Università di Roma
Tor Vergata

Università degli Studi di Roma “Tor Vergata”

Gestione di Big *RDF* Data

Manuel Fiorelli

fiorelli@info.uniroma2.it

Distributed RDF store

La taglia di un "big" RDF dataset può impedire di caricarlo e valutare query (SPARQL) usando soltanto le risorse computazionali di un singolo nodo di calcolo.

In questo caso, è necessario adottare un sistema distribuito:

- In queste slide l'enfasi è sui sistemi centralizzati per la gestione di dati RDF all'interno di un cluster
- Non vanno confusi con sistemi P2P in cui viene enfatizzata l'autonomia dei peer e la loro capacità di entrare/uscire dalla rete

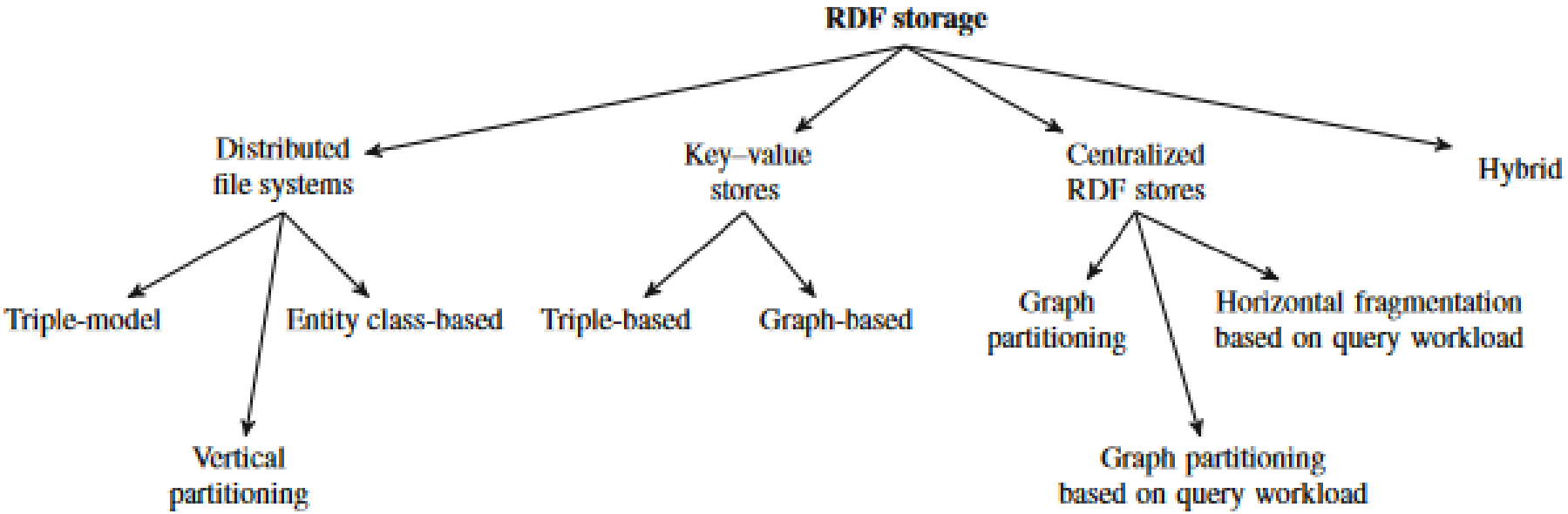
Quanto grande?

Con le opportune risorse si possono gestire anche 20 miliardi di triple su un singolo nodo

Statements	Unique resources	Java heap (min)	Java heap (opt)	Off heap	OS	Total	Repository image
100M	33.3M	1.2GB	3.6GB	370M	2	6GB	12GB
200M	66.6M	2.4GB	7.2GB	740M	3	11GB	24GB
500M	166.5M	6GB	18GB	1.86GB	4	24GB	60GB
1B	333M	12GB	30GB	3.72GB	4	38GB	120GB
2B	666M	24GB	30GB	7.44GB	4	42GB	240GB
5B	1.665B	30GB	30GB	18.61GB	4	53GB	600GB
10B	3.330B	30GB	30GB	37.22GB	4	72GB	1200GB
20B	6.660B	30GB	30GB	74.43GB	4	109GB	2400GB

http://graphdb.ontotext.com/documentation/8.7/standard/requirements.html

RDF Storage



(Kaoudi & Manolescu, 2015)

RDF Storage – triple model

Immagazzino le triple in un file all'interno di un filesystem distribuito (che lo suddividerà in blocchi ripartiti tra i vari data node)

Ciascuna riga del file contiene:

- Una tripla
- Tutte le triple di cui una certa risorsa è soggetto

RDF Storage – Vertical Partitioning

- È una reminiscenza dei primi sistemi che immagazzinavo dati RDF in database relazionali usando una tabella diversa per ciascuna proprietà.
- Le triple sono salvate in tanti file, una per ogni proprietà:
  - Lo scopo è permette un accesso più selettivo ai dati
  - La partizione è basata sulle proprietà anziché sui soggetti o gli oggetti, perché questi ultimi sono generalmente molto più numerosi e di conseguenza darebbero luogo ad un gran numero di file di piccole dimensioni (non ben supportati dal DFS)
- Quando accediamo ai dati, se la proprietà non è nota, occorre scansionare tutti i file

RDF Storage – Vertical Partitioning

I file possono essere divisi ulteriormente:

- Nel caso della proprietà *rdf:type* si usa l'oggetto, che è una classe e tipicamente la T-BOX è piccola

  type#painter

- Nel caso di altre proprietà si usa la classe dell'oggetto (perché gli oggetti sono in genere troppo numerosi)

  paints#artifact

RDF Storage – Vertical Partitioning

Extended vertical partitioning:

Per ogni coppia di tabelle VP_{p_1} e VP_{p_2} calcola diversi semi-join e li materializza se non sono vuoti e se la loro selettività (riduzione delle tabelle VP) è sufficientemente alta

$$\textbf{SS:} \quad VP_{p_1} \ltimes_{s=s} VP_{p_2} \;,\; VP_{p_2} \ltimes_{s=s} VP_{p_1}$$
$$\textbf{OS:} \quad VP_{p_1} \ltimes_{o=s} VP_{p_2} \;,\; VP_{p_2} \ltimes_{o=s} VP_{p_1}$$
$$\textbf{SO:} \quad VP_{p_1} \ltimes_{s=o} VP_{p_2} \;,\; VP_{p_2} \ltimes_{s=o} VP_{p_1}$$

Si basa sulla seguente decomposizione di una join tra due tabelle

$$T_1 \bowtie_{A=B} T_2 = (T_1 \ltimes_{A=B} T_2) \bowtie_{A=B} (T_1 \rtimes_{A=B} T_2)$$

Manuel Fiorelli fiorelli@info.uniroma2.it
http://art.uniroma2.it/fiorelli

RDF Storage – entity class

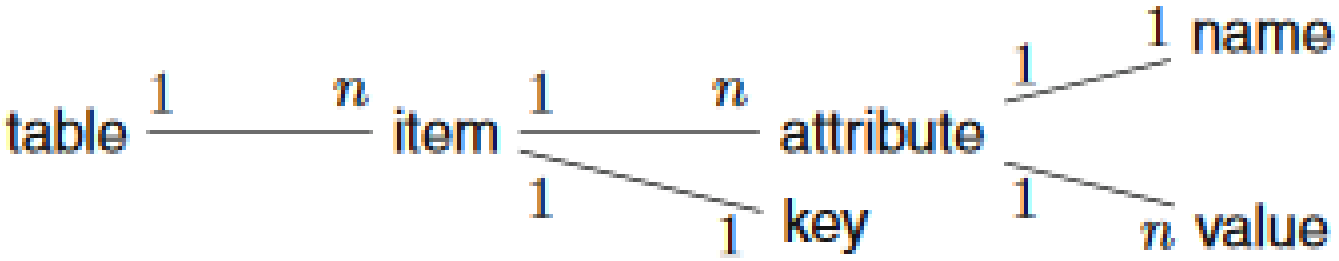
Produce un *summary* di un grafo di entity class:

Una entity class rappresenta un insieme di risorse sufficientemente simili (secondo una certa metrica)

Il grafo compresso viene partizionato, e le entità sono assegnate alla partizione corrispondente alla loro classe.

RDF Storage – key-value database

Modello di dati chiave/valore



(Kaoudi & Manolescu, 2015)

Due operazioni supportate:

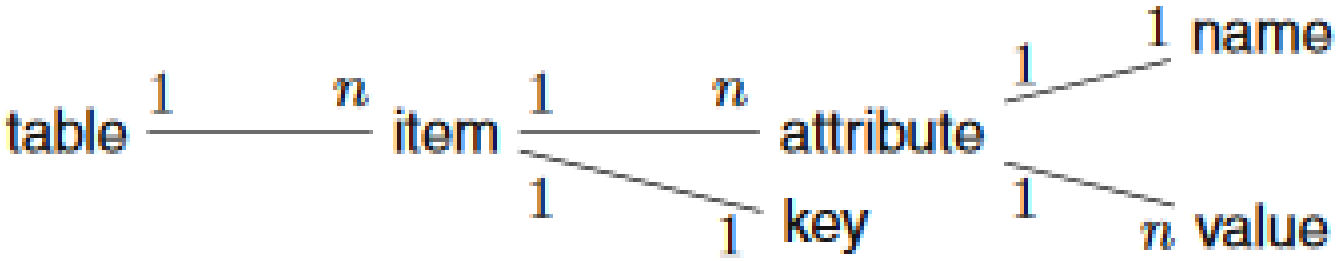
- Aggiungere un valore per una chiave
- Recuperare i valori per una chiave

