Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.202072639
Vol. 7, No. 2, April 2020, hlm. 391-398 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No.30/E/KPT/2018 e-ISSN: 2528-6579

OPTIMASI PROSES KLASTERISASI DI *MYSQL DBMS* DENGAN MENGINTEGRASIKAN ALGORITME *MIC-KMEANS* MENGGUNAKAN BAHASA *SQL* DALAM *STORED PROCEDURE*

Issa Arwani *[1]

[1] Fakultas Ilmu Komputer , Universitas Brawijaya
Email: [1] issa.arwani@ub.ac.id
*Penulis Korespondensi

(Naskah masuk: 31 Oktober 2019, diterima untuk diterbitkan: 11 Februari 2020)

Abstrak

Proses klasterisasi data di *DBMS* akan lebih efisien jika dilakukan langsung di dalam *DBMS* itu sendiri karena *DBMS* mendukung untuk pengelolaan data yang baik. *SQL-Kmeans* merupakan salah satu metode yang sebelumnya telah digunakan untuk mengintegrasikan algoritme klasterisasi *K-means* ke dalam *DBMS* menggunakan *SQL*. Akan tetapi, metode ini juga membawa kelemahan dari algoritme *K-means* itu sendiri yaitu lamanya iterasi untuk mencapai konvergen dan keakuratan hasil klasterisasi yang belum optimal akibat dari proses inisialisasi *centroid* awal secara acak. Algoritme *Median Initial Centroid (MIC)-Kmeans* merupakan pengembangan dari algoritme *K-means* yang bisa memberikan solusi optimal dalam menentukan awal *centroid* yang berdampak pada keakuratan dan lamanya iterasi. Dengan keunggulan yang dimiliki algoritme *MIC-Kmeans*, maka dalam penelitian ini dipilih sebagai alternatif algoritme yang diintegrasikan dalam proses klasterisasi data secara langsung di *DBMS* menggunakan *SQL*. Proses integrasinya meliputi 4 tahap yaitu tahap inisialisasi tabel *dataset*, tahap pemetaan algoritme *MIC-Kmeans* pada *SQL* dan tabel *dataset*, tahap perancangan *SQL* untuk tiap hasil pemetaan dan tahap implementasi rancangan *SQL* dalam *MySQL stored procedure*. Hasil pengujian menunjukkan bahwa metode *SQL MIC-Kmeans* bisa mengurangi 43% jumlah iterasi dan mengurangi 39% waktu yang dibutuhkan dari metode *SQL-Kmeans* untuk mencapai konvergen. Selain itu, nilai rata-rata *silhouette coefficient* metode *SQL MIC-Kmeans* adalah 0,79 dan masuk dalam kategori *strong structure* (nilai rentang 0,7 sampai 1). Sedangkan nilai rata-rata *silhouette coefficient* metode *SQL-Kmeans* adalah 0,68 dan masuk dalam kategori *medium structure* (nilai rentang 0,5 sampai 0,7).

Kata kunci: *clustering, SQL-kmeans, MIC-kmeans, MySQL stored procedure*

OPTIMIZATION OF THE CLASTERIZATION PROCESS IN MYSQL DBMS BY INTEGRATING MIC-KMEANS ALGORITHM USING SQL LANGUAGE IN STORED PROCEDURE

Abstract

The process of data clustering in the DBMS will be more efficient because the DBMS supports good data management. SQL-Kmeans is a method that has been used to integrate K-means clustering algorithms into DBMS using SQL. However, it carries the weakness of the K-means algorithm itself in the duration of iterations to reach convergence and the accuracy of clustering due to the centroid initialization process randomly. Median Initial Centroid (MIC)-Kmeans algorithm is a development of the K-means algorithm that can provide the optimal solution in determining the initial centroid which has an impact on the accuracy and duration of iterations. With the advantages of the MIC-Kmeans algorithm, the method was chosen as an alternative algorithm to be integrated in the DBMS using SQL for a clustering. The integration process includes 4 stages, there are dataset initialization, SQL algorithm mapping and dataset table, SQL design for each mapping result, and implementation SQL in the MySQL stored procedure. The test results show that the SQL MIC-Kmeans method can reduce 43% the number of iterations and reduce 39% of the time required from the SQL-Kmeans method to reach convergence. In addition, the average value of the coefficient SQL MIC-Kmeans method is 0.79 and categorized as strong structure (value ranges from 0.7 to 1). While, the average value of the coefficient SQL-Kmeans method is 0.68 and categorized as medium structure (value ranges from 0.5 to 0.7).

