Enumération Randomisée des Triangles dans des Graph à Grande Echelle à base de SQL

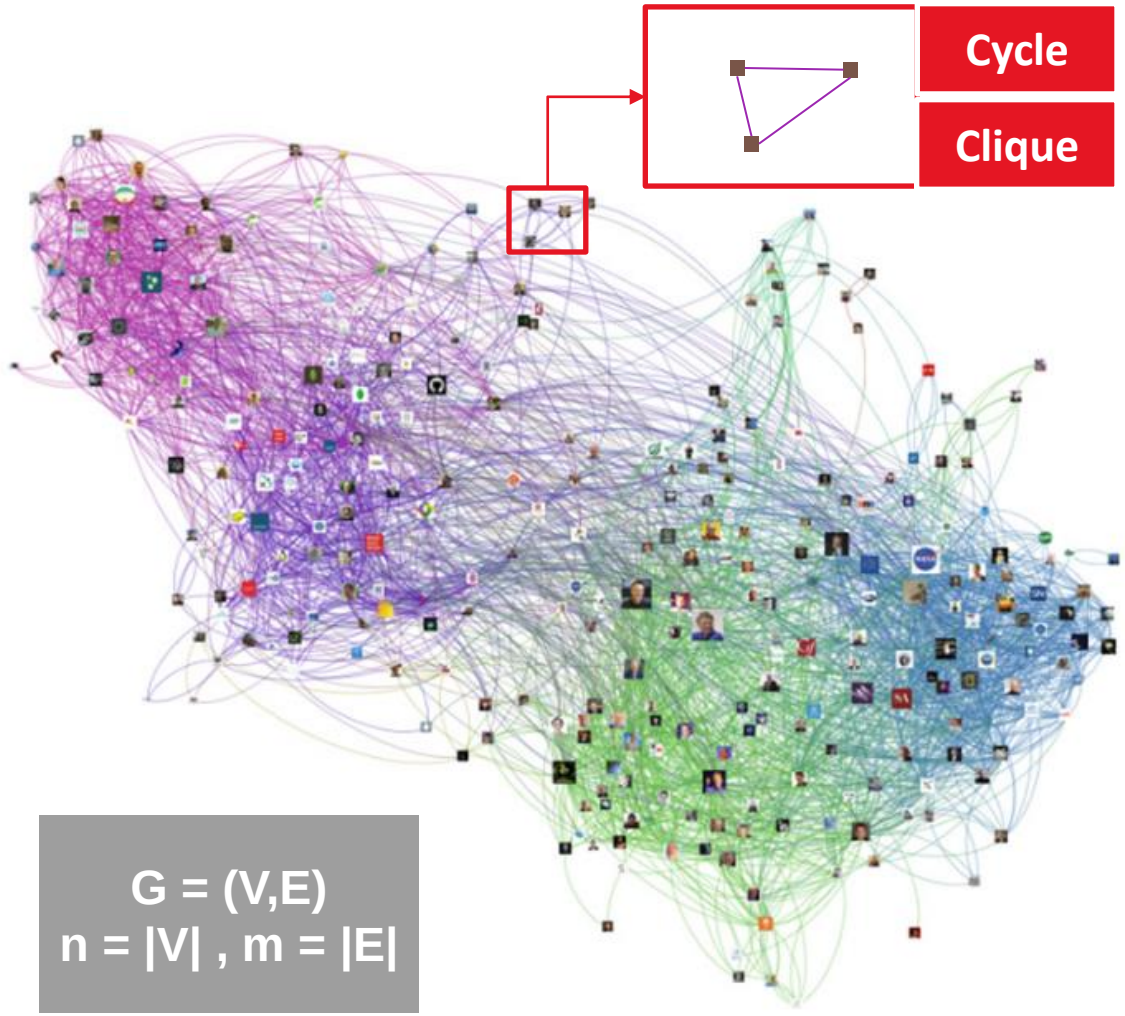
(Randomized Triangle Enumeration on Large Graph based on SQL)

Abir Farouzi[1,3], Ladjel Bellatreche[1], Carlos Ordonez[2], Gopal Pandurangan[2], Mimoun Malki[3]

[1] **LIAS/ISAE-ENSMA,** Poitiers, France
abir.farouzi@ensma.fr, bellatreche@ensma.fr
[2] Université de Houston, Texas, USA
carlos@central.uh.edu , gopal@cs.uh.edu
[3] **ESI-SBA,** Sidi Bel Abbès, Algérie
a.farouzi@esi-sba.dz , m.malki@esi-sba.dz

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Motivations: Graph



G = (V,E)
n = |V| , m = |E|

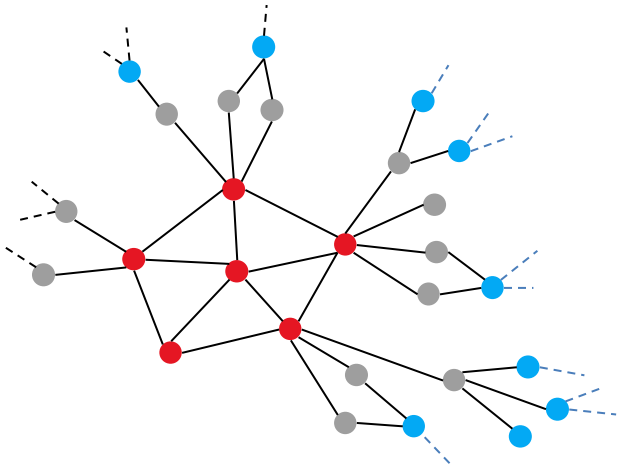
- Path Analytics
- Connectivity Analytics
- Community Analytics
- Centrality Analytics

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Motivations: Triangles

- Social process analysis
- Dense sub-graph extraction
- Database joins
- **Anomalies detection**
- Community age detection
- Recommandations

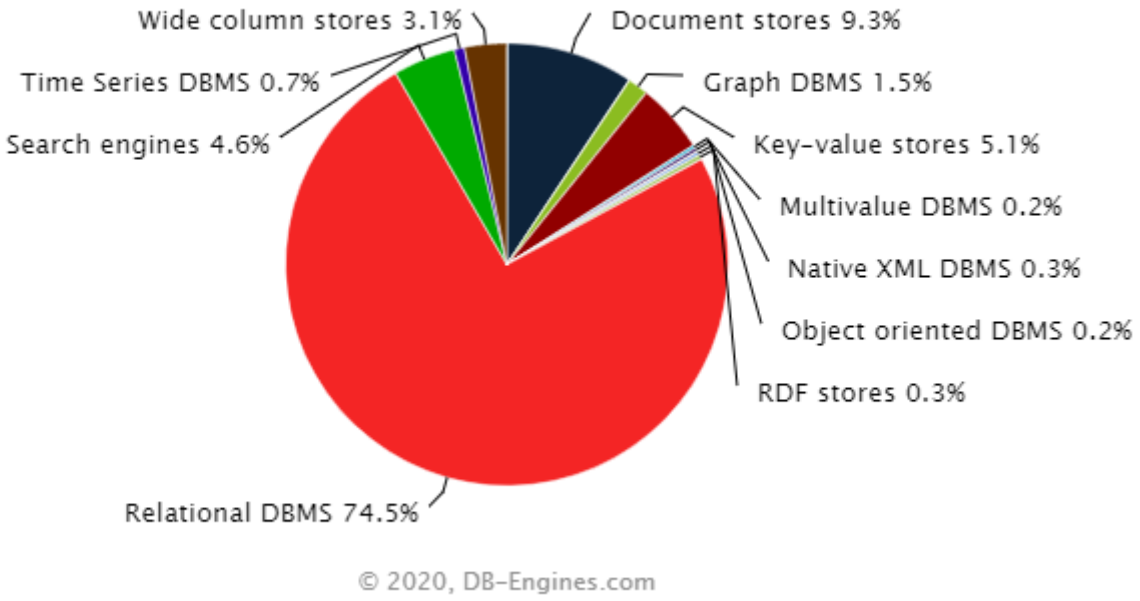
Spam filtering



- **Attacking node**
- Victim node
- **Safe node**

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Motivations: Relational databases [1]