In base al tipo di indicizzazione ho pattern di accesso diversi:

- Hash: lookup diretto → devo cercare la chiave esatta
- Sorted: prefix lookup → posso cercare un prefisso della chiave

RDF Storage – key-value database - triple

Modello di dati chiave/valore



(Kaoudi & Manolescu, 2015)

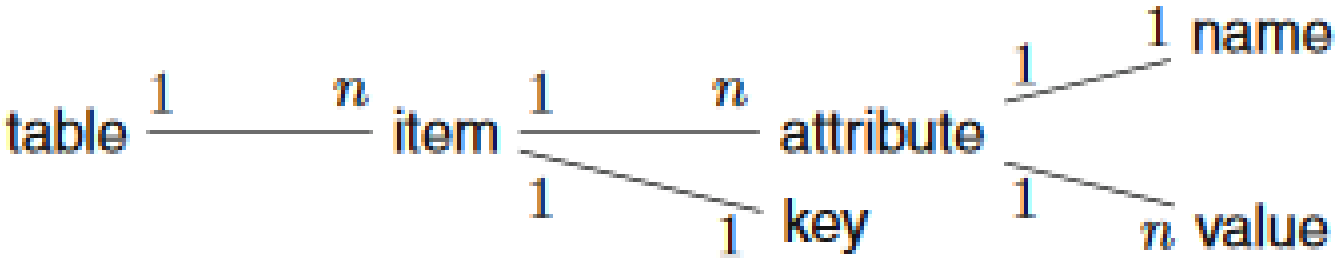
Posso codificare le triple RDF in vari modi per simulare diversi tipi di indici:

(SPO|e|e) con un indice *sorted* mi permette di cercare triple per S, SP o SPO

Manuel Fiorelli fiorelli@info.uniroma2.it
http://art.uniroma2.it/fiorelli

RDF Storage – key-value database - graph

Modello di dati chiave/valore



(Kaoudi & Manolescu, 2015)

Anziché ragionale in termini "relazionali", uso il key/value store per rappresentare il grafo RDF in termini di liste di adiacenza.

RDF Storage – federazione di store centralizzati

Partiziono il dataset RDF tra diversi triple store "tradizionali":

Avendo partizionato i nodi del grafo, possono assegnare le triple alla partizione del soggetto (1-hop directed guarantee) ed anche dell'oggetto (1-hop unidirected guarantee)

- Ciò mi permette di valutare in locale star-pattern (un insieme triple pattern con lo stesso soggetto)

Replicando le triple, possono avere n-hop (uni)directed guarantee, la quale aumenta i tipi di graph pattern che si possono valutare in locale

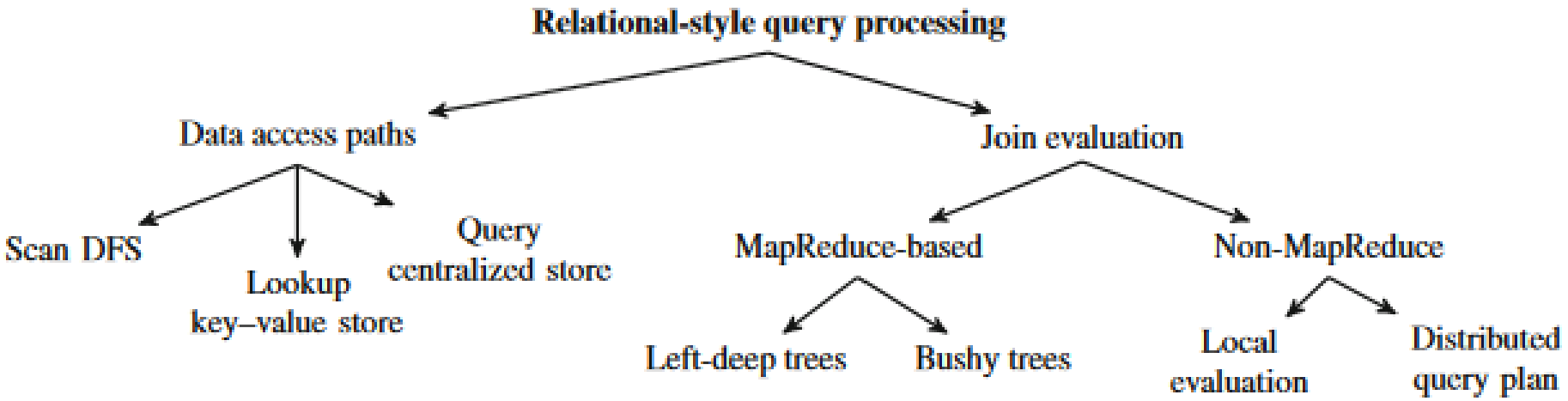
RDF Storage – federazione di store centralizzati

La collocazione delle triple può essere guidata anche dall'analisi delle query:

- Collocando le triple che spesso sono usate insieme per rispondere alle query

Un approccio diverso consiste nel *partizionamento orizzontale delle triple*, analizzando come essere sono usate per rispondere alle query.