Keywords: *clustering, SQL-kmeans, MIC-kmeans, MySQL stored procedure*

1. PENDAHULUAN

Secara umum, proses klasterisasi data yang tersimpan di dalam *Database Management System* (*DBMS)* dilakukan oleh aplikasi di luar *DBMS* dengan cara mengambil data dari *DBMS* terlebih dahulu, kemudian disimpan sementara dalam struktur data program (misal dalam sebuah *array* atau *list*) untuk diproses lebih lanjut dengan menggunakan algoritme klasterisasi. Proses klasterisasi yang dilakukan di luar *DBMS* perlu mempertimbangkan beberapa aspek, diantaranya waktu yang dibutuhkan dalam proses pengambilan data, kualitas data, keamanan data, fleksibilitas dimensi data dan kemampuan aplikasi mengolah data dalam jumlah yang besar. Sehingga, proses klasterisasi data akan lebih efisien jika dilakukan langsung di dalam *DBMS* itu sendiri karena *DBMS* mendukung untuk pengelolaan data yang baik (Ordonez, García 2016).

Standard Query Language (SQL) adalah bahasa standar yang digunakan untuk mengakses dan mengelola data serta tersedia pada semua *DBMS*. Proses klasterisasi data dalam *DBMS* bisa diimplementasikan dengan menggunakan *SQL* meskipun *SQL* sendiri memiliki kelemahan dalam mengoperasikan proses matematika yang komplek terutama dalam pengolahan data matrik. *SQL-Kmeans* merupakan salah satu metode yang sebelumnya telah diimplementasikan dengan mengintegrasikan algoritme klasterisasi *K-means* ke dalam *DBMS* menggunakan *SQL* (Arwani, 2015). Akan tetapi, algoritme *K-means* yang diimplementasikan menggunakan *SQL* akan membawa kelemahan dari algoritme *K-means* itu sendiri yaitu keakuratan dan lamanya iterasi untuk mencapai konvergen (Katara, Juhi, & Naveen, 2015). Sehingga, diperlukan pemilihan algoritme alternatif dari pengembangan algoritme *K-means* yang bisa diimplementasikan menggunakan *SQL* dengan mempertimbangkan kelemahan dari *SQL* itu sendiri.

Pengembangan algoritme *K-means* dalam penentuan awal *centroid* yang berdampak pada keakuratan dan lamanya iterasi untuk mencapai konvergen sudah banyak dilakukan. Algoritme *Median Initial Centroid (MIC)-Kmeans* menggunakan perhitungan *median interkuartil* dari masing-masing atribut untuk menentukan *initial centroid* tiap klasternya (Premkumar, Ganesh, 2017). Algoritme *Radial and Angular Coordinates* (RAC)-*Kmeans* menggunakan perhitungan *n-spherical coordinate system* dan *Cartesian coordinate system* untuk menentukan awal *centroid* (Rahim, Ahmed, 2017). Algoritme *Distance Part (DP-KMeans)* menggunakan metode nilai tengah dari *dataset* yang sudah dinormalisasi dan diurutkan untuk menentukan awal *centroid* (Ilham, Ibrahim, 2018).

Dari ketiga referensi pengembangan algoritme *K-means* diatas, algoritme *MIC-Kmeans* memiliki proses matematika yang paling sederhana dan kompleksitas waktu *asimptotik* yang paling rendah dalam menentukan nilai awal *centroid*. Sehingga, algoritme *MIC-Kmeans* dalam penelitian ini dipilih sebagai alternatif algoritme yang diintegrasikan dalam proses klasterisasi data secara langsung di *DBMS* menggunakan *SQL*. Selanjutnya, permasalahan yang diangkat dalam penelitian ini adalah bagaimana inisialisasi tabel *dataset* dan perancangan *SQL* untuk memetakan setiap tahap algoritme klasterisasi *MIC-Kmeans*. Hasil dari setiap tahap rancangan *SQL* kemudian diintegrasikan dalam *MySQL stored procedure* untuk bisa diimplementasikan di *DBMS*. Terakhir, pengujian dan evaluasi dilakukan pada data uji coba dengan berbagai varian jumlah *dataset* dan klaster.

2. METODE PENELITIAN

Metode yang digunakan dalam penelitian ini ditunjukkan pada Gambar 1 yang meliputi 5 tahap yaitu tahap inisialisasi tabel *dataset*, tahap pemetaan algoritme pada *SQL* dan tabel *dataset*, tahap perancangan *SQL* untuk tiap hasil pemetaan, tahap implementasi rancangan *SQL* dalam *MySQL stored procedure* dan tahap pengujian beserta analisis hasilnya. Berikut penjelasan detail tentang tiap tahap dari metode penelitian yang digunakan:

a. Inisialisasi tabel *dataset*

Pada tahap ini, didefinisikan tabel-tabel yang merupakan representasi dari *dataset* yang dibutuhkan dalam setiap tahap algoritme klasterisasi *MIC-Kmeans*.

b. Pemetaan algoritme *MIC-Kmeans* pada *SQL* dan tabel *dataset*

Pada tahap ini, dilakukan pemetaan *SQL* yang dibutuhkan dalam pengoperasian setiap tahap algoritme klasterisasi *MIC-Kmeans* terhadap tiap tabel *dataset* yang bersesuaian sebagai tempat penyimpanan hasil pengolahan data.

