RICERCHE TOP-K EFFICIENTI IN SISTEMI DISTRIBUITI

A. Vlachou C. Doulkeridis K. Nørvåg M. Vazirgiannis

Gruppo 11

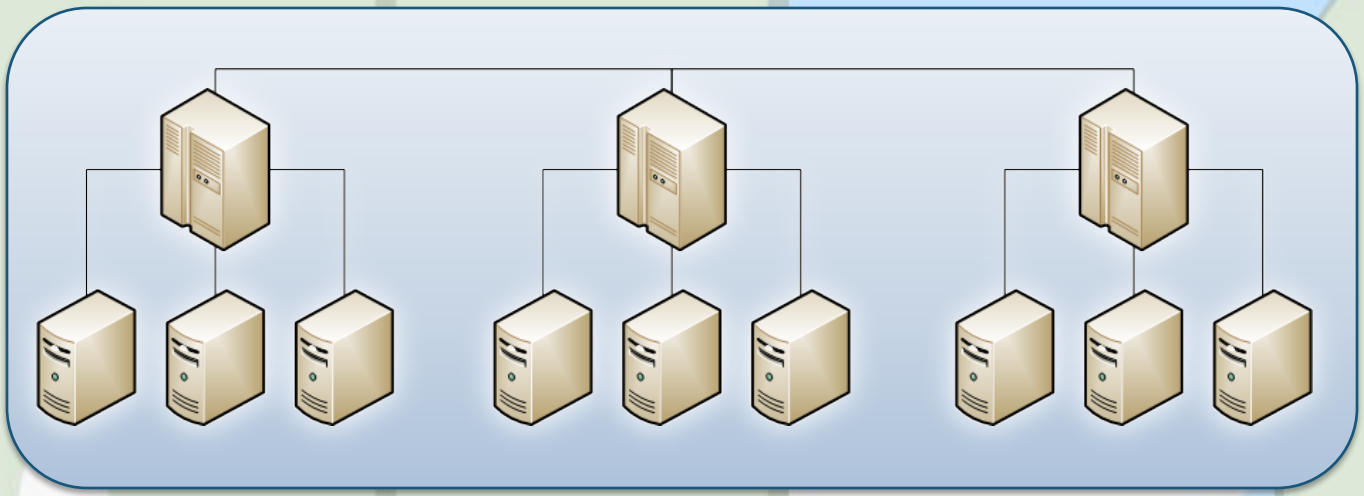
Denis Billi Davide Pompeo

DEIS
Università di Bologna

20 Maggio 2010

PROBLEMA

- Effettuare query top-k su un sistema altamente distribuito, sfruttando il P2P

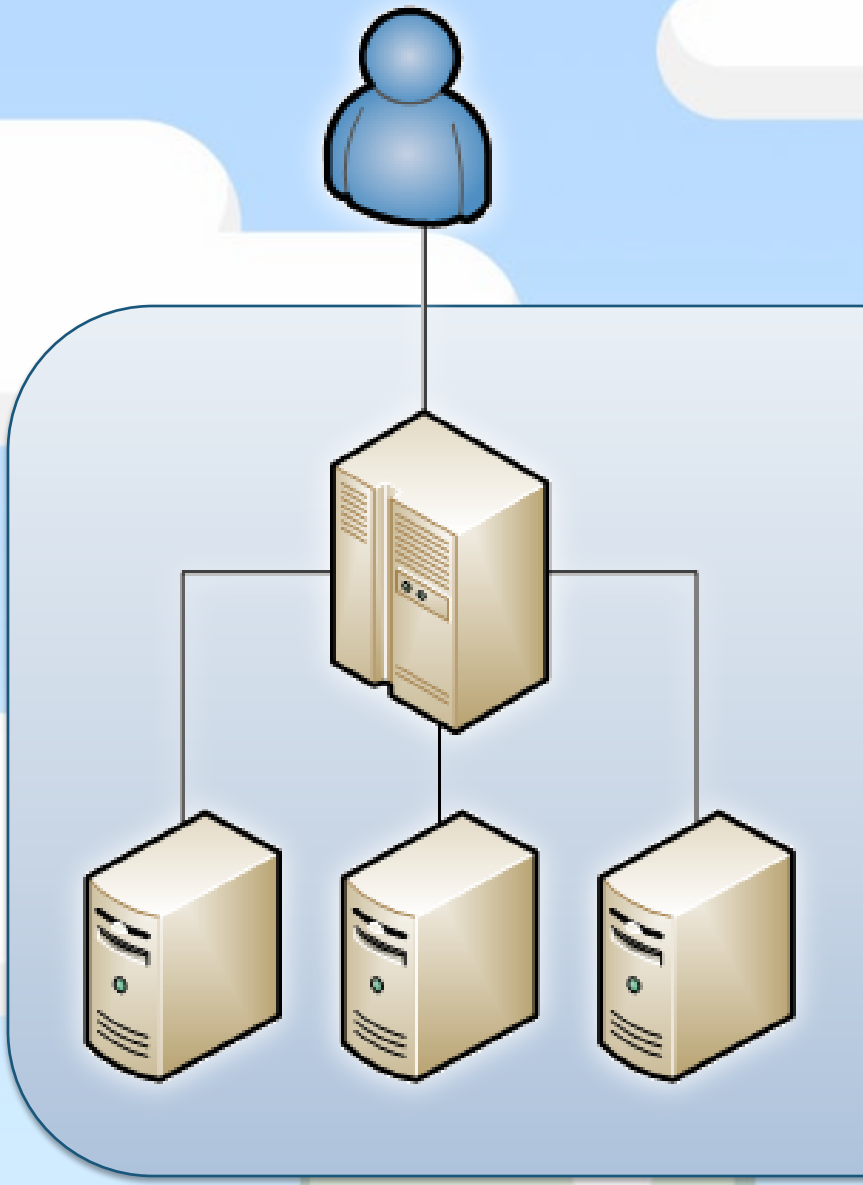


Nodi Principali
(SP = Super Peers)

Nodi Secondari
(P = Peers)

SCENARIO

- I dati che si intende cercare sono divisi tra i database locali dei nodi secondari
- Gli SP mantengono una **cache** di parte dei dati dei propri peer
- Un utente può inviare una **k-query** ad un SP casuale (chiamato SPq)
- Gli SP computano **insieme** il risultato della query dell'utente



OBIETTIVO

Limitare l'overhead dovuto al trasferimento di *dati* superflui

Limitare la comunicazione tra un SP ed i rispettivi peer

Limitare la comunicazione tra un SP e l'altro

Soluzione:

SPEERTO

Skyline-based Peer to Peer Top-K Query Processing

CASO PRATICO

MARIO E IL SUO KART

- Mario deve andare a salvare la principessa Peach (come al solito).
- Dopo 20 anni di salti e piroette a piedi, stavolta decide di usare il suo kart
- Ci sono diverse strade per giungere a destinazione
- Mario è un convinto ambientalista quindi il suo kart funziona a metano:
  i distributori sono sparsi per tutta la mappa

I SUPER PEER

- Mario chiede ai suoi amici (i Super Peer) un elenco delle **migliori k strade** secondo le sue preferenze in termini di lunghezza del percorso e numero di distributori presenti nel tragitto
- La richiesta può essere effettuata ad **uno qualsiasi degli SP**, che ha la possibilità di consultarsi con gli altri nel caso possiedano dati migliori

L Lunghezza in decine di Km	N 40/NumDistributori
4	2,86
0,5	6,67
8	1,48
5	4
2	8
3	8
7	4

KSKY

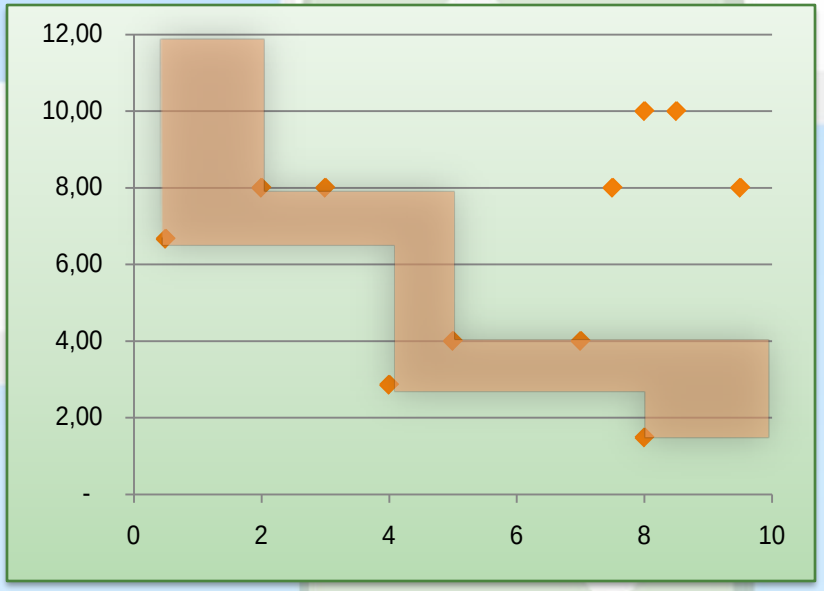
COSA CONOSCE OGNI SP?

- Ogni SP prepara preventivamente la lista dei "***percorsi migliori***" a loro conosciuti (KSKY) facendo una query ai suoi peer, gli unici possessori delle mappe dei percorsi (data objects)

- I percorsi **vengono ordinati in base agli scores**, in questo caso **L** (lunghezza in decine di Km) ed **N** (40/ nd, dove nd è il numero di distributori sul percorso)

- Nel risultato di ogni query ci sono solo i percorsi che sono dominati al massimo da **K - 1 percorsi (dalla definizione di K-Skyband)**

- Un percorso A "domina" un percorso B se si verificano le seguenti condizioni:
  - $L_A \leq L_B$
  - $N_A \leq N_B$
  - $L_A < L_B$ oppure $N_A < N_B$



PERCHE' LE KSKY?

- **Limitano le comunicazioni tra SP e rispettivi peer**, ovvero limitano la quantità di dati da trasferire tra SP e peer sottostanti, che porterebbe ad una perdita di efficienza.