Relational-style query processing



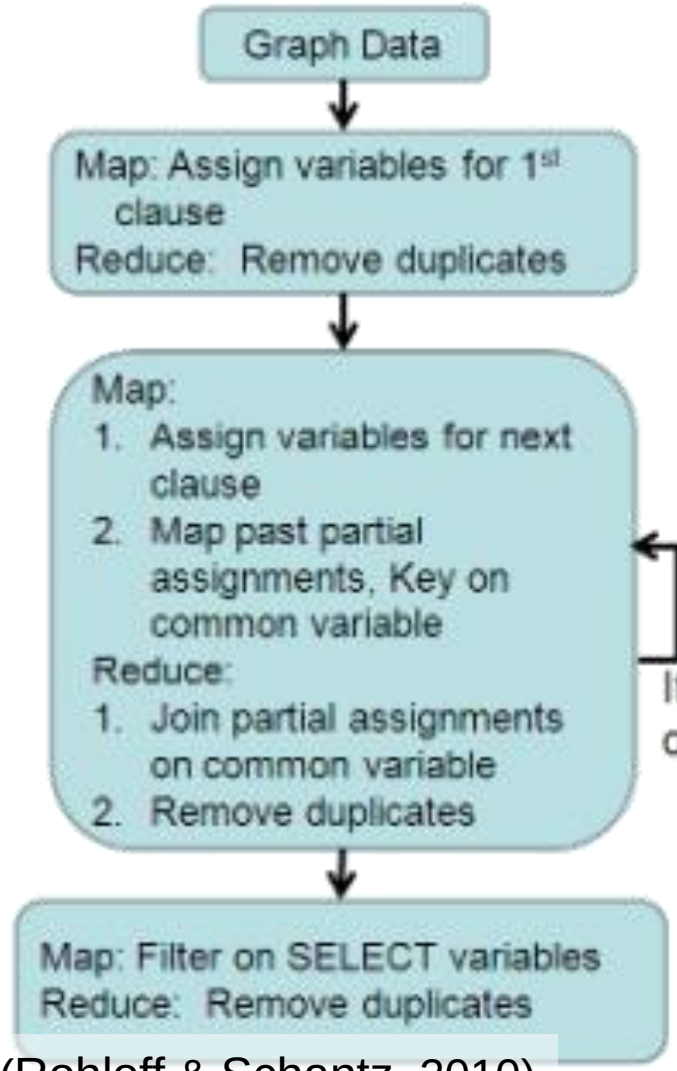
(Kaoudi & Manolescu, 2015)

Relational-style query processing – data access

L'accesso ai dati consiste nel valutare un semplice triple pattern (una tripla le cui componenti possono essere variabili):

