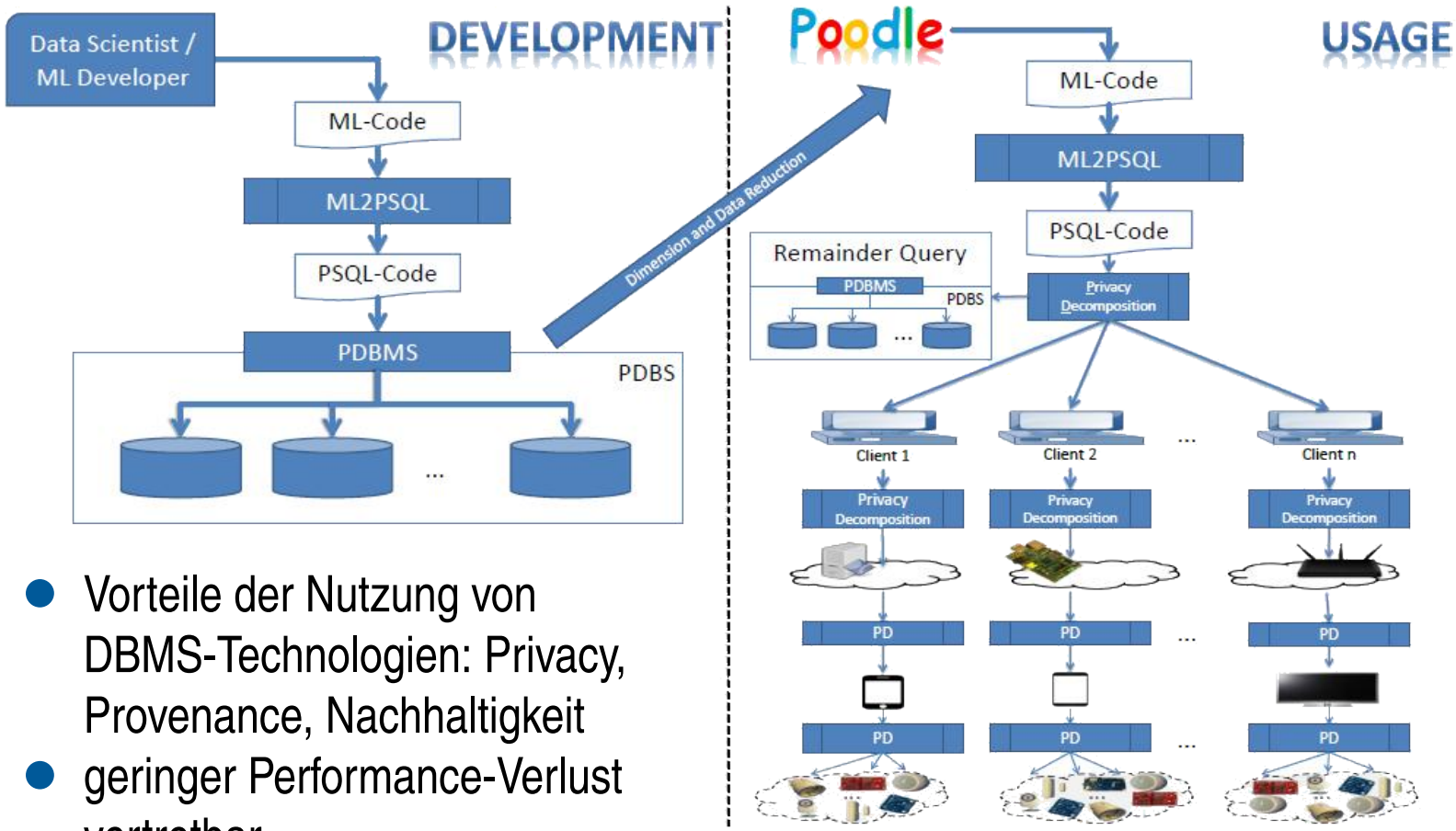
Ein Vergleich von Big-Data-Analytics-Plattformen

Stonebraker gegen Google:
Das 2:0 fällt in Rostock

Daniel Dietrich & Ole Fenske

Inhaltsverzeichnis

Motivation: PArADISE



- Vorteile der Nutzung von DBMS-Technologien: Privacy, Provenance, Nachhaltigkeit
- geringer Performance-Verlust vertretbar

Stonebraker gegen Google[1] – Das 1:0

- Cluster mit 100 Knoten: 2.4 GHz Intel Core 2 Duo Prozessor, 4 GB RAM
- Grep-Task: 10 Milliarden Datensätze (1 TB), 10 GB pro Knoten
- WebLog-Task: Aggregation mit GROUP BY auf einem Web Log: 2 TB Log mit 155 Millionen Datensätze, 20 GB pro Knoten, kein Index
- Join Task: Verbund über WebLog und PageRank, eine Selektion und eine Aggregation, PageRank-Tabelle: 18 Millionen Datensätze, 100 TB

[1] Michael Stonebraker u. a. „MapReduce and parallel DBMSs: friends or foes?“ In: Communications of the ACM 53.1 (2010), S. 64–71.

Stonebraker gegen Google – Das 1:0

- Hadoop 0.19.0
- DBMS-X: nicht benanntes, kommerzielles, zeilenorientiertes und paralleles DBMS
- Vertica: spaltenorientierte und parallele Architektur
- Beachtet wurden nur Laufzeiten
- Kriterien wie Administrationsaufwand, Skalierbarkeit, Easy-of-Use und Cost-of-Ownership wurden vernachlässigt
- keine genauen Details bekannt

	Hadoop	DBMS-X	Vertica
Grep	284 s	194 s	108 s
Web Log	1146 s	740 s	268 s
Join	1158 s	32 s	55 s