c. Perancangan *SQL* untuk tiap tahap hasil pemetaan

Pada tahap ini, dilakukan perancangan *SQL* berdasarkan hasil pemetaan setiap tahap proses operasi algoritme klasterisasi *MIC-Kmeans*.

d. Implementasi rancangan *SQL* dalam *MySQL stored procedure*

Pada tahap ini, dilakukan implementasi rancangan *SQL* pada tahap sebelumnya ke dalam *stored procedure* supaya bisa dioperasikan dalam *DBMS MySQL* melalui pemanggilan *stored procedure* utama.

e. Pengujian dan analisis hasil pengujian

Pada tahap ini, dilakukan pengujian dan analisis hasil dari berbagai varian jumlah *dataset*. Pengujian dilakukan dengan membandingkan hasil kinerja klasterisasi *dataset* antara metode *SQL-Kmeans* dengan *SQL MIC-Kmeans* di *DBMS MySQL* dari sisi banyaknya iterasi dan waktu yang diperlukan untuk mencapai konvergen serta kualitas

hasil klasterisasi yang ditunjukkan dengan nilai rata-rata *silhouette coefficient* (Govinda, Varaprasada, & Rambabu, 2018).

Gambar 1 Metode Penelitian

3. PROSES INTEGRASI ALGORITME *MIC-KMEANS KE* DALAM *DBMS MySQL*

3.1. Inisialisasi tabel *dataset*

Ada beberapa istilah tabel yang digunakan dalam penyimpanan hasil pemrosesan setiap tahap algoritme *MIC-Kmeans* yang direpresentasikan dalam tabel *dataset*. Pada tahap ini dilakukan inisialisasi tabel menggunakan pernyataan *Data Definition Language* (*DDL*). Ada sepuluh tabel *dataset* yang didefinisikan diawal seperti ditunjukkan pada Tabel 1. Kolom yang diberi tanda garis bawah dari masing – masing tabel merupakan *primary key* dari tabel tersebut. Sedangkan *subscript i,j,l* secara berturut-turut adalah jumlah *dataset*, jumlah klaster dan jumlah dimensi dari *dataset*.

3.2. Pemetaan algoritme *MIC-Kmeans* terhadap *SQL dan* tabel *dataset*

Komponen utama dalam memetakan algoritme *MIC-Kmeans* ke dalam bahasa *SQL* yaitu bagaimana mengekspresikan perhitungan matrik dari algoritme *MIC-Kmeans* ke dalam tabel *dataset* dengan mendesain *SQL* untuk setiap tahapnya. Tabel 2 menggambarkan *SQL* yang dibutuhkan berdasarkan pemetaan setiap tahap algoritme *MIC-Kmeans* yang mana hasilnya akan disimpan dalam tabel-tabel *dataset* yang bersesuaian.

Tabel 1. Dataset

Tabel *Dataset*	Kolom	Keterangan
YH	i,Y1,Y2,..., Y*l*	Tabel yang berisi *dataset* sebanyak *i* baris data dan *l* kolom (dimensi).
YV	i, *l*, val	Tabel yang berisi nilai *val* untuk masing-masing baris ke *i* dan dimensi *l* dari tabel *YH*.
YVsort	i, *l*, val	Tabel yang berisi data *val* yang sudah diurutkan nilainya dari tabel *YV* untuk masing-masing dimensi *l*.
Cmed	j, m, val	Tabel yang berisi baris ke *i* dari tabel *YVsort* untuk menentukan nilai *val* berdasarkan nilai *median* interkuartil *m* tiap klaster *j*.
CV	j, *l*, val	Tabel yang berisi *val* dari nilai *median* interkuartil tiap klaster *j* untuk masing-masing dimensi *l*.
YD	i, j, dist	Tabel yang berisi nilai jarak *dist* untuk masing-masing data ke *i* dengan klaster *j*
YNN	i, j	Tabel yang berisi hasil klaster *j* untuk masing-masing data baris ke *i*
R	j, *l*, val	Tabel yang berisi nilai varian *val* berdasarkan nilai *centroid* yang baru pada tabel *CV*
W	j, w	Tabel yang berisi jumlah/bobot *w* dari data untuk setiap klaster *j*
model	Avg_q, iteration	Tabel yang berisi nilai error *Avg_q* untuk menentukan konvergen dan nilai *iteration* untuk menyimpan banyaknya iterasi

Tabel 2. Pemetaan tiap tahap algoritme *MIC-Kmeans* terhadap *SQL* dan tabel *dataset*