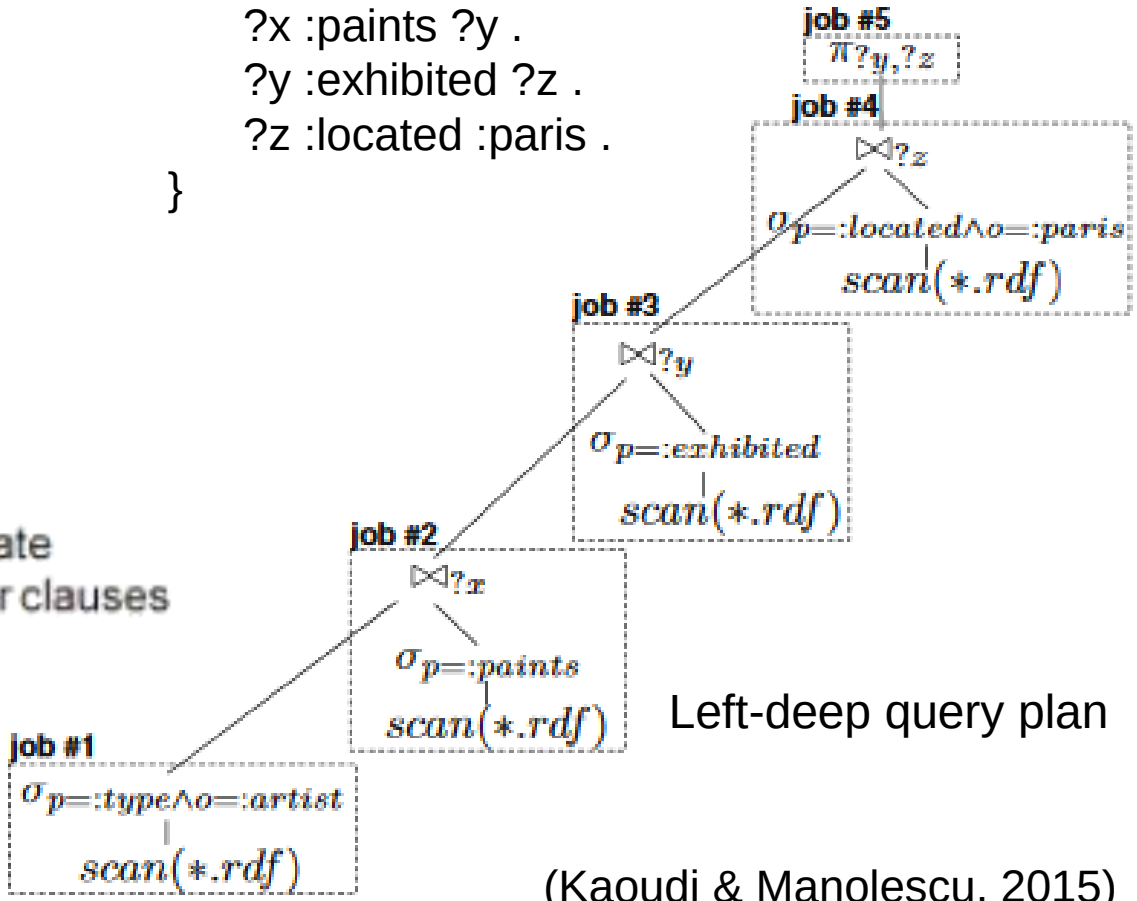
- Scansione degli opportuni file nel DFS (es. nel caso del vertical partitioning, se la proprietà è ground si può scansionare soltanto la VP di quella proprietà)
- Lookup in un key-value store usando la chiave opportuna (come abbiamo visto dipende dal tipo di indicizzazione)
- Sottomissione di una query ad un triple store federato

Relational-style query processing – join con MapRed



(Rohloff & Schantz, 2010)

```
Select ?y ?z WHERE {
      ?x :type :artist .
      ?x :paints ?y .
      ?y :exhibited ?z .
      ?z :located :paris .
}
```



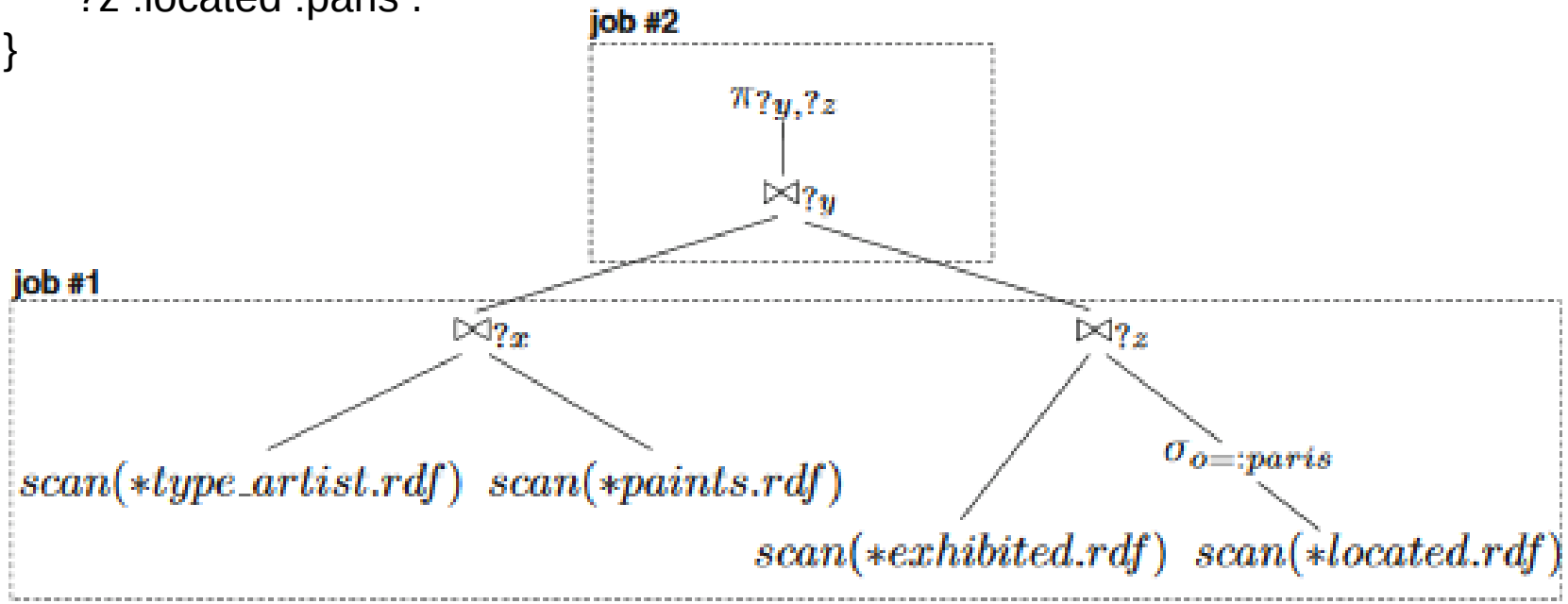
Left-deep query plan

(Kaoudi & Manolescu, 2015)