- **Questo non funziona sempre**, ma solo nel caso di Top-k query con **k ≤ K**
  - nel caso che Mario chieda un numero di percorsi superiori a K ogni SP è costretto a chiedere altri dati ai suoi **peer**: perdita di efficienza

- Nonostante tutto è buona abitudine **aggiornare periodicamente la K-Skyband** di ogni SP
  - con il passare del tempo i **peer** potrebbero imparare nuovi percorsi!

SKY

Cosa condividono gli SP

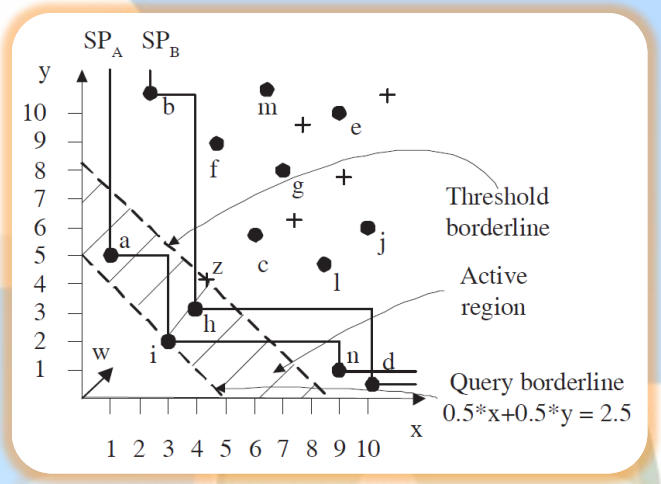
Questo ci permette di avere una rete sempre aggiornata di ciò che accade tra gli SP, che condividono le **skyline** composte da **routing object** (SKY)

- Una **skyline** è una versione particolare di K-skyband (con K=1) in cui ogni suo dato non è dominato da nessun altro
- Un **routing object** è una versione ottimizzata di data object, tale che sia molto più leggera da trasferire tra un SP e l'altro
- *ogni SP conosce gli oggetti appartenenti alle altre skyline*, senza tuttavia possedere i veri e propri data object
- ogni volta che un SP viene a conoscenza di un nuovo percorso migliore, *avverte tutti gli altri*

Una costellazione di SP permette ad ognuno di essi di conoscere gli oggetti delle skyline degli altri senza conoscere effettivamente i dati ed inoltre ogni volta che viene aggiunto un percorso in uno qualsiasi degli SP, gli altri ne vengono subito a conoscenza

PERCHE' SKY?

- Essendo che, per definizione, la skyline è composta da una serie di oggetti non dominati da nessun altro, allora sappiamo con certezza che essa contiene la **Top-1** di ogni possibile query
- Per ogni oggetto trovato dalla query nello spazio dei routing objects sappiamo che l'SP a cui appartiene contiene **almeno un data object da immettere nel risultato finale**
- A prescindere dalla query che verrà fatta:
  - Sappiamo quali SP interrogare
  - Conosciamo la regione attiva in cui i risultati sono contenuti



> PERFORMANCE

Algoritmo SPEERTO

Query processing on super-peer SP_q

```
01: Input: Query q_k(f)
02: list = {∅}
03: list = SP_Q.query ∪ SKY_i(q_k(f))
04: threshold = f(list[k])
05: c = 0
06: while (c < k) do
07:    next obj = list.pop()
08:    if next obj is a routing object then
09:          SP = next obj.super-peer()
10:          temp = SP.query(q_{k-c}(f), threshold)
11:          list.removeRoutObj(SP)
12:          list.add(temp)
13:    else
14:          return next object to the user
15:          c = c + 1
16:    end if
17:    threshold = f(list[k - c])
18: end while
```


Andiamo con ordine!

PASSO 1

SCELTA SP_q

SPq

	L	N
f	4	2,86
c	0,5	6,67
d	8	1,48
	5	4
	2	8
	3	8
	7	4

	L	N
g	3	2
i	9	1
a	1	8
	4	2,5
	7	3,08
	5	6,67
	4	10

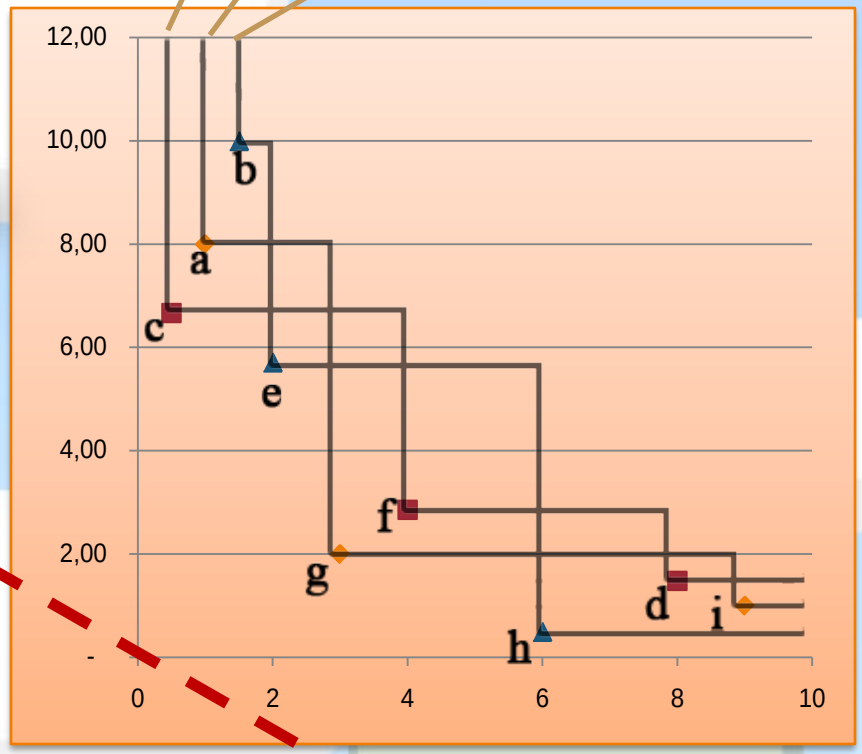
	L	N
e	2	5,71
h	6	0,5
b	1,5	10
	8	1
	5	5,71
	4	8
	3	10

DK

Yoshi

Luigi

k=3



Threshold: ?

Lista risultati: { }

PASSO 1

SCELTA SP_q

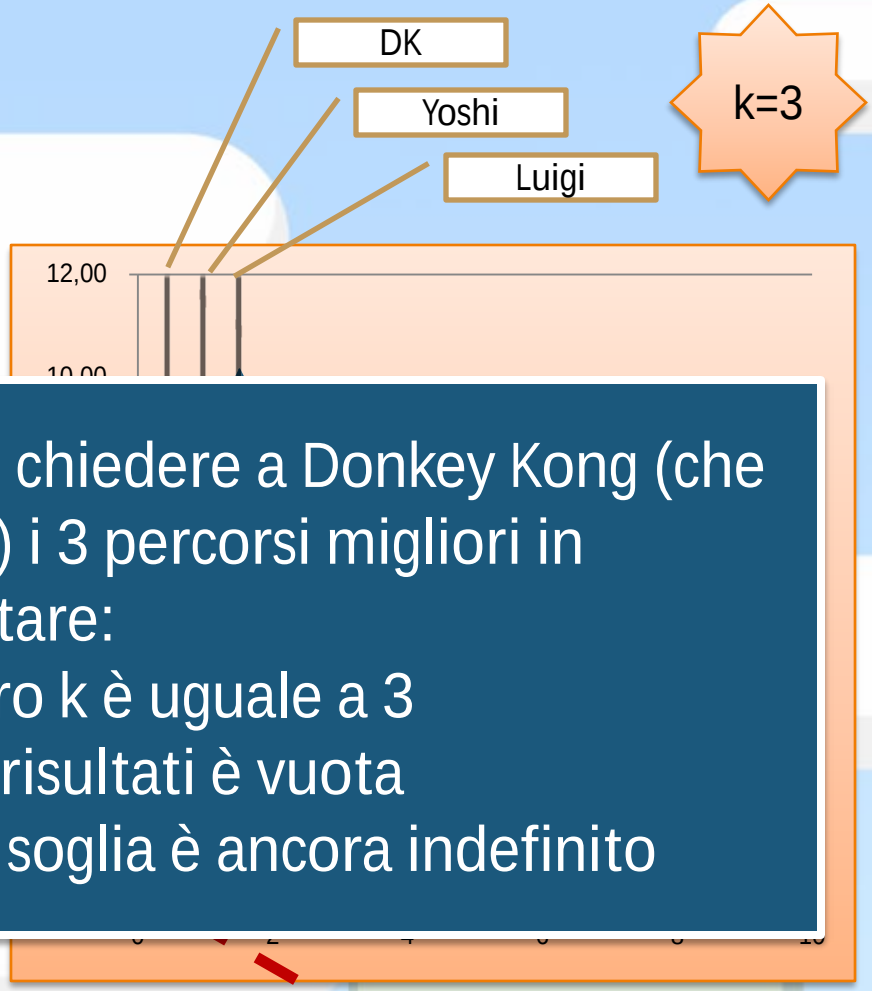
	L	N
f	4	2,86
c	0,5	6,67
d	8	1,48
	5	4
	2	8
	3	8
	7	4

	L	
g	3	
i	9	
a	1	
	4	
	7	
	5	6,67
	4	10

4	8
3	10

Mario decide di chiedere a Donkey Kong (che chiamiamo **SPq**) i 3 percorsi migliori in assoluto. Da notare:

1) il parametro k è uguale a 3
2) la lista dei risultati è vuota
3) il valore di soglia è ancora indefinito