Tahap	*MIC-Kmeans*	*SQL MIC-Kmeans*
1	Inisialisasi *dataset*	*SQL* untuk inisialisasi data dari *dataset* yang diisikan pada tabel *YH* dan *YV*
2	*Sorting dataset* untuk masing-masing atribut	*SQL* untuk *sorting* tiap atribut data dari tabel *YV* yang kemudian diisikan pada tabel *Yvsort*.
3	Menentukan nilai *median* interkuartil klaster untuk masing-masing atribut sebagai initial *centroid*	*SQL* untuk menghitung nilai *median* dari tabel *Yvsort* tiap klaster sebagai *initial centroid* dan diisikan pada tabel *Cmed* dan *CV*
4	Menghitung jarak setiap data pada *centroid*	*SQL* untuk menghitung jarak antara data tabel *YV* dengan tabel *CV* dan diisikan pada tabel *YD*
5	Mengelompokkan data berdasarkan jarak terdekat dengan *centroid*	*SQL* untuk memilih jarak terdekat dari tabel *YD* untuk masing-masing data terhadap *centroid* dan diisikan pada tabel *YNN*
6	Memutakhirkan *centroid* baru	*SQL* untuk memperbarui data pada tabel *CV*
7	Cek nilai konvergen	*SQL* untuk memperbarui data pada tabel *W*, *R* dan *model*
8	Tahap 4-7 diulang hingga konvergen bernilai *true*	Tahap 4-7 diulang hingga nilai *Avg_q* pada tabel *model* konvergen

3.3. Perancangan *SQL* tiap tahap hasil pemetaan

Perancangan *SQL* dilakukan berdasarkan hasil pemetaan setiap tahap proses operasi algoritme klasterisasi *MIC-Kmeans*. Berikut detail perancangan *SQL* untuk setiap tahapnya:

Tahap 1: Perancangan *SQL* untuk inisialisasi *dataset* yang diisikan pada tabel *YH* dan *YV*. Tabel *YH* diisi dengan pernyataan *SQL* sebagai berikut:

```
INSERT INTO YH
SELECT @n:=@n+1 i,Y1,Y2, ..., Yl
FROM <dataset_table>,(SELECT @n:=0) m;
```

Pernyataan `@n:=@n+1 i` dan `(SELECT @n:=0) m` digunakan untuk fungsi *incremental* yang memberikan nilai unik dari kolom *primary key i* pada tabel *YH*. Sedangkan Tabel *YV* diisi dengan pernyataan *SQL* sebagai berikut:

```
INSERT INTO YV SELECT i,1,Y1 FROM YH;
...
INSERT INTO YV SELECT i,l,Yl FROM YH;
```

Tahap 2: Perancangan *SQL* untuk *sorting* tiap atribut data dari tabel *YV* yang kemudian diisikan pada tabel *Yvsort*. Tabel *Yvsort* diisi dengan pernyataan *SQL* sebagai berikut:

```
INSERT INTO YVsort
SELECT (@n1:=@n1+1) i,l,val
FROM YV, (SELECT @n1 := 0) tb WHERE l=1
ORDER BY val;
...
INSERT INTO YVsort
SELECT (@n1:=@n1+1) i,l,val
FROM YV, (SELECT @n1 := 0) tb WHERE l=d
ORDER BY val;
```

Tahap 3: Perancangan *SQL* untuk menghitung nilai *median* dari tabel *Yvsort* tiap klaster *k* sebagai *initial centroid* dan diisikan pada tabel *Cmed* dan *CV*. Tabel *Cmed* diisi dengan pernyataan *SQL* sebagai berikut:

```
INSERT INTO Cmed (
SELECT 1,1, FLOOR(c/(2*k))
FROM (SELECT COUNT(*) AS c FROM YH) tb
)
UNION(
SELECT 1,2, CEIL(c/(2*k))
FROM
(SELECT COUNT(*) AS c FROM YH) tb);
...
INSERT INTO Cmed (
SELECT @j,1, FLOOR((2*@j-1)*c/(2*k))
FROM (SELECT COUNT(*) AS c FROM YH) tb
)
UNION(
SELECT @j,2, CEIL((2*@j-1)*c/(2*k))
FROM
(SELECT COUNT(*) AS c FROM YH) tb);
```

Sedangkan Tabel *CV* diisi dengan pernyataan *SQL* sebagai berikut:

```
INSERT INTO CV
SELECT j,l, AVG(yvsort.val)
FROM yvsort,ch WHERE yvsort.i=ch.val
GROUP BY j,l;
```

Tahap 4: Perancangan *SQL* untuk menghitung jarak antara data tabel *YV* dengan tabel *CV* dan diisikan pada tabel *YD*. Tabel *YD* diisi dengan pernyataan *SQL* sebagai berikut:

```
INSERT INTO YD
SELECT i,j,
SUM(pow ((YV.val-CV.val),2))AS dist
FROM YV,CV WHERE YV.l =CV.l
GROUP BY i,j;
```