Relational-style query processing – join con MapRed

```
Select ?y ?z WHERE {
      ?x :type :artist .
      ?x :paints ?y .
      ?y :exhibited ?z .
      ?z :located :paris .
}
```

Bushy query plan che sfrutta inter-operator parallelism



(Kaoudi & Manolescu, 2015)

Relational-style query processing – join fuori da MapRed

- Client-side join

  Tipico dei sistemi che usano key-value store per la persistenza che non supportano la join

- Distributed query plan

  Tipico dei sistemi federati, in cui la query viene suddivisa in frammenti eseguiti localmente in ciascun nodo, mentre i risultati parziali sono combinati nel coordinatore

Relational-style query processing – graph style

Le query sono valutate attraverso un meccanismo di attraversamento dei grafi, che tipicamente richiede la presenza di un soggetto o di un oggetto ground.

Reasoning

Tre tipologie di approcci:

- Materializzazione della chiusura delle implicazioni logiche
- Riscrittura delle query
- Ibridi

Reasoning

Tre tipologie di approcci:

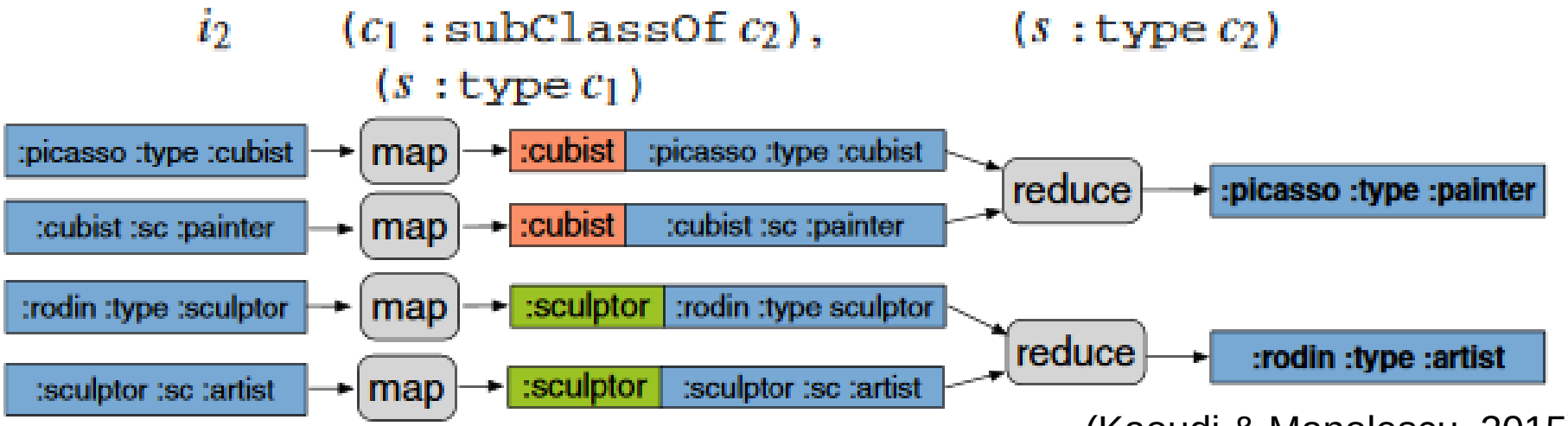
- Materializzazione della chiusura delle implicazioni logiche
- Riscrittura delle query
- Ibridi