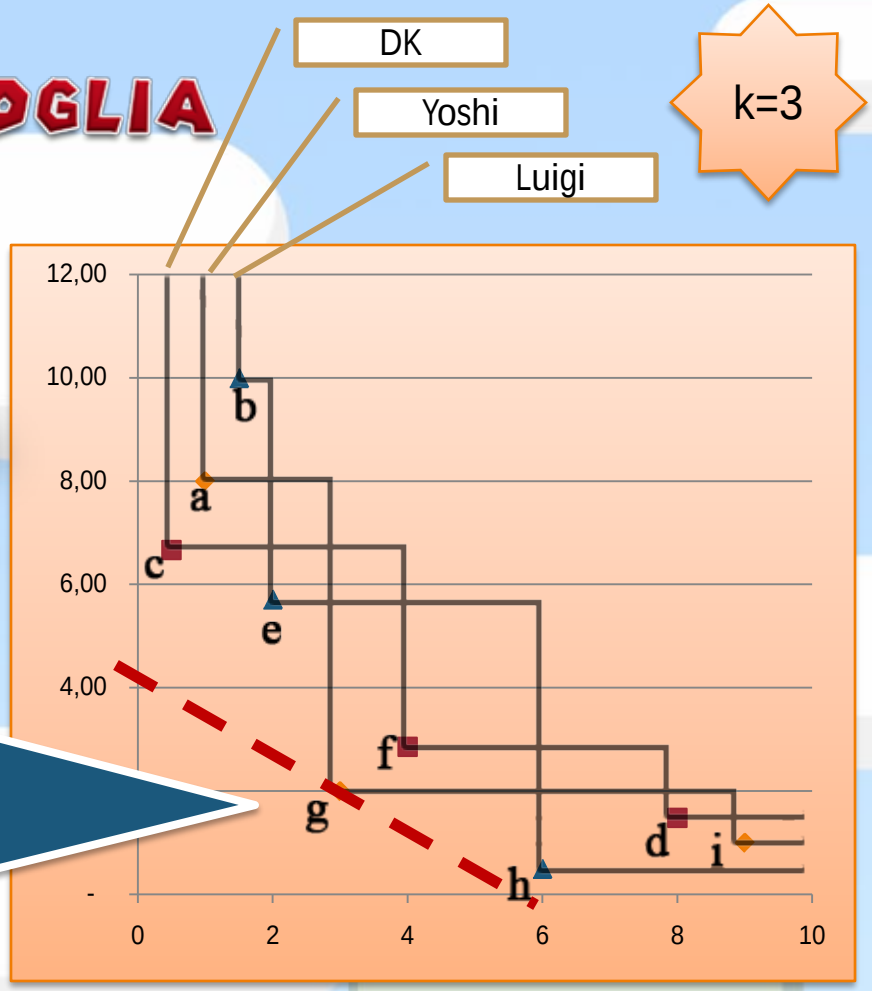
Threshold: ?

Lista risultati: { }

PASSO 2

LISTA RISULTATI E SOGLIA

1. Donkey Kong scorre i routing object della SKY…
2. Il primo che trova è g e pertanto lo aggiunge alla lista dei risultati.



7	4
4	10
3	10

Threshold: ?

Lista risultati: { *g (3; 2)*}

PASSO 2

LISTA RISULTATI E SOGLIA

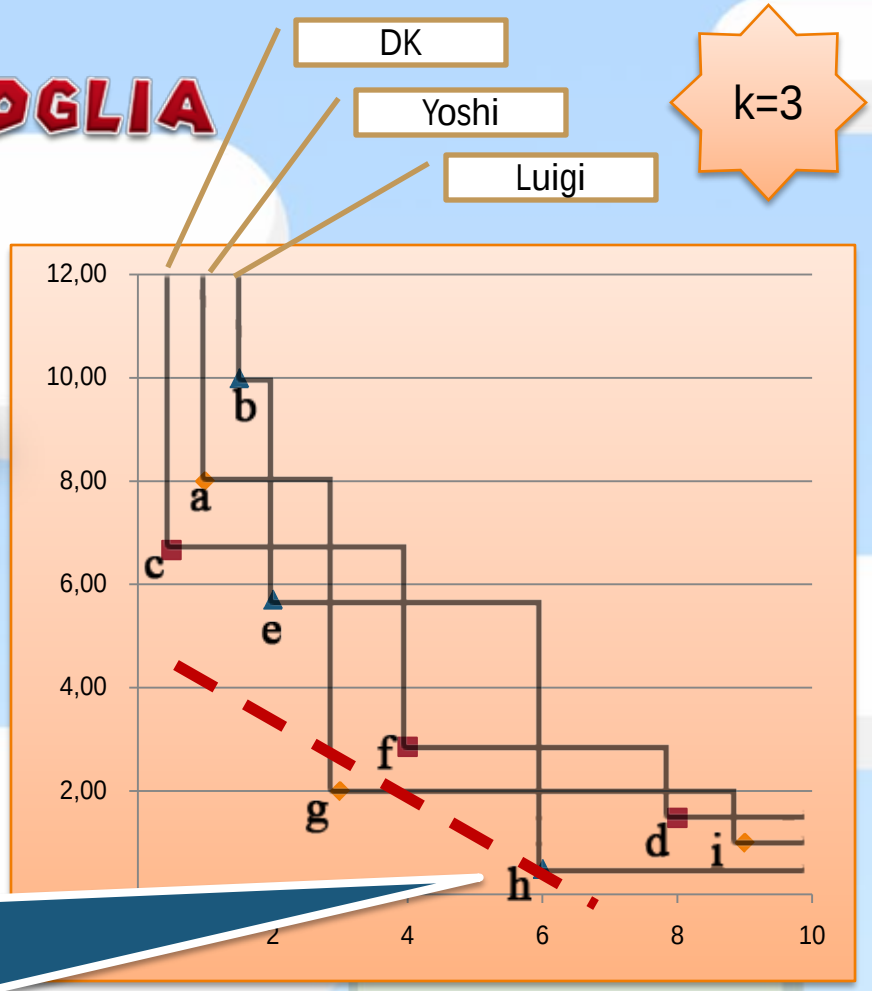
SPq

	L	N
f	4	2,86

	L	N
g	3	2

	L	N
e	2	5,71

Il secondo che trova è h e pertanto lo aggiunge alla lista dei risultati.



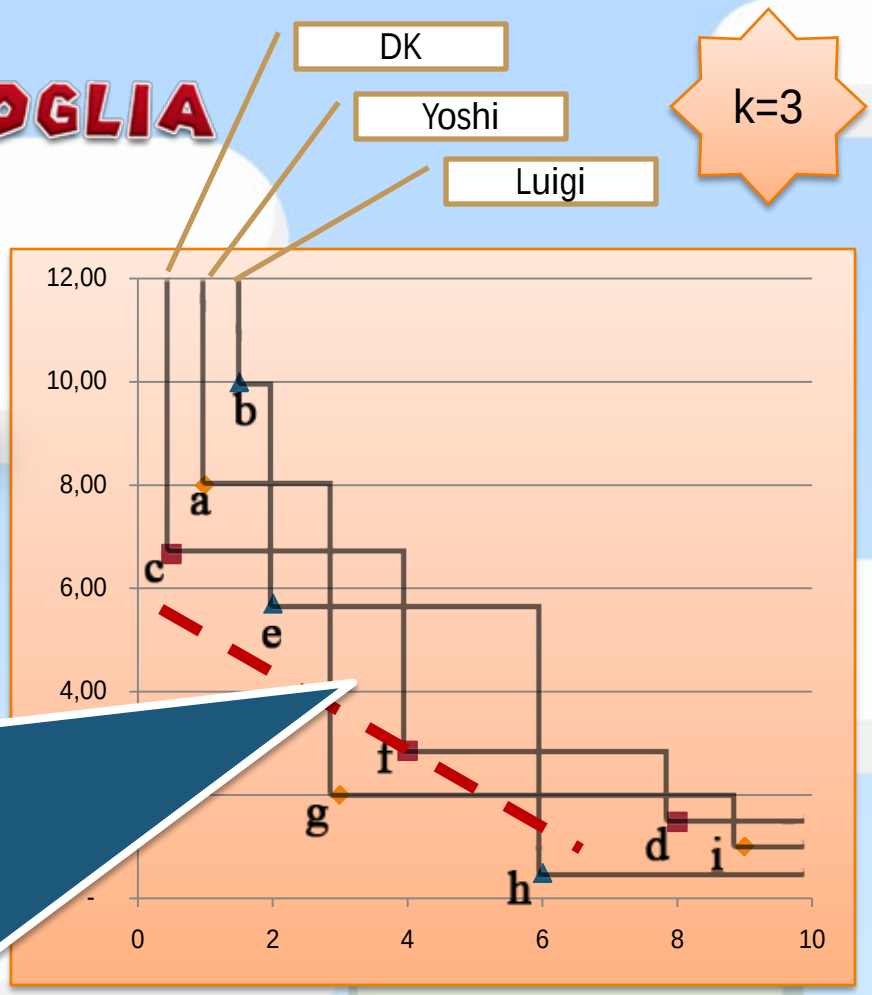
Threshold: ?

Lista risultati: { *g (3; 2) h (6; 0,5)* }

PASSO 2

LISTA RISULTATI E SOGLIA

1. Arriva ad f e lo aggiunge in lista.
2. Essendo k=3 la lista temporanea di risultati è completa e possiamo pertanto passare a mandare le query ai singoli SP che li contengono.
3. Prima di questo però settiamo (finalmente) il valore di soglia al punteggio del punto più grande (f in questo caso)



Threshold: (0,5*4+0,5*2,86) = 3,43

Lista risultati: { *g (3; 2) h (6; 0,5) f (4; 2,86)* }

PASSO 3

DONKEY KONG CHIEDE A YOSHI

k=3

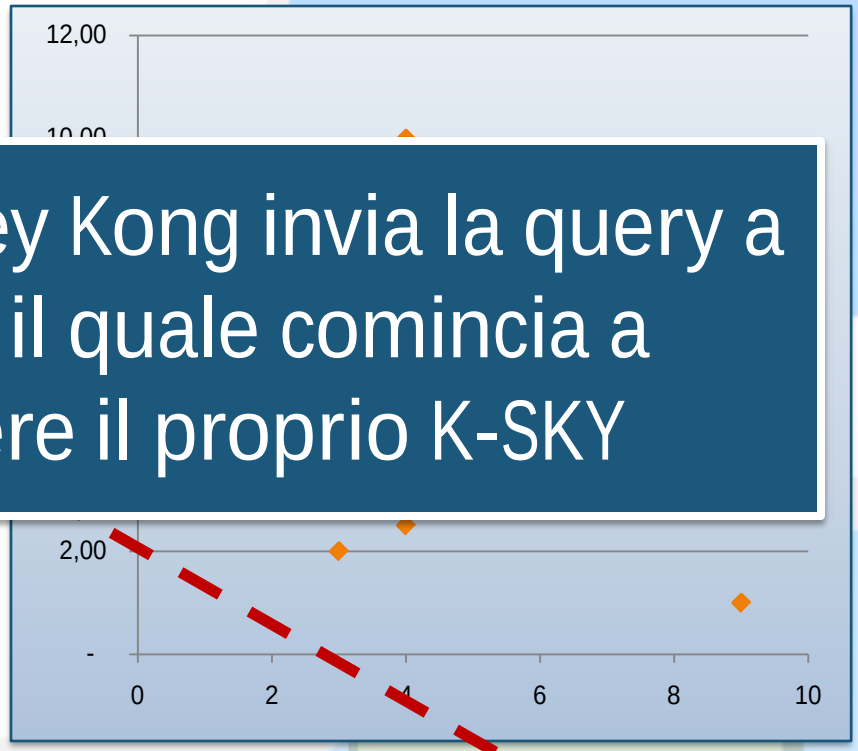
SPq

Donkey Kong invia la query a Yoshi, il quale comincia a scorrere il proprio K-SKY