Tahap 5: Perancangan *SQL* untuk memilih jarak terdekat dari tabel *YD* untuk masing-masing data terhadap *centroid* dan diisikan pada tabel *YNN*. Tabel *YNN* diisi dengan pernyataan *SQL* sebagai berikut:

```
INSERT INTO YNN
SELECT YD.i,YD.j
FROM YD,(SELECT i,MIN(dist) AS min_dist
FROM YD GROUP BY i) YMIND
WHERE YD.i = YMIND.i and YD.dist =
YMIND.min_dist GROUP BY YD.i;
```

Tahap 6: Perancangan *SQL* untuk memperbarui data tabel *CV*. Tabel *CV* diisi dengan pernyataan *SQL* sebagai berikut:

```
INSERT INTO CV
SELECT j,l,AVG(YV.val)
FROM YV,YNN
WHERE YV.i=YNN.i GROUP BY j,l;
```

Tahap 7: Perancangan *SQL* untuk memperbarui data tabel *W*, *R* dan *model*. Tabel *W*, *R* dan *model* diisi dengan pernyataan *SQL* sebagai berikut:

```
INSERT INTO W SELECT j,COUNT(*)
FROM YNN GROUP BY j;

INSERT INTO R
SELECT CV.j,CV.l ,
AVG (pow((YV.val-CV.val),2))As val
FROM CV,YV,YNN
WHERE YV.i=YNN.i and YV.l =CV.l AND
YNN.j=CV.j
GROUP BY CV.j, CV.l;

UPDATE model
SET avg_q= (SELECT SUM(W.w*R.val) AS
avg_q FROM R,W WHERE R.j=W.j),
iteration= iteration+1;
```

3.4. Implementasi rancangan *SQL* dalam *MySQL Stored Procedure*

Setiap perancangan *SQL* yang telah didefinikan pada tahap 1 sampai dengan tahap 7 kemudian diimplementasikan dalam *MySQL stored procedure*. Tabel 3 menggambarkan detail lengkap untuk masing-masing isi dari *stored procedure*. Nama *stored procedure* diambilkan dari setiap tahap pemetaan algoritme terhadap tabel *dataset* yang berkolerasi. Di akhir tabel, ditambahkan sebuah *stored procedure* utama untuk memanggil setiap tahap *stored procedure* hingga proses klasterisasi konvergen.

Tabel 3. Implementasi rancangan *SQL* dalam *MySQL Stored Procedure*

No	Implementasi *Stored Procedure*

1	**Nama Stored Procedure: Step_1_SP_YH**
	`CREATE PROCEDURE Step_1_SP_YH()`
	`BEGIN`
	`TRUNCATE YH;`
	`INSERT INTO YH SELECT @n:=@n+1 i,Y1,Y2`
	`FROM Y,(SELECT @n:=0) m;`
	`END`
	Nama Stored Procedure: Step_1_SP_YV
	`CREATE PROCEDURE Step_2_SP_YV`
	`(IN j INT)`
	`BEGIN`
	`SET @d=1;`
	`TRUNCATE YV;`
	`WHILE (@d<=j)`
	`DO`
	`SET @s = CONCAT('INSERT INTO YV`
	`SELECT i,',@d,',Y',@d,' FROM YH');`
	`PREPARE stmt FROM @s;`
	`EXECUTE stmt;`
	`DEALLOCATE PREPARE stmt;`
	`SET @d=@d+1;`
	`END WHILE;`
	`END`

2	**Nama Stored Procedure: Step_2_SP_ YVsort**
	`CREATE PROCEDURE Step 2 SP YVsort(IN j INT)`
	`BEGIN`
	`SET @d=1;`
	`TRUNCATE YVsort;`
	`WHILE (@d<=j)`
	`DO`
	`INSERT INTO YVsort`
	`SELECT (@n1 := @n1 + 1) i,l, val`
	`FROM YV, (SELECT @n1 := 0) tb`
	`WHERE l=@d ORDER BY val;`
	`SET @d=@d+1;`
	`END WHILE;`
	`END`

3	**Nama Stored Procedure: Step_3_SP_ Cmed**
	`CREATE Step_3_SP_Cmed (IN k INT)`
	`BEGIN`
	`SET @j=1;`
	`TRUNCATE Cmed;`
	`WHILE (@j<=k)`
	`DO`
	`insert into Cmed (`
	`select @j,1, FLOOR((2*@j-1)*c/(2*k))`
	`from(select count(*) as c from YH) tb)`
	`UNION(`
	`select @j,2, CEIL((2*@j-1)*c/(2*k))`
	`from(select count(*) as c from YH) tb);`
	`SET @j=@j+1;`
	`END WHILE;`
	`END`
	Nama Stored Procedure: Step_3_SP_ CV
	`CREATE Step_3_SP_ CV()`
	`BEGIN`
	`TRUNCATE CV;`
	`INSERT INTO CV SELECT j,l, AVG(yvsort.val)`
	`FROM yvsort,ch WHERE yvsort.i=ch.val`
	`GROUP BY j,l;`
	`END`

