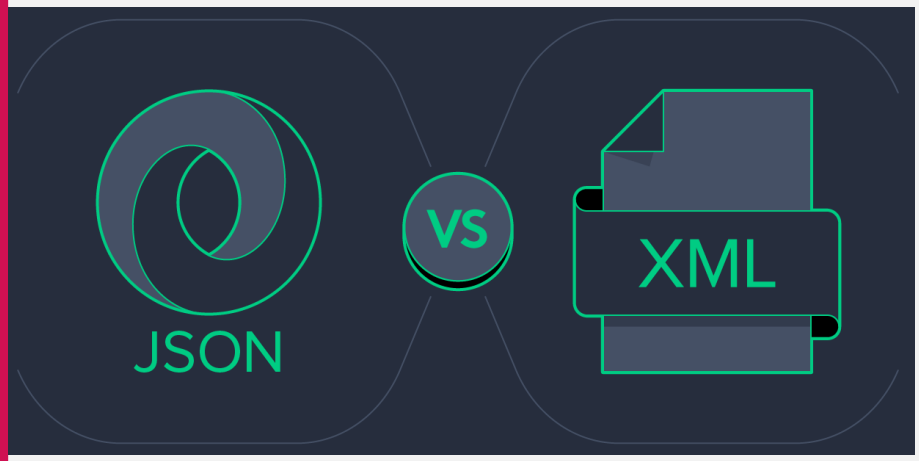
➢ The use case can guide the denormalization

- Data models compatible with queries
- How to handle complex and evolving use case?

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr



DOCUMENTS & JSON



JSON: JAVASCRIPT OBJECT NOTATION

- XML initially designed for standardizing computer communications
  - Too verbose & space greedy
  - Dedicated to *Web Services*

vs

- JSON (JavaScript Object Notation)
  - Designed for web server to web browser communications
  - Lightweight, text oriented, language independent
  - Used for APIs (Google API, Twitter API) or dynamic web (Ajax)

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

JSON: (TOO) SIMPLE GRAMMAR

J ::=	$B \mid O \mid A$	JSON expressions
B::=	null|true|false|n|s	Basic values
	$n \in$ Num,$s \in$ Str	
O ::=	$\{l_1 : J_1,\ldots,l_n : J_n\}$	Objects
	$n\geq 0$, $i\neq j \Rightarrow l_i \neq l_j$	
A::=	$[J_1,\ldots,J_n]$	Arrays
	$n\geq 0$	

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

JSON: KEY-VALUES & TYPING

Key + Value

- "lastname" : "Travers"
- Keys with **quotations**

Identifiers

- **" _id "** commonly used
- Overwrite already stored ids
- Can be automaticaly generated
  - Ex MongoDB: "_id" : ObjectId(1234567890)

Objects/Documents

- Collection of **key/values – unique keys**

```
{ "_id" : 1234,
  "lastname" : "Travers",
  "firstname" : "Nicolas",
  "kind" : 1}
```

Scalar

- String, Integer, float, boolean, null…

Documents

- Objects **{…}**

List

- Arrays **[…]**
- No typing
  - "lessons" : **[** "SQL", 1, 4.2, null, "NoSQL" **]**
- Can nest documents
  - "doc" : **[**{"test" : 1},
    {"test" : {"nesting" : 1.0}},
    {"key" : "text", "value" : null} **]**

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr

JSON: COMPLETE EXAMPLE

```
{
        "_id" : 1234,
        "lastname" : "Travers", "firstname" : "Nicolas",
        "employers" : [
                { "company" : "ESILV", "starting_date" : "2018-09",
                  "location" : {"street" : "12 avenue Léonard de Vinci","city" : "Paris La Défense", "zip" : 92916}},
                { "company" : "CNAM", "starting_date" : "2007-09", "end_date" : "2018-08",
                  "location" : {"street" : "2 rue Conté","city" : "Paris", "zip" : 75141}},
                { "company" : "UCP", "starting_date" : "2006-09", "end_date" : "2007-08",
                  "location" : {"street" : "2 rue Conté","city" : "Cergy-Pontoise", "zip" : 91000}},
                { "company" : "UVSQ", "starting_date" : "2004-01", "end_date" : "2006-08",
                  "location" : {"street" : "45 avenue des Etats-Unis","city" : "Versailles", "zip" : 78000}}    ],
        "fields" : [ "DB" , "DB optimization", "XML", "NoSQL", "IR" ],
        "hobbies" : ["Star Wars", "BZH"]
}
```


LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr



JSON SCHEMA

JSON SCHEMA

- Needs a Schema to validate documents' structure
- Automatic check (exists, inclusion, syntax, queries)

- Requires a typed grammar
  - Objects, arrays,
  - Mandatory/optional keys

http://json-schema.org/

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

JSON SCHEMA: OBJECTS

```
{
        "type" : "object",
        "properties" : {
                "key1" : {"type": ... },
                "key2" : {"type": ... }
        }
}
```

Object's keys

Types of values
"number" (option – "min", "max")
"string" (option - "pattern")
"anyOf"
"not"

Other options : "definitions" + "$ref"

JSON SCHEMA: KEYS

```
{
        "type" : "object",
         "properties" : {
                "key1" : {"type": … },
                "key2" : {"type": … }
        },
        "required" : ["key1"],
        "additionalProperties" : false
}
```

Mandatory keys
Missing = optional

Forbids additional keys

Other options : "minProperties", "maxProperties", "patternProperties"

WARNING! NEGATION

```
{
        "type": "object",
        "properties": {
                "key1": { "type": "integer" }
        },
        "not": { "required": [ "key1" ] }
}
```



```
"arrayKey" : {
        "type" : "array",
        "items" : [ {"type" : ... } ],
        "minItems" : 1,
        "maxItems" : 10
}
```

JSON SCHEMA : LISTS

Other options
"uniqueItems"
"additionalItems"

JSON SCHEMA : EXAMPLE

```
{ "type": "object",
 "properties": {
  "_id": { "type": "integer" },
  "lastname": { "type": "string" },
  "firstname": { "type": "string" },
  "employers": {
    "type": "array",
    "items": {
      "type": "object",
      "properties": {
          "company": { "type": "string" },
          "starting_date": { "type": "string" },
          "end_date": { "type": "string" } },
...
```

```
...
          "location": {
              "type": "object",
              "properties": {
              "street": { "type": "string" },
              "city": { "type": "string" },
              "zip": { "type": "integer" } },
            "required": [ "street", "city", "zip" ] },
        "required": [ "company", "starting_date", "location" ] } },
  "fields": { "type": "array", "items": { "type": "string" } },
  "hobbies": {
      "type": "array",
      "items": { "type": "string" } } },
  "required": [ "_id", "lastname", "firstname", "employers",
              "fields", "hobbies" ]
}
```


LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

JSON SCHEMA: MISCELLANEOUS

PROGRAMMING LANGUAGE-ORIENTED SPECIFICATION

JOI

https://github.com/hapijs/joi

SCHEMA EXTRACTION

https://www.liquid-technologies.com/online-json-to-schema-converter

https://app.quicktype.io/#l=schema

SCHEMA VALIDATION

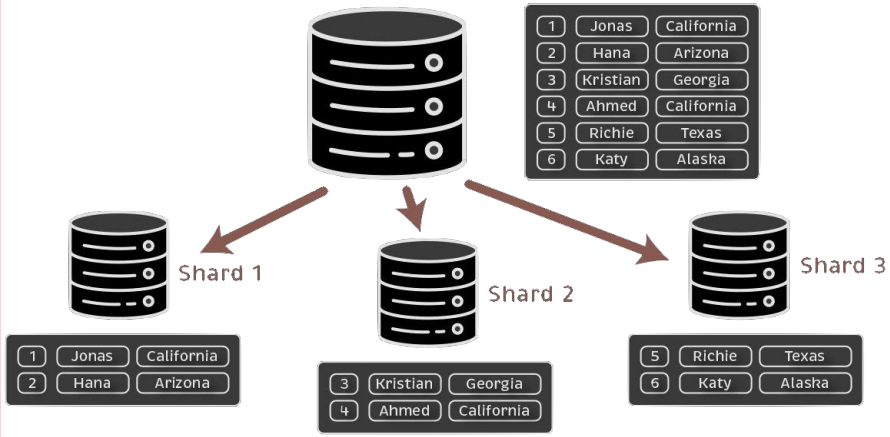
https://www.jsonschemavalidator.net/

JSON EXAMPLES GENERATOR

From a JSON Schema

https://jsonschematool.ew.r.appspot.com/

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr


1 Jonas California
2 Hana Arizona
3 Kristian Georgia
4 Ahmed California
5 Richie Texas
6 Katy Alaska
Shard 1
1 Jonas California
2 Hana Arizona
Shard 2
3 Kristian Georgia
4 Ahmed California
Shard 3
5 Richie Texas
6 Katy Alaska

SHARDING

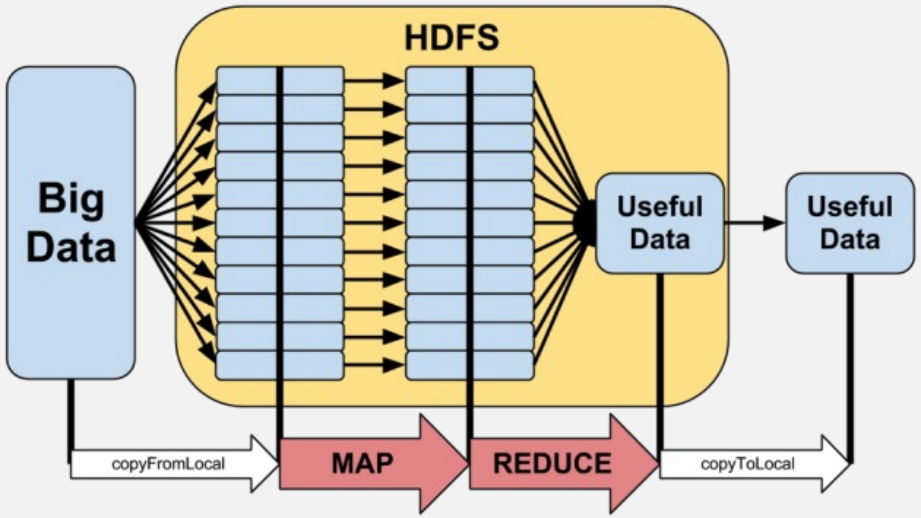
SHARDING: DATA DISTRIBUTION & REPLICATION STRATEGIES

- Files are segmented in chunks (64MB, 256MB…)
  - Chunks are distributed in the cluster
  - Data are placed according to a **sharding** strategy

- 3 types of technics of sharding:
  1. **HDFS:** Resource allocation based (racks, switches, datacenter)
     - *++ Fault tolerance and massive computations*
  2. ***Clustered index:*** Tree-based structure (total order)
     - *++ Physical data clustering and dynamicity*
  3. **DHT:** Hash-based structure
     - *++ elasticity and self-management*

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

SHARDING WITH HDFS



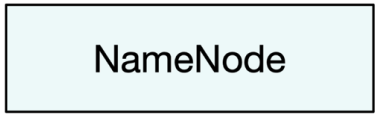
Resources allocation strategy

- Distributed file system
- Rely on servers load balancing
- Dedicated to fault tolerance
- Dynamic allocation and optimization

(1) See *Hadoop Distributed File System*
http://chewbii.com/transparents-hdfs-hadoop/

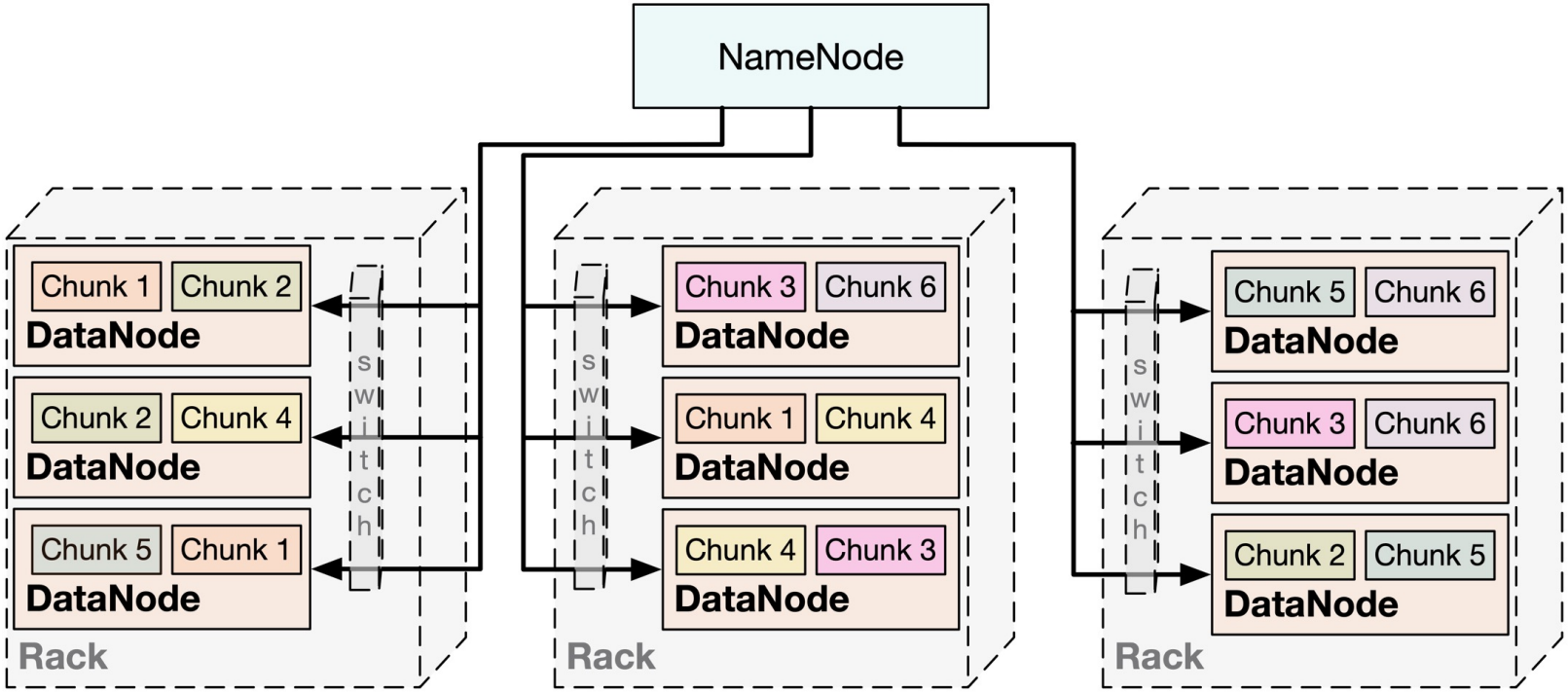
ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