Minimal RDFS

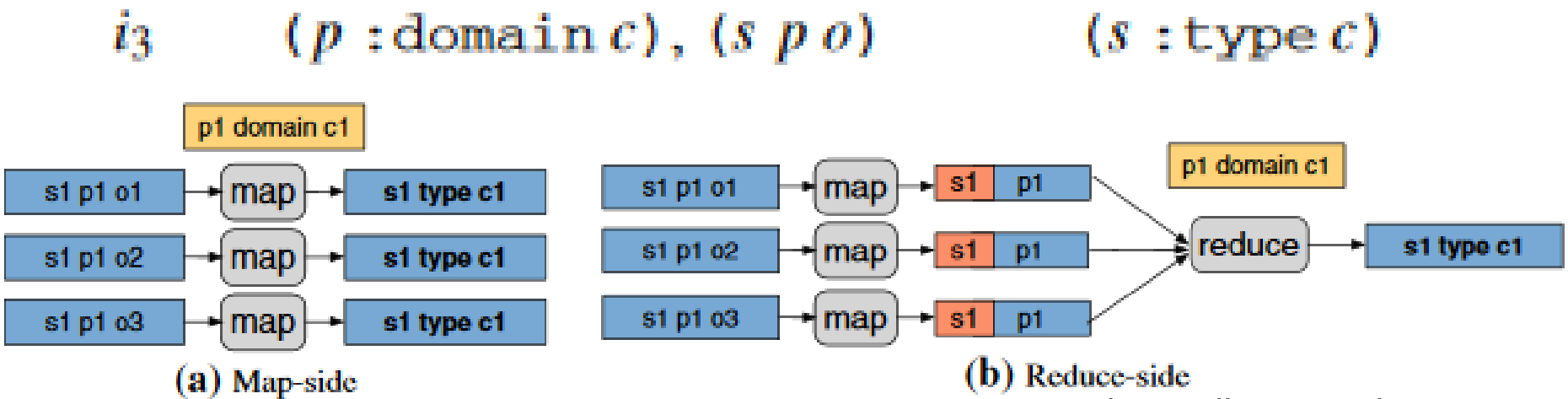
Rule	Triples	Entailed triple
s_1	$(c_1$:subClassOf $c_2)$, $(c_2$:subClassOf $c_3)$	$(c_1$:subClassOf $c_3)$
s_2	$(p_1$:subPropertyOf $p_2)$, $(p_2$:subPropertyOf $p_3)$	$(p_1$:subPropertyOf $p_3)$
i_1	$(p_1$:subPropertyOf $p_2)$, $(s\ p_1\ o)$	$(s\ p_2\ o)$
i_2	$(c_1$:subClassOf $c_2)$, $(s$:type $c_1)$	$(s$:type $c_2)$
i_3	$(p$:domain $c)$, $(s\ p\ o)$	$(s$:type $c)$
i_4	$(p$:range $c)$, $(s\ p\ o)$	$(o$:type $c)$

- Se ci sono due antecedenti, essi hanno una variabile in comune
- Alcune regole possono usare le inferenze prodotte da altre regole → necessità di ordinare l'applicazione delle regole per minimizzare le iterazioni

Reasoning with Map Reduce

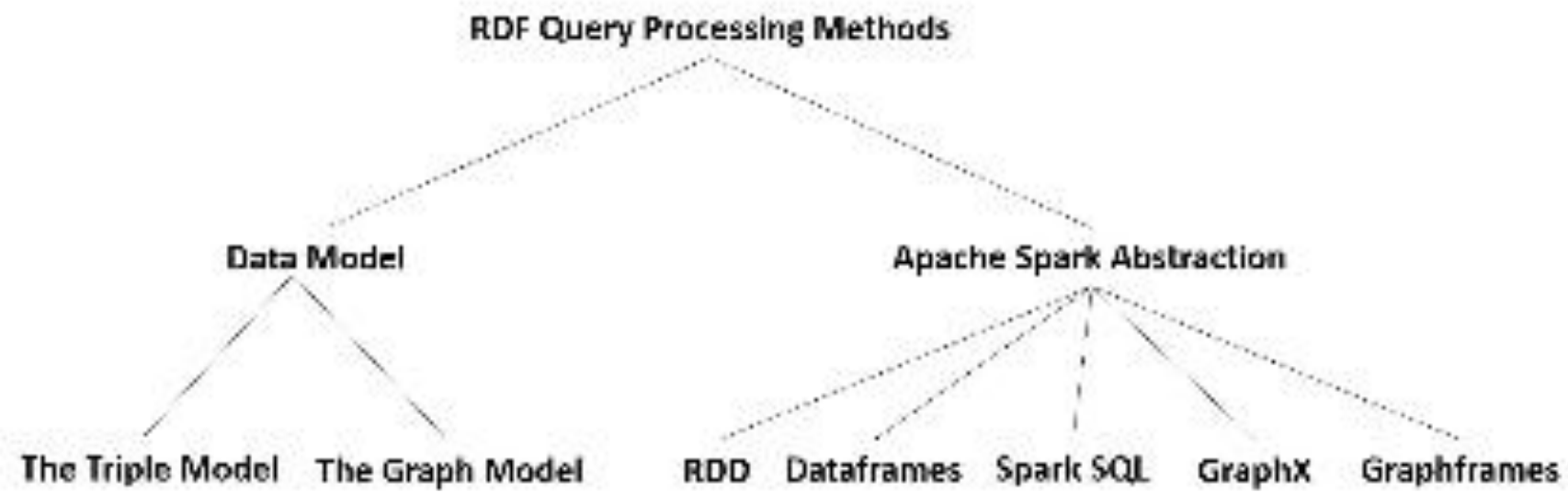


(Kaoudi & Manolescu, 2015)