4	**Nama Stored Procedure: Step_4_SP_ YD**
	`CREATE PROCEDURE Step_4_SP_ YD()`
	`BEGIN`
	`TRUNCATE YD;`
	`INSERT INTO YD`
	`SELECT i,j,SUM(pow ((YV.val-CV.val),2))AS dist FROM YV,CV`
	`WHERE YV.l =CV.l GROUP BY i,j;`
	`END`

5	**Nama Stored Procedure: Step_5_SP_ YNN**
	`CREATE Step_5_SP_ YNN()`
	`BEGIN`
	`TRUNCATE YNN;`
	`INSERT INTO YNN`
	`SELECT YD.i,YD.j`
	`FROM YD,(SELECT i,MIN(dist) AS min_dist FROM YD GROUP BY i) YMIND`
	`WHERE YD.i = YMIND.i AND YD.dist = YMIND.min_dist GROUP BY YD.i;`
	`END`

6	**Nama Stored Procedure: Step_6_SP_ CV**
	`CREATE PROCEDURE Step_6_SP_ CV()`
	`BEGIN`
	`TRUNCATE CV;`
	`INSERT INTO CV`
	`SELECT j,l,AVG(YV.val)`
	`FROM YV,YNN`
	`WHERE YV.i=YNN.i GROUP BY j,l;`
	`END`

7	**Nama Stored Procedure: Step_7_SP_ W_R_MODEL**
	`CREATE PROCEDURE Step_7_SP_ W_R_MODEL()`
	`BEGIN`
	`TRUNCATE W; TRUNCATE R;`
	`INSERT INTO W SELECT j,COUNT(*)`
	`FROM YNN GROUP BY j;`
	
	`INSERT INTO R`
	`SELECT CV.j,CV. l ,AVG (pow ((YV.val-CV.val),2))As val`
	`FROM CV,YV,YNN`
	`WHERE YV.i=YNN.i AND YV.l =CV.l AND YNN.j=CV.j GROUP BY CV.j, CV. l;`
	
	`UPDATE model`
	`SET avg_q= (SELECT SUM(W.w*R.val) AS avg_q`
	`FROM R,W WHERE R.j=W.j),`
	`iteration= iteration+1;`
	`END`

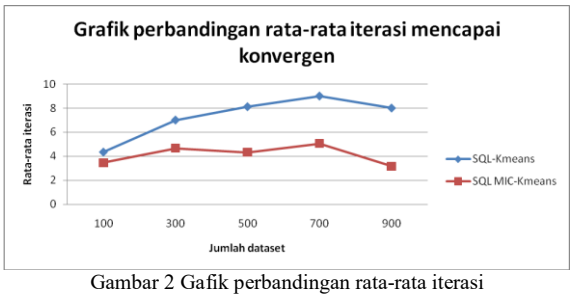
8	**Nama Stored Procedure: Step_8_SP_ *SQL_MIC_Kmeans***
	`CREATE PROCEDURE Step_8_SP_SQL_MIC_Kmeans(`
	`IN iterasi INT(11),`
	`IN klaster INT(11),`
	`IN dimensi INT(11))`
	`BEGIN`
	`SET @it=0;`
	`SET @x=-1;`
	
	`CALL Step 1 SP YH();`
	`CALL Step 1 SP YV(dimensi);`
	`CALL Step_2_SP_YVsort(dimensi);`
	`CALL Step_3_SP_Cmed(klaster);`
	`CALL Step_3_SP_CV();`
	`UPDATE model SET avg_q=0, iteration=0;`
	
	`WHILE (@x!=(SELECT avg q FROM model)AND @It<=iterasi)`
	`DO`
	`CALL Step_4_SP_YD();`
	`CALL Step_5_SP_YNN();`
	`SET @x=(SELECT avg_q FROM model);`
	`CALL Step 6 SP CV();`
	`CALL Step_7_SP_W_R_MODEL();`
	`SET @it=@it+1;`
	`END WHILE;`
	`END`

4. HASIL DAN PEMBAHASAN

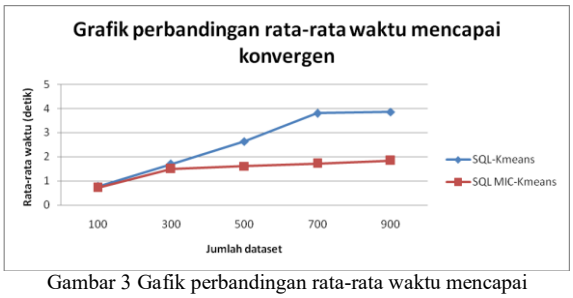
Pengujian dan evaluasi dilakukan dengan membandingkan kinerja klasterisasi *dataset* antara metode *SQL-Kmeans* dengan *SQL MIC-Kmeans* di *DBMS MySQL* dengan 3 skenario uji coba. Uji coba yang pertama adalah uji coba jumlah iterasi dan waktu yang dibutuhkan untuk mencapai konvergen dengan varian jumlah *dataset*. Ujicoba yang kedua