	L	N
f	4	2,86
c	0,5	6,67
d	8	1,48
	5	4
	2	8
	3	8
	7	4

	L	N
g	3	2
i	9	1
a	1	8
	4	2,5
	7	3,08
	5	6,67
	4	10

e		
h		
b	1,5	10
	8	1
	5	5,71
	4	8
	3	10



Lista risultati: { *g (3; 2) h (6; 0,5) f (4; 2,86)* }

PASSO 3

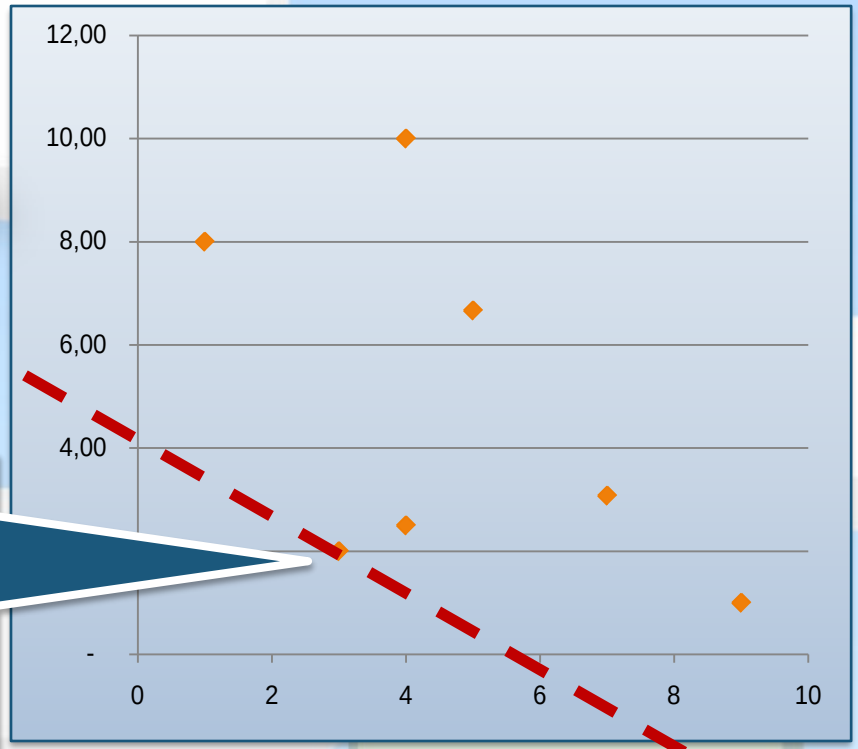
DONKEY KONG CHIEDE A YOSHI

k=3

SPq

	L	N
f	4	2,86
g	3	2
e	2	5,71

Il primo punto che Yoshi trova nel suo K-SKY è **g**, ed appartenendo a lui allora lo mette nel resultset sotto forma di data-object



Lista risultati: { ***g (3; 2)*** *h (6; 0,5)* *f (4; 2,86)* }

PASSO 3

DONKEY KONG CHIEDE A YOSHI

k=3

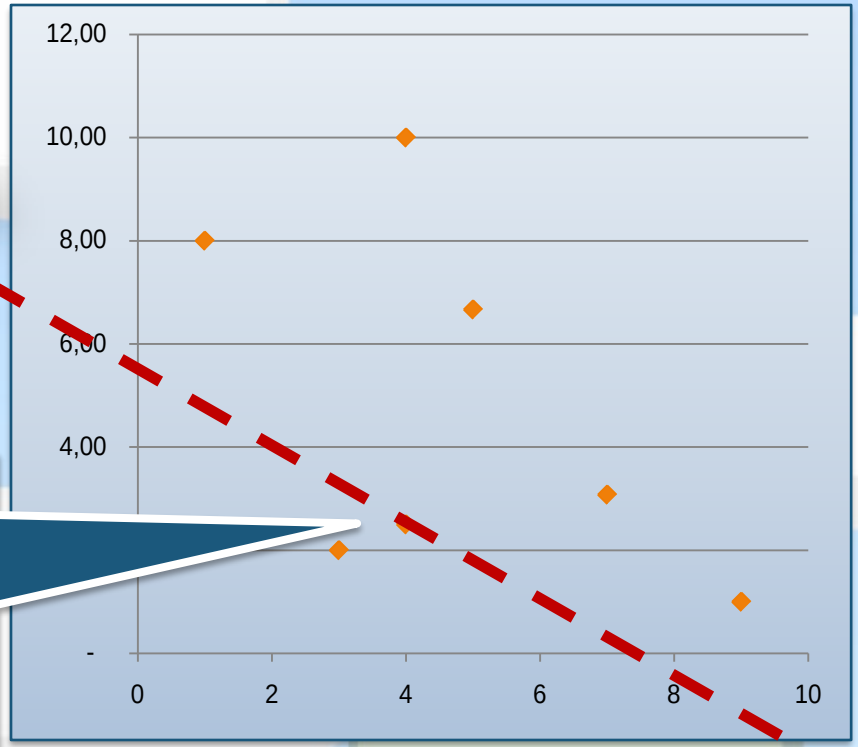
SPq

	L	N
f	4	2,86

	L	N
g	3	2

	L	N
e	2	5,71

Il secondo punto che trova non è presente nel resultset, quindi lo aggiungiamo direttamente sotto forma di data-object

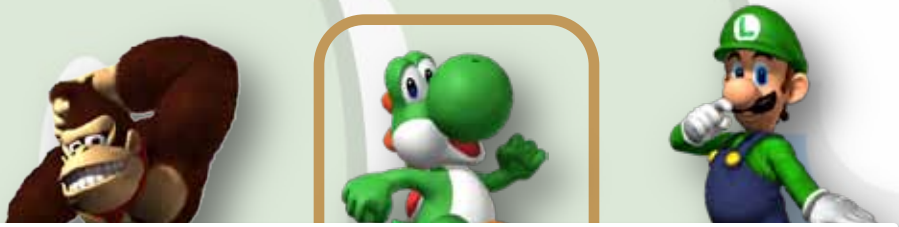


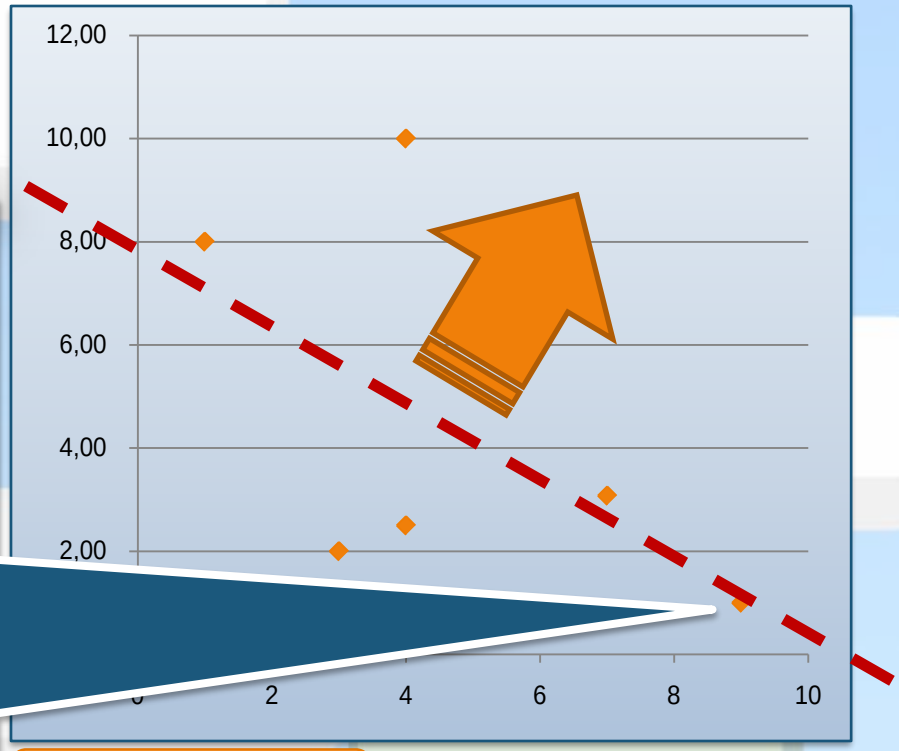
Lista risultati: { ***g (3; 2)*** *h (6; 0,5)* *f (4; 2,86)* ***(4; 2,5)*** }

PASSO 3

DONKEY KONG CHIEDE A YOSHI

k=3





Il terzo punto che trova ha punteggio maggiore della soglia che avevamo calcolato in precedenza e decide perciò di non processarlo esattamente come per i successivi, che non computiamo