- Lot of data are stored in relational DBMS
- Less data stored as Graphs
- Export data from relational DBMS to graph DBMS causes performance issues
- Exploit RDBMS capabilities like security, concurrence management..etc

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Contribution

Parallel Randomized Algorithm based on SQL queries

Graph Analysis Inside DBMS

Perfect load Balancing

Scalability

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Agenda

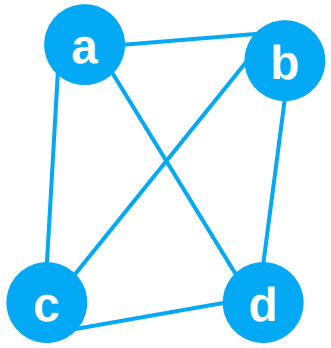
- **Motivations**
  - Why triangles?
  - Why relational databases
- **Solution**
  - Standard Algorithm
  - Randomized Algorithm
- **Experimental study**
  - Configuration and Data sets
  - Results
- **Conclusion**
  - Conclusions
  - Perspectives

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Solution

#standard_algorithm #randomized_algorithm #sql_queries

Graph definition



Undirected
unweighted Graph

i \ j	a	b	c	d
a	0	1	1	1
b	1	0	1	1
c	1	1	0	1
d	1	1	1	0

Adjacency matrix

i	j
a	b
a	c
a	d
b	a
b	c
b	d
c	a
c	b
c	d
d	a
d	b
d	c

Edge Table E
Adjacency list

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Standard Algorithm

Algorithm 1: Algorithme standard

Result: triangles (liste de tous les traingles dans le graphe)

chargement du graphe;

for $e_1 \leftarrow 1$ *to* m **do**
  for $e_2 \leftarrow 1$ *to* m **do**
    for $e_3 \leftarrow 1$ *to* m **do**
      if $(e_1.j == e_2.i \wedge e_2.j == e_3.i \wedge e_3.j == e_1.i)$ **then**
      else
        $triangles \leftarrow (e_1, e_2, e_3)$
      end
    end
  end
end

Complexity is $O(n^3)$

```
SELECT E1.i As v1, E1.j AS v2, E2.j As v3
  FROM
    E E1 JOIN E_dup E2 ON E1.j=E2.i
    JOIN E E3 ON E2.j=E3.i AND E3.j=E1.i
  WHERE E1.i<E1.j AND E2.i<E2.j;
```

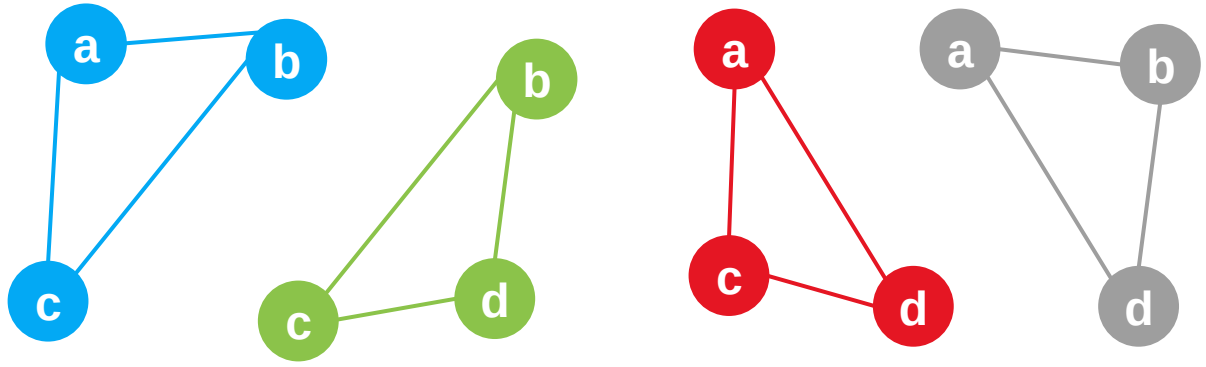
i	j
a	b
a	c
a	d
b	a
b	c
b	d
c	a
c	b
c	d
d	a
d	b
d	c

⋈

i	j
a	b
a	c
a	d
b	a
b	c
b	d
c	a
c	b
c	d
d	a
d	b
d	c

⋈

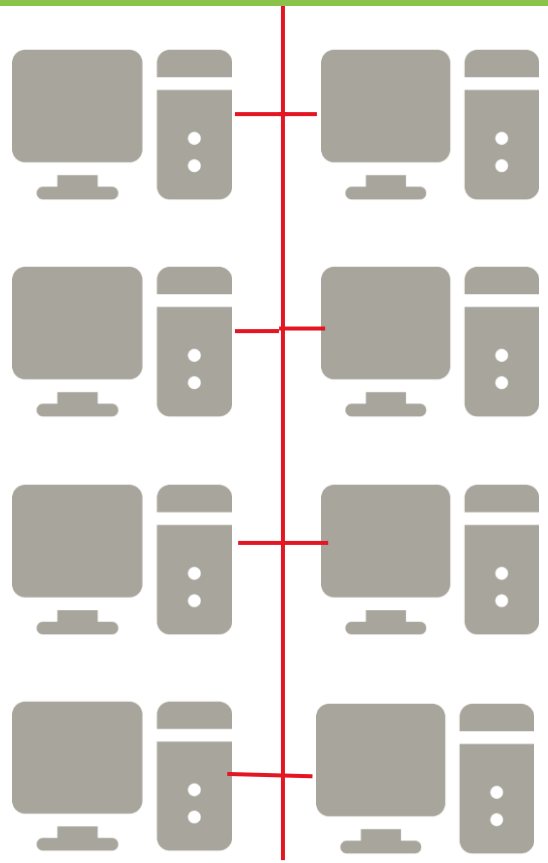
i	j
a	b
a	c
a	d
b	a
b	c
b	d
c	a
c	b
c	d
d	a
d	b
d	c