Projekt

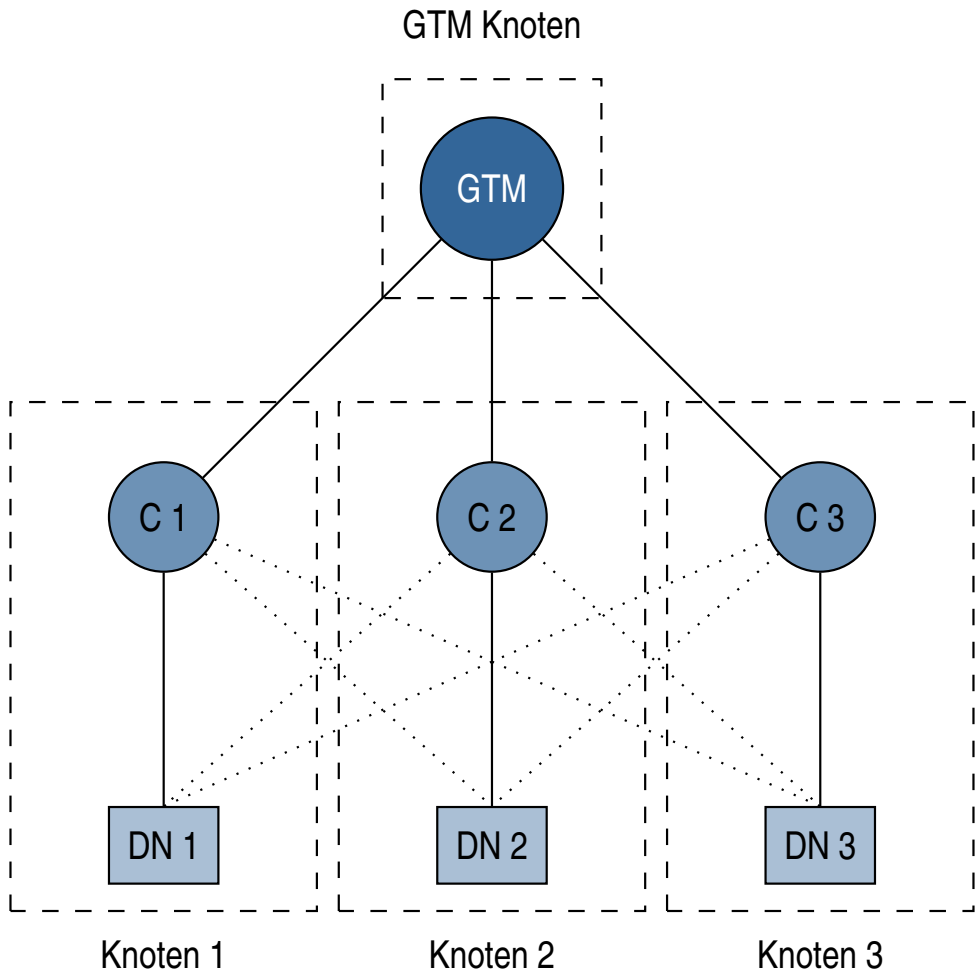
- Wirtschaftsinformatik-Gruppe (WIN)
  - 5 Personen
  - 160 Stunden Datenaufbereitung
  - 170 Stunden Implementierung der Tasks
- Informatik-Gruppe (INF)
  - 3 Personen Installation & Administration von Postgres-XL
  - 1 Person Implementierung der Tasks
  - Nutzung von WIN-Ergebnissen
  - 40 Stunden Datenaufbereitung
  - 60 Stunden Implementierung

Plattformen

Plattform	Jahr	Bibliotheken	Sprachen
Hadoop	2008		
Flink	2014	Table API & SQL, FlinkML, Flink CEP, Gelly	Python, Java
Tensorflow	2015		C++,Python
Spark	2013	Spark SQL, Spark Streaming, MLlib, GraphX	Python, Java
Naiad	2012		C#
Postgres-XL	2014	MadLib	SQL

- Entscheidung für Flink, Spark und Postgres-XL

Cluster



Cluster – Konfiguration

- CentOS 7
- Intel Haswell mit 4 Kernen
- 64 GB RAM
- 350 MB/s I/O-Geschwindigkeit

Parameter	Wert
Speicherblockgröße	256 MB
Heapsize Task Executor	1024 MB
Heapsize History Server	1024 MB
Heapsize DataNode	1024 MB
Rackawareness	deaktiviert
Replikate	3/Block
Kompression	deaktiviert

Tabelle: HDFS-Konfiguration

Parameter	Wert
Effective Cache Size	4 GB
Worker Memory	512 MB
Maintenance Memory	1 GB
Temporary Buffer	64 MB
Shared Buffer	1 GB
Segment Size	1 GB

Tabelle: Postgres-XL-Konfiguration

Datensätze

- Ausschnitt des Twitter-Follower-Graphen mit 26 GB
- errechneter PageRank-Datensatz: 1,15 GB
- Web-Log-Einträge: 4,29 GB

Aufgabenstellung

- Vergleich der Ausführungszeiten folgender Tasks auf dem Twitter-Follower-Graphen zwischen den beiden Projektgruppen
  - Grep-Task
  - Join-Task
  - *k*-Means

Grep-Task

- Suche nach einem bestimmten Substring in einem großen Datensatz
  - Standardtest von Google zur Messung der Ausführungsgeschwindigkeit von Parallelisierungsplattformen
- Bedingungen:
  - Keine Sortierung
  - Keine Indexierung

Import	
id1	id2
34097876	49356737
18943022	41501483
14681605	38541699
39504860	46790556
18725026	22932435
20472444	23795701
23988595	26022493
11597492	20859805
33469544	18009931
42753364	27327684
16738390	16446999
19404587	24854437

Anzahl Zeilen enthalten „1“

Anzahl
8

Grep-Task

Twitter-Follow-Graph

ID1	ID2
12343454	86968792
29656457	94665834
37695979	81632765

Anzahl Zeilen

die „4“ enthalten

Anzahl
2

Flink	Spark	Postgres-XL
100 s	53 s	121 s

Join-Task

1. IP-Adressen ermitteln, die die meisten Twitter-Accounts in einem bestimmten Zeitraum besucht haben
2. Join (PageRank $\bowtie$ Weblog): Die Summe über die PageRanks der von der IP-Adresse besuchten Twitter-Accounts berechnen
3. Aggregation: Den Durchschnitt über diese Summe bilden und ausgeben