Threshold: 3,43

Lista risultati: { ***g (3; 2)*** *h (6; 0,5)* *f (4; 2,86)* ***(4; 2,5)*** }

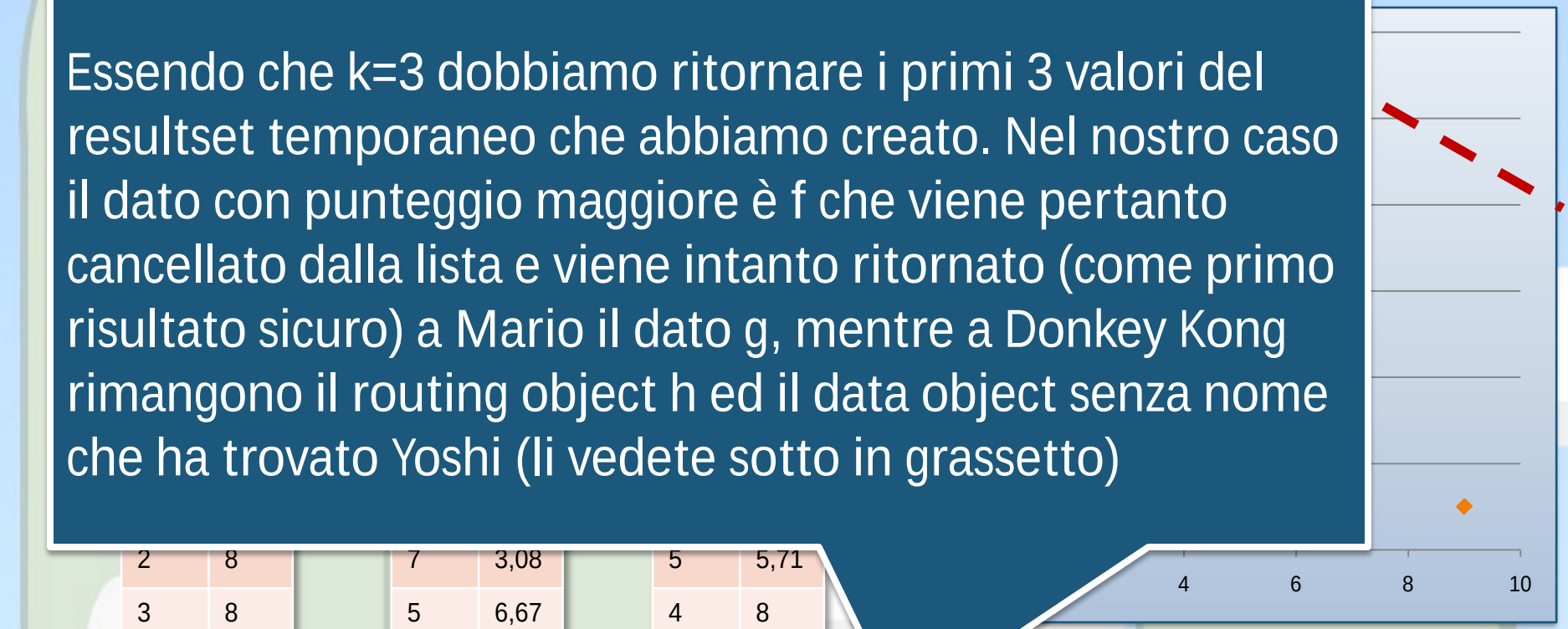
PASSO 3

DONKEY KONG CHIEDE A YOSHI

k=3

Yoshi

Essendo che k=3 dobbiamo ritornare i primi 3 valori del resultset temporaneo che abbiamo creato. Nel nostro caso il dato con punteggio maggiore è f che viene pertanto cancellato dalla lista e viene intanto ritornato (come primo risultato sicuro) a Mario il dato g, mentre a Donkey Kong rimangono il routing object h ed il data object senza nome che ha trovato Yoshi (li vedete sotto in grassetto)



2	8
3	8
7	4

7	3,08
5	6,67
4	10

5	5,71
4	8
3	10

…shold: (0,5*4+0,5*2,5) = 3,25

Lista risultati: { g (3) h (6; 0,5) f (4 86) **(4; 2,5)** }

PASSO 4

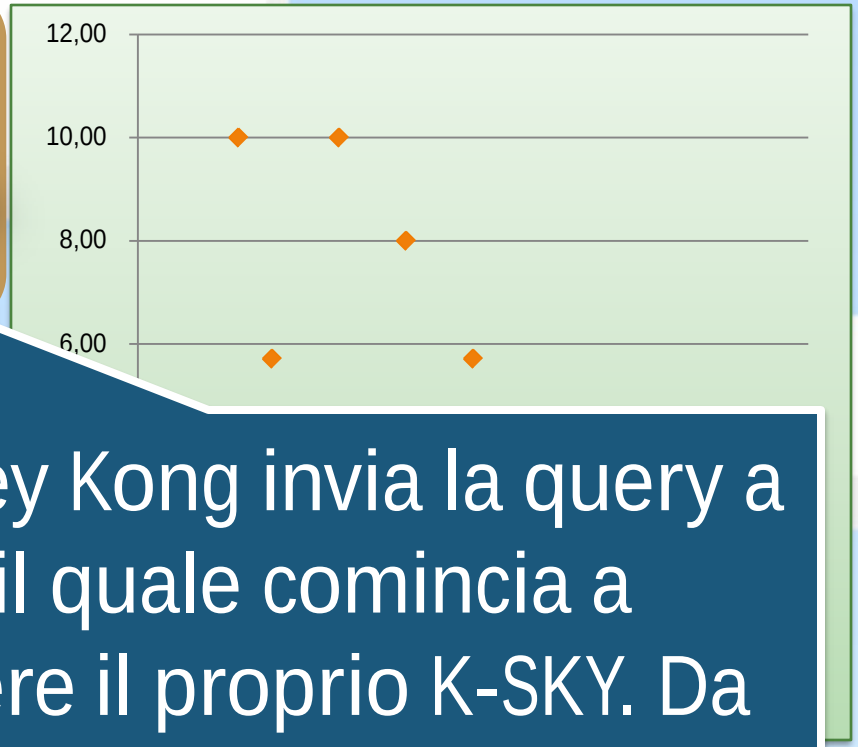
DONKEY KONG CHIEDE A LUIGI

k=2

	L	N
f	4	2,86
c	0,5	6,67
d	8	1,48
	5	4
	2	8
	3	8
	7	4

	L	N
g	3	2
i	9	1
a	1	8
	4	2,5
	7	3,08
	5	6,67
	4	10

	L	N
e		
h		
b		



Donkey Kong invia la query a Luigi, il quale comincia a scorrere il proprio K-SKY. Da notare che ora k=2

Lista risultati: { g (3; 2) h (6; 0,5) **(4; 2,5)** }

PASSO 4

DONKEY KONG CHIEDE A LUIGI

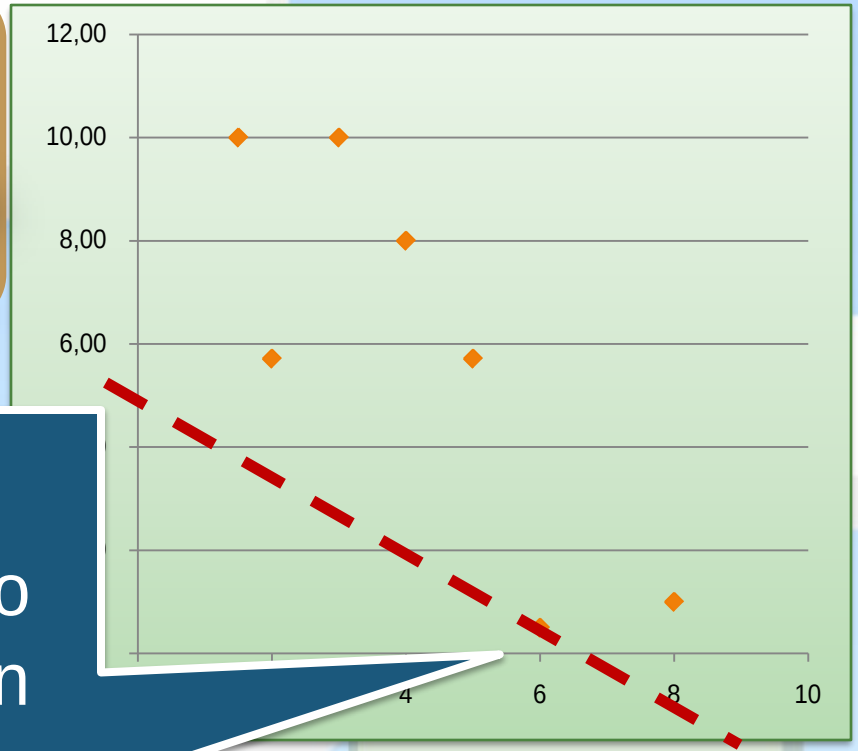
k=2

	L	N
f	4	2,86

	L	N
g	2	2

	L	N
o	2	5,71

Luigi



Trova h, che pertanto viene tramutato in data object pronto per essere ritornato (nel caso in cui risulti il migliore)

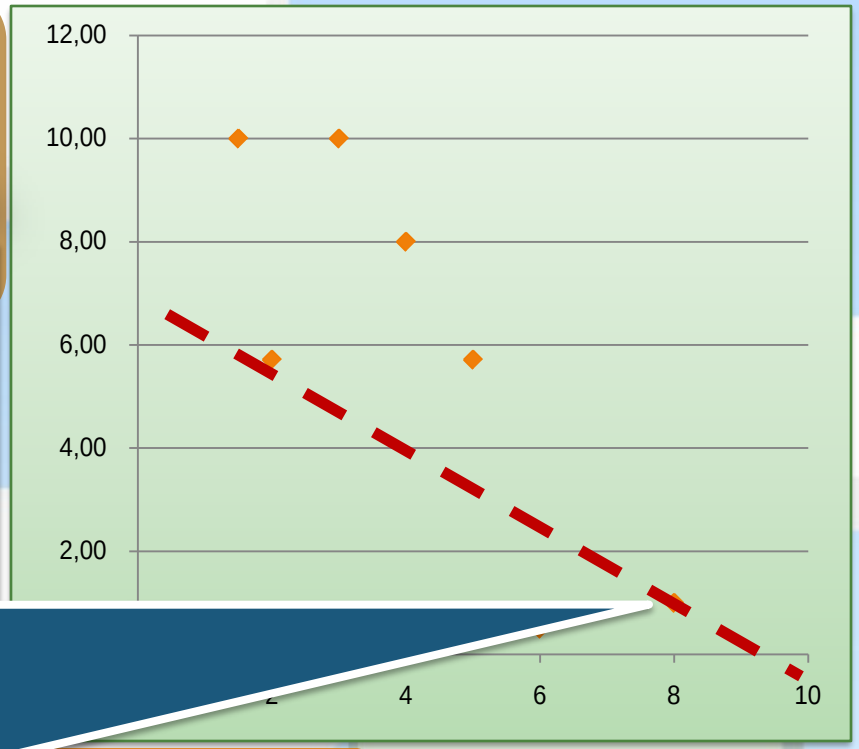
eshold: 3,25

Lista risultati: { g (2) h (6; 0,5) (4; 2,5) }

PASSO 4

DONKEY KONG CHIEDE A LUIGI

k=2



Poi trova un secondo punto che tuttavia, esattamente come prima, ha punteggio maggiore della soglia. Come è successo prima il dato viene scartato come i successivi