SHARDING WITH HDFS: EXAMPLE



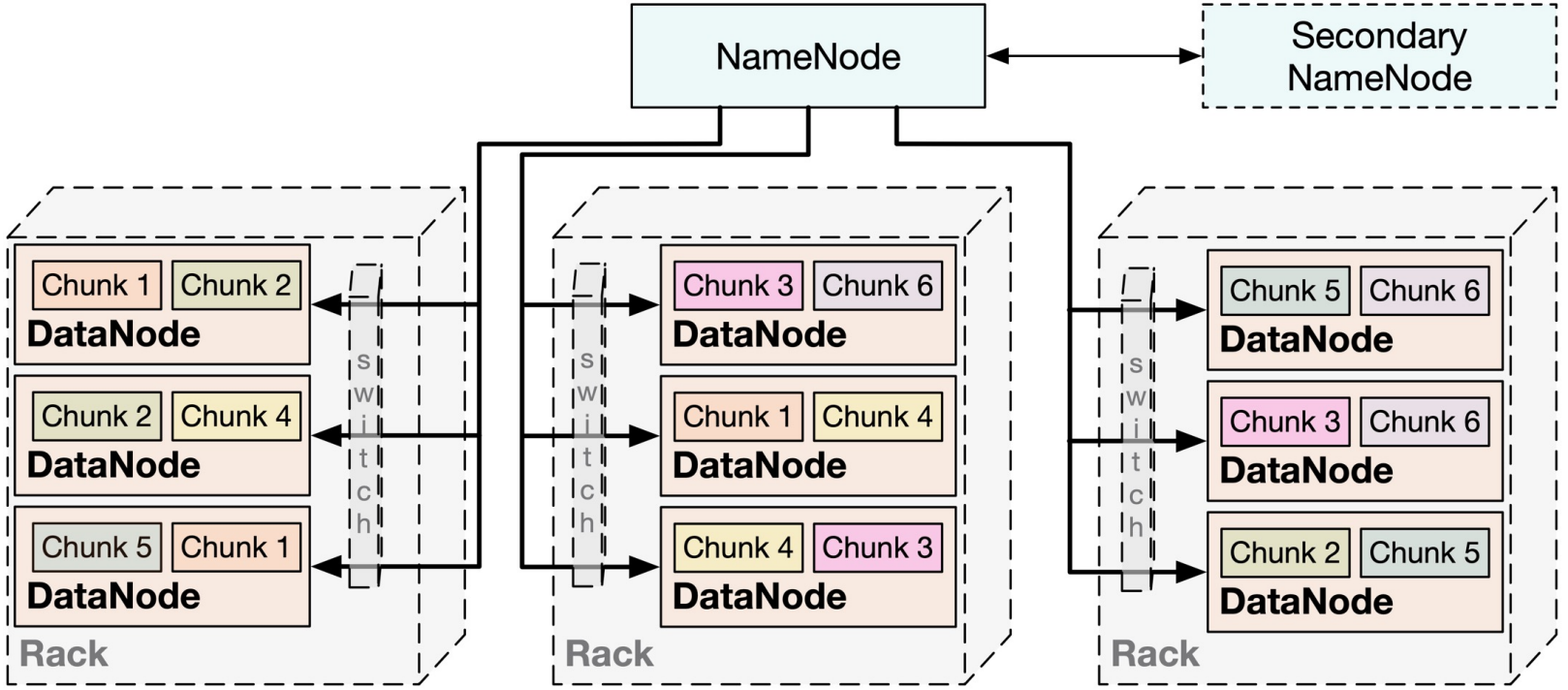
LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

SHARDING WITH HDFS: EXAMPLE



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

SHARDING WITH HDFS: EXAMPLE



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

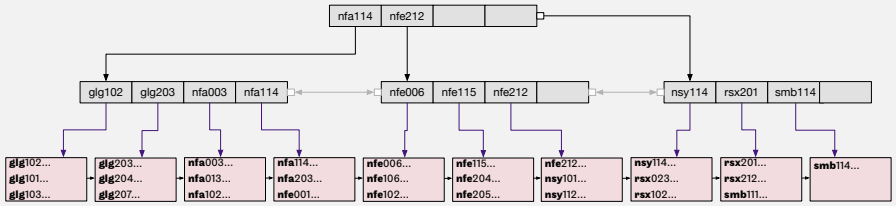
SHARDING WITH HDFS NOSQL SOLUTIONS

Highly distributed computation

Fault Tolerance



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr



SHARDING WITH CLUSTERED INDEX

Distributed Clustered Index

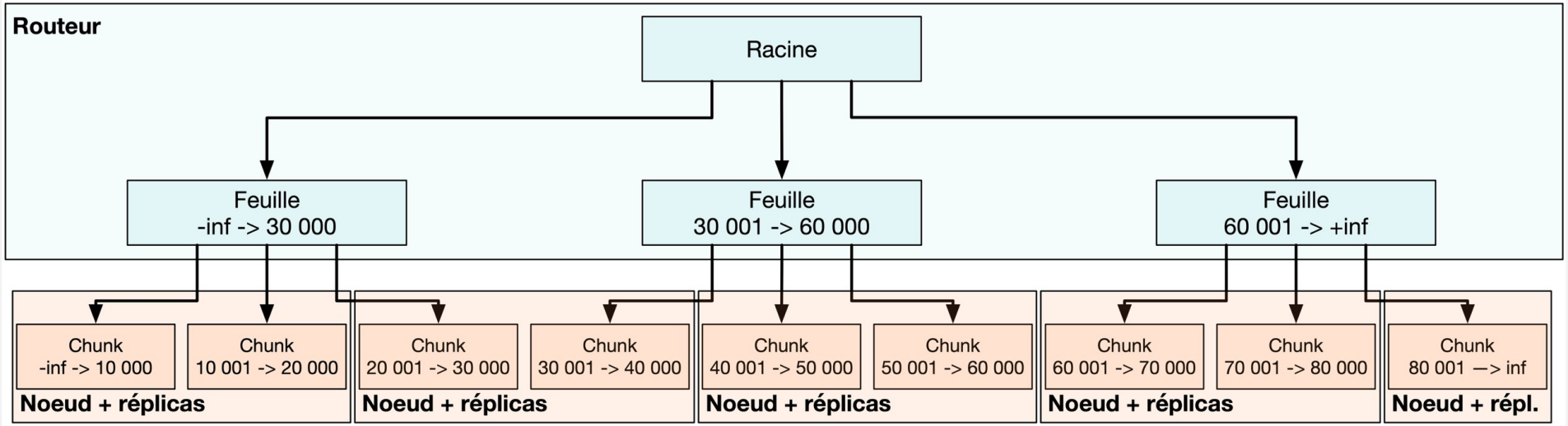
- Sorted data according to a key
  - Primary key
  - Build chunks (default 256Mo)
  - Distribute chunks
  - Replicate chunks

➡️ *Range* queries and *group by* queries

⚠️ Only one key can be chosen

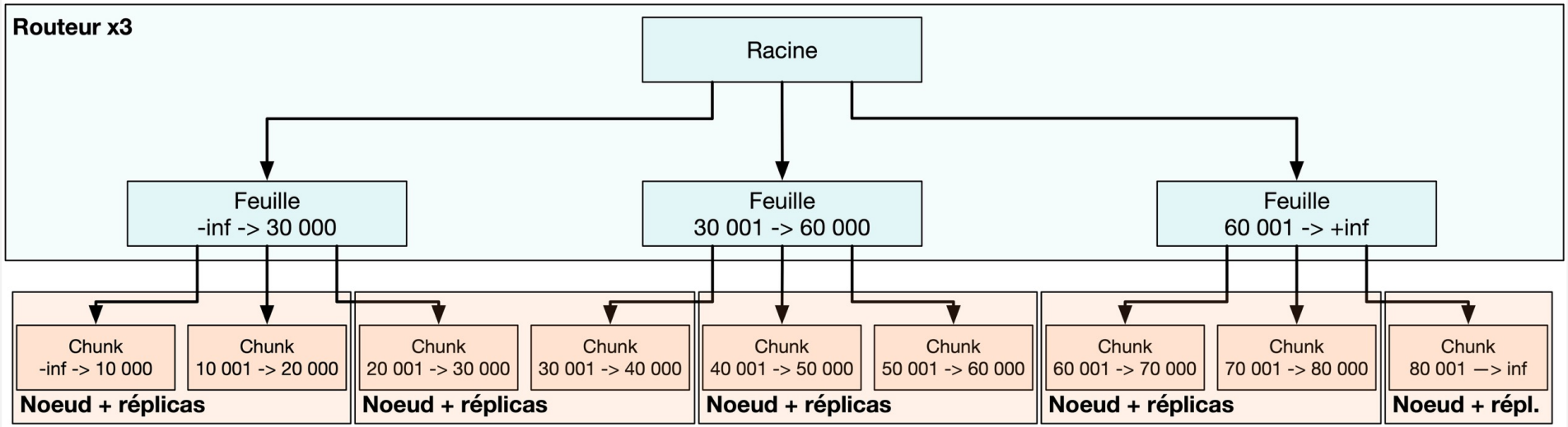
(2) See « index non-dense »
http://chewbii.com/videos-optimisation-bases-de-donnees/
(Vidéo 4)

SHARDING WITH CLUSTERED INDEX EXAMPLE



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

SHARDING WITH CLUSTERED INDEX EXAMPLE



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

SHARDING WITH CLUSTERED INDEX SOLUTIONS

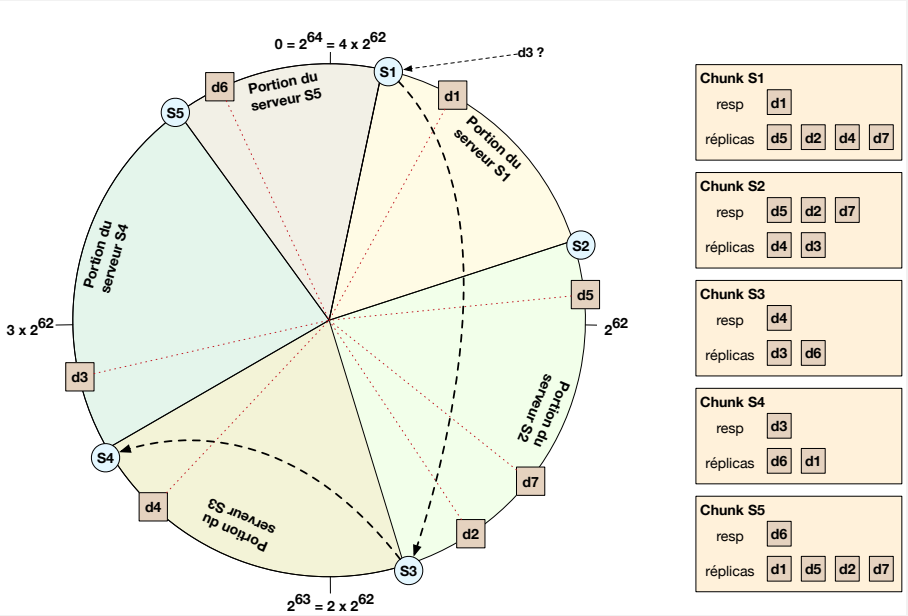
Data organization

Complex queries



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

SHARDING WITH DHT

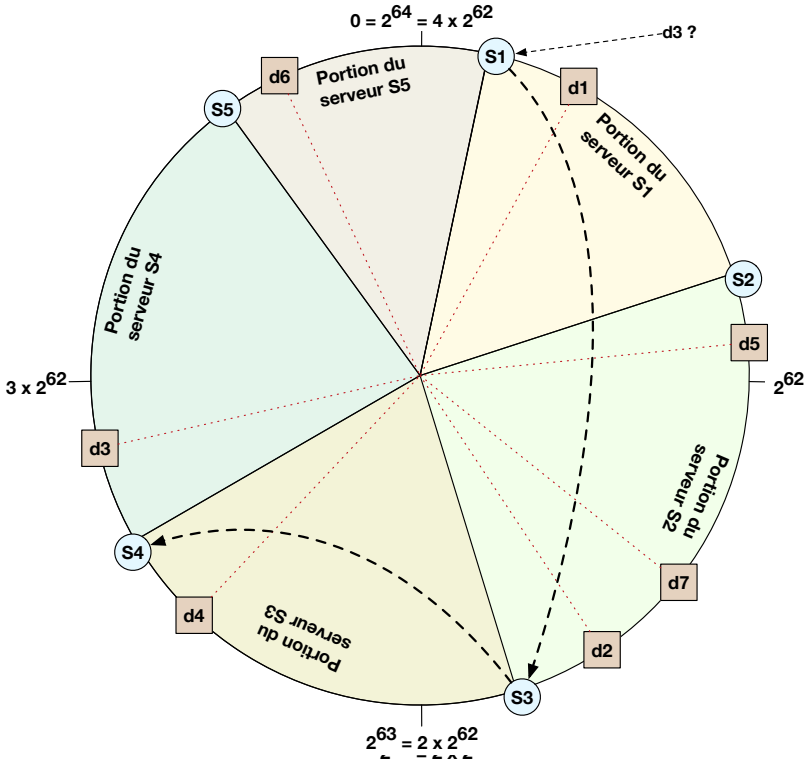


Distributed Hash Table (DHT[3])