adalah uji coba kualitas klaster dengan metode *silhouette coefficient* dengan varian jumlah *dataset*. Sedangkan uji coba yang ketiga adalah ujicoba waktu yang dibutuhkan untuk satu kali iterasi yang kemudian dianalisis dengan perhitungan kompleksitas waktu *asimptotik*. Pengujian skenario pertama dan kedua dilakukan 5 kali percobaan untuk setiap varian jumlah dan klaster *dataset*-nya kemudian diambil nilai rata-ratanya. Pengujian dilakukan pada komputer personal (bukan komputer server) dengan spesifikasi memori 4 GB, dan *processor intel core i5 cpu 3 Ghz*. *Dataset* yang digunakan adalah data perkembangan *Indeks Prestasi Kumulatif (IPK)* dan jumlah sks lulus mahasiswa untuk setiap semesternya dengan variasi jumlah dataset 100, 300, 500, 700, dan 900 data.

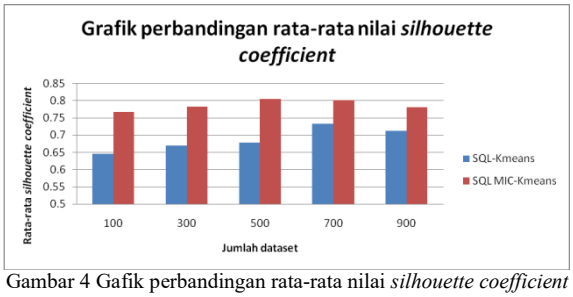
Dari grafik pada Gambar 2 dan Gambar 3, hasil penelitian menunjukkan bahwa dengan berbagai jumlah varian *dataset*, maka jumlah rata-rata iterasi dan rata-rata waktu proses klasterisasi untuk mencapai konvergen metode *SQL MIC-Kmeans* lebih kecil jika dibandingkan dengan metode *SQL-Kmeans*. Dalam perhitungan persentase, metode *SQL MIC-Kmeans* bisa mengurangi 43% jumlah iterasi dan mengurangi 39% waktu yang dibutuhkan dari metode *SQL MIC-Kmeans* untuk mencapai konvergen. Hal tersebut dikarenakan metode *SQL-Kmeans* pada proses inisialisasi *centroid* awal dilakukan secara acak yang menyebabkan algoritmenya tidak stabil. Sedangkan metode *SQL MIC-Kmeans* menggunakan perhitungan median tiap atribut (dimensi) untuk menentukan inisialisasi *centroid* awal masing-masing klaster sehingga hasilnya lebih optimal.



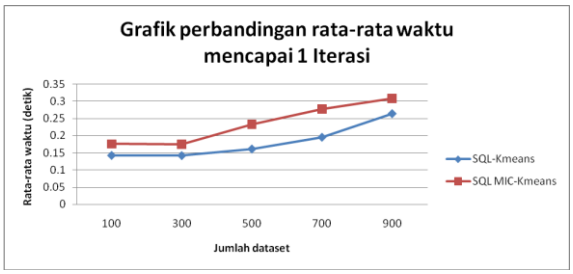
Gambar 2 Gafik perbandingan rata-rata iterasi



Gambar 3 Gafik perbandingan rata-rata waktu mencapai konvergen



Gambar 4 Gafik perbandingan rata-rata nilai *silhouette coefficient*



Gambar 5 Gafik perbandingan rata-rata waktu 1 iterasi

Dari grafik pada Gambar 4, hasil penelitian menunjukkan bahwa dengan berbagai jumlah varian *dataset*, maka nilai rata-rata *silhouette coefficient* metode *SQL MIC-Kmeans* lebih tinggi jika dibandingkan dengan metode *SQL-Kmeans*. Nilai rata-rata *silhouette coefficient* metode *SQL-Kmeans* dan *SQL MIC-Kmeans* masing-masing adalah 0,68 dan 0,79. Berdasarkan nilai rata-rata *silhouette coefficient* maka metode *SQL-Kmeans* masuk dalam kategori *medium structure* (nilai rentang 0,5 sampai 0,7) sedangkan metode *SQL MIC-Kmeans* masuk dalam kategori *strong structure* (nilai rentang 0,7 sampai 1). Hal tersebut juga dikarenakan metode *SQL MIC-Kmeans* sejak awal telah melakukan optimasi dalam inisialisasi nilai *centroid* awal sehingga hasil dari klasterisasi juga bisa optimal.