Threshold: 3,25

Lista risultati: { *g (2)* *h (6; 0,5)* *(4; 2,5)* }

PASSO 4

DONKEY KONG CHIEDE A LUIGI

k=2

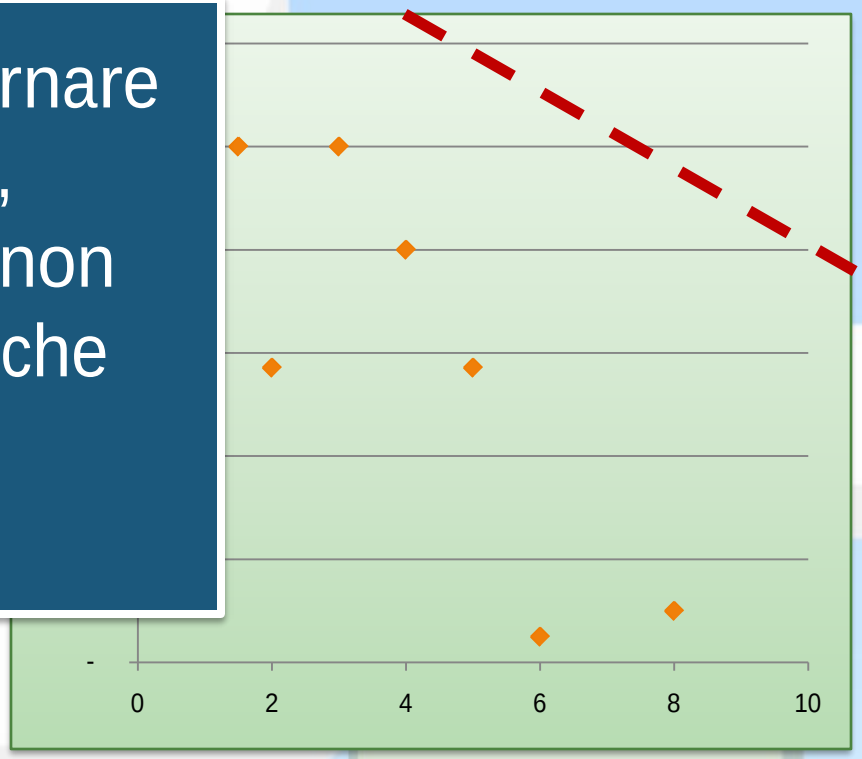
Luigi

Non resta altro che ritornare a Mario il data object h, mentre a Donkey Kong non resta altro che il valore che avevamo trovato prima ancora in lista

	L	
f	4	2
c	0,5	6
d	8	1
	5	4
	2	8
	3	8
	7	4

5	
4	10

5	5,71
4	8
3	10



Threshold: 3,25

Lista risultati: { g (4; 2) h (4; 2,5) **(4; 2,5)** }

PASSO 5
ULTIMO VALORE!!!

k=1

Non essendoci più nessun SP a cui chiedere, non ci sono più dati migliori, Il controllo ritorna a Donkey Kong.
Da notare che ora k=1

	L	N
f	4	2,86
c	0,5	6,67
d	8	1,48
	5	4
	2	8
	3	8
	7	4

	L	N
g	3	2
i	9	1
a	1	8
	4	2,5
	7	3,08
	5	6,67
	4	10

	L	
e	2	5
h	6	0
b	1,5	1
	8	1
	5	5
	4	8
	3	1

Lista risultati: { *g (3; 2)* *h (5; 2,5)* *(4; 2,5)* }

PASSO 5
ULTIMO VALORE!!!

k=1

A Mario viene finalmente ritornato anche l'ultimo data object.

Ora Mario conosce esattamente ben 3 percorsi che possono permettergli di arrivare da Peach con la certezza che potrà far metano ogni volta che ne avrà bisogno.

7	4

4	10

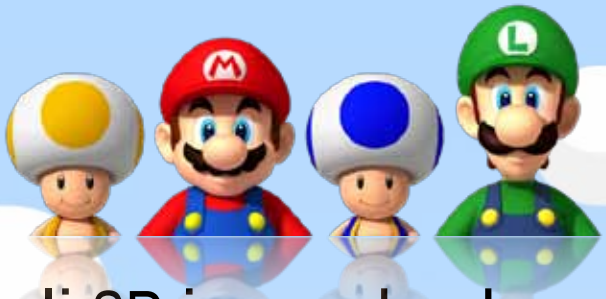
3	10

Lista risultati: { *g* (2) *h* (5) (5) }

POSSIAMO OTTIMIZZARE SPEERTO?

- E’ possibile
  - **Ridurre i tempi di risposta**
    Parallel Query Processing
  - **Ridurre i costi** dovuti alla quantità di dati memorizzati
    Abstract Skyline

COMPUTAZIONE PARALLELA



- Vogliamo poter inviare le query agli SP in modo che possano **computarle in parallelo**

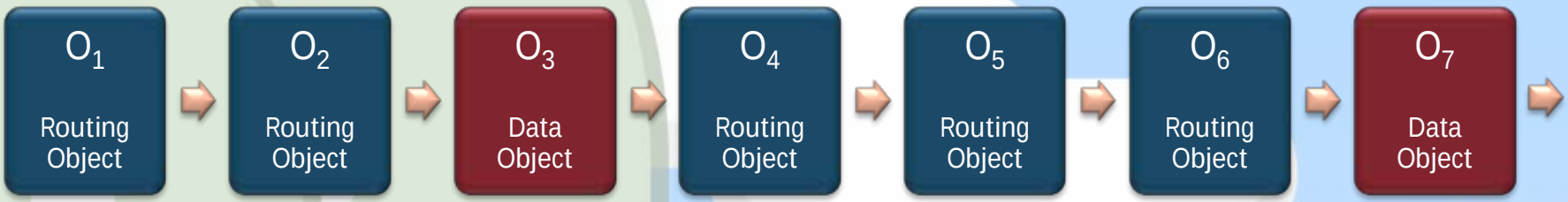
Problemi

- Non possiamo aggiornare la soglia in maniera continuativa; questo può portare a dei **falsi positivi**

Soluzione

- Sfruttiamo un sistema di parallelizzazione parziale: le *query in sospeso* vengono inviate tutte insieme ai rispettivi SP **soltanto quando si verificano determinati eventi.** Per mezzo di un sistema a soglie parziali ed a stime di conteggio dati, riusciamo ad ottenere quanto richiesto.

A GRANDI LINEE...



Il fungo non è altro che un indicatore di posizionamento della query.

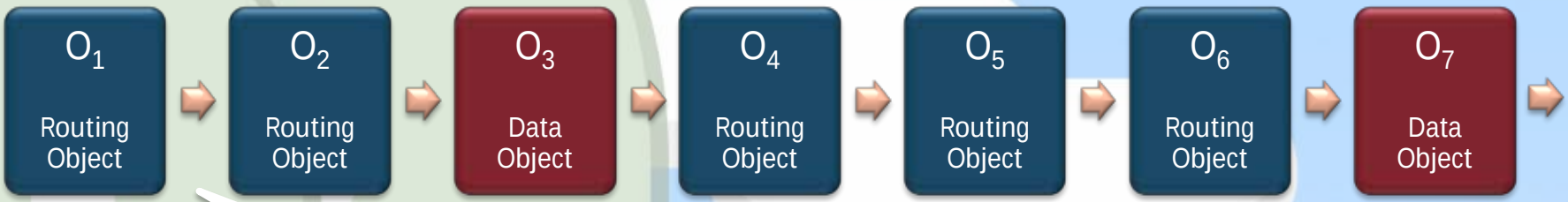
N_r: [numero di oggetti]

t: [valore del massimo ro]

m: 0

Resultset: { }

A GRANDI LINEE...



N_r: [numero di oggetti]

t: [valore del massimo ro]

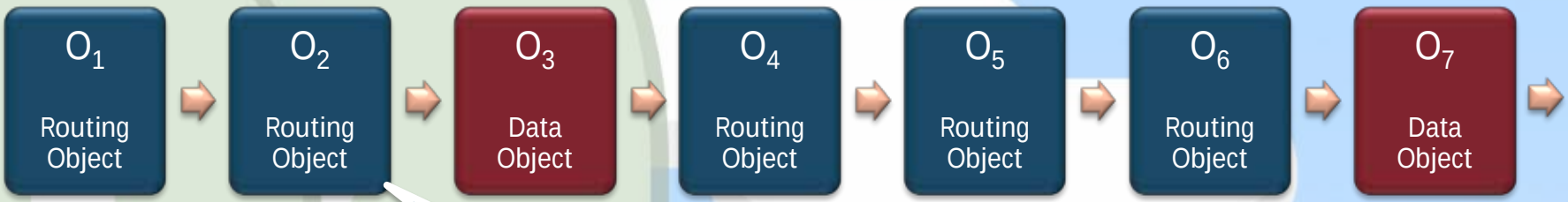
m: $(t - score(o_1))/N_r$

Resultset: { }

Il primo oggetto che incontriamo è un routing object. Cosa fa il sistema:

1. stima un punteggio medio di tutti gli oggetti che sono dominati da quel punto
2. setta il valore di soglia al valore massimo dei routing object che conosce

A GRANDI LINEE...



N_r: [numero di oggetti]

t: [valore del massimo ro]

m: $(t - score(o_1))/N_r$

N_{r2}: $(score(o_2)-score(o_1))/m$

Resultset: { }

Il secondo oggetto è ancora un routing object.