- Bedingungen:
  - Keine Sortierung

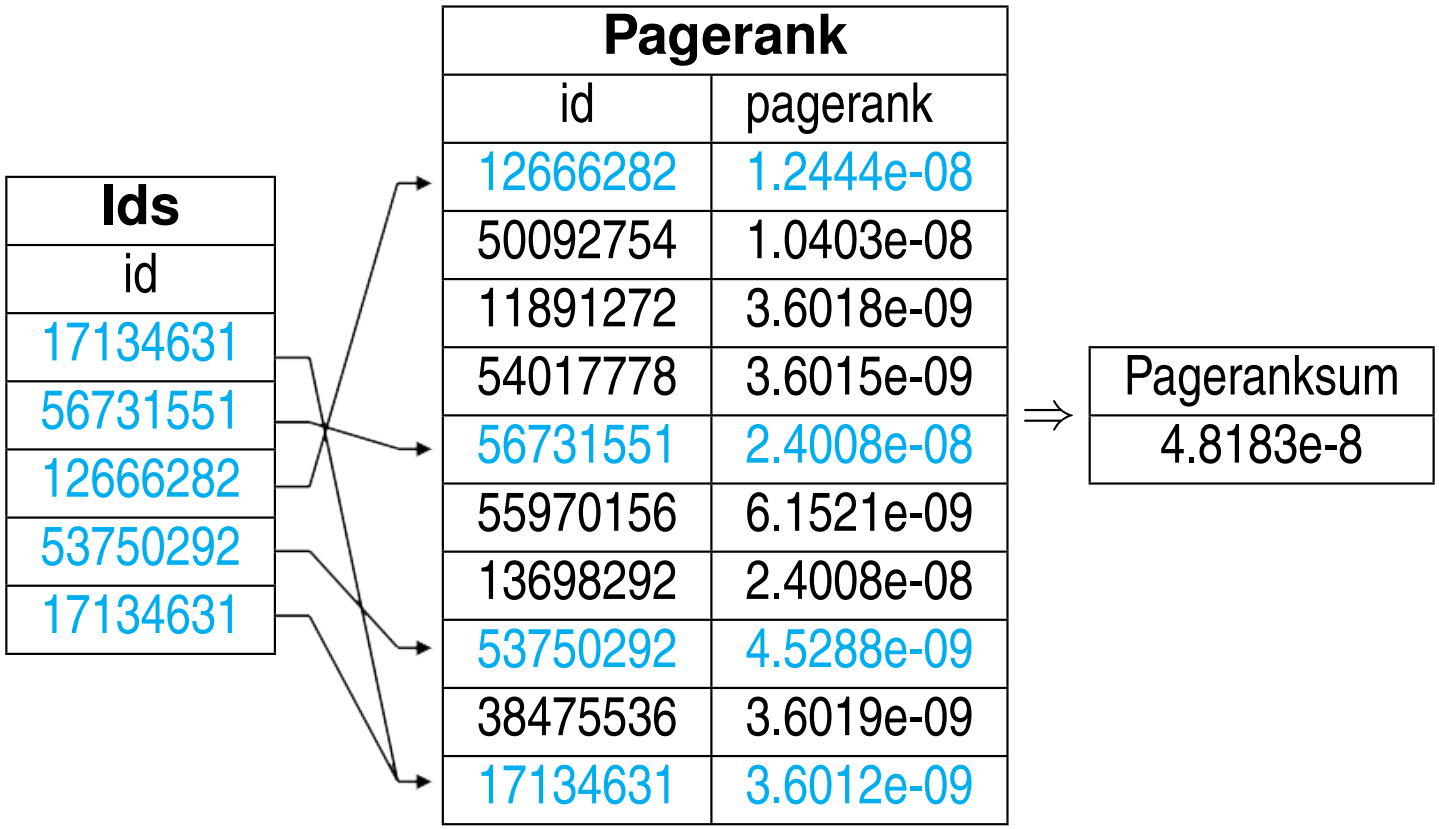
Join-Task

Weblog		
id	ip	timestamp
50092754	22	1466070299
17134631	33	1456360210
11891272	15	1466582325
56731551	33	1455847649
12666282	33	1479022032
54017778	80	1452428833
55970156	19	1480965586
53750292	33	1472903812
13698292	34	1460978591
17134631	33	1452392597

group by ip

IpCount	
ip	anzahl
22	1
33	5
15	1
80	1
19	1
34	1

Join-Task



Ids
id
17134631
56731551
12666282
53750292
17134631

Pagerank	
id	pagerank
12666282	1.2444e-08
50092754	1.0403e-08
11891272	3.6018e-09
54017778	3.6015e-09
56731551	2.4008e-08
55970156	6.1521e-09
13698292	2.4008e-08
53750292	4.5288e-09
38475536	3.6019e-09
17134631	3.6012e-09

⇒

Pageranksum
4.8183e-8

Flink	Spark	Postgres-XL
121 s	140 s	27 s

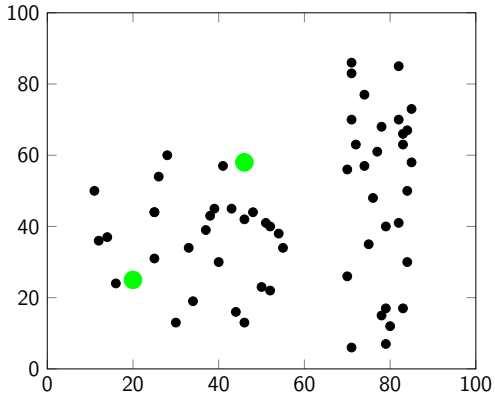
k-Means

0. Vorbereiten der Daten
1. Centroide wählen[2]
2. Distanzen berechnen
3. Daten den Centroiden zuordnen und somit Cluster bilden
4. Wiederhole ab 1., solange bis sich die Centroide nicht mehr ändern oder 10 Iterationen erreicht sind.