Dari grafik pada Gambar 5, hasil penelitian menunjukkan bahwa dengan berbagai jumlah varian *dataset*, ternyata waktu yang dibutuhkan untuk 1 iterasi proses klasterisasi dengan metode *SQL-Kmeans* lebih cepat dari pada metode *SQL MIC-Kmeans*. Hal tersebut disebabkan karena adanya tambahan proses dalam menentukan inisialisasi nilai *centroid* awal pada metode *SQL MIC-Kmean*. Sedangkan dalam metode *SQL-Kmeans*, penentuan inisialisasi nilai *centroid* awal dilakukan secara acak dan hanya membutuhkan satu proses. Hal ini juga bisa dijelaskan dalam perbandingan perhitungan kompleksitas waktu *asimptotik* untuk masing-masing tahap algoritme dari *SQL-Kmeans* maupun *SQL MIC-Kmean* pada Tabel 4. Pada metode *SQL MIC-Kmean*, proses dalam menentukan inisialisasi nilai *centroid* awal memiliki kompleksitas waktu *asimptotik* yang lebih tinggi yaitu *O(ndk)*, sedangkan kompleksitas waktu *asimptotik* pada metode *SQL-Kmeans* adalah *O(kd)*.

Tabel 4. Perbandingan kompleksitas waktu *asimptotik* tiap tahap algoritme *SQL-Kmeans dan SQL MIC-Kmeans*

LangkahUmum Algoritme	***SQL - Kmeans***	***SQL MIC-Kmeans***
1. Inisialisasi *dataset*	O(nd)	O(nd)
2. Inisialisasi *centroid*	O(kd)	O(ndk)
3. Menghitung jarak setiap data pada *centroid*	O(nkd)	O(nkd)
4. Mengelompokkan data berdasarkan jarak terdekat dengan *centroid*	O(nk)	O(nk)
5. Memutakhirkan *centroid* baru	O(nd)	O(nd)
6. Cek nilai konvergen	$O(n^2dk)$	$O(n^2dk)$

5. KESIMPULAN

Proses klasterisasi secara langsung dalam *DBMS MySQL* berhasil dilakukan dengan mengintegrasikan algoritme klasterisasi *MIC-Kmeans* menggunakan *SQL*. Proses integrasi dilakukan dengan beberapa tahap yang diawali dengan tahap inisialisasi tabel-tabel *dataset* dalam *DBMS* menggunakan *DDL*. Kemudian dilanjutkan dengan pemetaan tahap-tahap algoritme *MIC-Kmeans* terhadap pendefinisian *SQL* yang dibutuhkan untuk setiap proses operasinya. Perancangan *SQL* kemudian dilakukan berdasarkan proses dari setiap tahap algoritme *MIC-Kmeans* dengan melakukan *join* tabel-tabel *dataset* yang bersesuaian. Proses implementasi dilakukan dengan membuat *stored procedure* untuk masing-masing rancangan *SQL* dan dipanggil dalam sebuah *stored procedure* utama. Hasil pengujian menunjukkan bahwa metode *SQL MIC-Kmeans* bisa mengurangi 43% jumlah iterasi dan mengurangi 39% waktu yang dibutuhkan dari metode *SQL-Kmeans* untuk mencapai konvergen. Selain itu, nilai rata-rata *silhouette coefficient*-nya masuk dalam kategori *strong structure*. Akan tetapi waktu yang dibutuhkan untuk 1 kali iterasi, metode *SQL-Kmeans* lebih cepat dari pada metode *SQL MIC-Kmeans*. Sehingga, untuk penelitian selanjutnya disarankan untuk bisa dilakukan optimasi *SQL*-nya.

DAFTAR PUSTAKA

ARWANI, I., 2015. Integrasi Algoritme K-Means Dengan Bahasa Sql Untuk Klasterisasi Ipk Mahasiswa (Studi Kasus: Fakultas Ilmu Komputer Universitas Brawijaya), 2(2), p. 143-151, Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK).

GOVINDA, R., VARAPRASADA, R.,RAMBABU, R., 2018. A Novel Approach in Clustering Algorithm to Evaluate the Performance of Regression Analysis, 42(6), p.143-146, IEEE 8th International Advance Computing Conference (IACC).

ILHAM, A., IBRAHIM, D., 2018. Tackling Initial Centroid of K-Means with Distance Part (DP-KMeans), 42(6), p.185-189, International Symposium on Advanced Intelligent Informatics (SAIN).

KATARA, JUHI, dan NAVEEN C.A., 2015. Modified Version of the K-means Clustering Algorithm, vol. 15, no. 7, Global Journal of Computer Science and Technology.

ORDONEZ, C., GARCÍA, J., 2016. Managing Big Data Analytics Workflows with a Database System, 18(2), p. 649-655, 16th IEEE/ ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid).

PREMKUMAR, M.S., GANESH, S.H, 2017. A Median Based External Initial Centroid Selection Method for K-Means Clustering, 42(6), p.143-146, Computing and Communication Technologies (WCCCT).

RAHIM, S.M.D., AHMED, T, 2017. An Initial Centroid Selection Method based on Radial and Angular Coordinates for K-means Algorithm, p.22-24, International Conference of Computer and Information Technology (ICCIT).

Halaman ini sengaja dikosongkan