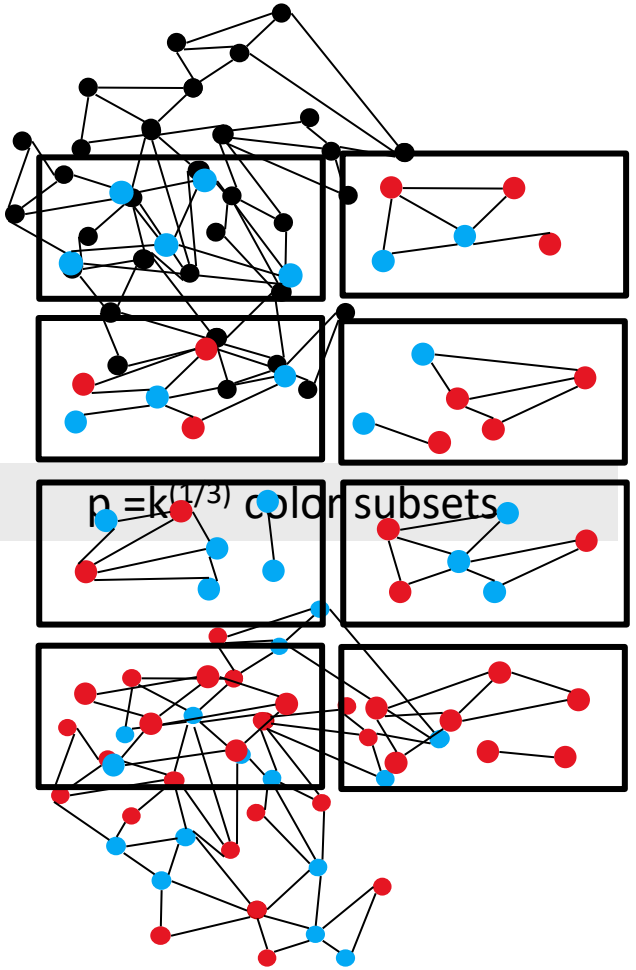


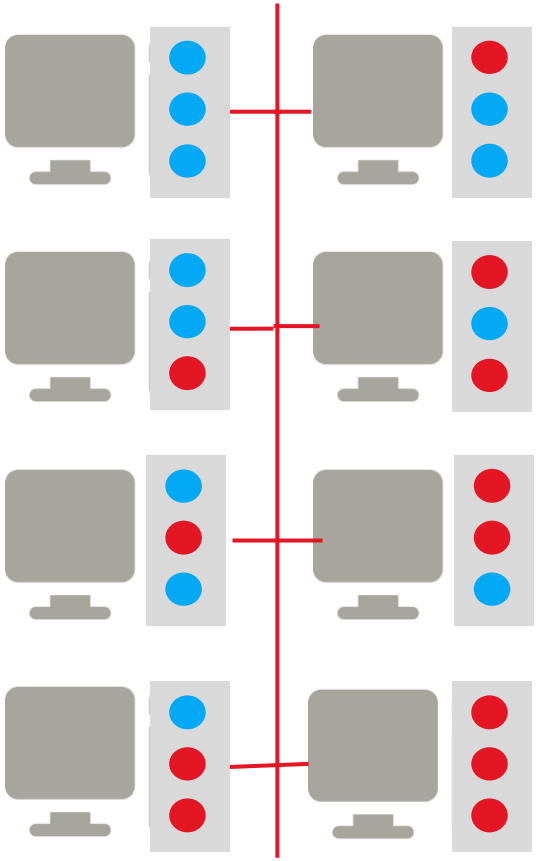
#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Randomized Algorithm [2,3]

Complexity is
$\tilde{O}(\max\{m/k^{2/3}, n/k^{1/3}\})$



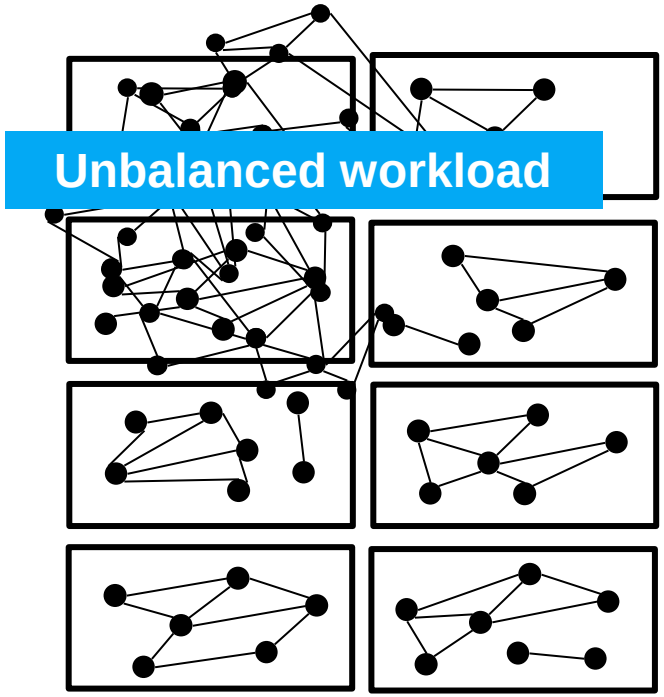




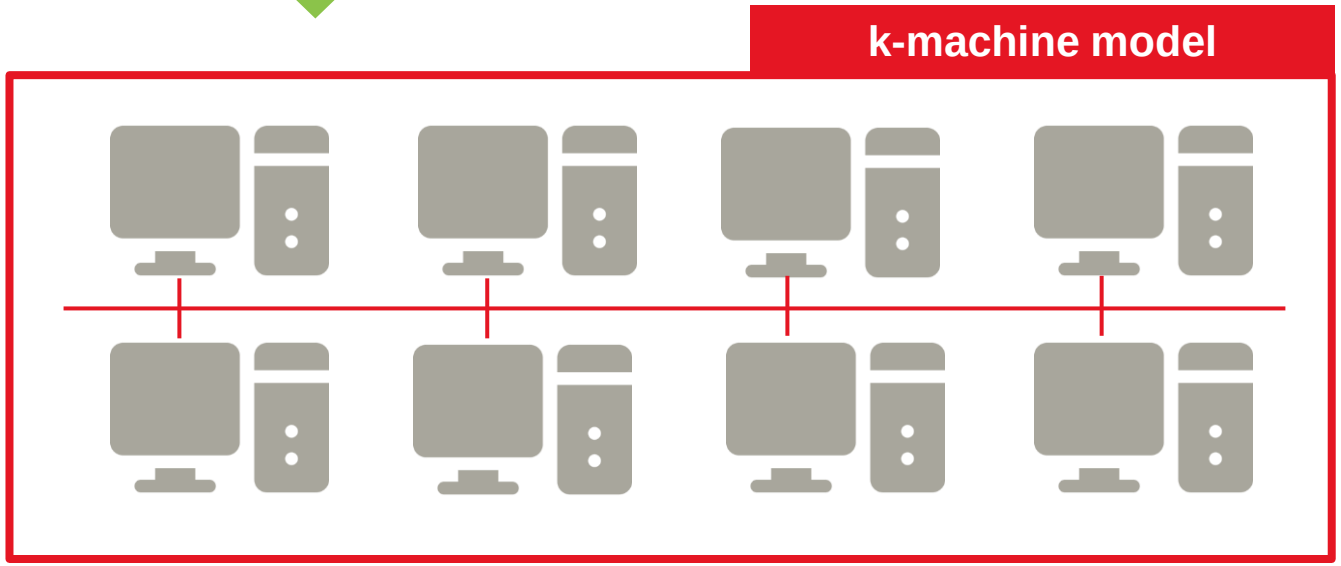
#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Randomized Algorithm