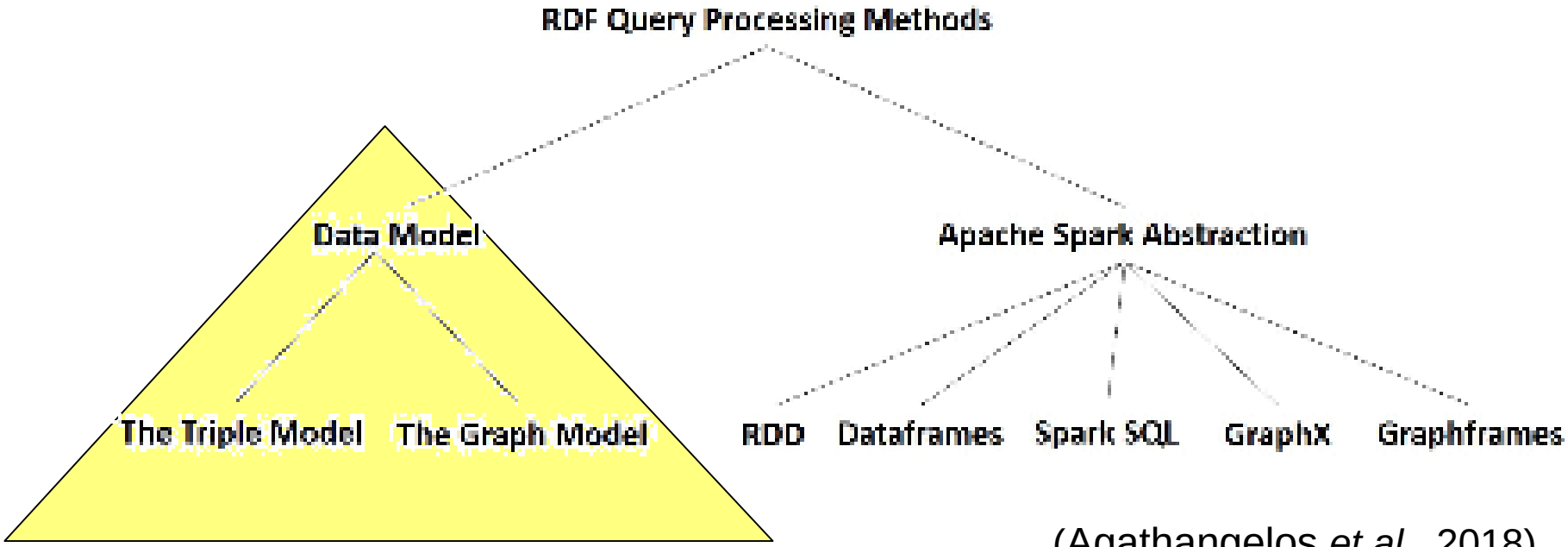
(Kaoudi & Manolescu, 2015)

Un'altra classificazione basata su Spark



(Agathangelos *et al.*, 2018)

Un'altra classificazione basata su Spark



(Agathangelos *et al.*, 2018)

Il punto è se vediamo un dataset come un insieme di triple, oppure come un grafo. Cio si rifletterà nella scelta di diverse API:

- Triple model →RDD, DataFrame
- Graph model → GraphX

RDF Processing con Spark

Gli RDD, i DataFrame e SparkSQL ci offrono un'API per cui è relativamente facile esprime una query SPARQL.

Quando usiamo GraphX, le query sono valutate mediante algoritmi iterativi:

- Ciascun nodo deve determinare la validità di una soluzione parziale dal proprio punto di vista
- Tramite scambio di messaggi si propagano le soluzioni parziali e si giunge
- Si converge verso le soluzioni

Riferimenti

- Agathangelos, G., Troullinou, G., Kondylakis, H., Stefanidis, K., & Plexousakis, D. (2018). RDF Query Answering Using Apache Spark: Review and Assessment. In *IEEE ICDE*.
- Kaoudi, Z., & Manolescu, I. (2015). RDF in the clouds: a survey. *The VLDB Journal—The International Journal on Very Large Data Bases*, *24*(1), 67-91.
- Rohloff, K., & Schantz, R. E. (2010, October). High-performance, massively scalable distributed systems using the MapReduce software framework: the SHARD triple-store. In *Programming support innovations for emerging distributed applications* (p. 4). ACM.
- Urbani, J., Kotoulas, S., Oren, E., & Van Harmelen, F. (2009, October). Scalable distributed reasoning using mapreduce. In *International Semantic Web Conference* (pp. 634-649). Springer, Berlin, Heidelberg.