- Ring of virtual servers
  - Max 2^{64}
- Unique hash table: splitted & distributed
  - Routing
  - Efficiency
  - Self-Management (no main server)

(3) See DHT http://chewbii.com/hachagedynamique/ (Vidéo 4 & 5)

SHARDING WITH DHT: EXAMPLE



Chunk S1
resp d1
réplicas d5 d2 d4 d7

Chunk S2
resp d5 d2 d7
réplicas d4 d3

Chunk S3
resp d4
réplicas d3 d6

Chunk S4
resp d3
réplicas d6 d1

Chunk S5
resp d6
réplicas d1 d5 d2 d7

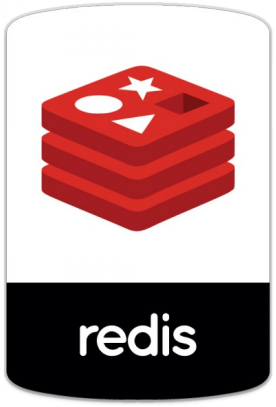
LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

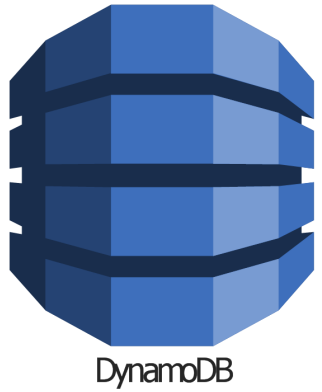
SHARDING WITH DHT: SOLUTIONS

Self-Management

Elasticity



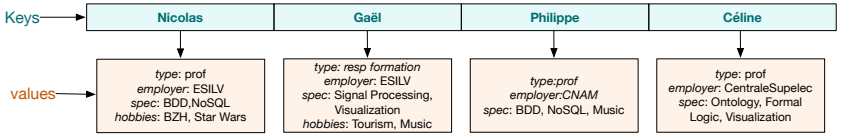




LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

NOSQL DATA TYPE FAMILIES

Key-Value stores



Wide Column-oriented

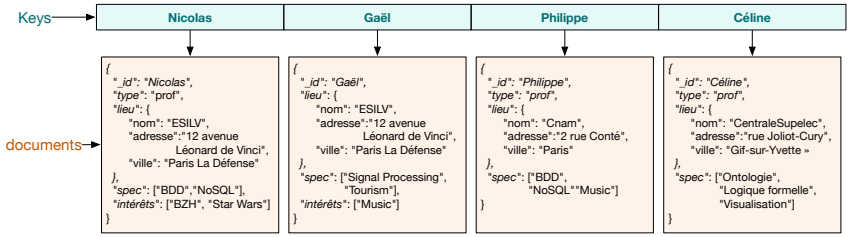
id	status
Nicolas	prof
Gaël	resp formation
Philippe	prof
Céline	prof

id	employer
Nicolas	ESILV
Gaël	ESILV
Philippe	CNAM
Céline	CentraleSupelec

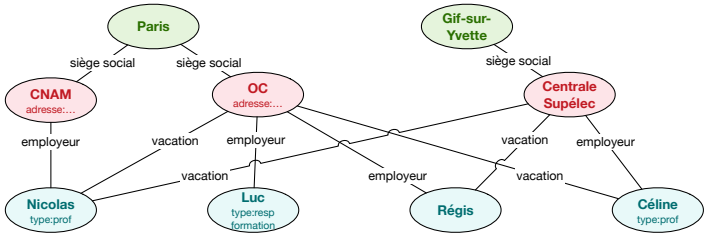
id	spec
Nicolas	BDD
Nicolas	NoSQL
Gaël	Signal Processing
Gaël	Visualization
Philippe	BDD
Philippe	NoSQL
Philippe	Music
Céline	Ontology
Céline	Formal Logic
Céline	Visualization

id	hobbies
Nicolas	BZH
Nicolas	Star Wars
Gaël	Tourism
Gaël	Music

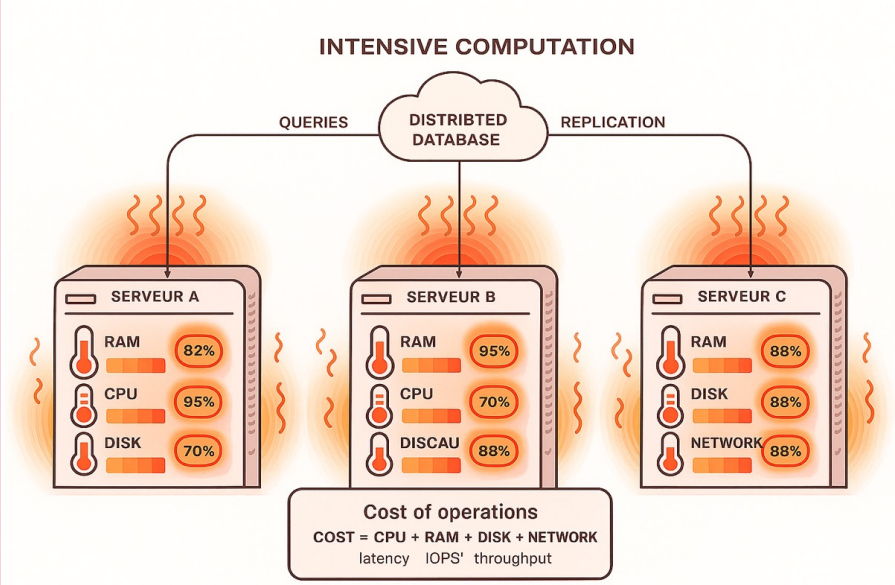
Document-oriented



Graph oriented



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr



DISTRIBUTED COST MODEL

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr

HOW TO COMPARE DATA MODELS?

- Require a generic cost model based on the meta-model
  - Execution time
  - Environmental impact
  - Financial cost
- Contexte modeling
  - Queries
  - Statistics (data & queries)
  - Cluster information (data center, servers spec, budget, etc.)
  - GDPR: Private vs Public Cloud
  - Data across several database types: Polystore

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

CLOUD'S COST MODEL

Need for a cost model

- Estimate the cost of queries submitted to the DB
  - 1 query = 1 sequence of operations
  - 1 operation = 1 algorithm = 1 cost formula
- Operations:
  - Filter / Selection - reduces number of documents = *Map*
  - Projection - reduces document size = *Map/Reduce output*
  - Grouping - aggregates results = Reduce
- Cost impacted by:
  - Communications – Network distribution operations
  - CPU & RAM access – Standard Database operations
    - Disk is ignored after "cold start"

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

TIME COST

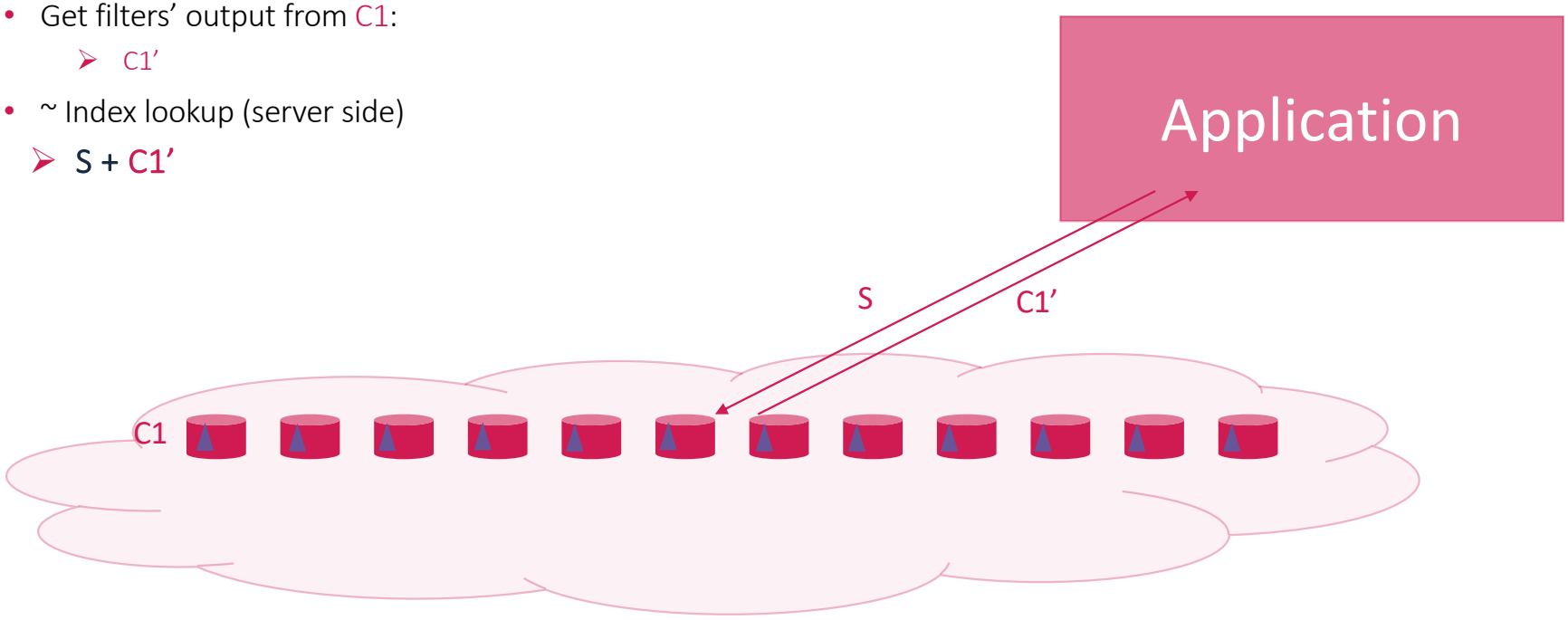
- For a query (q):
  - Communication time: *Bandwidth*
    - $\boldsymbol{bandwidth_{network}} = 100$ MB/s – 1GB/s
  - Server-side computation in main memory: Read/Write *RAM*
    - $\boldsymbol{bandwidth_{RAM}} = 25$ GB/s
    - CPU negligeable
  - ➢ Calculate data volumes
    - Messages size, documents size
    - (see data types size)



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

DISTRIBUTED FILTER

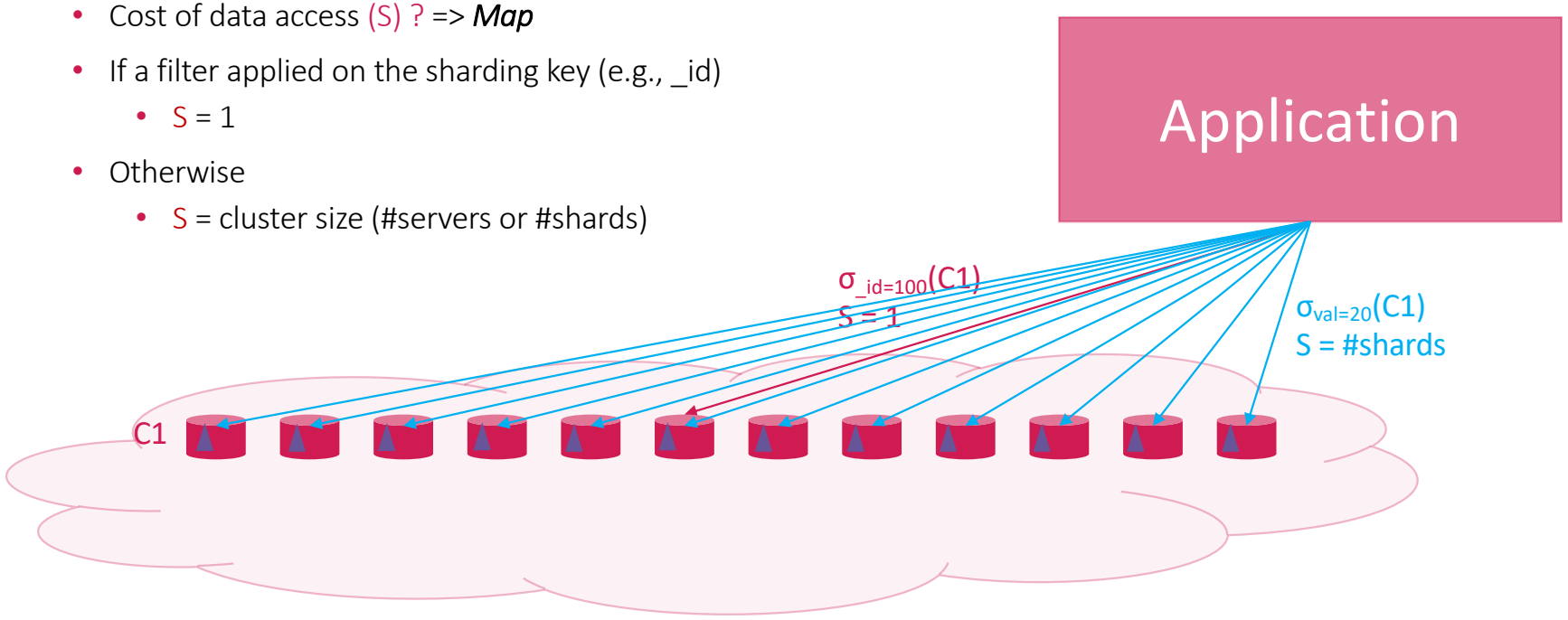
- Data access of servers (S) => *Map on collection C1*
- Get filters' output from C1:
  - C1'
- ~ Index lookup (server side)
  - S + C1'