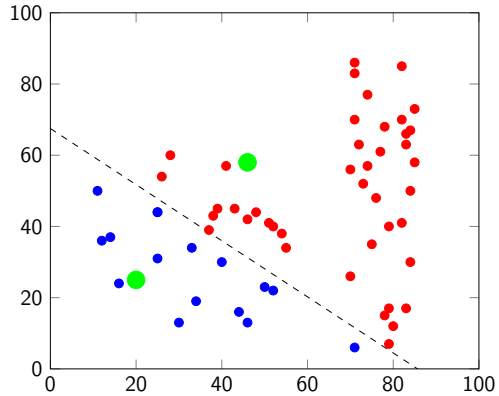
[2] kein k-**Means++** (Cluster-Schwerpunkte nicht zufällig) um Vergleichbarkeit mit WIN zu erreichen

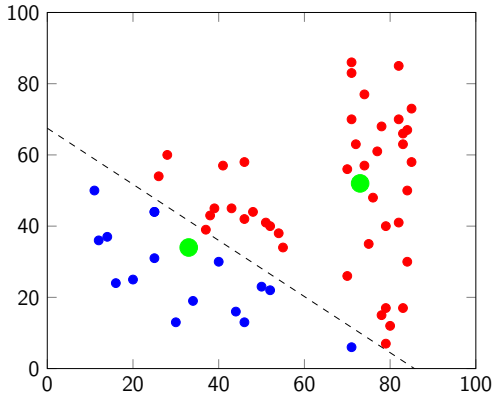
k-Means

1: Beliebige Punkte als Clusterzentren wählen

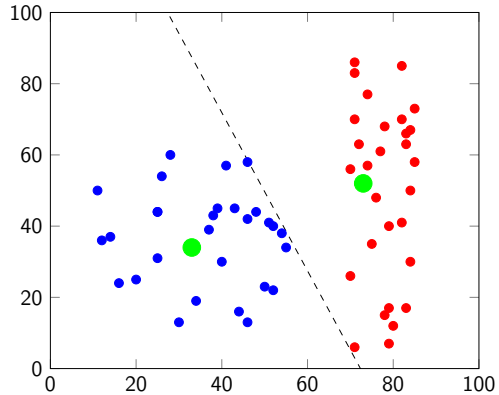


2: Distanzen ermitteln und Cluster bilden



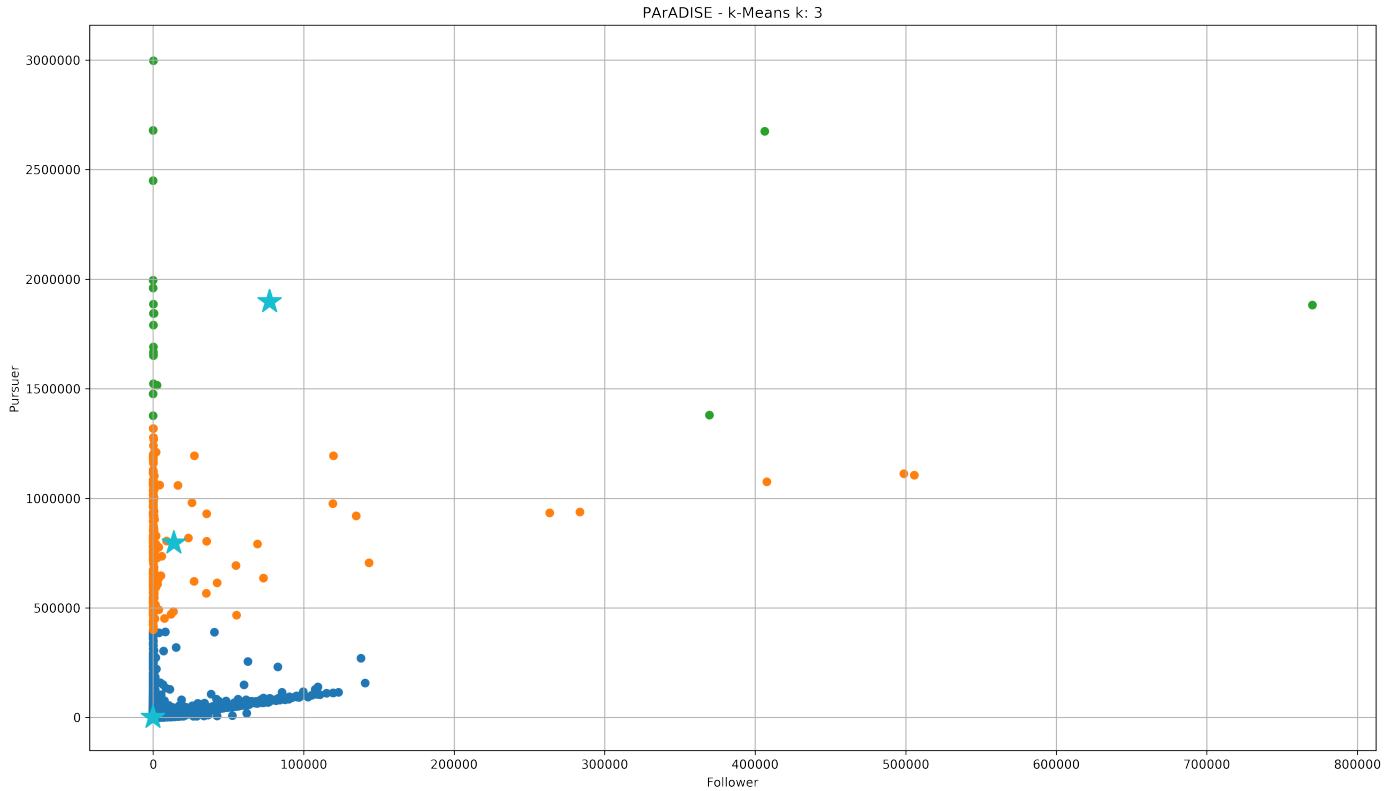


3: Neue Clusterzentren berechnen



4: Distanzen ermitteln und Cluster bilden

k-Means



Flink	Spark	Postgres-XL
705 s	917 s	335 s

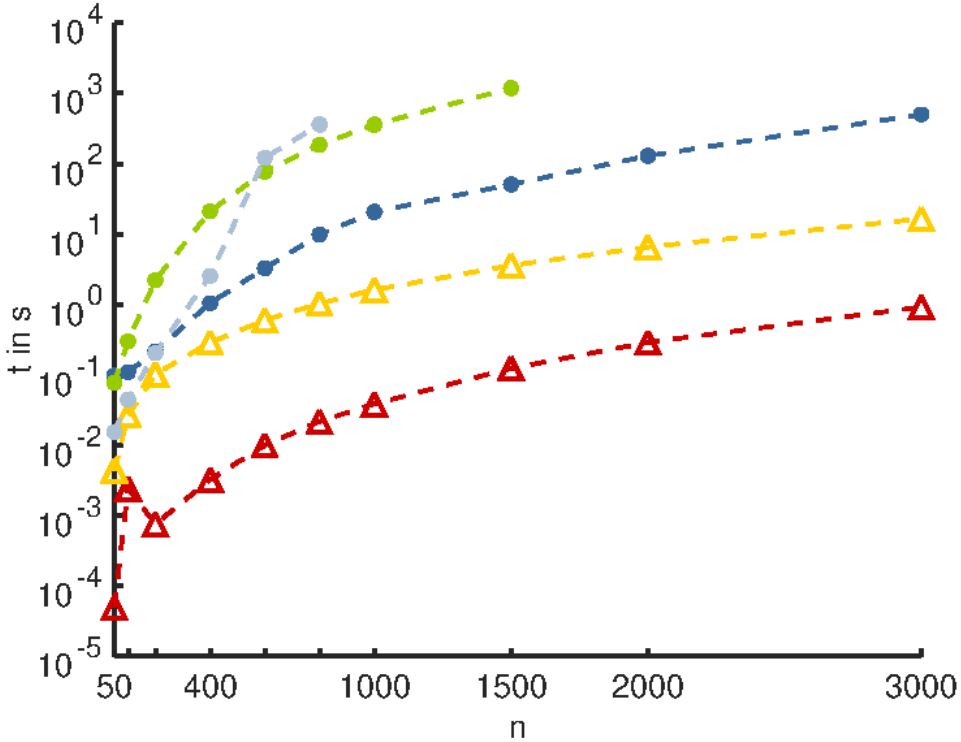
Resultate

- Postgres-XL ist nicht wesentlich langsamer als Spark und Flink