Graph loading



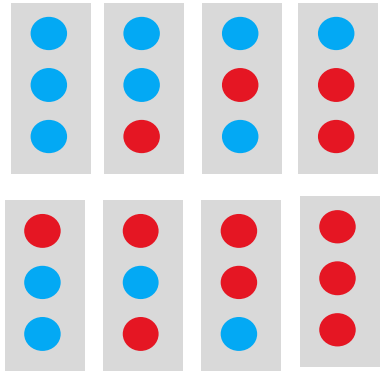

```
CREATE TABLE E_s (i int, j int);
COPY E_s FROM “Lien/vers/graph_data_set“;
/*Si le graphe est non orienté */
INSERT INTO E_s SELECT j,i FROM E_s;
```



#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

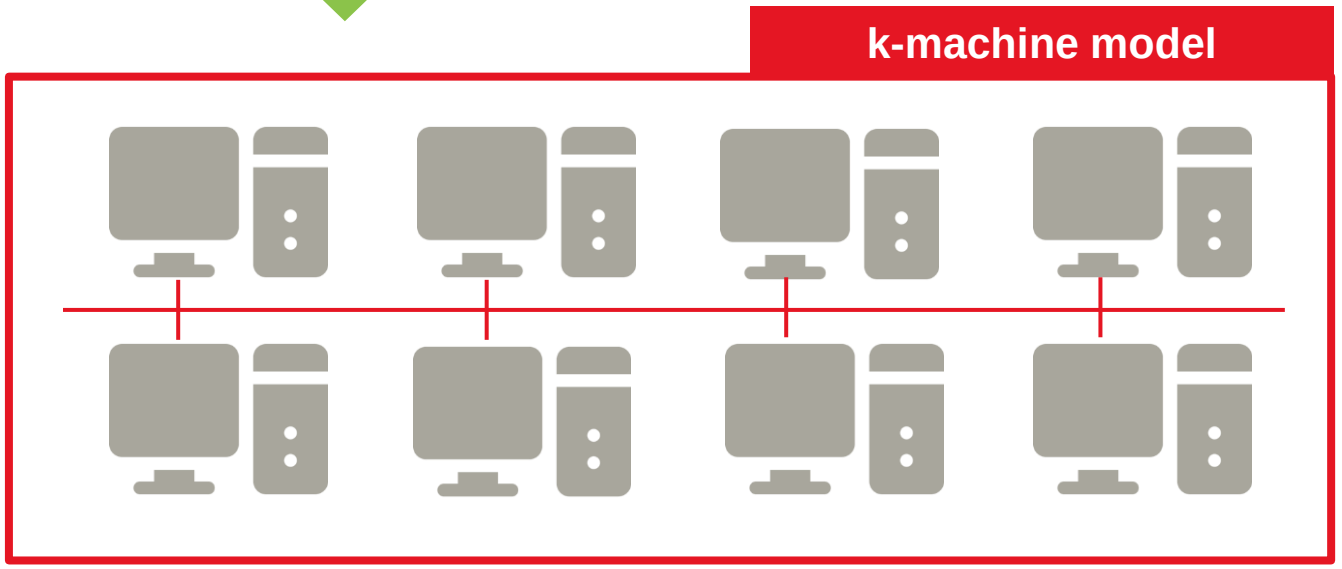
Randomized Algorithm

Color triplet assignment to machines



```
CREATE TABLE Triplet(machine int,color1 int,color2 int,color3 int)
        UNSEGMENTED ALL NODES;
COPY Triplet FROM “lien/vers/triplet_file";
```

```
1,1,1,1
2,1,1,2
3,1,2,1
4,1,2,2
5,2,1,1
6,2,1,2
7,2,2,1
8,2,2,2
```

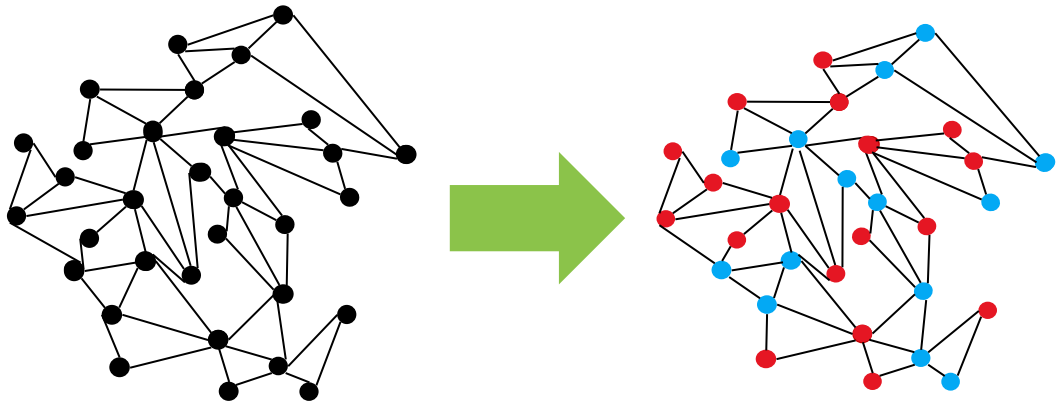


#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Randomized Algorithm

Send edges to proxies (1)

p = $k^{(1/3)}$ color subsets

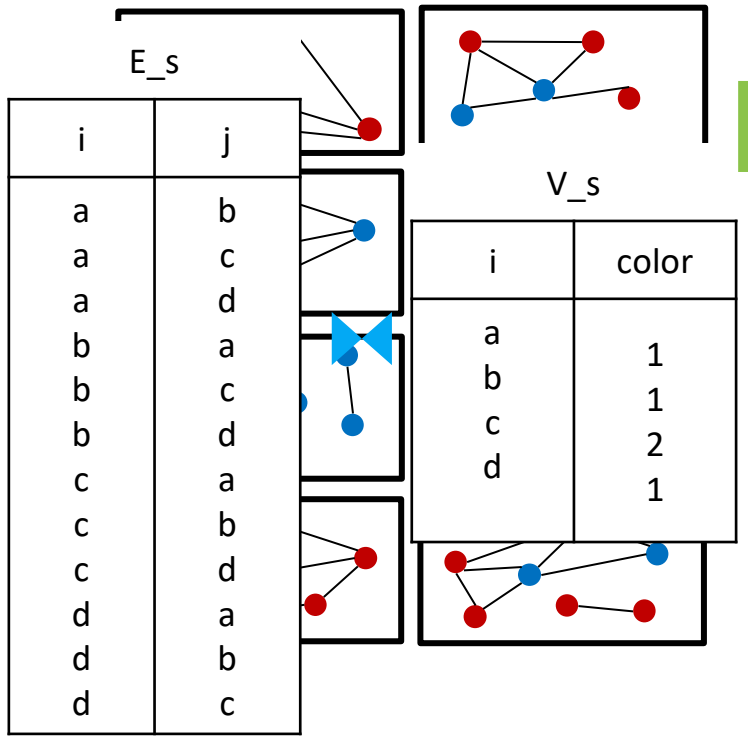


```
CREATE TABLE V_s(i int, color int);
INSERT INTO V_s
   SELECT i, randomint(p)+1
      FROM
         (SELECT DISTINCT i FROM E_s
         UNION
          SELECT DISTINCT j FROM E_s
         )V;
```