1. Basandoci sul calcolo del punteggio medio m di prima, conoscendo il suo punteggio possiamo stimare il numero di oggetti che quell'SP può ritornarci
2. Ci salviamo questo valore in Nr2

A GRANDI LINEE...

N_r: [numero di oggetti]

t: [valore del massimo ro]

m: $(t - score(o_1))/N_r$

N_{r2}: $(score(o_2)-score(o_1))/m$

m_2: $(score(o_3)-score(o_2))/N_{r2}$

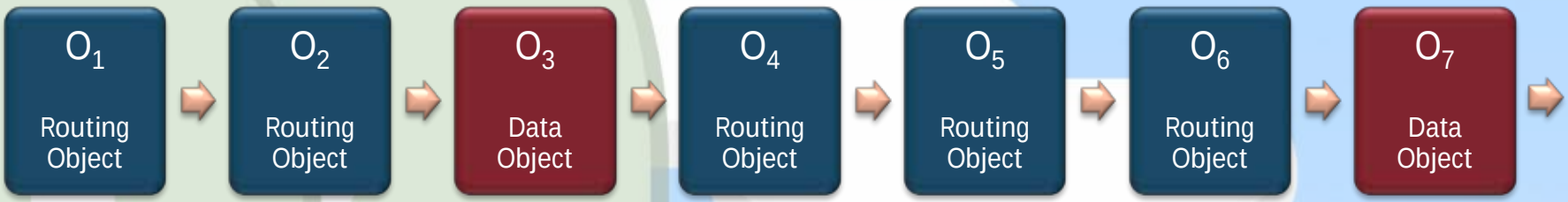
m': $(m+m_2)/2$

Resultset: { RSO_1 RSO_2 O3 }

Il terzo oggetto è un data object.

1. Partono le due query direttamente ad SP1 ed SP2 (l'animazione era un po' troppo veloce spero l'abbiate vista)
2. Qui calcoliamo m2 che altro non è (come per m) che una stima del punteggio medio degli oggetti che sono dominati da O2
3. Aggiorniamo il punteggio medio del sistema a m' = m+m2/2 (dove il diviso 2 è perché abbiamo passato 2 routing object)
4. Aggiorniamo la soglia ad un valore t' (calcolato come prima sul punteggio massimo nuovo spazio trovato)
5. Il resultset infine viene aggiornato contenendo i risultati della query a SP1, a SP2 ed al data object O3

A GRANDI LINEE...



Si continua fin quando $|RS| \geq k$

A questo punto il ciclo ricomincia. t' diventa t, m' diventa m e si continua così fintanto che la cardinalità del resultset non è maggiore di k, ovvero finchè non abbiamo trovato almeno k risultati.

... IL PARALLELO

- Se nel percorso non troviamo nessun data object, la query non prosegue nello SKY all'infinito.
- Le stime parziali di risultato Nr2, Nr3 ecc. ci servono proprio per tenere traccia di quando possiamo essere quasi certi di poter ottenere dagli SP k risultati.
- Pertanto nel caso in cui non esista nessun data object, ma:

$$\sum_{1<r<n} N_r \geq (k-c)$$

(dove c è il numero di oggetti già ritornati all'utente ed n il numero di routing object passati), allora viene scatenato comunque l'evento.

ABSTRACT SKYLINE

- L'abstract skyline è un'approssimazione lower bound dello skyline

- Dobbiamo limitare ad **U** il numero massimo di punti che devono essere contenuti in una SKY_i

Come funziona?

Due punti vicini dello skyline vengono sostituiti con un nuovo punto, finchè il numero totale degli oggetti è minore di **U**

Algoritmo Abstract Skyline

Skyline abstraction on super-peer SPi

```
01: Input: SKY_i
02: p = argmax_{∀t∈SKYi}(∑_{1≤i≤d} ln(t[i] + 1))
03: min_dist = ∞
04: for ((∀t ∈ SKY_i) and (p ≠ t)) do
05:     dist = min_{1≤i≤d}(|p[i] − t[i]|)
06:     if (dist ≤ min_dist) then
07:             min_dist = dist
08:             q = t
09:     end if
10: end for
11: for (1 ≤ i ≤ d) do
12:     r[i] = min(p[i], q[i])
13: end for
```

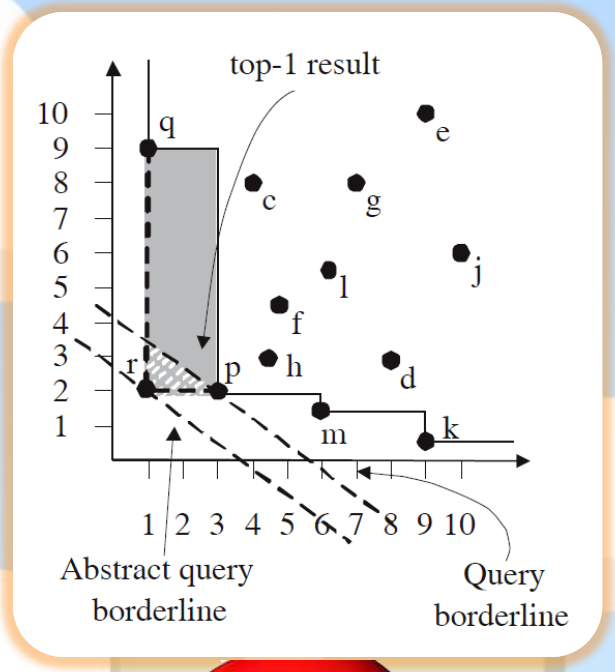
ANDIAMO CON CALMA...

COSA SUCCEDE AD OGNI CICLO?

- Scegliamo un punto **p** che ha la minore probabilità di essere dominato da tutti gli altri (quindi sostituibile più facilmente)
- Questa probabilità è decisa dal maggior valore di entropia del punto:

$$E(p) = \sum_{1 \le i \le d} \ln (p[i] + 1)$$

- Scegliamo un punto **q** che abbia distanza minima, in tutte le dimensioni, rispetto a **p**
- Sostituiamo **p** e **q** con un nuovo punto **r** che ha come coordinate i valori minimi di **p** e **q** in tutte le dimensioni
- I valori dominati **r** ma non dominati da **p** e **q** vengono cancellati



ABSTRACT SKYLINE

VANTAGGI E SVANTAGGI

- Riduce la memoria utilizzata nelle operazioni

- Rende la SKY più robusta agli aggiornamenti

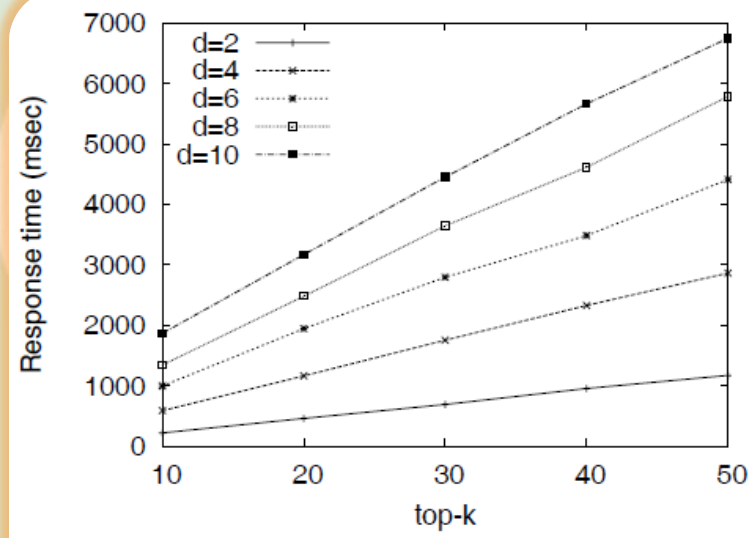
- Non è più possibile sfruttare il concetto di soglia
  (aumento di query inutili tra i SP)

SPEERTO IN AZIONE

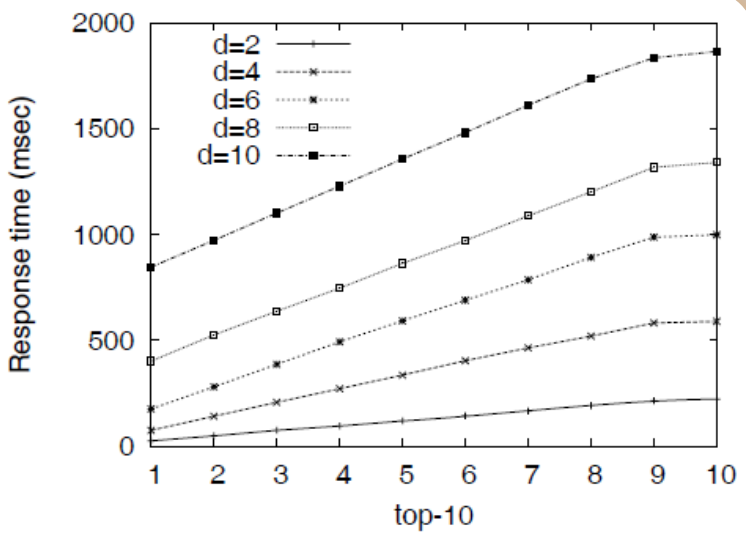
- I test effettuati sono stati di due tipi:
  - Con dati distribuiti uniformemente tra loro in tutti i peer (uniform)
  - Con dati distribuiti secondo una funzione gaussiana gaussiana con baricentro nei SP e poi in maniera casuale tra i restanti peer (clustered)
- Sono stati utilizzati come valori di default:
  - $d = 4$;
  - $K = 50$;
  - $10 \leq k \leq 50$;
  - $n = 10^6$;
  - $n_p = 2000$;
  - $N_{sp} = 0.1N_p$

SPEERTO IN AZIONE

TEMPI DI RISPOSTA



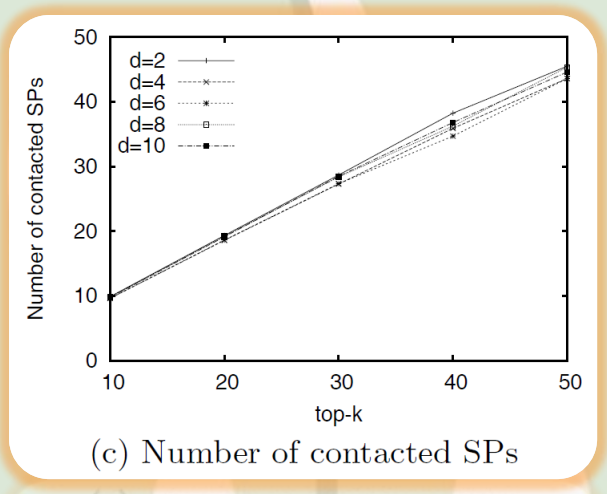
(a) Response time with d



(b) Response time for first 10 objects

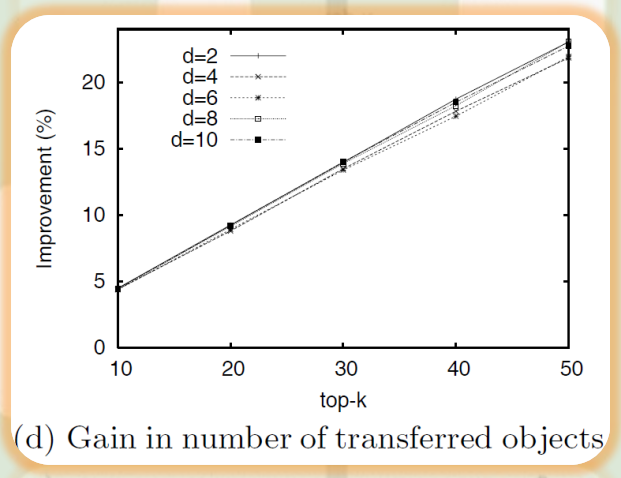
- I tempi di risposta incrementano con il numero delle dimensioni
- I risultati vengono restituiti in tempo reale

SPEERTO IN AZIONE
CORRETTEZZA DELL'ALGORITMO



(c) Number of contacted SPs

Il numero di SP contattati aumenta con k ma rimane quasi invariato con l'aumentare del numero di dimensioni

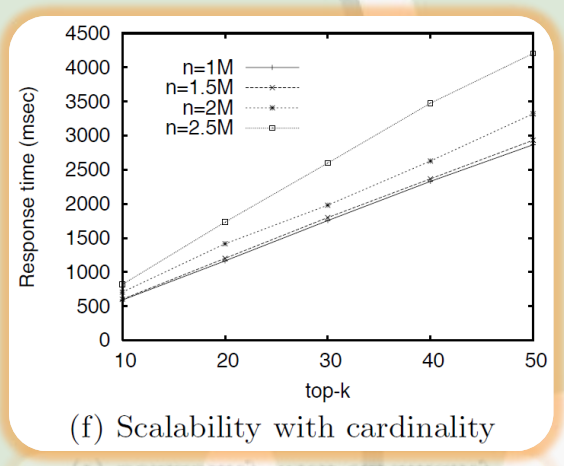


(d) Gain in number of transferred objects

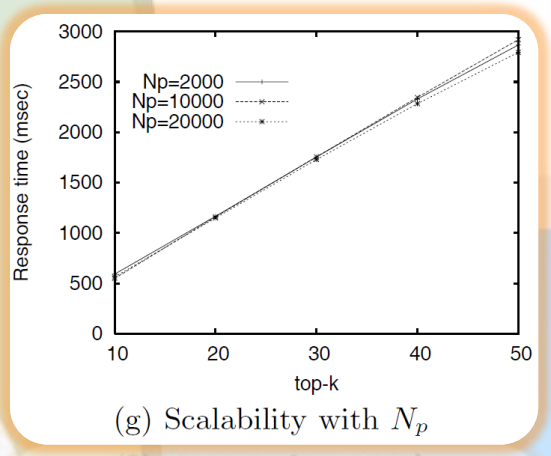
Il numero di oggetti trasferiti è circa 23 volte in meno rispetto a quelli che sarebbero effettivamente trasferiti senza l'utilizzo della soglia

SPEERTO IN AZIONE

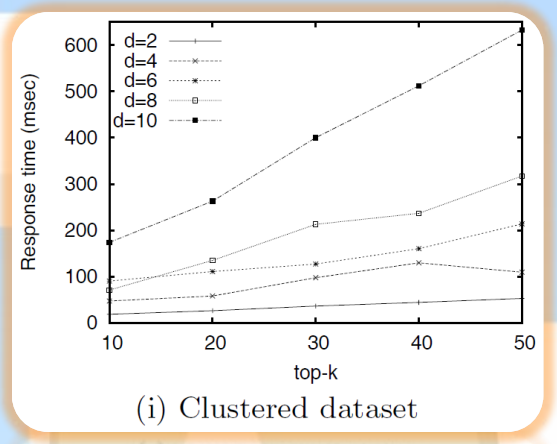
SCALABILITA'



(f) Scalability with cardinality



(g) Scalability with N_p

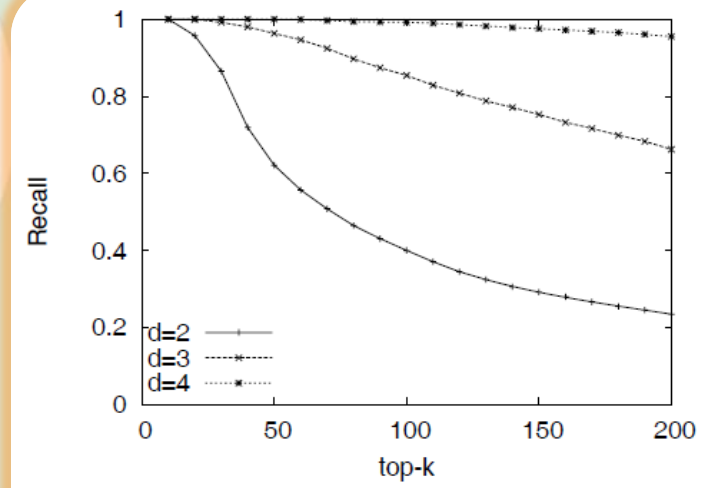


(i) Clustered dataset

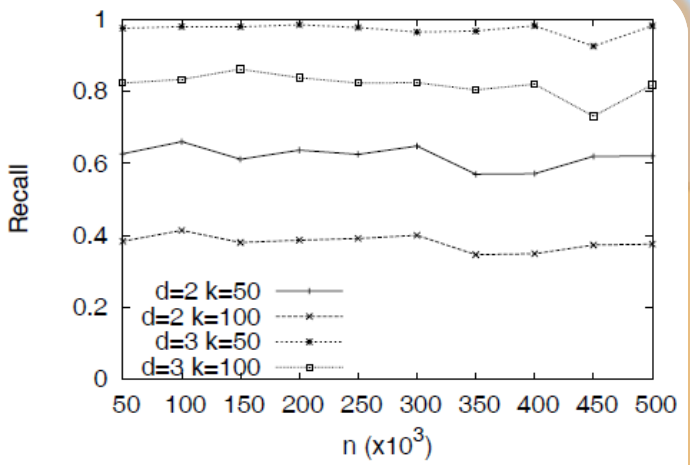
- Il tempo di risposta cresce velocemente all'aumentare di k
- Con dati clustered il numero di query che l'algoritmo deve fare agli altri SP per trovare altri valori plausibili è tendenzialmente molto più basso perchè ad ogni query vengono restituiti più data object

SPEERTO IN AZIONE

TOP-K CON k>K



(b) Recall for clustered data

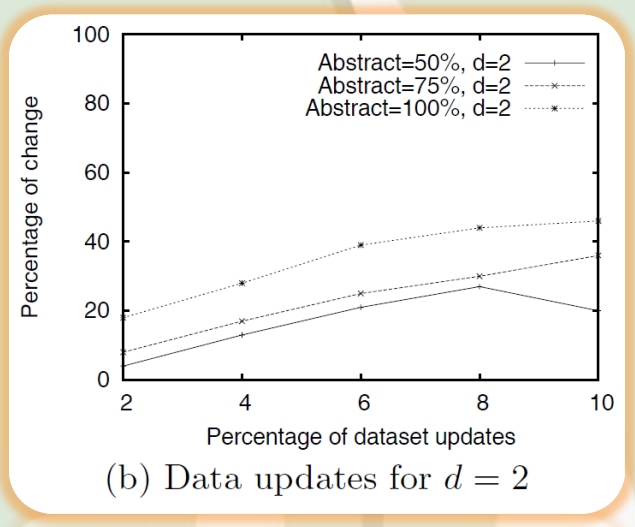


(c) Recall for varying n

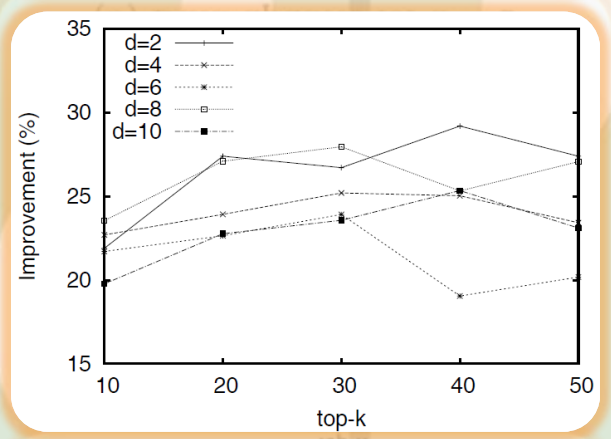
- Sia con **k»K** che con **d crescente**, diminuisce la quantità di dati corretti per la query rispetto alla quantità di dati trovati
- All'aumentare della **cardinalità del dataset**, la probabilità di ottenere dati sbagliati non cambia significativamente

SPEERTO IN AZIONE

PARALLELIZZAZIONE E ABSTRACT SKYLINE



(b) Data updates for $d = 2$



- Con d=2, 2% di dati modificati e un'astrazione del 50%, solo il 4% degli SP devono aggiornare la KSKY
- Il tempo di risposta diminuisce del 20% ma il numero di oggetti trasferiti aumenta leggermente

ALTERNATIVE...

Non esistono alternative, l'unica via è Chuck Norris!

GRAZIE!