- Dafür ist der Aufwand, Postgres-XL zu skalieren bedeutend größer als bei Spark oder Flink
- In gewissen Anwendungsfällen können DBMS-Lösungen eine konkurrenzfähige Alternative zu MapReduce-Programmierparadigmen und anderen spezialisierten Big-Data-Analytics-Umgebungen sein

	Grep/s	**Join**/s	k-**Means**/s
Spark	53	140	917
Flink	100	121	705
Postgres-XL	121	27	335

Weiterführende Arbeit

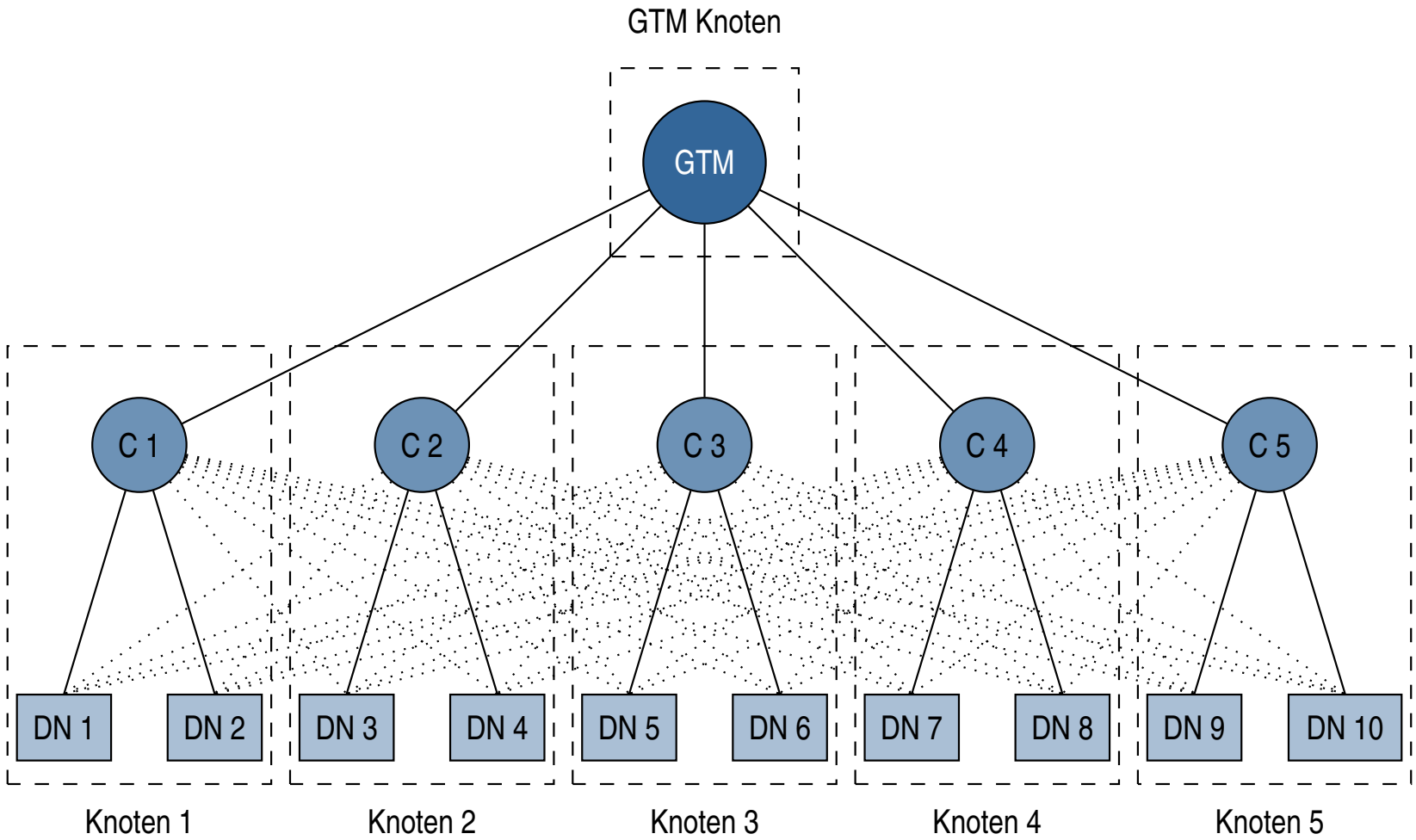
- Vektorraumoperationen auf lokalen Systemen
- Matrixmultiplikation AB mit $A, B \in \mathbb{R}^{n \times n}$ auf verschiedenen Systemen:



R (3.4.2) ohne Speicherung
R mit Speicherung in CSV
Actian Vector 5.0
PostgreSQL 10.1
MonetDB v11.27.5

PostgreSQL Faktor 50 langsamer als Actian Vector

Ausblick – Cluster



Ausblick

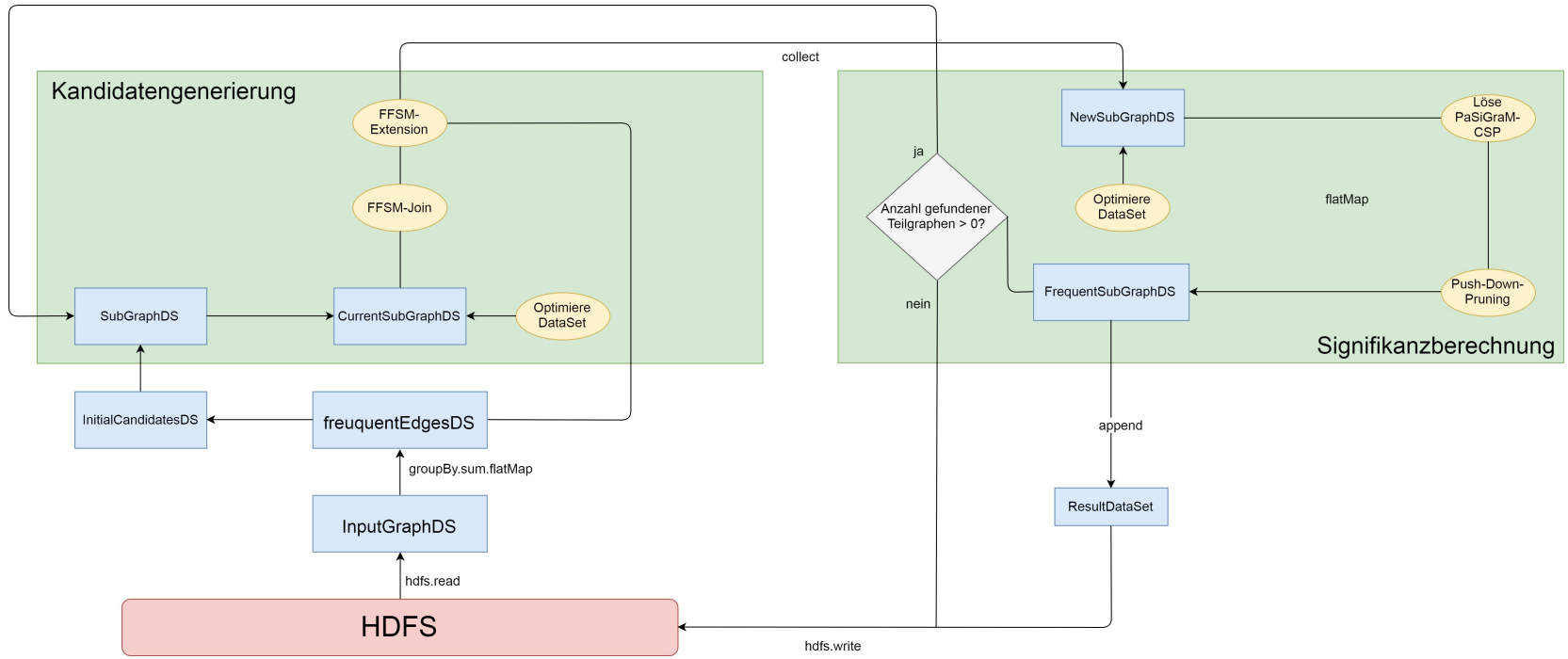
- Vektorraumoperationen auf parallelen Datenbanken:
  - Actian Vector in Hadoop (spaltenorientiert)
  - Postgres-XL (zeilenorientiert)

Weiterführende Arbeit

- Thematik: Parallele Graph-Mining-Techniken zur Auswertung von Hypergraph-Strukturen
- Was gibt es für Graph-Mining-Techniken?
- Wie sind diese auf Hypergraphen anwendbar?
- Wie lässt sich eine Graph-Mining-Technik parallelisieren?

Weiterführende Arbeit

- Umsetzung Frequent-Subgraph-Mining auf Grundlage von Flink und Spark
  - Entwicklung von PaSiGraM[3]



[3] http://eprints.dbis.informatik.uni-rostock.de/959/1/BA_Ole_Fenske_2018.pdf

Herzlichen Dank für Ihre

AufmerksamkeIT

Kurzschreibweisen

CREATE$^?$ *

CREATE * IF NOT EXISTS

$\mathfrak{N}$

TO NODE (dn_3, dn_6, dn_9)

$\mathfrak{T}$

TABLESPACE mounted_tablespace

$\mathfrak{D}_H$

DISTRIBUTE BY HASH

$\mathfrak{D}_R$

DISTRIBUTE BY REPLICATION

Grep-Task

- CREATE TABLE import (*row*) $\mathfrak{D}_H$ (*row*$_d$) TO NODE (*nodes*);
- \copy import from 'twitter_rv.net' delimiter ';';
- SELECT count(*) FROM import WHERE (line ~~'%6125785%'::text);

	row	row$_d$	space	time	#nodes
Test	id1 bigint, id2 bigint	id1	60,555 GB	103 s	9
Test	line char(8)	line	71,355 GB	36 s	9
Test	line varchar(9)	line	71,013 GB	36 s	9
Test	line text	line	71,013 GB	36 s	9
⇒	line text	line	71,005 GB	121 s	3