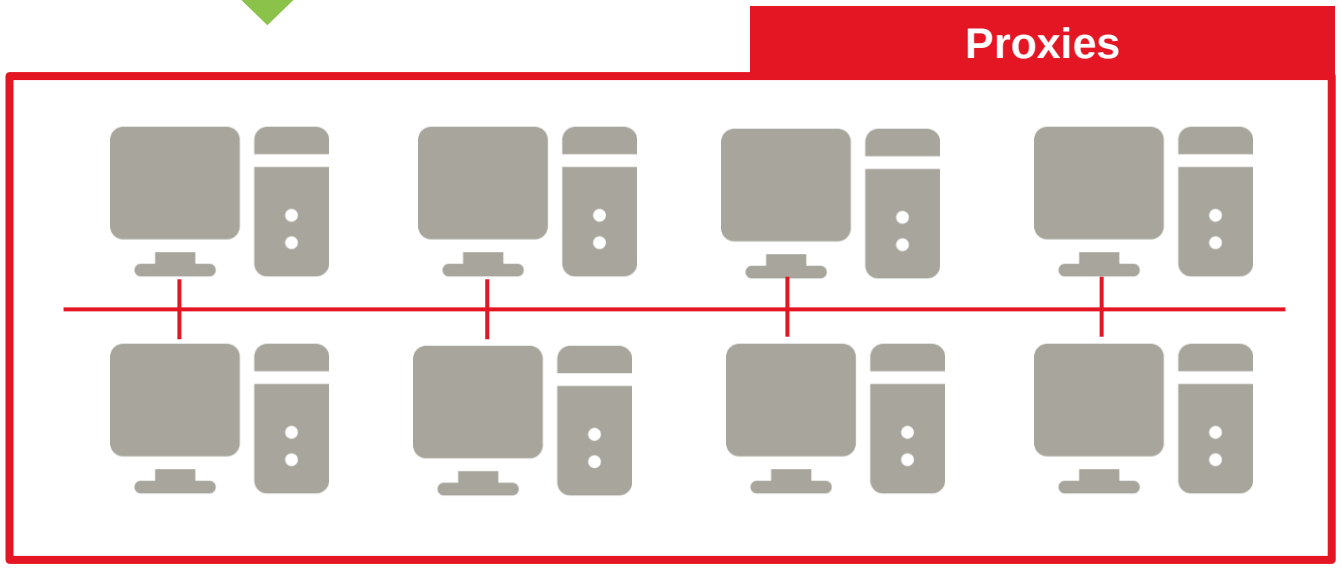

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Randomized Algorithm

Send edges to proxies (2)



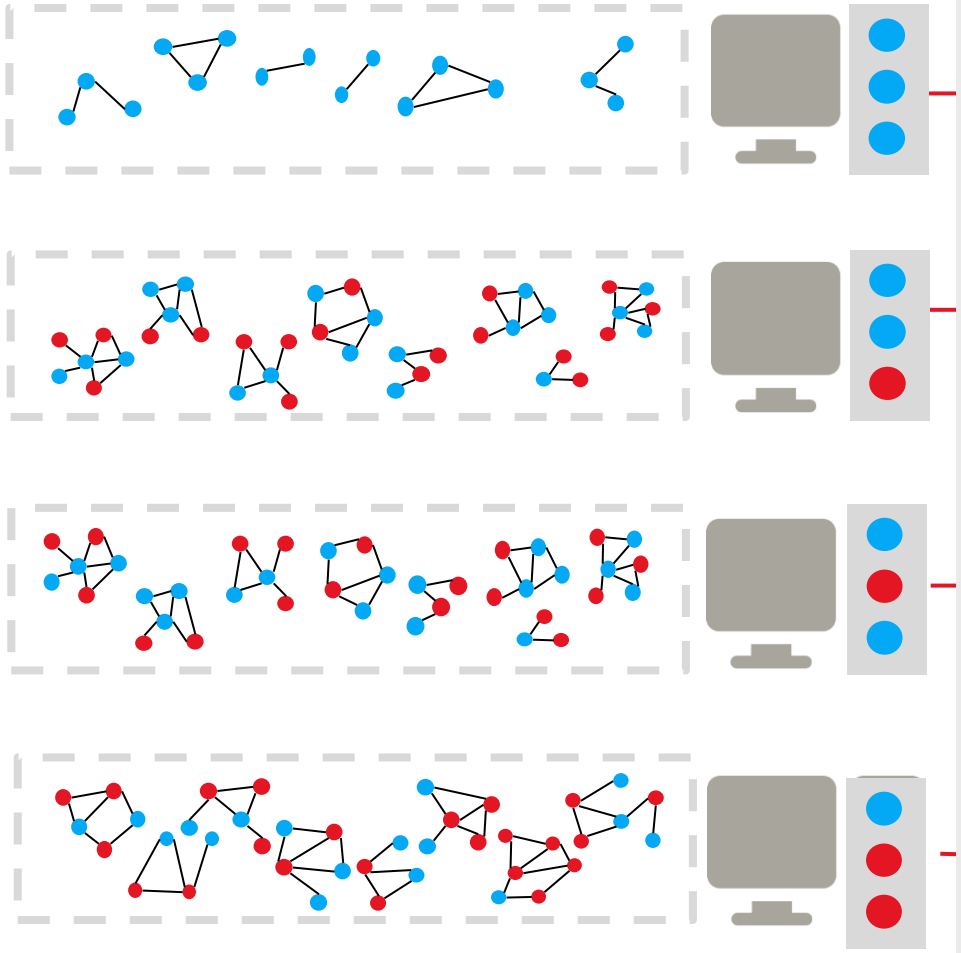

```
CREATE TABLE E_s_proxy (i_color int, j_color int, i int, j int);
INSERT INTO E_s_proxy
   SELECT Vi.color, Vj.color, E.i, E.j
      FROM
         E_s E JOIN V_s Vi ON E.i = Vi.i
         JOIN V_s Vj ON E.j = Vj.i;
```



#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Randomized Algorithm

Edges collecting by local machines



```
CREATE TABLE E_s_local(machine int,i int,j int,i_color int,j_color int);
INSERT INTO E_s_local
  SELECT machine, i, j, i_color, j_color
    FROM
      E_s_proxy E JOIN Triplet edge1 ON E.i_color=edge1.color1
        AND E.j_color=edge1.color2 WHERE E.i<E.j
  UNION
  SELECT machine, i, j, i_color, j_color
    FROM
      E_s_proxy E JOIN Triplet edge2 ON E.i_color=edge2.color2
        AND E.j_color=edge2.color3 WHERE E.i<E.j
  UNION
  SELECT machine, i, j, i_color, j_color
    FROM
      E_s_proxy E JOIN Triplet edge3 ON E.i_color=edge3.color3
        AND E.j_color=edge3.color1 WHERE E.i>E.j;
```

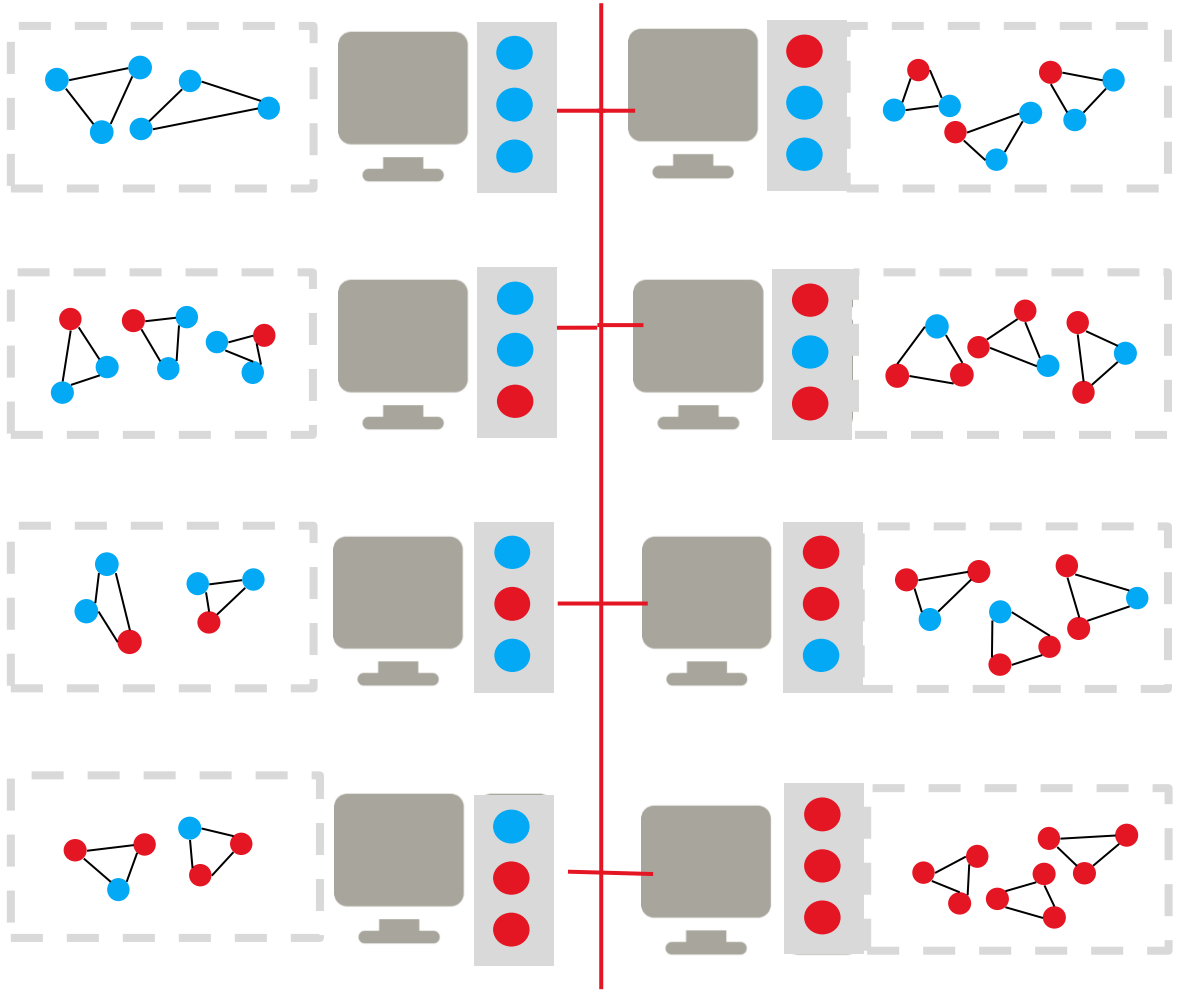

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Randomized Algorithm

Local triangle enumeration

Balanced load

```
SELECT E1.machine, E1.i AS v1, E1.j AS v2, E2.j AS v3
  FROM
    E_s_local E1 JOIN E_s_local E2
      ON E1.machine=E2.machine AND E1.j=E2.i
    JOIN E_s_local E3 ON E2.machine=E3.machine
      AND E2.j=E3.i
    JOIN Triplet T on T.machine = E3.machine
  WHERE E1.i<E1.j AND E2.i<E2.j AND E1.i=E3.j
    AND E1.i_color=T.color_1 AND E1.j_color=T.color_2
    AND E2.j_color=T.color_3
    AND local_node_name()='node_name'
  ORDER BY v1,v2,v3;
```