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

DATA ACCESS « S »

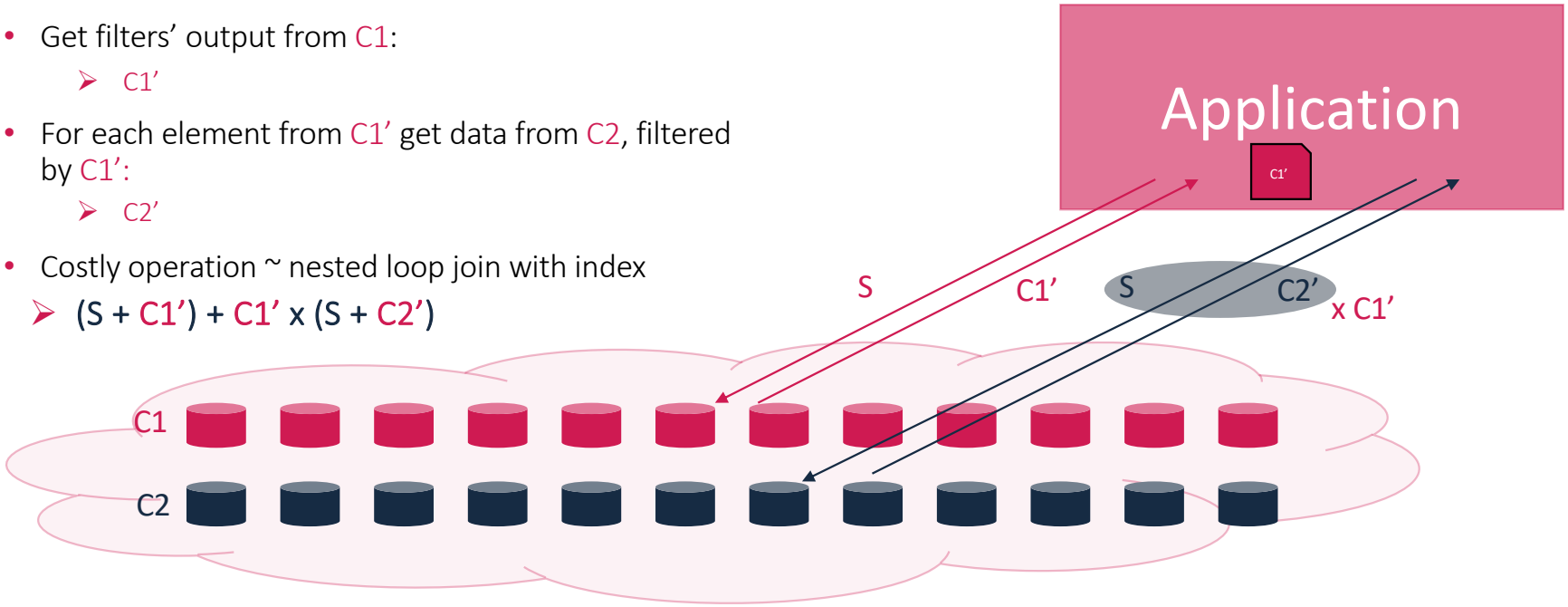
- Cost of data access (S) ? => *Map*
- If a filter applied on the sharding key (e.g., _id)
  - S = 1
- Otherwise
  - S = cluster size (#servers or #shards)



Nicolas.travers@devinci.fr

DISTRIBUTED JOIN NESTED LOOP

- Data access of servers (S) => *Map on collection C1*
- Get filters' output from C1:
  - C1'
- For each element from C1' get data from C2, filtered by C1':
  - C2'
- Costly operation ~ nested loop join with index
  - (S + C1') + C1' x (S + C2')

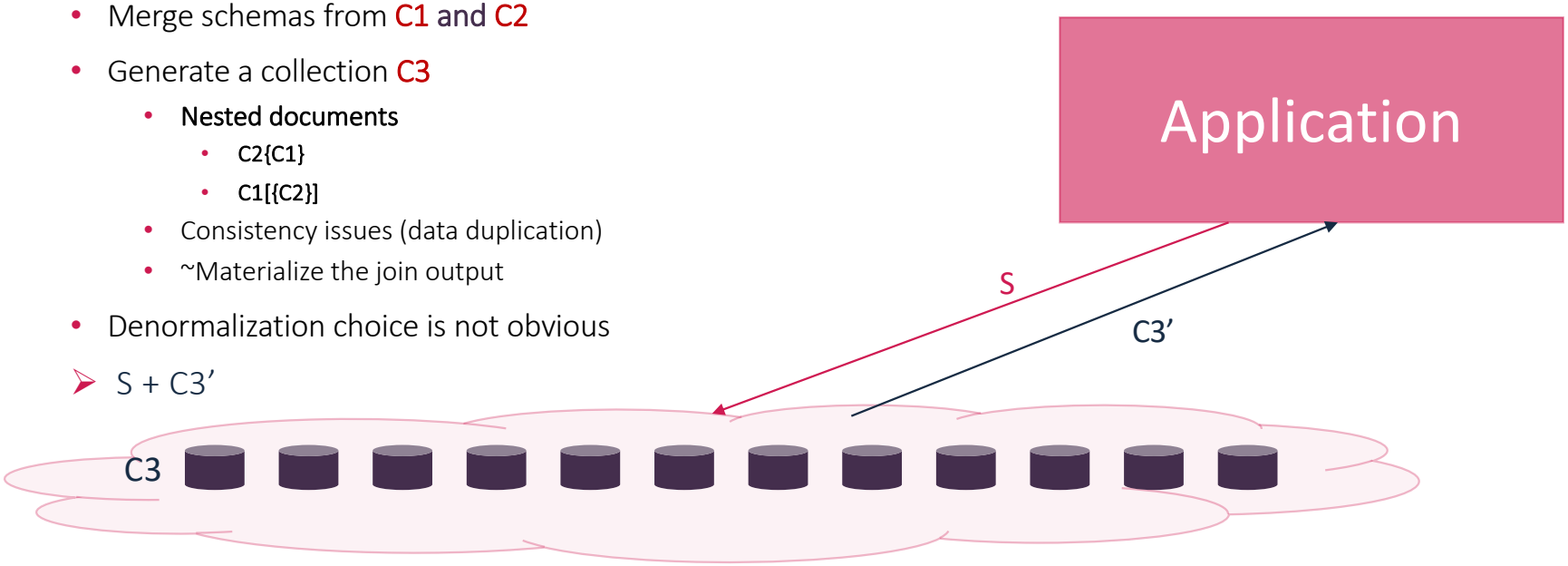


LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

APPLIED TO A DENORMALIZED COLLECTION

- Merge schemas from C1 and C2
- Generate a collection C3
  - Nested documents
    - C2{C1}
    - C1[{C2}]
  - Consistency issues (data duplication)
  - ~Materialize the join output
- Denormalization choice is not obvious

➢ S + C3'



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

TIME COST: FILTER & JOIN QUERIES

- Communications
  - $vol_{network}(q) = S * size_{query} + |res(q)| * size_{msg}$
  - Joins: $vol_{network}(q_1) + |res(q_1)| * vol_{network}(q_2)$

- Server-side computation
  - Parallelism = max time of each server (*n* servers)
  - $vol_{RAM}(q) = max(vol_{RAM}(q,1), \dots, vol_{RAM}(q,n))$
  - $vol_{RAM}(q,n) = index(q) + sel(att) * |coll_{q,n}| * size_{doc(q)}$
    - Secondary local index (~1 MB) -> pointers on pieces of data
    - Data access
    - **Sel**: Filter query selectivity (q) applied to the local storage

Volume of data read by the query

Volume of data on server *n*

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

JOIN APPLICATION AT SERVER SIDE

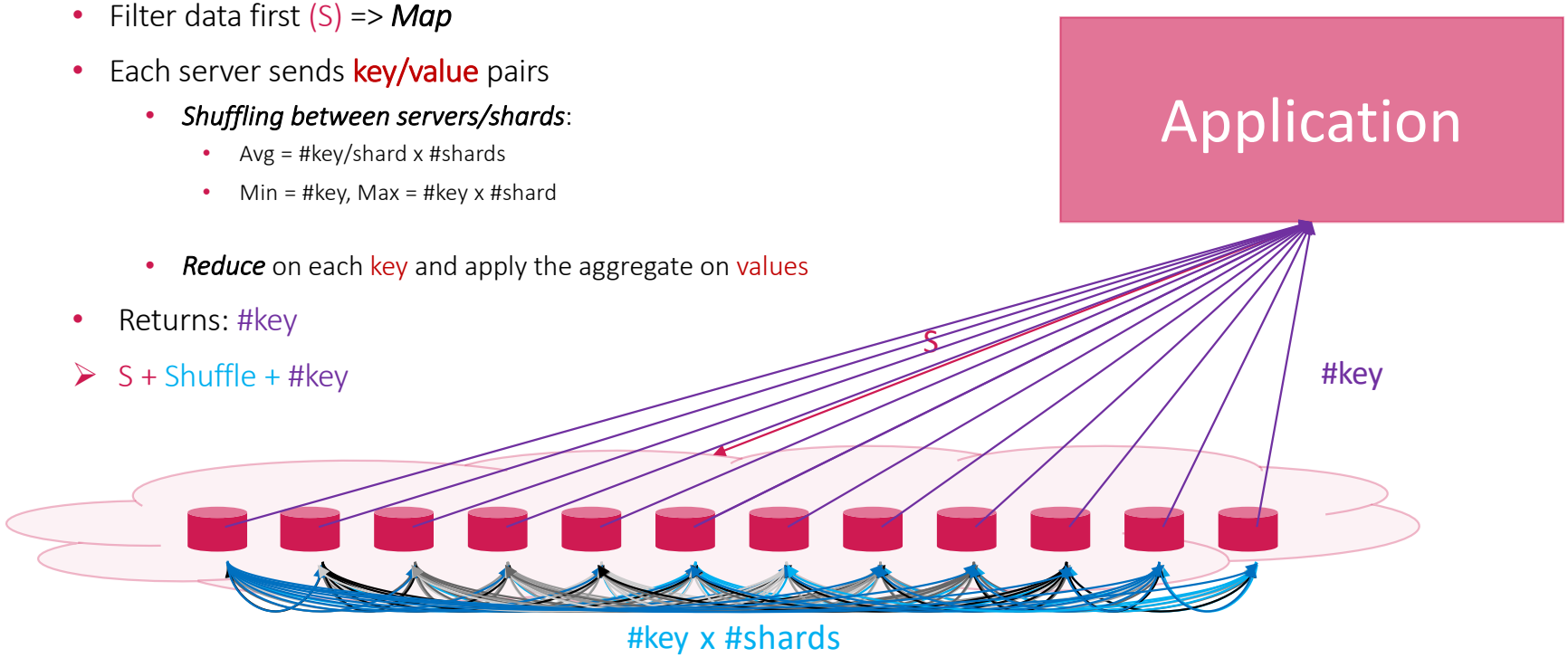
- Join data can be distributed on servers for local computation
  (same amount of data transfer)
- Few NoSQL databases enable this distributed join
  - e.g., *Hadoop/Pig, Spark, CouchBase, MongoDB 6.1, Cassandra (future version)*
  - Similar to application join (distributed)
  - Depends on collection sizes to be distributed
  - Relational-based join algorithms (hash, tri-merge)

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

AGGREGATES "*GROUP BY + COUNT*" ?

- Filter data first (S) => ***Map***
- Each server sends **key/value** pairs
  - ***Shuffling between servers/shards***:
    - Avg = #key/shard x #shards
    - Min = #key, Max = #key x #shard
  - ***Reduce*** on each key and apply the aggregate on values
- Returns: #key

➢ S + Shuffle + #key

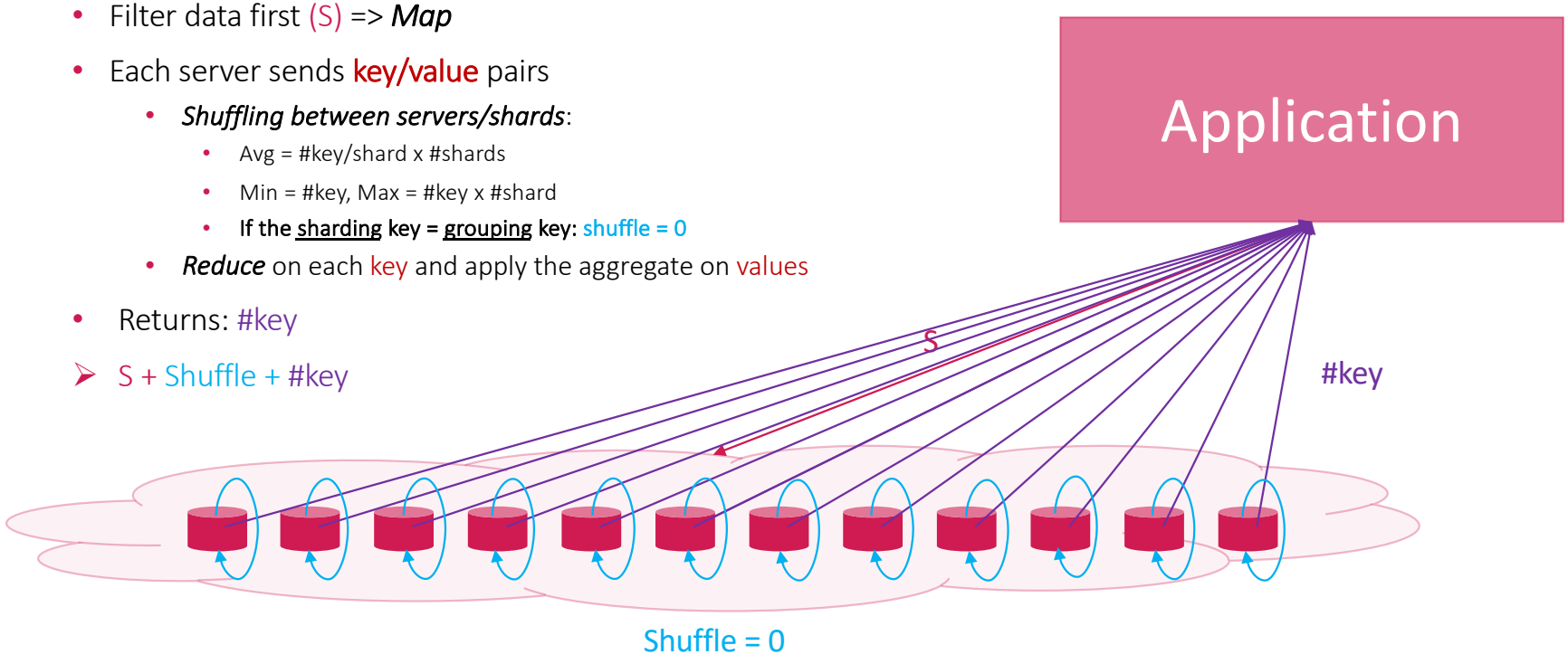


LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

AGGREGATES "*GROUP BY + COUNT*" ?

- Filter data first (S) => ***Map***
- Each server sends **key/value** pairs
  - ***Shuffling between servers/shards***:
    - Avg = #key/shard x #shards
    - Min = #key, Max = #key x #shard
    - **If the sharding key = grouping key:** shuffle = 0
  - ***Reduce*** on each key and apply the aggregate on values
- Returns: #key

➢ S + Shuffle + #key



TIME COST: AGGREGATE QUERIES

- Communications during the **Map** & **Reduce** phases
  - $\boldsymbol{vol_{network}(q) = S * size_{query} + shuffle * size_{msg} + |res(q)| * size_{msg}}$
  - Joins: $\boldsymbol{vol_{network}(q_1) + |res(q_1)| * vol_{network}(q_2)}$
  - Can be combined with filter queries

- Server-side computation for **Map/Reduce**
  - $\boldsymbol{vol_{RAM}(q, n) = reduce_{local} + reduce_{global}}$
  - $\boldsymbol{reduce = \# doc * size_{doc} * 2}$
    - Local, $\# doc = nb\ de\ doc\ agrégés$ (par serveur)
    - Global, $\# doc = nb\ de\ doc\ reçus\ par\ le\ shuffle$ (par serveur)

Read & write for each reduce stage

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

MULTIDIMENSIONAL COST MODEL* [MALI ET AL. – ARXIV'23,BDA'23,EDBT'24,IS'25]

TIME COST

- Query time
  - Communication
  - RAM
  - Storage
- Use case time (frequencies)

FINANCIAL COST

- Cluster size
  - #servers
  - Server spec
- Queries
  - External communications (Cloud fees)

ENVIRONMENTAL COST

- Cluster/storage carbon footprint (life cycle)
- Queries' carbon footprint
  - RAM
  - Storage
  - Communications

Based on data volumes (GB)

*The generic cost model gives estimations for comparison purposes. No real values

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr

TIME COST: TOTAL

- Take into account the whole use case
  - Set of queries
  - Query frequencies

- $$time(DB) = \sum_{q=1}^{Q} \left(\frac{vol_{network}(q)}{bandwidth_{network}} + \frac{vol_{RAM}(q)}{bandwidth_{RAM}} \right) * freq(q)$$

Internal vs external cloud communications can be split (S & res)

FINANCIAL COST



- **External** bandwidth fees:
  - $externalFees$ = 0,011 €/GB
  - Split inter/extern: $time_{network}(q)$

- Virtual machines fees (VMs) :
  - Watch the configuration:
    - RAM
    - Storage (SSD)
    - Bandwidth
    - Price
  - Cluster dimensioning
    - Data volumes
    - Must fit in main memory (avoid cold start)
    - Storage capacity (replication x3 !!!)

Azure server	capacity RAM	capacity SSD	Price (per month)	Bandwidth
B1ms	1 GB	4 GB	25,63 €	100 Mbps
B1s	2 GB	4 GB	30,13 €	100 Mbps
L8as	64 GB	1,9 TB	313,25 €	32 Gbps

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

ENVIRONMENTAL IMPACT

- Carbon footprint measure
  - kg CO2-eq
  - Variation between countries (DataCenter)
- Infrastructure impact (fixed cost)
  - Storage: 0,0031 kg CO2-eq/GB
  - Lifecycle of servers: 1 283kg CO2-eq/serveur
- Queries Impact
  - Bandwidth
    - $CO2_{network}$ = 0,0110 kg CO2-eq/GB
  - Computation RAM/CPU
    - $CO2_{RAM}$ = 0,0280 kg CO2-eq/GB



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

ENVIRONMENTAL IMPACT: THE USE CASE

Network impact

- $impact_{network}(q) = vol_{network}(q) * CO2_{network}$

Simplification of computation in RAM

RAM/CPU impact

- $impact_{RAM}(q) = vol^{*}_{RAM}(q) * CO2_{RAM} = \sum_{n=1}^{N} vol_{RAM}(q,n) * CO2_{RAM}$

Global impact

- $impact(DB) = \sum_{q=1}^{Q}(impact_{network}(q) + impact_{RAM}(q)) * freq(q)$

LIKE,SHARE, FOLLOW ! @esiv_paris | esilv.fr
Nicolas.travers@devinci.fr

MULTI-DIMENSIONAL COST MODEL

$$time(DB) = \sum_{q=1}^{Q} \left(\frac{vol_{network}(q)}{bandwidth_{network}} + \frac{vol_{RAM}(q)}{bandwidth_{RAM}} \right) * freq(q)$$

$$price(DB) = price(s) * max\left(\left\lceil \frac{vol_{DB} * 3}{capacity_{storage}} \right\rceil, \left\lceil \frac{vol_{DB}}{capacity_{RAM}} \right\rceil \right) * 2 + externalFees * \sum_{q=1}^{Q} vol_{external\ network}(q) * freq(q)$$

$$impact(DB) = \sum_{q=1}^{Q} \left(vol_{network}(q) * CO2_{network} + \sum_{n=1}^{N} vol_{RAM}(q,n) * CO2_{RAM} \right) * freq(q)$$

Not true for environmental impact ($vol^{*}_{RAM}(q)$)

In practice:

- $vol_{RAM}(q) \ll vol_{network}(q)$
- $bandwidth_{RAM} = 250 * bandwidth_{network}$
- $CO2_{RAM} \approx 2{,}5 * CO2_{network}$

➢ The network part of the cost model remains important

➢ Better target **environmental** impact optimization with time and price constraints (QoS & budget)

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr



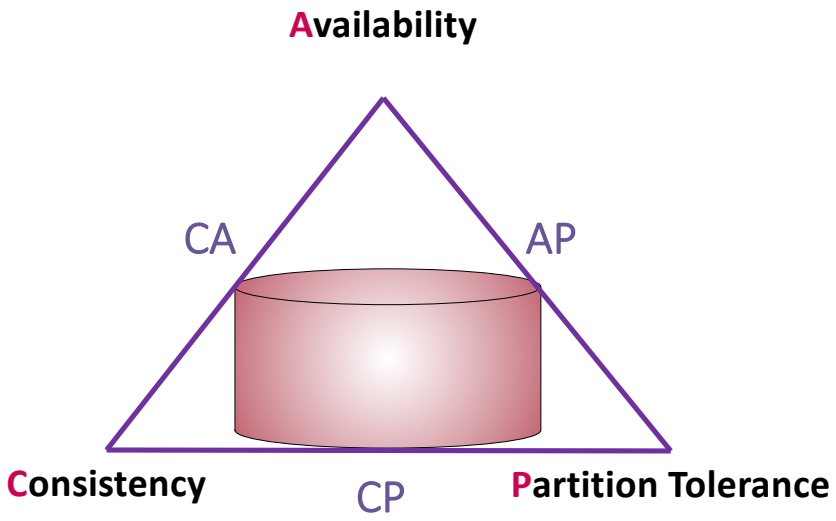
CAP: CONSISTENCY VS AVAILABILITY

THE CAP THEOREM [BREWER 2000]

3 main properties for distributed management

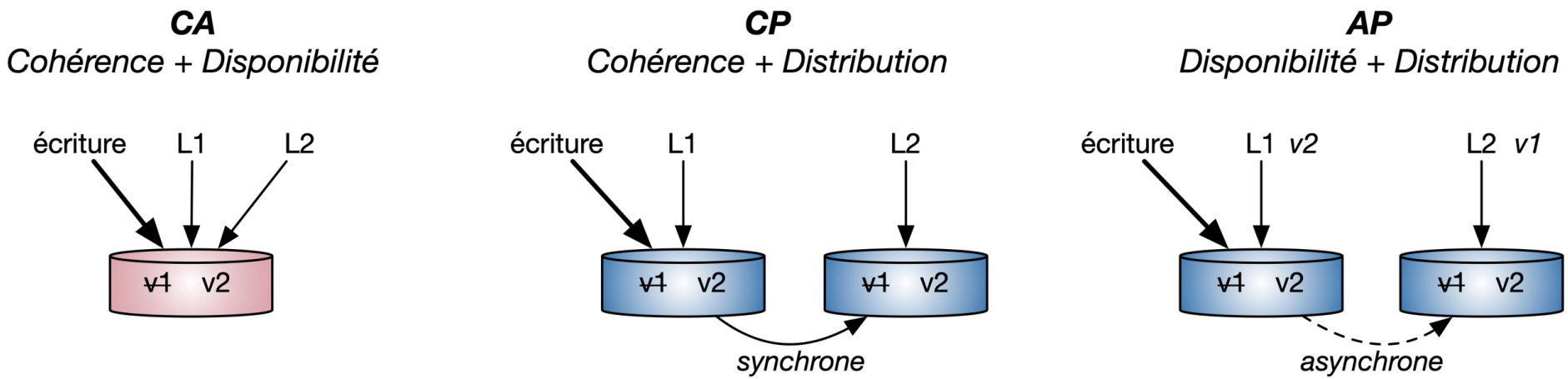
1. *Consistency*:
   - A data returns the proper value at any time (coherency)
2. *Availability*:
   - Even if a server is down, data is available
3. *Partition Tolerance*:
   - Even if the system is partitioned, a query must have an answer (unless for global failures)

Theorem: *A distributed, networked system can have only <u>two</u> of these three properties.*



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

THE CAP THEOREM: ILLUSTRATION



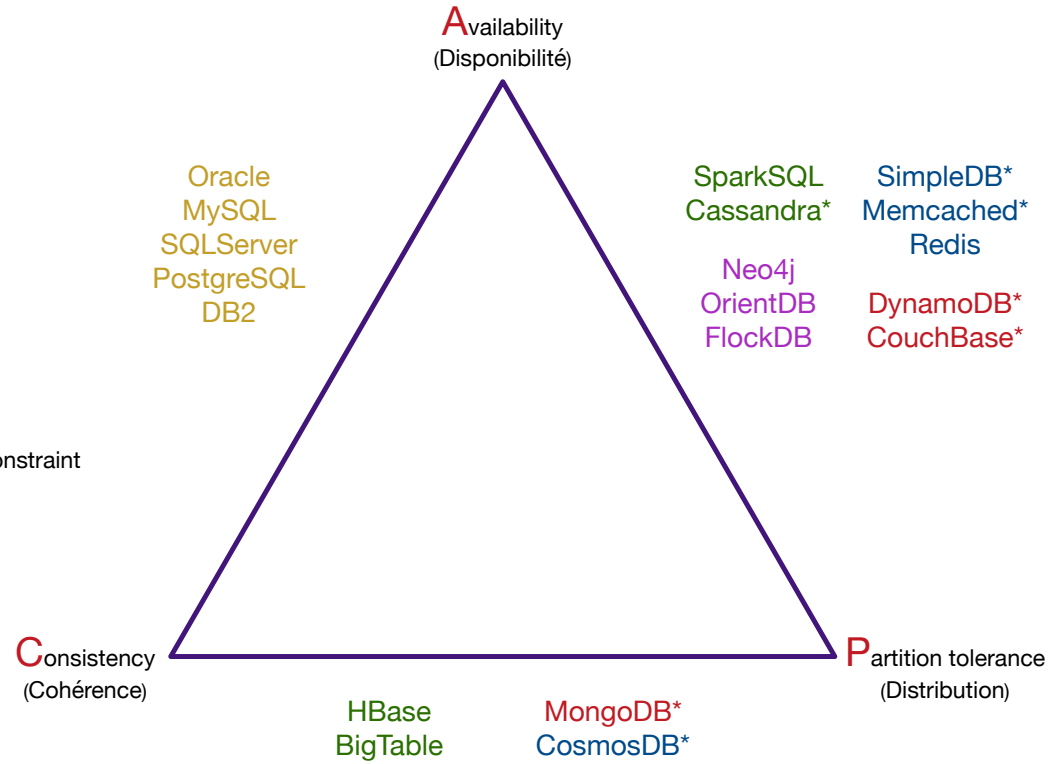
LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