Join-Task

1. CREATE INDEX ips ON weblog (ip) $\mathfrak{T}$;
   CREATE TABLE ipcount (ip smallint, anzahl int) $\mathfrak{D}_H$ (ip) $\mathfrak{N}$;
   CREATE TABLE ids (id bigint) $\mathfrak{D}_H$ (id) $\mathfrak{N}$;
2. INSERT INTO ipcount (SELECT ip, count(*) anzahl FROM weblog wl GROUP BY wl.ip);
3. INSERT INTO ids (SELECT id FROM weblog WHERE ip=(SELECT ip FROM ipcount WHERE anzahl=(SELECT max(anzahl) FROM ipcount)));
4. SELECT sum(pagerank)/(SELECT count(*) FROM ids) pageranksum FROM pagerank WHERE id IN (SELECT id FROM ids);

Type	Index	t_2/s	t_3/s	t_4/s	t_Σ/s
bigint	x	18	40	6	64
text	x	18	44	7	69
bigint	✓	18	3	6	27
text	✓	19	3	6	28

k-Means

n	t/s
1	328
2	341
3	337
4	336
5	338
6	334
7	337
8	334
9	335
10	333

k	#nodes	t/s
1	3	180
2	3	224
3	3	335
3	9	58
4	9	73
5	9	82
8	9	124

k-Means

Data		
id	follower	pursuer

- CREATE TABLE data (id bigint, follower bigint, pursuer bigint) $\mathfrak{T}$ $\mathfrak{D}_H$ (id) $\mathfrak{N}$;
- CREATE INDEX importid1 ON import (id1) $\mathfrak{T}$;
- CREATE INDEX importid2 ON import (id2) $\mathfrak{T}$;
- INSERT INTO data (SELECT id1, a.pursuer, b.follower FROM (SELECT id1, count(*) pursuer FROM import GROUP BY id1) a INNER JOIN (SELECT id2, count(*) follower FROM import b GROUP BY id2 b ON a.id1=b.id2);
- CREATE INDEX dataid ON data (id) $\mathfrak{T}$;

k-Means

```
CREATE OR REPLACE FUNCTION kmeans_setup() RETURNS void AS $$
BEGIN
  CREATE? TABLE centroid (id smallserial, follower bigint, pursuer bigint) 𝔗 𝔇_R 𝔑;
  CREATE? TABLE distance (id bigint, cid smallint, distance bigint) 𝔗 𝔇_H (id) 𝔑;
  CREATE? TABLE cluster (id bigint, cid smallint) 𝔗 𝔇_H (id) 𝔑;
  CREATE? TABLE temp_centroid (id smallint, follower bigint, pursuer bigint) 𝔗 𝔇_H
(id) 𝔑;
END;
$$ LANGUAGE plpgsql;
```

k-Means

```
CREATE OR REPLACE FUNCTION kmeans_initial(k smallint) RETURNS void AS
$$
BEGIN
  TRUNCATE centroid;
  INSERT INTO centroid (follower, pursuer) (SELECT d.follower, d.pursuer FROM
data d group by d.follower, d.pursuer ORDER BY random() LIMIT k);
END;
$$ LANGUAGE plpgsql;
```

k-Means

```
CREATE OR REPLACE FUNCTION kmeans(k smallint) RETURNS TABLE (id
smallint, follower bigint, pursuer bigint) AS $$
DECLAIRE
b smallint := 1; – distance between centroid and temp_centroid
 – loop-counter, max iterations = 10, wie beim WIN-project
c smallint := 1;
BEGIN
 PERFORM kmeans_initial(k);
 WHILE b > 0 AND c < 11 LOOP
  b := kmeans_update();
  c := c + 1;
 END LOOP;
 RETURN QUERY SELECT * FROM centroid;
END;
$$ LANGUAGE plpgsql;
```

k-Means

```
CREATE OR REPLACE FUNCTION kmeans_update() RETURNS smallint AS $$
DECLAIRE
  anz smallint;
BEGIN
  TRUNCATE distance, cluster, temp_centroid;
  INSERT INTO distance (SELECT d.id, c.id, (d.follower-c.follower)*(d.follower-
c.follower)+(d.pursuer-c.pursuer)*(d.pursuer-c.pursuer) FROM data d, centroid
c);
  INSERT INTO cluster (SELECT d.id, d.cid FROM distance d, (SELECT d.id,
min(distance) mindist FROM distance d GROUP BY d.id) mind WHERE d.id =
mind.id AND d.distance = mind.mindist);
...
```