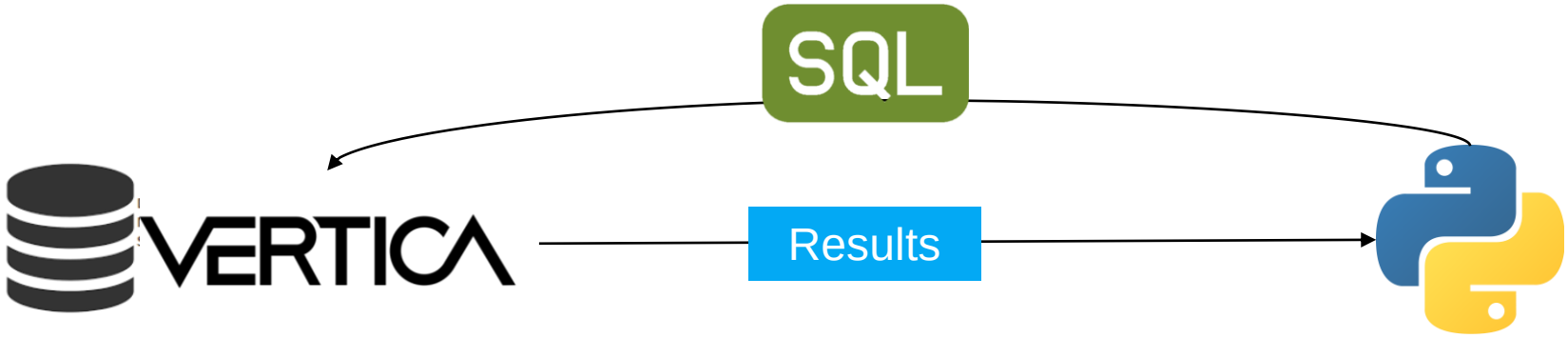


#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Experimental study

#data_sets #cluster #vertica_dbms

Hardware and Software configuration



Operating System: **Ubuntu server 18.04**

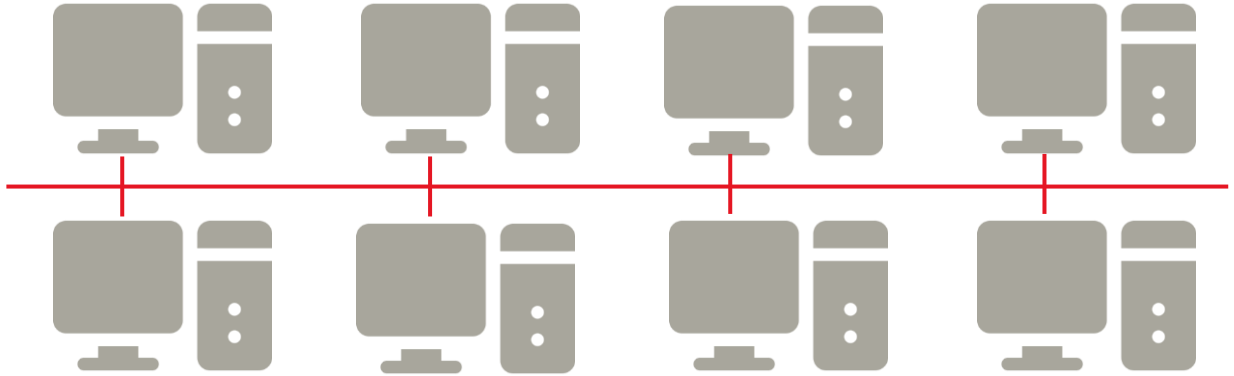
RAM: **48 Go/host**

Storage: **1 To/ host**

CPU: **Intel 4 cores/host**

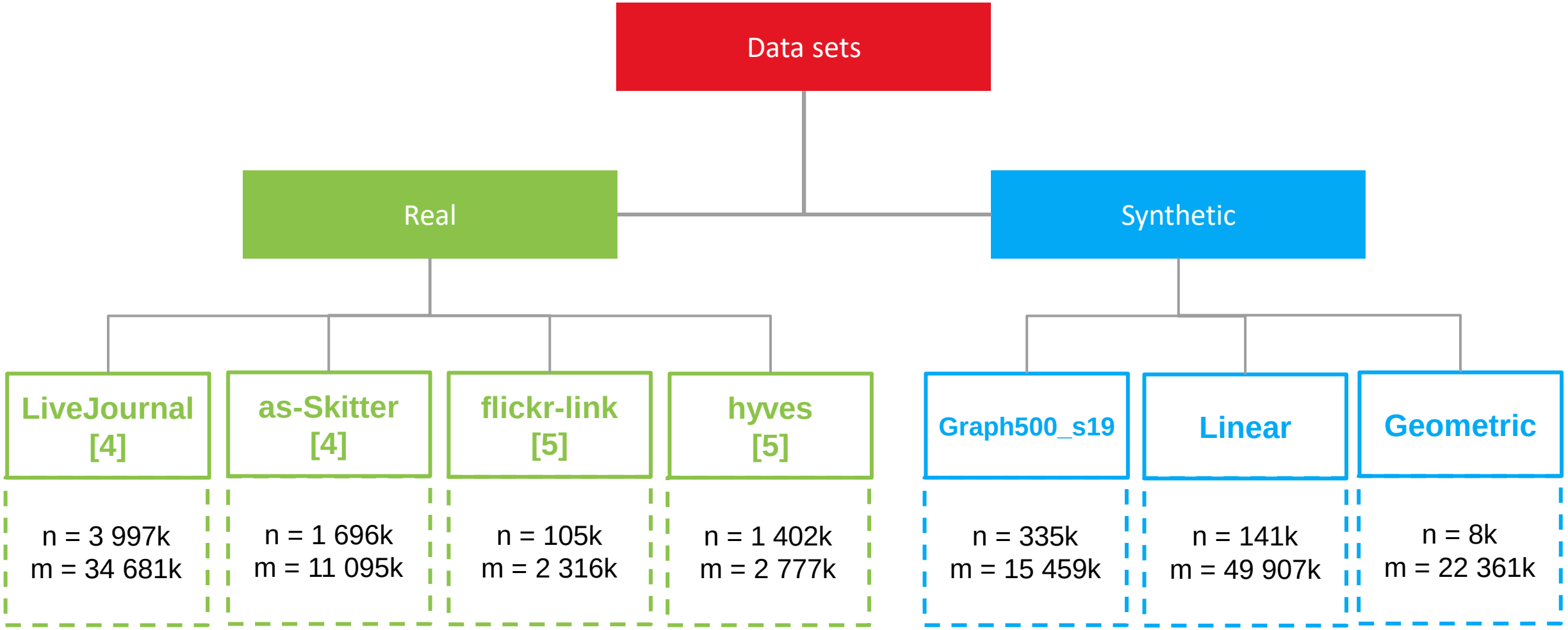
Architecture: **Share-Nothing**

k-machine model



#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

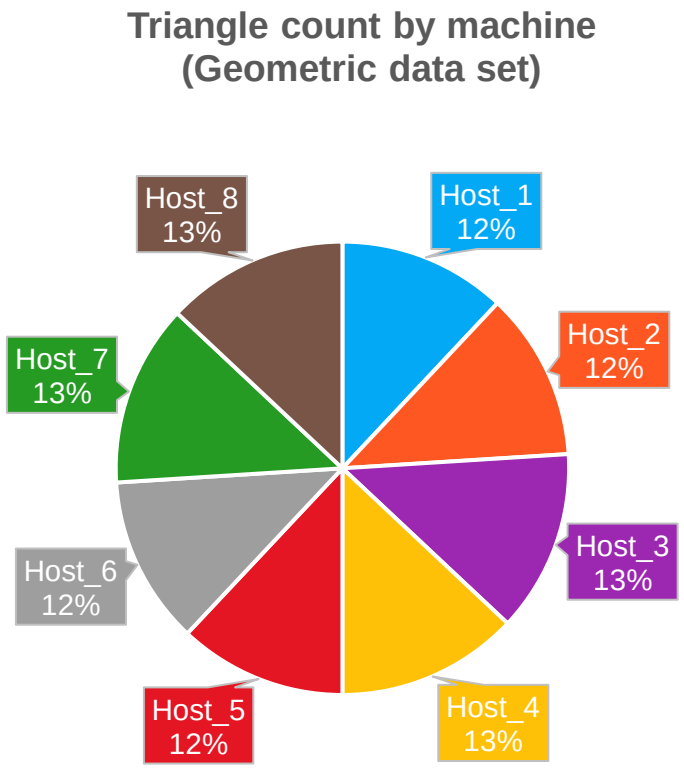
Data sets

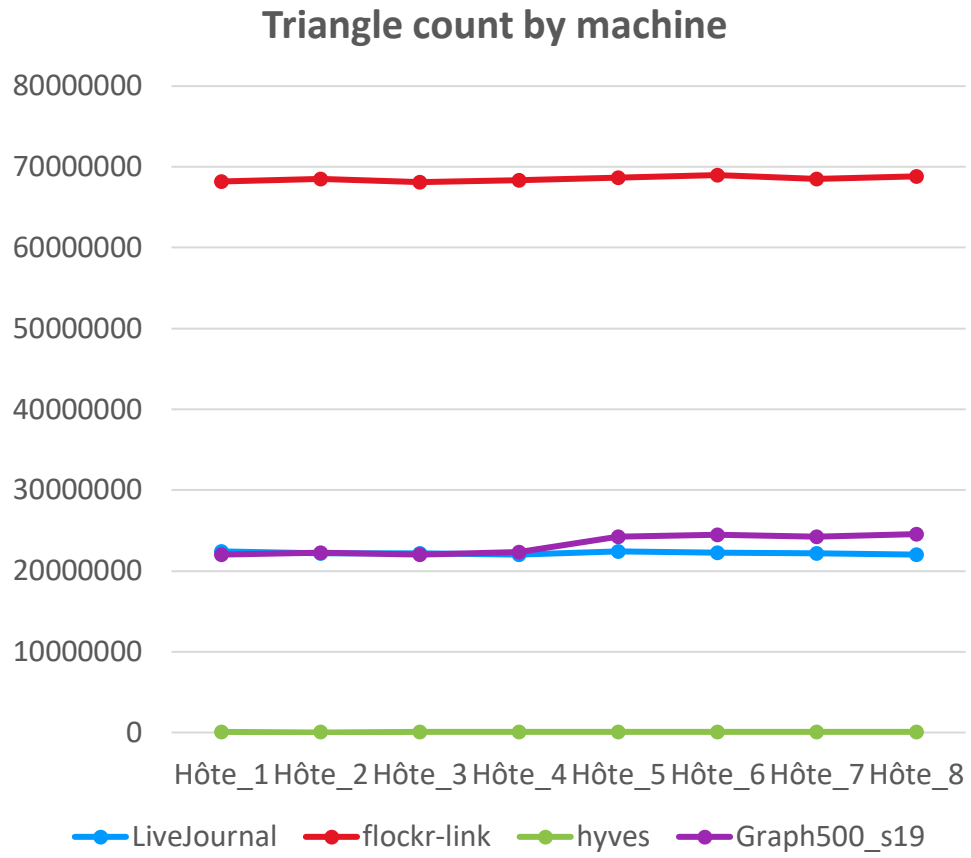


#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Results

Randomized triangle enumeration by machine

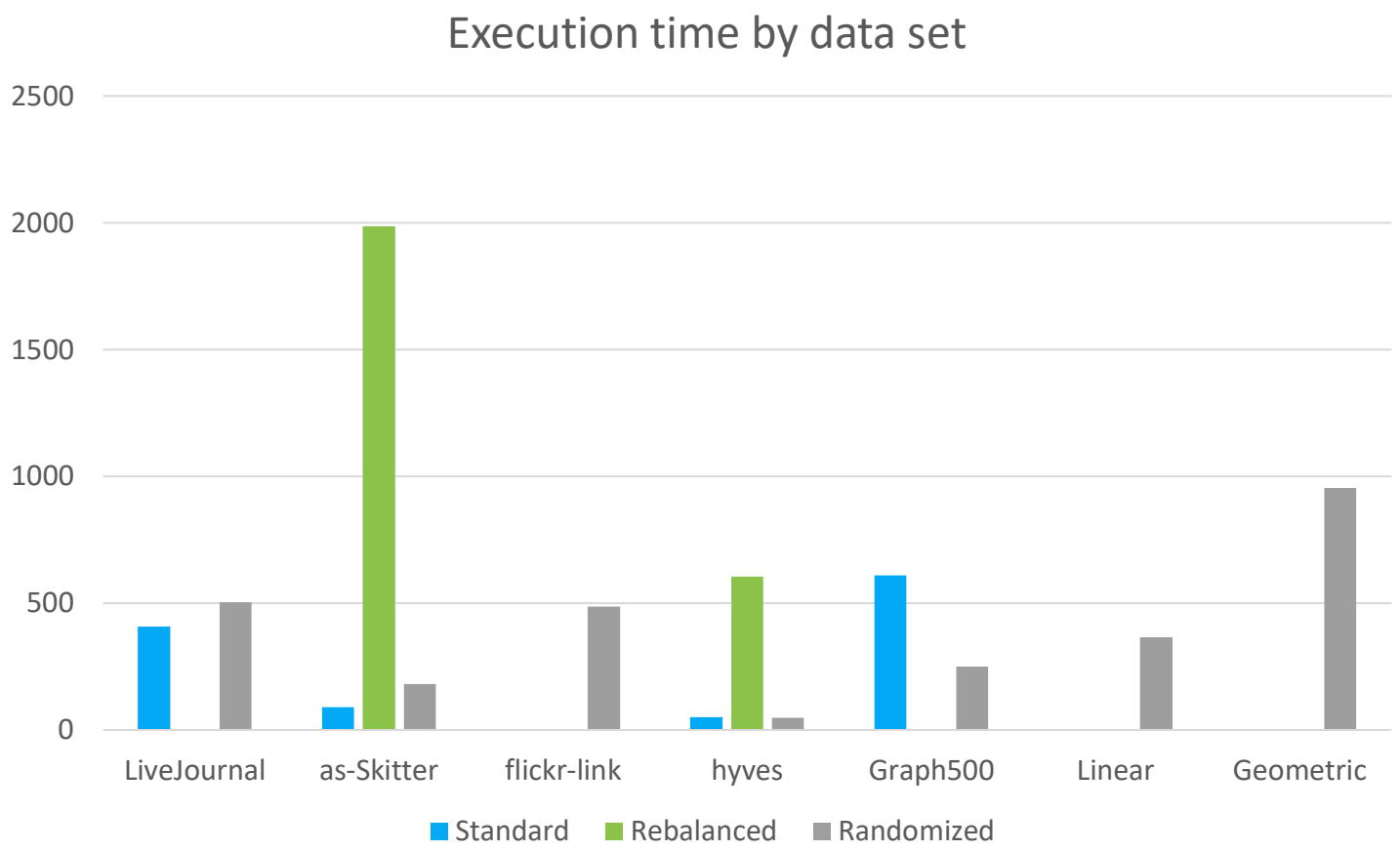




#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Results

Standard algorithm and randomized algorithm comparaison



- Standard algorithm queries are **less efficient** with large-scale graphs and cause **performance problems when scaling.**
- Rebalancing is **expensive** and often **causes memory issues**.
- Randomized algorithm queries **perform better** with graphs having ***skewed data distribution***

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Conclusions

#conclusion #perspectives

Conclusions

Parallel Randomized Algorithm for Triangles Enumeration on Large Graphs Using SQL Queries

Randomized algorithm for triangles enumeration ensures load balancing

The algorithm queries were able to process very large graphs and therefore they ensure scalability

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Perspectives

- Deep study of Randomized Algorithms
- Comparaison with graph analysis engines
- Clique detection using triangles

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

References

[1] https://db-engines.com/en/ranking_categories

[2] Pandurangan, G., P. Robinson, et M. Scquizzato (2018). On the distributed complexity oflarge-scale graph computations. InSPAA, pp. 405–414.

[3] Farouzi, A., L. Bellatreche, C. Ordonez, G. Pandurangan, et M. Malki (2020). A scalable ran-domized algorithm for triangle enumeration on graph based on SQL queries. InTo Appearin DaWaK Conference.

[4] Leskovec, J. et A. Krevl (2014). SNAP Datasets : Stanford large network dataset collection. http://snap.stanford.edu/data

[5] Kunegis, J. (2013). Konect : The koblenz network collection. Association for ComputingMachinery.

#Inside_DBMS #randomized_algorithm #graph #triangle_enumeration

Thank you for your attention

Stay safe!