THE CAP THEOREM: CAP TRIANGLE

Models

Relational/Column
Key-Value
Wide Column
Document
Graph

* can change consistency constraint



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

CAP VS PACELC [ABADI 2012]

Data synchronization

- PAC
  - Partitioning/Data movements
  - Between Availability and Consistency
- ELC
  - Normal state of the network
  - Between Latency and Consistency

- **PA/EL**
  - *Dynamo, Cassandra, Riak*
  - Always **available** with a minimum of latency
- **PC/EC**
  - *VoltDB/H-Store, MegaStore*
  - **Consistency** whatever happens
- **PA/EC**
  - *MongoDB*
  - While partitioning: **availability**
  - In normal state: **consistency**

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

RECAP: HOW TO CHOOSE A NOSQL SOLUTION? 225+ NOSQL SOLUTIONS + RELATIONAL

- NoSQL Family
  - Querying languages: *Expressivity vs simplicity*
  - *Consistency issues*
- *Sharding* strategy
  - vs secondary indexes
  - Primary key / foreign key / join key / group key
- Availability vs Consistency: CAP
- Community support

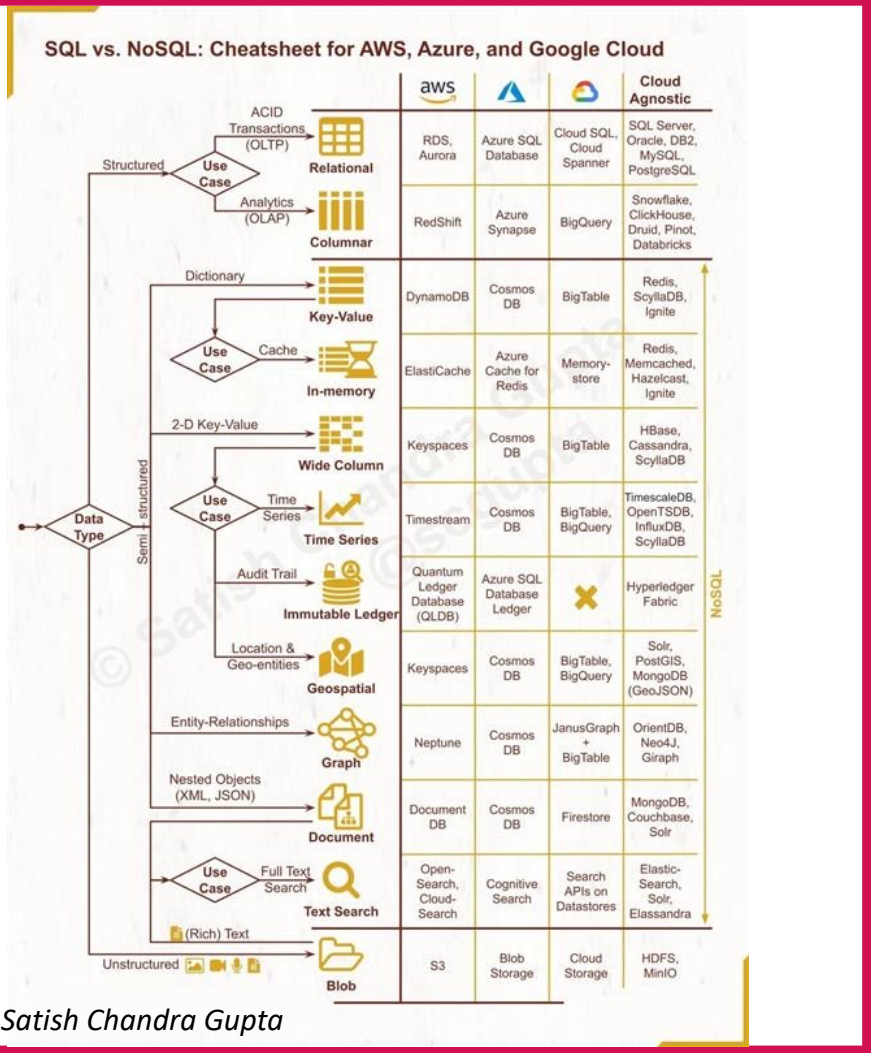
Complex use cases

- Best denormalization / tradeof ?
- Filter/Join/Group queries + frequencies
- Data statistics (Data distribution)
- Other constraints :
  - QoS
  - Environmental impact
  - Financial cost
  - GDPR
  - Use case evolution

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

DATABASE CHOICE

A Simple choice decision tree



Credits - *Satish Chandra Gupta*

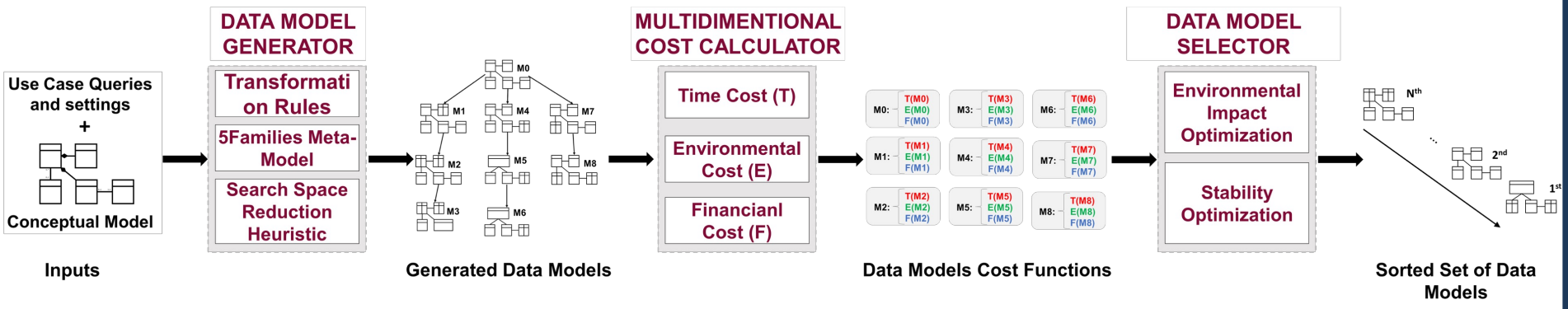
LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

HOW TO CHOOSE THE OPTIMAL DATA MODEL ON THE CLOUD?

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

DaMOop

DAMOOP – A GLOBAL DENORMALIZATION APPROACH



Nicolas.travers@devinci.fr

ESILV
ENGINEERING SCHOOL DE VINCI PARIS

DaMOop

DATA MODEL GENERATOR

DATA MODELS TRANSFORMATION SCHEMA

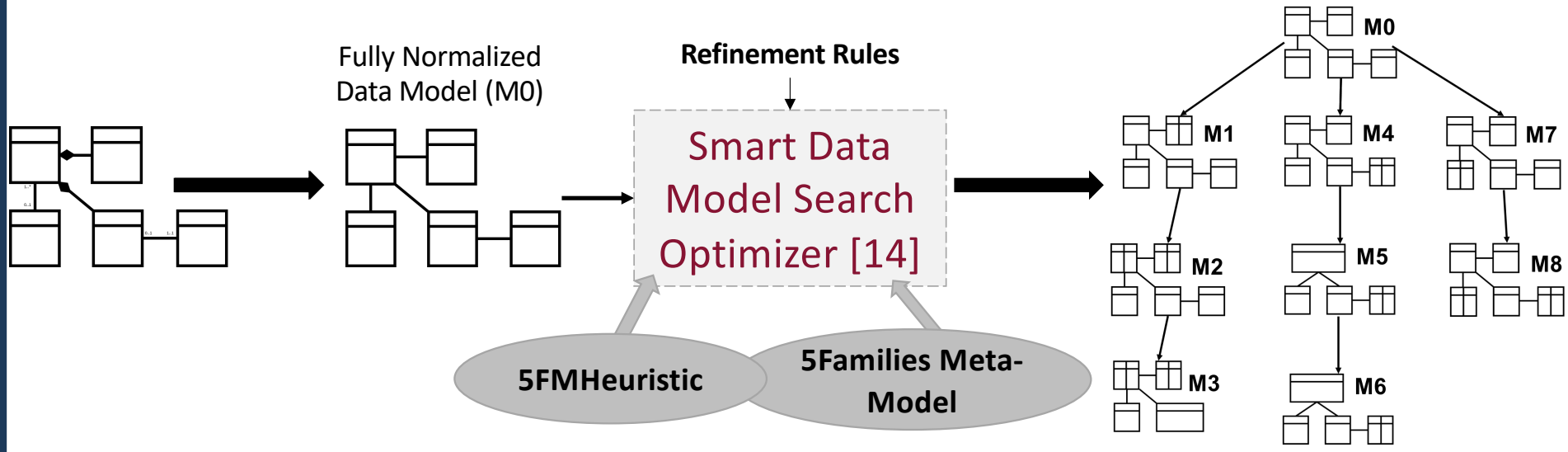
J. Mali, S.Ahvar, F.Atigui, A.Azough, and N.Travers. 2022. A Global Model-Driven Denormalization Approach for Schema Migration. In RCIS'22.

Conceptual Model

Logical Models

Search Space

Tree of relevant data models



Nicolas.travers@devinci.fr

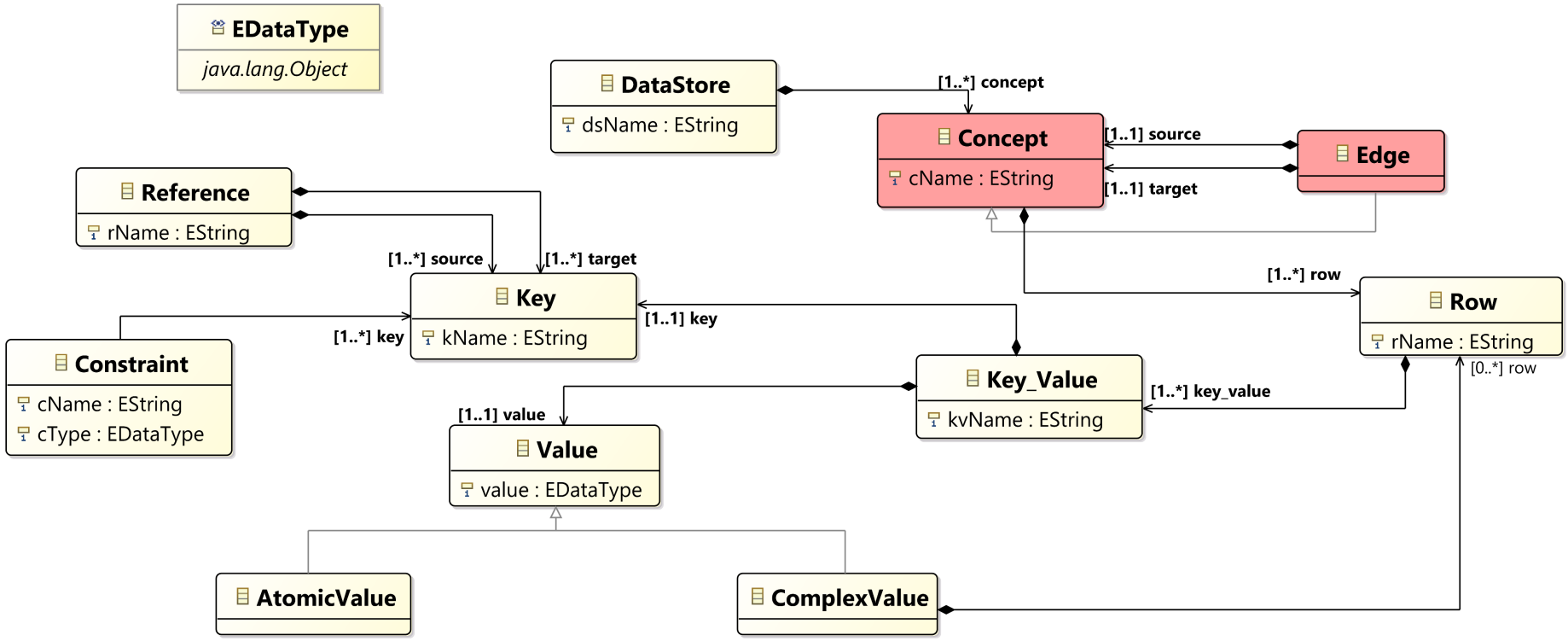
ESILV
ENGINEERING SCHOOL DE VINCI PARIS

DaMOop

[15] J.Mali, F.Atigui, A.Azough, and N.Travers. 2020. ModelDrivenGuide: An Approach for Implementing NoSQL Schemas. In DEXA'20.

DATA MODEL GENERATOR

5FAMILIES META-MODEL



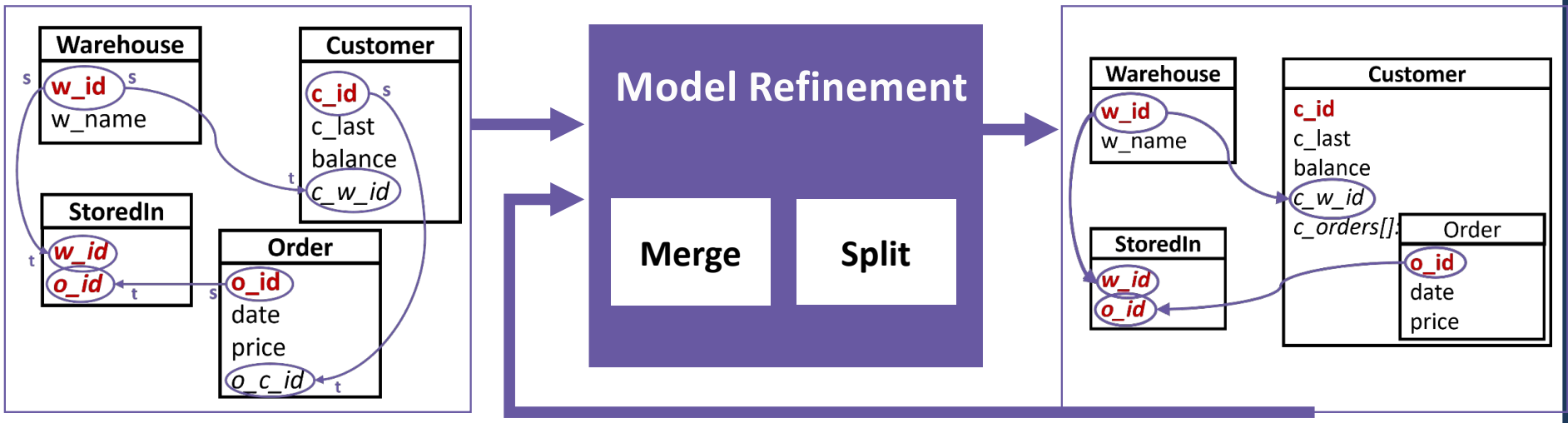
Nicolas.travers@devinci.fr

ESILV
ENGINEERING SCHOOL DE VINCI PARIS

DaMOop

DATA MODEL GENERATOR

LOGICAL MODEL REFINEMENT RULES



Nicolas.travers@devinci.fr

DATA MODEL GENERATOR

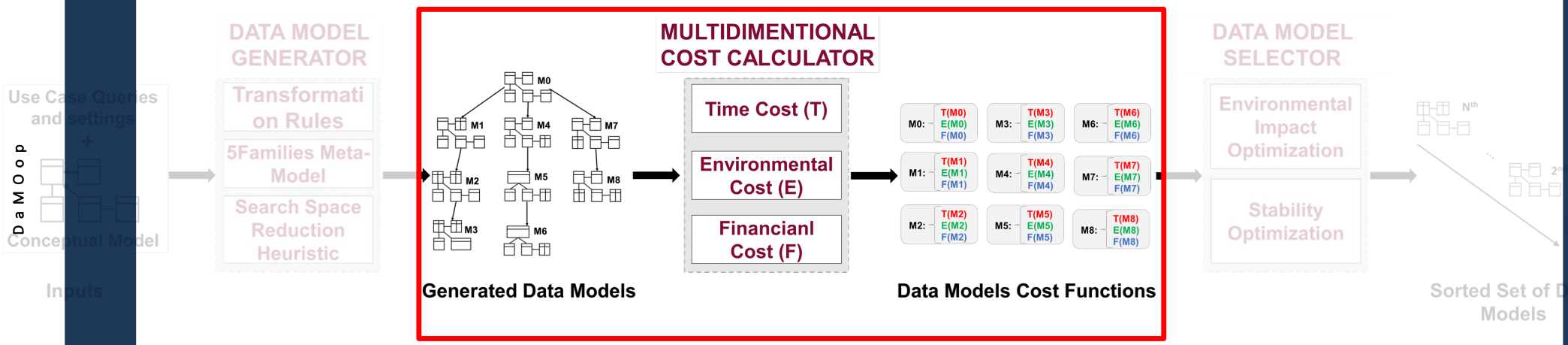
STATE OF THE ART

	Relational	DO	CO	KVO	GO	Approach	
[1], [2], [3]			✓			Dedicated	Manual
[4]		✓				Dedicated	Manual
[5]					✓	Dedicated	Manual
[6]		✓	✓			Semi-Global	Semi-Automated
[7],[8]		✓	✓	✓	✓	Dedicated	Semi-Automated
Our approach	✓	✓	✓	✓	✓	Global	Automated

- [5] : Daniel, G. et al. (2016)
- [6] : de la Vega, A. et al. (2020)
- [7] : Abdelhedi, F. et al. (2017)
- [8] : de Freitas, M.C. et al. (2016)

LIKE,SHARE, FOLLOW ! @nane_Mali et al.
Nicolas.travers@devinci.fr

DAMOOP – COST CALCULATOR



Nicolas.travers@devinci.fr

MULTI-DIMENSIONAL COST MODEL

[Mali et al. – arXiv'23,BDA'23, EDBT'24,IS'25]

$$time(DB) = \sum_{q=1}^{Q} \left(\frac{vol_{network}(q)}{bandwidth_{network}} + \frac{vol_{RAM}(q)}{bandwidth_{RAM}} \right) * freq(q)$$

$$price(DB) = price(s) * max\left(\left\lceil \frac{vol_{DB} * 3}{capacity_{storage}} \right\rceil, \left\lceil \frac{vol_{DB}}{capacity_{RAM}} \right\rceil \right) * 2 + externalFees * \sum_{q=1}^{Q} vol_{external\ network}(q) * freq(q)$$

$$impact(DB) = \sum_{q=1}^{Q} \left(vol_{network}(q) * CO2_{network} + \sum_{n=1}^{N} vol_{RAM}(q,n) * CO2_{RAM} \right) * freq(q)$$

Not true for environmental impact ($vol^{*}_{RAM}(q)$)

In practice:

- $vol_{RAM}(q) \ll vol_{network}(q)$
- $bandwidth_{RAM} = 250 * bandwidth_{network}$
- $CO2_{RAM} \approx 2{,}5 * CO2_{network}$

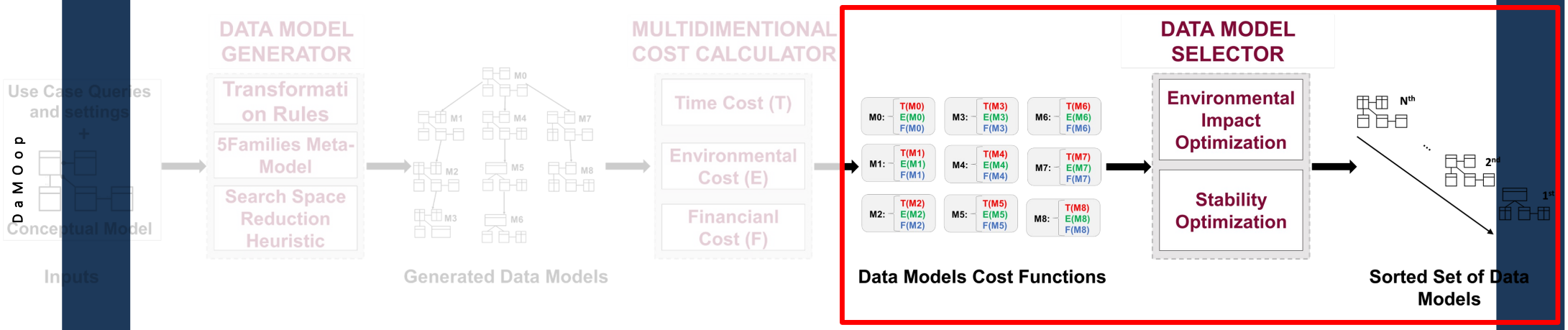
➢ The network part of the cost model remains important

➢ Better target **environmental** impact optimization with time and price constraints (QoS & budget)

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr

Nicolas.travers@devinci.fr

OUR APPROACH



Nicolas.travers@devinci.fr

ESILV
ENGINEERING SCHOOL DE VINCI PARIS

DaMOop

DATA MODELS SELECTOR

[Mali et al. – arXiv'23, IS'25]

DATA MODEL SELECTOR INPUT: USE CASE

- **Queries** and their **frequencies**
- Settings which are the couple:
  - **Data volume:** number of documents #doc
  - **Number of servers:** #srv
- Constraints for each dimension:
  - **Time constraint τ_{tq} :** max execution time of query q
  - **Environmental constraint τ_e :** carbon emission budget
  - **Financial constraint τ_f :** budget

Qualified data models M^A respect the constraints

$$M \in \mathsf{M}^A \mid \forall M \in \mathsf{M},\ q \in Q,$$
$$\boldsymbol{T(M,q) \leq \tau_{tq} \wedge \mathbf{E}(\mathbf{M}) \leq \tau_e \wedge\ \mathbf{F}(\mathbf{M}) \leq \tau_f}$$

Nicolas.travers@devinci.fr

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

DaMOop

DATA MODELS SELECTOR

DATA MODEL SELECTOR INPUTS: COST SCORE

- Time, environmental and financial dimensions are **not** at the same level
- Cost score $\tilde{C}(M,Q)$ is defined using a weighted sum of normalized cost dimensions :

  $\tilde{T}(M,Q)$, $\tilde{E}(M,Q)$ and $\tilde{F}(M,Q)$

$$\tilde{C}(M,Q) = wT \times \tilde{T}(M,Q) + wE \times \tilde{E}(M,Q) + wF \times \tilde{F}(M,Q)$$

Nicolas.travers@devinci.fr

DATA MODELS SELECTOR

DATA MODEL STABILITY OPTIMIZATION

- Offer stability for all dimensions at the same time on different settings $\mathbf{S}$
- Settings are : **Data Volume** #doc & **number of servers** #srv
- Compute density Δ of each data model M for all settings $\mathbf{S}$
- Compute average score of each data model M for all settings $\mathbf{S}$

$$\Delta(M,Q) = \frac{1}{|S|\times(|S|-1)} \sum_{k=1}^{|S|} \sum_{l=k+1}^{|S|} \left| \tilde{C}^k(M,Q) - \tilde{C}^k(M,Q) \right|$$

$$\mathrm{Avg}(M,Q) = \frac{1}{|S|} \sum_{k=1}^{|S|} \tilde{C}^k(M,Q)$$

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

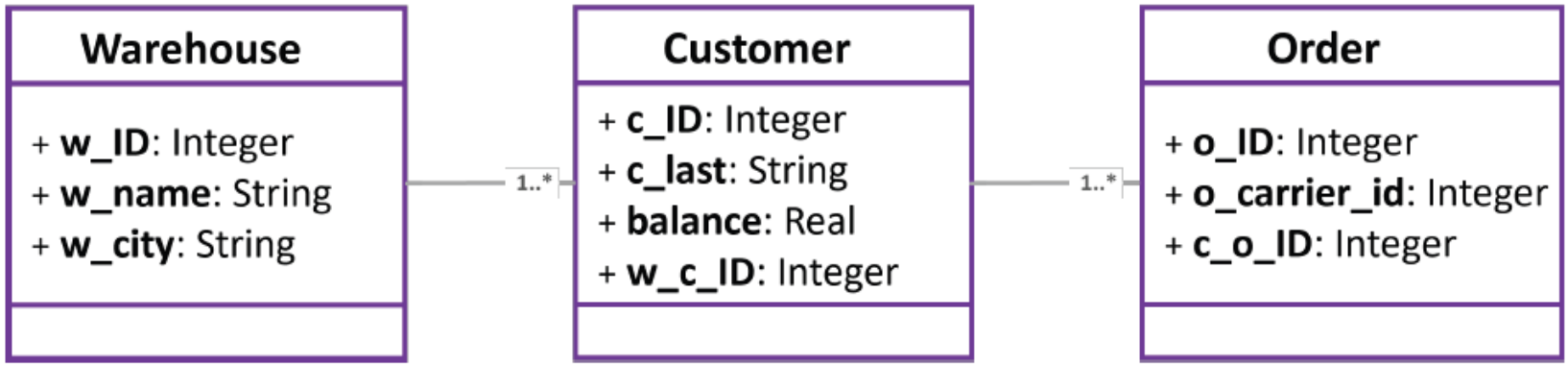
DATA MODELS SELECTOR

DATA MODEL STABILITY OPTIMIZATION

- Select most stable data model M_{opt}^{S} out of qualified data models M^{A} in different settings
- Calculate a stability score $\widetilde{D}(M, Q)$: weighted sum of normalized density and average scores
- M_{opt}^{S} is the data model with the lowest stability score among all other data models

$$M_{opt}^{S} = M \in \mathsf{M}^{A} \mid \forall Mi \in \mathsf{M}^{A},\ \widetilde{D}(M_i, Q) \geq \widetilde{D}(M, Q)$$

IMPLEMENTATION
TPC-C BENCHMARK



LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

IMPLEMENTATION

USE CASE QUERIES

Query	Type	Filter Keys	Projection Keys	Join Keys	Sharding	Occurrences /day	Constraint τ_{tq} (in s)
Q1	Filter	$balance$	c_last	N/A	N/A	500	0,1
Q2	Filter	o_ID	$o_carrier_id$	N/A	N/A	1,000	10
Q3	Filter	w_city	w_name	N/A	On w_city	100	1
Q4	Join	c_last	c_last, $o_carrier_id$	c_ID, c_o_ID	On c_last	50	0,1
Q5	Join	c_last, w_city	$balance$, w_name, $o_carrier_id$	c_ID, c_o_ID, w_ID, w_c_ID	On w_city & c_last	10	10

IMPLEMENTATION

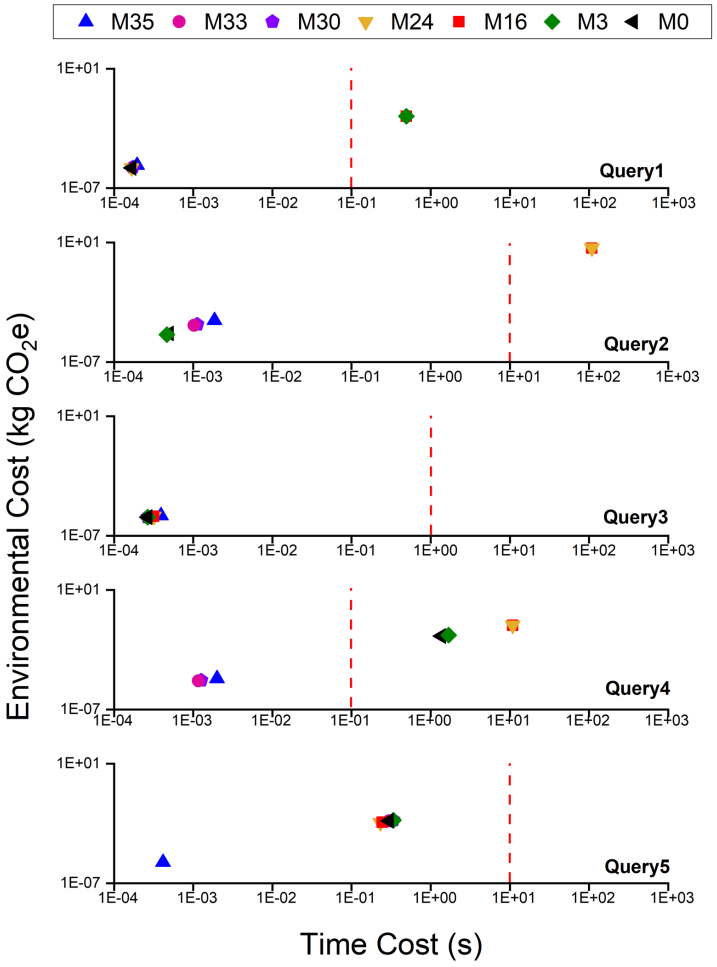
	Nb Models	Nb Splits	Nb Merges	Nb Unique Models
Naïve Approach	2731	2226	504	1012
5FMHeuristic	36	3	32	36

Data Model	Signature	Signature Definition	Produced by competitors
M0	W,C,O	3 rows W, C and O	
M3	$C1,C2,C3,O1,O2,W$	2 splits on C and one split on O	
M16	$C1,C2,C3\{O,W\}$	2 splits on C, merge of O and W in C3	
M24	$C1,C2\{O,W\}$	1 split on C, merge of O and W in C2	
M30	$C\{O\},W$	Merge O in C	
M33	O{C},W	Merge C in O	Chebotko[1]
M35	$O\{C\{W\}\}$	Merge W in C, then merge C in O	Abdelhadi[8]

IMPLEMENTATION
QUERY'S COST

- Study how costs vary on different types of queries
- Consider queries ' frequency for a daily cost
- Test on a fixed setting: **1M warehouses & 1k servers**

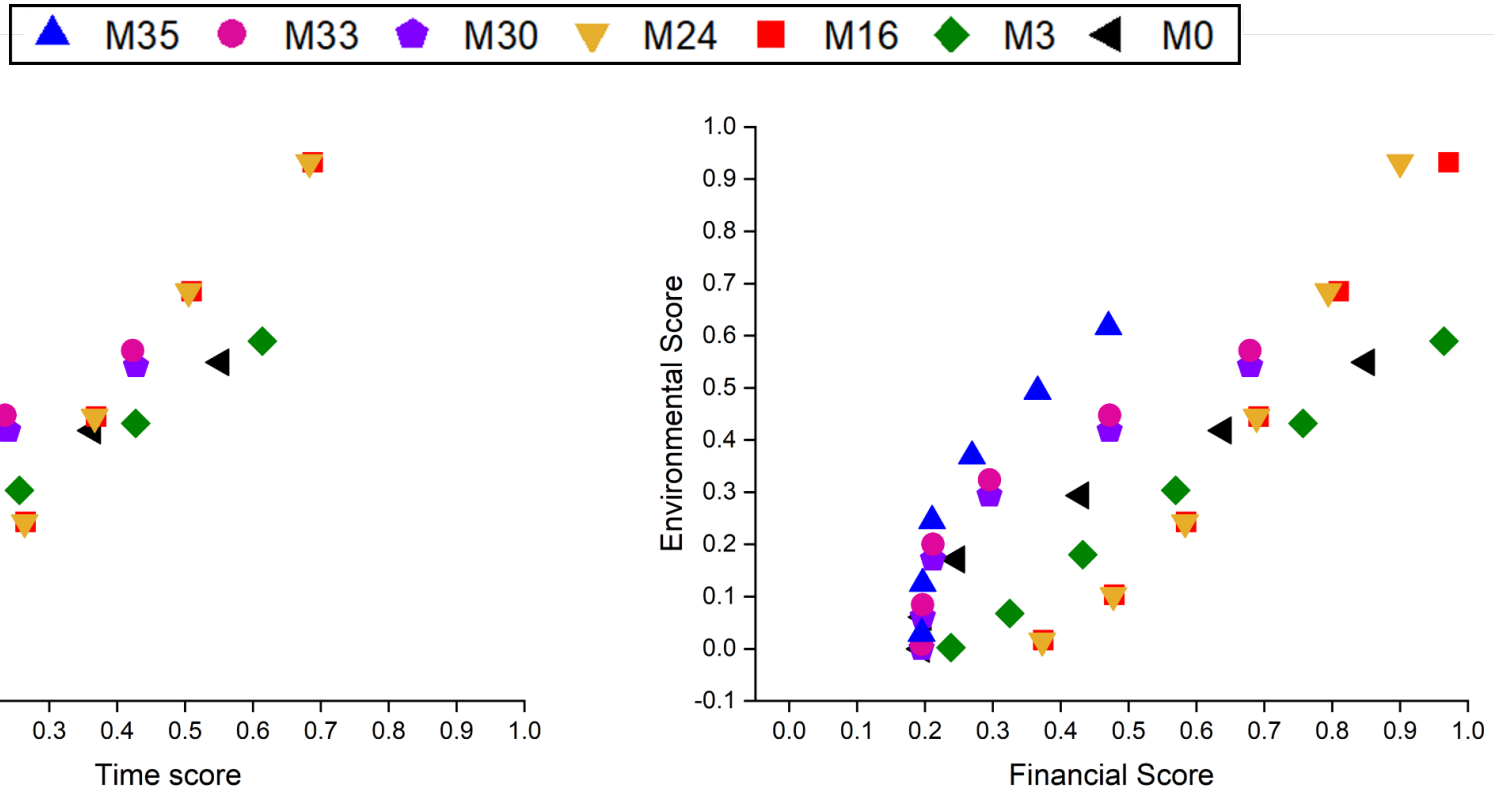
Data Model	Signature
M0	W,C,O
M3	$C1,C2,C3,O1,O2,W$
M16	$C1,C2,C3\{O,W\}$
M24	$C1,C2\{O,W\}$
M30	$C\{O\},W$
M33	O{C},W
M35	$O\{C\{W\}\}$



Nicolas.travers@devinci.fr

IMPLEMENTATION

TOTAL DATA MODEL'S COST SCORE



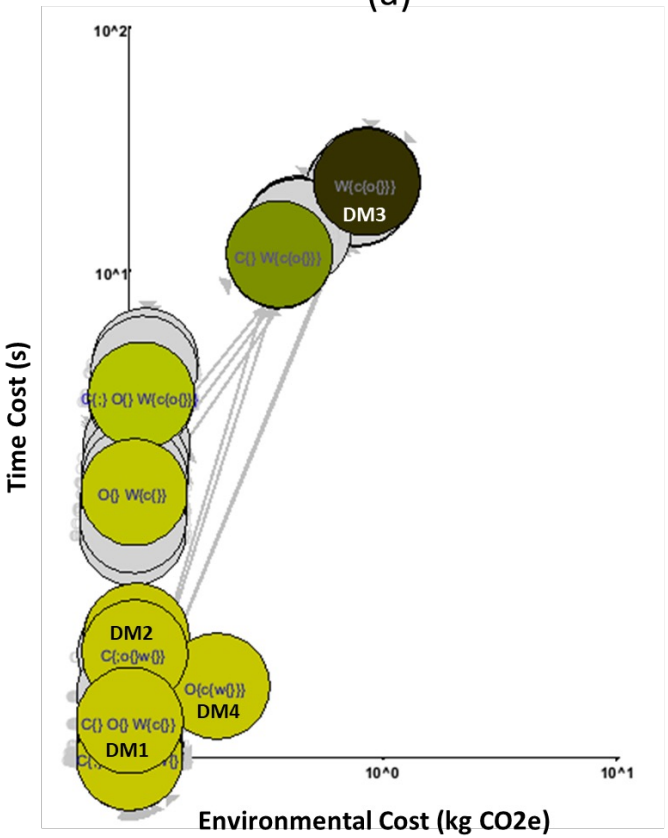
Nicolas.travers@devinci.fr

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

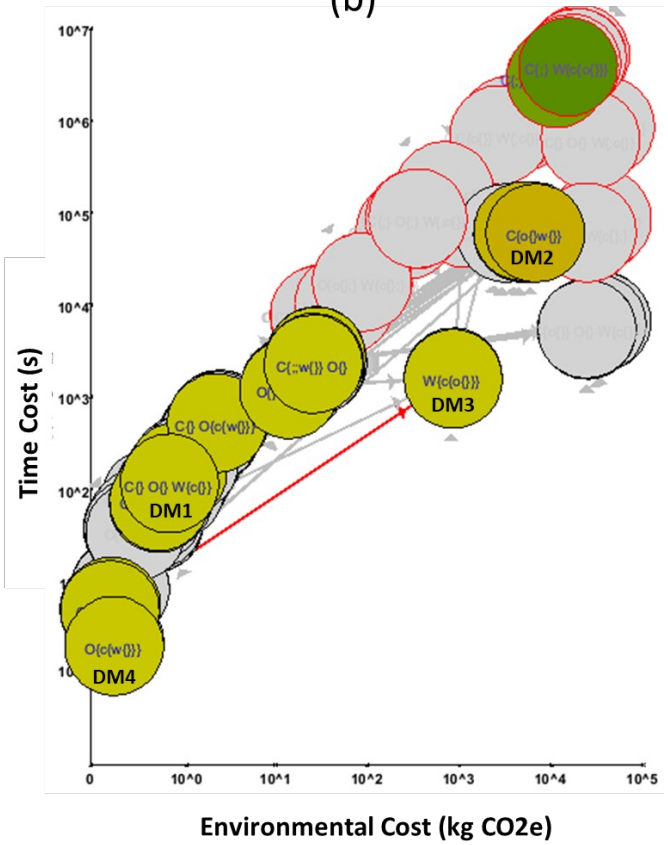
DaMOop

IMPLEMENTATION

(a)



(b)



Nicolas.travers@devinci.fr

ESILV
ENGINEERING SCHOOL DE VINCI PARIS

DaMOop

IMPLEMENTATION

Settings variations:

- Data Volumes : from 1k to 100M factored by 10 each time
- Number of servers: {10, 50, 100, 500, $1k$, $5k$, $10k$}

Data Model	Use case constraints		Environmental score $\tilde{C}(M,Q)$ With w_E = 1	*Stability*		
	QoS	Budget		Density score	Average score	$\tilde{D}(M,Q)$
M35[1]			3.6895 ×10−1	6.8679 ×10−2	2.0349 ×10−1	1.3609 ×10−1
M30[8]			2.9379×10−1	9.2863 ×10−2	2.2173 ×10−1	1.5729 ×10−1
M33			3.2335 ×10−1	9.3518 ×10−2	2.2904 ×10−1	1.6128 ×10−1
M0	~~T(M,q4)~~		2.9385 ×10−1	1.1544 ×10−1	2.6928 ×10−1	1.9236 ×10−1
M3	~~T(M,q1), T(M,q4)~~	~~F(M)~~	3.0346 ×10−1	1.2499 ×10−1	3.3019 ×10−1	2.2759 ×10−1
M24	~~T(M,q2), T(M,q4)~~	~~F(M)~~	4.4428 ×10−1	1.3533 ×10−1	4.3228 ×10−1	2.8381 ×10−1
M16	~~T(M,q1), T(M,q2) T(M,q4)~~	~~F(M)~~	4.4449 ×10−1	1.4249 ×10−1	4.4376 ×10−1	2.9313 ×10−1

CONCLUSION & PERSPECTIVES

DamOop Approach

- Find a data structure that is the most relevant to a use case
- Generate all the possible data models
- Calculate the cost of data models based on a multidimensional cost model
- Decision making based on Data Model's stability

Future Works

- Data models shift on query logs / data model evolution
- Orbital Cloud data placement vs data model optimization
- Sustainable stream processing orchestration on continuum computing

LIKE,SHARE, FOLLOW ! @esilv_paris | esilv.fr
Nicolas.travers@devinci.fr

THANK YOU!
OPENED TO QUESTIONS