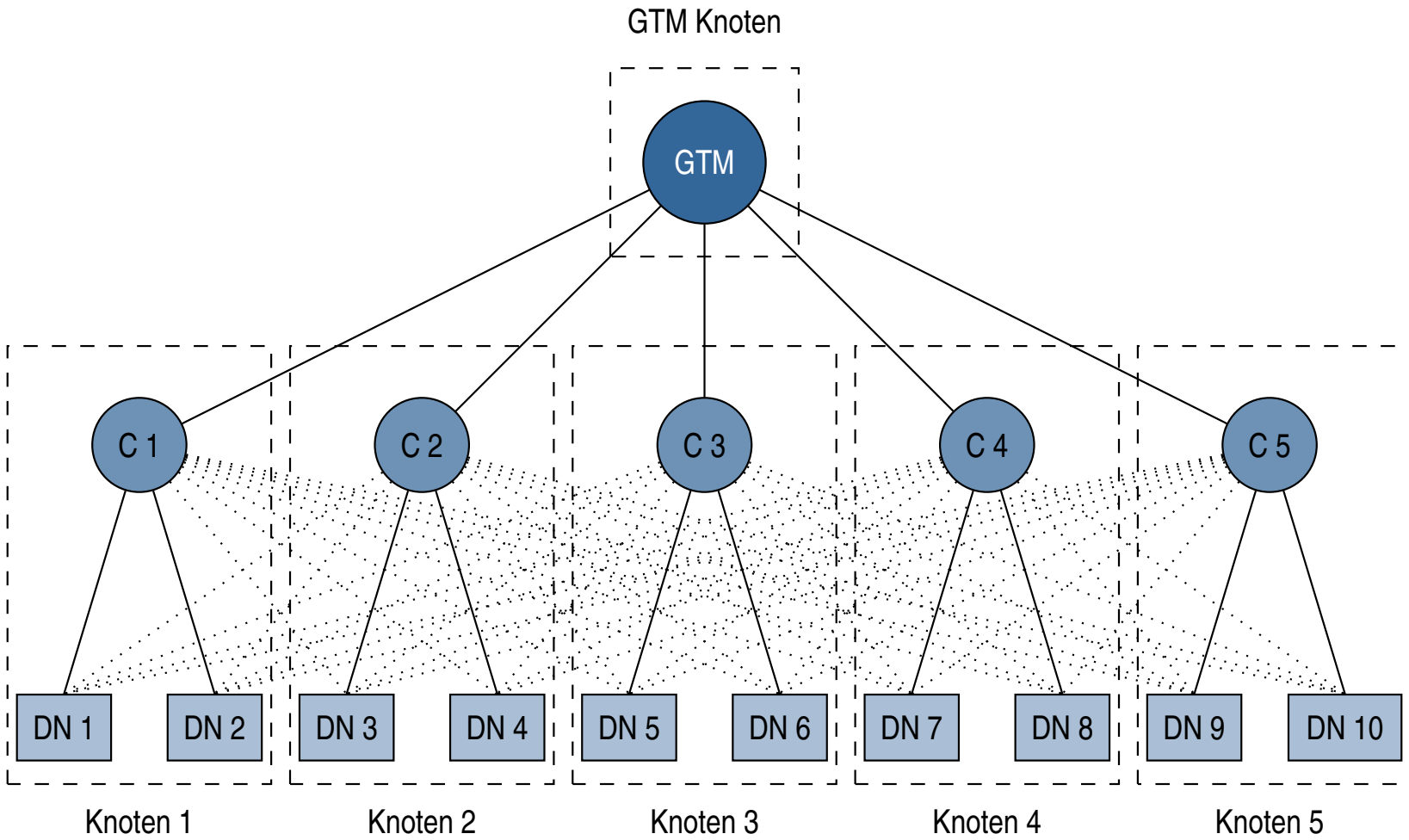
k-Means

```
. . .
  INSERT INTO temp_centroid (SELECT cl.cid id, avg(d.follower), avg(d.pursuer)
FROM data d, cluster cl WHERE d.id=cl.id GROUP BY cl.cid;
  anz := count(*) FROM ((SELECT id, follower, pursuer FROM centroid c FULL
OUTER JOIN temp_centroid t USING (id, follower, pursuer)) EXCEPT (SELECT id,
c.follower, c.pursuer FROM centroid c INNER JOIN temp_centroid t USING (id)))
tmp;
  TRUNCATE centroid;
  INSERT INTO centroid SELECT * FROM temp_centroid;
  RETURN anz;
END;
$$ LANGUAGE plpgsql;
```

k-Means

- SELECT kmeans_setup();
  EXPLAIN ANALYSE SELECT * FROM kmeans('3');
- SELECT kmeans_setup();
  SELECT kmeans_initial('3');
  EXPLAIN ANALYSE SELECT kmeans_update();
  SELECT * FROM centroid;

Postgres-XL Cluster – neu



Postgres-XL Cluster – neu

- Spezifikation eines Knotens mit 1 Koordinator (C) und 2 Datenknoten (DN):

Parameter	Wert
Betriebssystem (BS)	CentOS 7
Prozessoren	$2 \times 1{,}9$ GHz
Cache Größe	2×4 MiB
Arbeitsspeicher	16 GiB DDR4 (2133 Mhz)
Sekundärspeicher (Daten)	1 TB
Sekundärspeicher (BS)	50 GB
Sekundärspeicher (Temporäre Tabellen)	20 GB SSD

Konfiguration – neu

- Standard Konfiguration so, dass PostgreSQL auch auf Raspberry Pi läuft.[4]
  $\Rightarrow$ Enormes Optimierungspotenzial

Parameter	**Standard**	C_i	DN_i
shared_buffers	32 MiB	1 GiB	1 536 MiB
effective_cache_size	4 GiB	3 GiB	4 608 MiB
work_mem	4 MiB	52 428 KiB	78 643 KiB
maintenance_work_mem	64 MiB	512 MiB	768 MiB
min_wal_size	80 MiB	4 GiB	4 GiB
max_wal_size	1 GiB	8 GiB	8 GiB
checkpoint_completion_target	0,5	0,9	0,9
wal_buffers	-1	16 MiB	16 MiB
default_statistics_target	100	500	500
random_page_cost	4	4	4

[4] Tomas Vondra, 2ndquadrant, https://blog.2ndquadrant.com/basics-of-tuning-checkpoints/

	row	row_d	space	time	#nodes
Test	id1 bigint, id2 bigint	id1	60,555 GB	103 s	9
Test	line char(8)	line	71,355 GB	36 s	9
Test	line varchar(9)	line	71,013 GB	36 s	9
Test	line text	line	71,013 GB	36 s	9
⇒	line text	line	71,005 GB	121 s	3

Type	Index	t/s
bigint	x	64
text	x	69
bigint	✓	27
text	✓	28

PageRank

ID	PageRank
1	0,0016546
2	0,0857657
3	0,4534646

$\bowtie_{ID}$

Weblog

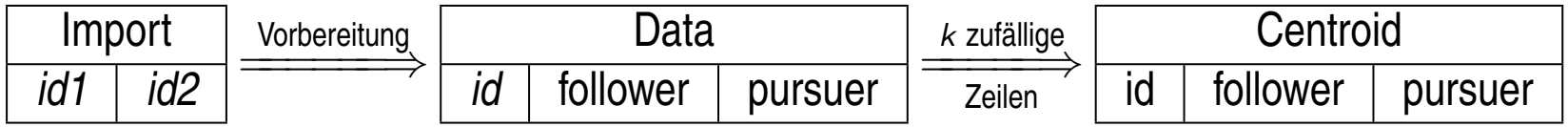
ID	IP	t
1	10	150
2	08	10
3	10	30

$\Rightarrow$

$Ø_{IP=10}$ PageRank
0,2275596

k-Means

kmeans_initial:



kmeans_update: