Analisis Kinerja MySQL Cluster Dengan Metode Load Balancing

[1]Yosep Dwi Irawan, [2] Dr.F. Yudi Limpraptono, ST,. MT., [3] Sotyohadi, ST,.MT.
Institut Teknologi Nasional, Malang, Indonesia
[1]mydwiirawan@gmail.com

Abstract—MySQL Cluster merupakan gabungan beberapa MySQL Server yang berdiri sendiri dan membentuk suatu sistem tunggal sebagai solusi untuk ketersediaan data yang tinggi *(High Availbility*) dalam *database*. Namun pada praktiknya sistem tersebut mempunyai kelemahaan ketika terjadi *request* data bersamaan dari *user* maka *server* akan mengalami *overloading* dan penurunan performa karena tidak mampu melayani *request* data dengan jumlah besar.

Pada penelitian ini dibuat suatu mekanisme untuk mengatasi masalah tersebut dengan menambahkan *load balancing* pada *server cluster. Load balancing* digunakan untuk meningkatkan performa suatu *server* dengan membagi *request* ke beberapa *server* lainnya, sehingga membuat beban masing-masing server menjadi lebih ringan. Pada penelitian ini menggunakan algoritma *Round Robin* dan *Least Connection.* Pengujian menggunakan aplikasi *Sysbench* dengan parameter yang diuji adalah *Transaction per Seconds* dan *Response Time.* Cara kerjanya dengan memberikan beban kerja 8 hingga 512 *threads* ke *server* pada kondisi sebelum dan sesudah *load balancing.*

Hasil pengujian menunjukan bahwa MySQL Cluster tanpa *load balancing* saat beban 512 *threads* tidak mampu melayani transaksi data karena *overload,* namun setelah menggunakan *load balancing* MySQL Cluster dapat melayani transaksi data tanpa mengalami masalah dan tidak *overload*. Nilai dari parameter TPS dan *Response Time* MySQL Cluster dengan *load balancing* lebih baik dari MySQL Cluster tanpa *load balancing*.

Kata Kunci— MySQL Cluster, Load Balancing, Threads, Round Robin, Least Connection

I. PENDAHULUAN

A. Latar Belakang

Berkembangnya teknologi informasi saat ini dan tingginya penggunaan *database*, suatu sistem penyimpanan data berbasis komputer dituntut untuk dapat melayani kebutuhan *database* secara terus menerus dari penggunanya tanpa mengalami masalah yang berarti. Oleh karena itu dibutuhkan suatu sistem pengolahan *database* yang dapat melayani transaksi data yang tinggi. Dengan melihat banyaknya *database* dengan tipe MySQL yang digunakan secara umum, salah satu teknologi komputer yang mempunyai fasilitas mengelola data tersebut adalah MySQL Cluster.

Semakin banyak jumlah *client* yang mengakses suatu *server*, maka semakin berat pula performa sebuah *server* dalam menerima *request.* Banyaknya jumlah *request* pada suatu *server* akan mengakibatkan terjadinya *overloading* bahkan kemungkinan *server* akan mengalami *down* . Oleh karena itu dibutuhkan sebuah penyeimbang beban *server* yang akan mengatur dan membagi kerja server. Karena pada penerapan sistem *cluster* ketika ada *request* dari *client* hanya menggunakan satu *server* tunggal dalam fungsinya karena server yang lain hanya siaga sebagai *backup* ketika terjadi kegagalan pada *server* sebelumnya[1].

Load balancing server merupakan salah satu cara yang digunakan untuk meningkatkan kinerja dan tingkat ketersediaan *server*, yaitu dengan membagi *request* yang datang ke beberapa *server* sekaligus, sehingga beban yang ditanggung oleh masing-masing *server* lebih ringan. Tingkat ketersediaan *server* bisa tetap terjaga dengan penggunaan *load balancing* ini, yaitu ketika salah satu *server* tidak dapat melayani permintaan pengguna (*server down*), maka secara otomatis *server* yang lain langsung menggantikannya, sehingga pengguna seakan-akan tidak mengetahui bahwa *server* tersebut *down*[2].

Disisi lain penelitian tentang algoritma *load balancing* juga banyak dilakukan untuk mengetahui perbandingan kinerja dalam pembagian beban kerja *server*. Dari hal tersebut pada penelitian ini juga diimplementasikan dua algoritma penjadwalan *Round Robin* dan *Least Connection* pada sistem *load balancing* untuk menganalisis perbandingan kinerjannya pada setiap beban *threads* yang diberikan.

B. Rumusan Masalah

Berdasarkan latar belakang di atas maka dapat dirumuskan masalah sebagai berikut :

1. Bagaimana rancangan sistem MySQL Cluster ?
2. Bagaimana cara menyeimbangkan beban kerja *server* pada MySQL Cluster ?
3. Bagaimana pengaruh dari penambahan *load balancer* terhadap kinerja *server* MySQL Cluster ?
4. Bagaimana perbandingan kinerja dari algoritma *load balancing* yang berbeda terhadap kinerja *server* MySQL Cluster ?

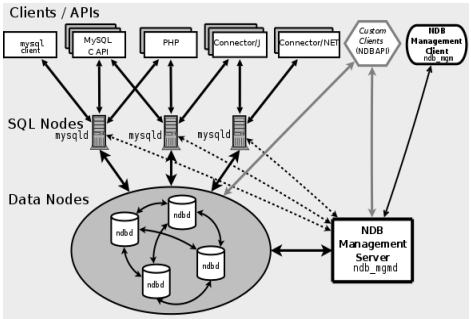
C. Tujuan Penelitian

1. Mengatasi beban *server* pada sistem MySQL Cluster.
2. Mengetahui pengaruh dari penggunaan *load balancing* terhadap sistem MySQL Cluster.

3. Menganalisis perbandingan kinerja *server* dari *MySQL Cluster* sebelum dan sesudah menggunakan *load balancing,* dengan menganalisa parameter *transaction per seconds* dan *response time.*
4. Menganalisis perbandingan kinerja *server* dari *MySQL Cluster load balancing* dengan menggunakan algoritma penjadwalan yang berbeda yaitu *Round Robin* dan *Least Connection,* dengan menganalisa parameter *transaction per seconds* dan *response Time.*

II. TINJAUAN PUSTAKA

A. *MySQL Cluster*

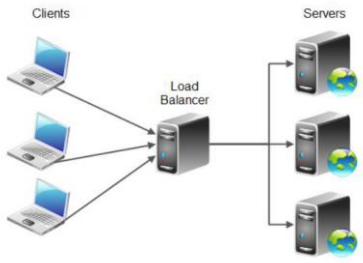


Gambar 2.1 : Arsitektur pada MySQL Cluster

Ada 3 jenis node cluster utama dalam komponen MySQL Cluster antara lain :

1. *Management Node*, peran jenis *node* ini adalah untuk mengelola *node* lain dalam *Cluster NDB*, melakukan fungsi seperti menyediakan data konfigurasi, mulai *(start)* dan menghentikan *node*, dan menjalankan *backup.* Karena tipe node ini mengelola konfigurasi *node* lain, sebuah node dari tipe ini harus dimulai terlebih dahulu sebelum node lainnya. Sebuah *node MGM* diawali dengan perintah *ndb_mgmd.*
2. *Data node*, jenis *node* ini menyimpan *data cluster* digunakan untuk menyimpan semua data transaksi pada *MySQL cluster* dan data tersebut direplikasi pada *node* ini.
3. *SQL Node*, merupakan *interface* yang digunakan oleh *client* untuk mengakses data atau supaya terkoneksi kedalam database yang terdapat pada *node* sistem *cluster.*

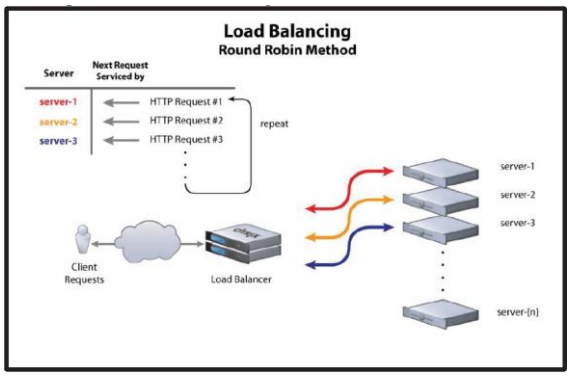
B. *Load Balancing*



Gambar 2.2 : Gambaran *Load Balancing* [3]

Load balancing adalah teknik jaringan komputer yang menggunakan metode pendistribusian terhadap beban trafik untuk membagi beban dalam beberapa jalur koneksi atau *link* secara seimbang. Tujuannya agar tidak ada *link* atau koneksi yang mendapatkan beban yang lebih besar dari *link* atau koneksi lainnya. Dengan membagi beban atau *load* kedalam beberapa *link* tersebut diharapkan dapat mengelola keseimbangan atau *balancing* untuk pengguna *link* tersebut dan menghindari *overload* pada salah satu jalur koneksi jaringan [2].

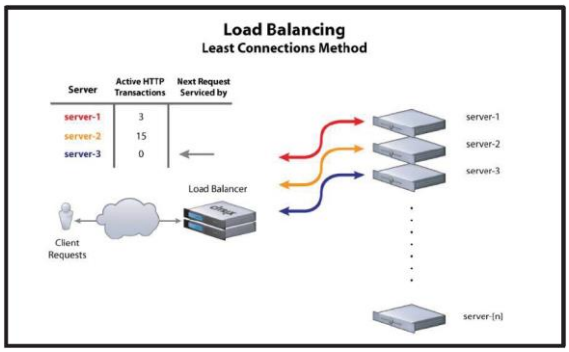
C. *Algoritma Round Robin*



Gambar 2.3 : Cara kerja *Round Robin* [4]

Algoritma *Round Robin* merupakan algoritma yang paling sederhana dan paling banyak digunakan oleh perangkat *load balancing*. Algoritma *Round Robin* bekerja dengan cara membagi beban secara bergiliran dan berurutan dari satu *server* ke *server* lainnya. Konsep dasar dari algoritma *Round Robin* ini adalah dengan menggunakan *time sharing*, pada intinya algoritma ini memproses antrian secara bergiliran[5].

D. *Algoritma Least Connection*



Gambar 2.4 : Cara kerja *Least Connection* [4]

Algoritma *Least Connection* melakukan pembagian beban berdasarkan banyaknya koneksi yang sedang dilayani oleh sebuah *server. Server* dengan koneksi yang paling sedikit akan diberikan beban berikutnya, begitu pula *server* dengan koneksi banyak akan dialihkan bebannya ke *server* lain yang bebannya lebih rendah[4].

E. *SysBench*

Sysbench benchmark tool adalah perangkat lunak yang untuk mengukur performansi sebuah sistem operasi. *Sysbench* adalah sebuah *modular*, *cross platform*, *multi-thread tool benchmark* yang digunakan untuk mengevaluasi parameter sistem operasi yang penting pada sistem database yang berjalan dibawah beban intensif [5]. Salah satu pengukuran database tersebut adalah pengukuran dengan mode *Online Transaction Processing* (OLTP) yaitu suatu sistem yang melakukan suatu transaksi melalui komputer yang terhubung dalam jaringannya [6].

F. *Parameter Pengukuran Sysbench*

1. *Threads*

Threads adalah suatu proses *computing* yang terdiri dari suatu data dan instruksi yang dijalankan secara paralel dan terpisah yang akan mempengaruhi kemampuan *server* dalam mengolah data. *Server* akan menerima beban kerja yang dihasilkan oleh instruksi-instruksi tersebut dan akan berubah tergantung modus uji yang telah digunakan [6].

2. *Transaction per Second*

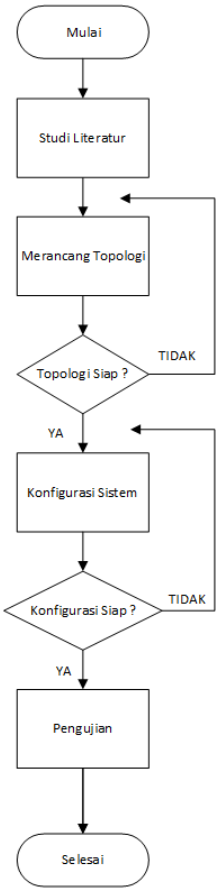
Transaction per Second (TPS) merupakan kemampuan *database* dalam melayani transaksi yang dibutuhkan *client* dalam hitungan detik. Transaksi adalah suatu proses *read* dan *write* yang dilakukan oleh *client* dalam suatu *database*. Proses ini meliputi perintah dan bahasa SQL yang dapat menyebabkan nilai dari tabel atau *database* mengalami perubahan [7].

3. *Response time*

Response time juga dijadikan sebagai salah satu parameter pengukuran OLTP pada *SysBench*. *Response time* dapat didefinisikan sebagai waktu yang dibutuhkan oleh sistem ketika *client* atau pengguna layanan mengirim permintaan menuju *server*. Parameter ini tidak dapat ditentukan secara pasti kecuali melalui pengukuran karena dipengaruhi oleh beberapa faktor antara lain kecepatan prosesor, jumlah memori serta interkoneksi sistem [8].

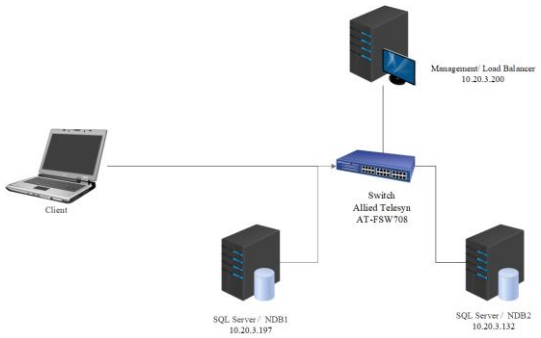
III. METODOLOGI PENELITIAN

A. *Diagram Alir Analisis*



Gambar 3.1 : Diagram Alir Penelitian

B. *Desain Topologi*



Gambar 3.2 : Desain Topologi Sistem

Pada gambar 3.2 merupakan desain topologi yang digunakan dalam penelitian ini yang dimana dari topologi tersebut akan dikonfigurasi sebagai sistem MySQL Cluster dan Sistem *Load Balancing*

Pada tahapan kebutuhan sistem ini dibutuhkan 1 buah komputer sebagai *client* dan 3 buah komputer lainnya digunakan sebagai *server*. Pada 3 buah *server*

sendiri mempunyai fungsi yang berbeda dengan fungsi *server* pertama digunakan sebagai *Management-Node* untuk menghubungkan antara *server NDB-Node* dan *server SQL-Node.* 2 *server* lainnya digunakan sebagai server *NDB-Node* dan server *SQL-Node.*

Pada *server* NDB-Node dan SQL-Node perlu dihubungkan agar terjadi replikasi database antara kedua *server* dan juga berfungsi untuk mengatasi kegagalan sistem database pada salah satu sisi *server* agar *server* yang lain bisa menggantikan tugas *server* yang lainnya.

Tabel 1. Spesifikasi Perangkat Keras

Sistem	Prosesor	Sistem Operasi	Konektivitas
Server	Intel ® Core i3 Dual-Core 7100 @3.90 GHz	Ubuntu Server 16.04 LTS i386	Allied Telesyn AT-FSW708 Switch
Client	AMD ® Ryzen 5 Quad-Core 2500U @2.0 GHz	Ubuntu Server 16.04 LTS i386	Allied Telesyn AT-FSW708 Switch

Tabel 2. IP Pada Setiap Perangkat

Perangkat Keras	Alamat IP	Keterangan
Server	enp1s0 : 10.20.3.200 enp1s0 :1 : 10.20.3.17	*Management-Node/Load Balancer*
Server	enp1s0 : 10.20.3.197	*SQL(API) Node/NDB Node*
Server	enp1s0 : 10.20.3.132	*SQL(API) Node/NDB Node*
PC Dekstop	enp1s0 : 10.20.3.201	*Client*

C. Konfigurasi MySQL Cluster

Dalam mengkonfigurasi MySQL Cluster pada topologi yang sudah disiapkan adalah dengan sebelumnya mengunduh terlebih dahulu paket MySQL Cluster secara manual pada websitenya dan bukan menginstall menggunakan paket *repository* yang ada di linux. Dalam hal ini paket MySQL tersebut harus ada pada setiap *server* yang kan digunakan sebagi sistem *cluster*.

Tahapan selanjutnya adalah paket MySQL Cluster yang sudah di unduh dalam bentuk ekstensi *tar.gz* di ekstraksi terlebih dahulu maka akan dihasilkan sebuah *file* dengan nama mysql-cluster-gpl-XXX-linux-VERSI-OS, dari file tersebut kemudian disalin ke direktori yang ditentukan dilanjutkan dengan membuat *file* dengan nama *config.ini* dalam file tersebut adalah tempat mengkonfigurasi agar pada setiap nodenya bisa saling terhubung satu sama lain meliputi *Management node, NDB Node dan SQL(API) Node.*

```
[ndbd default]
NoOfReplicas=2
DataMemory=80M
IndexMemory=18M

[mysqld default]

[ndb_mgmd default]

[tcp default]

# Management node
[ndb_mgmd]
NodeId=1
HostName=10.20.3.200
DataDir=/var/lib/mysql-cluster

# Data Node
[ndbd]
NodeId=2
HostName=10.20.3.197
DataDir=/usr/local/mysql/data
[ndbd]
NodeId=3
HostName=10.20.3.132
DataDir=/usr/local/mysql/data

# SQL Node
[mysqld]
NodeId=4
HostName=10.20.3.197
[mysqld]
NodeId=5
HostName=10.20.3.132
```

Gambar 3.3 : Konfigurasi *Management Node*

Konfigurasi pada *Management Node* bertujuan untuk informasi pada *server Management* untuk mengidentifikasi fungsi dari node yang akan dihubungkan melalui alamat IP masing-masing *server*.

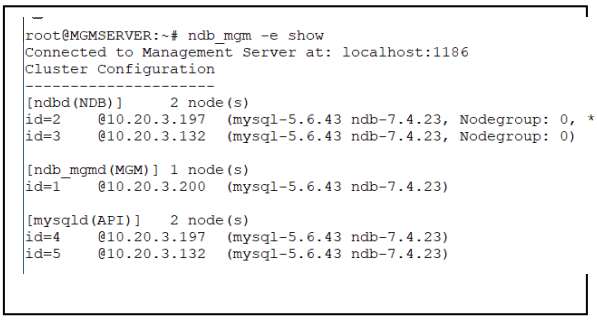
```
# MySQL Config
[mysqld]
ndbcluster
socket=/tmp/mysql.sock
ndb-connectstring=10.20.3.200
datadir=/usr/local/mysql/data
default_storage_engine=ndbcluster


[mysql_cluster]
ndb-connectstring=10.20.3.200

# MySQL Pid and Log
[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

Gambar 3.4 : Konfigurasi NDB Node & SQL Node

Konfigurasi pada sisi server NDB Node dan SQL Node bertujuan untuk memberi informasi kepada *Management Server* bahwa kedua node ini adalah bagian dari sistem MySQL Cluster agar setiap node dapat saling terhubung dengan konfigurasi yang sebelumnya di atur pada sisi *Management Server*.

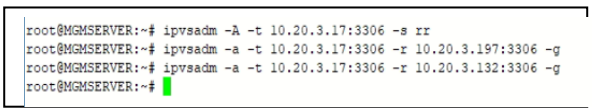


Gambar 3.5 : Konfigurasi Cluster yang sudah terkoneksi

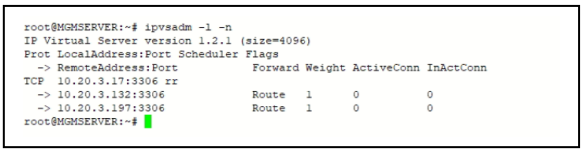
Pada Gambar 3.5 merupakan tampilan pada *Management Node* MySQL Cluster yang sudah terkoneksi dengan baik, ditandai dengan nama "id=X" dan "@10.20.3.x" yang menandakan alamat IP tersebut sudah terkoneksi ke sistem *cluster*. Dari gambar 3.4 juga terlihat *node* dengan id=1 digunakan sebagai *Management Nod*e dengan alamat IP 10.20.3.200 dan node dengan id=2-3 sebagai NDB Node dengan alamat IP 10.20.3.197, 10.20.132, sedangkan pada node dengan id=4-5 digunakan sebagai interface MySQL atau *SQL(API) Node* dengan alamat IP 10.20.3.197, 10.20.132.

D. Konfigurasi IPVSadm

Setelah mekonfigurasi semua *node cluster* selanjutnya dilakukan dengan konfigurasi IPVSadm, konfigurasi ini dumaksud agar *load balancing* dapat berjalan pada topologi yang digunakan.



Gambar 3.6 : Konfigurasi *Load Balancing*



Gambar 3.7 : Tabel layanan *Load Balancer*

E. Pengujian

Pengujian kinerja dilakukan menggunakan aplikasi SysBench. Aplikasi SysBench akan membuat suatu *request* secara simultan yang diberikan kepada sistem *load balancing* dengan menggunakan *thread yang* ditentukan.

SysBench memiliki dua proses utama dalam pengujian.Proses pertama adalah mempersiapkan isi *database* buatan. Persiapan *database* menggunakan perintah berikut :

```
Sysbench --test=oltp --oltp-table-
size=100000 --mysql-host=10.20.3.17
--db-driver=mysql --mysql-db=sbtest
--mysql-port=3306 --mysql-
user=sysbench --mysql-password=1234
--mysql-table-engine=ndbcluster
prepare
```

Proses kedua adalah dengan mengeksekusi tabel yang telah dibuat di dalam *database* yang menggunakan perintah berikut :

```
sysbench --test=oltp --num-
threads=10 --max-time=60 --max-
requests=0 --oltp-test-mode=simple
--oltp-read-only=on --oltp-table-
size=100000 --mysql-host=10.20.3.17
--db-driver=mysql --mysql-db=sbtest
--mysql-port=3306 --mysql-
user=sysbench --mysql-password=1234
--mysql-table-engine=ndbcluster run
```

IV. *ANALISA HASIL PENGUJIAN*

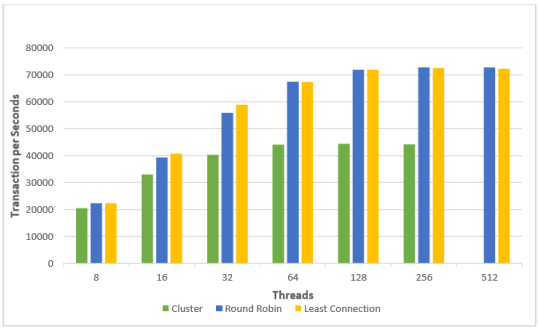
Setelah melakukan pengujian maka akan diperoleh nilai *Transaction per Seconds* dan *Response Time* dari topologi yang diterapkan dan akan dilakukan perbandingan dari peningkatan jumlah *thread* yang diberikan. Selain itu juga akan dilakukan perbandingan antara MySQL Cluster default dan MySQL Cluster *load balancing*, serta juga akan dilakukan perbandingan dari 2 jenis penjadwalan *load balancing* algoritma *Round Robin* dan *Least Connection.* Kemudian dari data pengujian setiap node *simple* dan *complex* yang diperoleh akan diambil rata-rata dan akan dilakukan perbandingan dengan setiap skenario yang sudah dilakukan.

A. Skenario Simple-Mode

Pada mode ini akan ditampilkan data *Transaction per Second* dan *Response Time* dari hasil transaksi *read-only* yang selengkapnya dapat dilihat pada Tabel 3. Dari tabel tersebut menununjukan penurunan nilai TPS mulai dari 256 *thread* pada *cluster default* dan pada 512 *thread server* sudah tidak mampu memproses transaksi data.

Tabel 3. Rata – Rata pengujian MySQL Cluster *simple-mode*

THREADS	CLUSTER		ROUND ROBIN		LEAST CONNECTION	
	TPS	RESPONSE TIME (ms)	TPS	RESPONSE TIME (ms)	TPS	RESPONSE TIME (ms)
8	20493,68	0,63	22342,32	0,54	22375,67	0,54
16	33081,34	0,80	39292,29	0,60	40689,54	0,59
32	40303,66	2,12	55930,21	0,82	58875,30	0,82
64	44109,25	3,05	67457,23	2,00	67369,52	2,03
128	44384,87	5,78	71896,48	3,26	71.897,12	3,26
256	44212,57	11,08	72771,83	6,53	72518,19	6,53
512	-	-	72739,69	13,30	72244,14	13,48

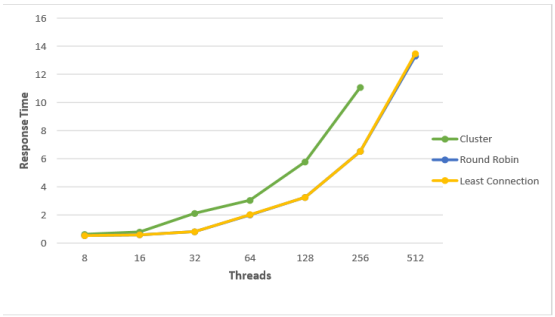


Gambar 4.1 : Grafik TPS *Simple-Mode*

Gambar 4.1 menunjukkan perbandingan 3 skenario pengujian dari pengaruh penambahan *threads* terhadap jumlah TPS (*Transaction per Seconds*). Pada parameter ini untuk nilai TPS yang baik akan diperoleh dengan jumlah yang lebih tinggi. Pada Gambar 4.1 menunjukkan jumlah TPS tertinggi berjumlah 72771,83 *transaction per seconds* pada beban 256 *threads*. Pada MySQL Cluster *default* jumlah TPS pada beban 8 hingga128 *threads* meningkat mengikuti beban *threads* yang diberikan, sebelum pada beban 256 *threads* jumlah TPS mengalami penurunan dan pada beban 512 *threads* tidak ada nilai TPS karena pada saat pengujian pada beban tersebut *server* sudah tidak mampu melayani transaksi data dengan kata lain *server overload*. Hal ini terjadi karena proses pengujian *database*, transaksi hanya terjadi pada 1 *node* sedangkan *node* yang lain berfungsi sebagai *backup* data. Berbeda pada MySQL Cluster *load balancing* jumlah TPS dari beban 8 hingga 256 *threads* meningkat mengikuti penambahan beban *threads* yang diberikan, pada beban 512 *threads server* tetap mampu memproses transaksi data dan tidak terjadi kegagalan atau *server overload* seperti pada MySQL Cluster *default* sebelumnya meskipun hanya terjadi penurunan jumlah TPS saja. Hal ini terjadi karena pada MySQL Cluster *load balancing* proses transaksi data terjadi pada jumlah *node* yang dijadikan *cluster* sehingga akan berbagi beban kerja.

Menurunan jumlah TPS pada masing-masing kondisi MySQL Cluster menandakan bahwa pada beban *threads* tertentu yang mengalami penurunan merupakan titik jenuh dari MySQL Cluster tersebut yang dipengaruhi oleh beban yang diberikan. Sedangkan untuk MySQL Cluster *load balancing* dengan algoritma masing-masing yang berbeda nilai TPS saat beban 32 *threads* algoritma *Least Connection* lebih unggul dari *Round Robin* kemudian keduanya menjadi seimbang dan saat beban 512 *threads* algoritma *Round Robin* lebih unggul sedikit dari *Least Connection*. Namun untuk kondisi MySQL Cluster sebelum dan sesudah ditambahkan *load balancer*, nilai TPS MySQL Cluster *load balancing* lebih baik dibandingkan MySQL Custer *default* dan perbandingannya sangat besar yang menandakan bahwa penambahan *load balancer* pada MySQL Cluster berpengaruh pada kinerja *server* yang lebih baik.

Parameter lain yang berpengaruh pada pengujian ini adalah *response time*. Hasil yang diperoleh dari tabel dijelaskan pada Gambar 4.2.



Gambar 4.2 : Grafik *Response Time Simple-Mode*

Gambar 4.2 menunjukkan perbandingan 3 skenario pengujian dari pengaruh penambahan *threads* terhadap nilai *response time* yang merupakan waktu yang dibutuhkan *server* dalam menyelesaikan suatu transaksi data. Pada parameter ini untuk nilai *response time* yang baik akan diperoleh dengan jumlah yang lebih kecil. Pada Gambar 4.2 menunjukkan nilai *response time* terbaik dengan nilai 0,54 ms pada beban 8 *threads*. Pada MySQL Cluster *default* nilai pada beban 512 *threads* tidak ada nilai TPS karena pada saat pengujian pada beban tersebut *server* sudah tidak mampu melayani transaksi data dengan kata lain *server overload*. Hal ini terjadi karena proses pengujian *database*, transaksi hanya terjadi pada 1 *node* sedangkan *node* yang lain berfungsi sebagai *backup* data. Berbeda pada MySQL Cluster *load balancing* pada beban 512 *threads server* tetap mampu memproses transaksi data dan tidak terjadi kegagalan atau *server overload* seperti pada MySQL Cluster *default*.

Semua nilai *response time* MySQL Cluster dengan *load balancing* memiliki nilai yang lebih baik dibandingkan nilai *response time* MySQL Cluster *default*. Hal ini terjadi karena pada MySQL Cluster *load balancing* proses transaksi data terjadi pada jumlah *node* yang dijadikan *cluster* sehingga sistem *cluster* akan berbagi beban kerja dan akan membuat waktu tanggap menjadi lebih cepat. Kemudian untuk titik jenuh pada masing-masing kondisi berada pada beban *threads* yang sama seperti halnya pengujian parameter TPS (*Transaction per Seconds*).

B. Skenario Complex-Mode

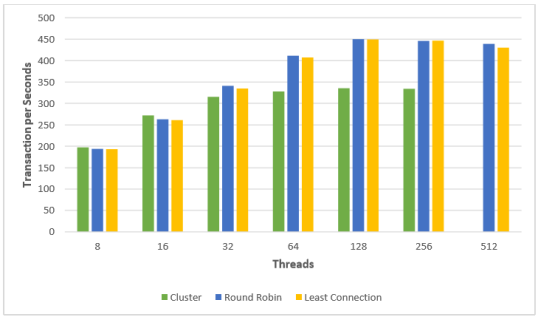
Pada mode ini akan ditampilkan data *Transaction per Second* dan *Response Time* dari skenario *Complex-Mode*. Terdapat beberapa perbedaan dibandingkan dengan skenario *Simple-Mode*. Jumlah *Transaction per Second* pada kondisi *default* mengalami penurunan bahkan pada beban *thread* 512 masih terjadi kegagaln transaksi data. Namun juga mengalami kenaikan pada kondisi 2 algoritma *load balancing*. Jumlah *Transaction per Second* kondisi

default hanya tinggi pada beban *thread* 8-16 dari kondisi *load balancing* selebihnya pada beban *thread* 32-512 kondisi 2 algoritma *load balancing* jumlahnya lebih tinggi dari pada kondisi *default.*

Tabel 4. Rata – Rata pengujian MySQL Cluster *Complex-Mode*

THREADS	CLUSTER		ROUND ROBIN		LEAST CONNECTION	
	TPS	RESPONSE TIME (ms)	TPS	RESPONSE TIME (ms)	TPS	RESPONSE TIME (ms)
8	197,09	52,11	194,02	52,51	192,91	52,59
16	272,15	77,43	262,66	72,29	260,73	77,64
32	315,38	135,01	340,94	122,57	335,05	122,66
64	327,59	260,99	411,13	208,71	407,60	210,60
128	335,31	537,84	450,02	403,58	449,56	399,16
256	334,01	1161,26	445,80	847,92	446,60	854,04
512	-	-	439,04	2033,51	430,20	2103,26

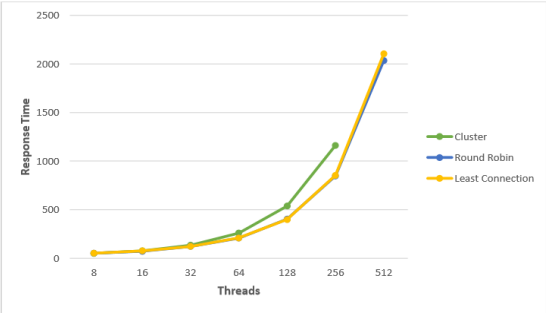
Nilai yang diperoleh pada Tabel 4 menunjukan perbedaan dari Tabel 3. Pada tabel diatas dapat dilihat pada kondisi *default* pada beban *thread* 512 tetap terjadi kegagalan transaksi data, namun pada kondisi 2 algoritma *load balancing* tetap mampu mengatasi beban lebih dari 512 *thread*.



Gambar 4.3 : Grafik TPS *Complex-Mode*

Gambar 4.3 menunjukkan perbandingan 3 skenario pengujian dari pengaruh penambahan *threads* terhadap jumlah TPS (*Transaction per Seconds*). Pada parameter ini untuk nilai TPS yang baik akan diperoleh dengan jumlah yang lebih tinggi. Pada Gambar 4.3 menunjukkan jumlah TPS tertinggi berjumlah 450,02 *transaction per seconds* pada beban 128 *threads*. Pada MySQL Cluster *default* jumlah TPS pada beban 8 hingga128 *threads* meningkat mengikuti beban *threads* yang diberikan, sebelum pada beban 256 *threads* jumlah TPS mengalami penurunan dan pada beban 512 *threads* tidak ada nilai TPS karena pada saat pengujian pada beban tersebut *server* sudah tidak mampu melayani transaksi data dengan kata lain *server overload.* Berbeda dengan MySQL Cluster *load balancing* pada beban 512 *threads server* tetap mampu memproses transaksi data dan tidak terjadi kegagalan atau *server overload.*

Menurunan jumlah TPS pada masing-masing kondisi MySQL Cluster menandakan bahwa pada beban *threads* tertentu yang mengalami penurunan merupakan titik jenuh dari MySQL Cluster tersebut yang dipengaruhi oleh beban yang diberikan. Sedangkan untuk MySQL Cluster *default* nilainya TPS-nya lebih baik dibandingkan MySQL Cluster *load balancing* saat diberikan beban 8 hingga16 *threads* kemudian saat diberikan beban 32 hingga 512 *threads* MySQL Cluster *load balancing* lebih baik dibandingkan MySQL Cluster *default* dengan perbandingan yang besar pada setiap penambahan beban, hal ini juga menandakan bahwa penambahan *load balancer* pada MySQL Cluster berpengaruh pada kinerja *server* yang lebih baik.



Gambar 4.4 : Grafik *Response Time Complex-Mode*

Gambar 4.4 menunjukkan perbandingan 3 skenario pengujian dari pengaruh penambahan *threads* terhadap nilai *response time* yang merupakan waktu yang dibutuhkan *server* dalam menyelesaikan suatu transaksi data. Ketiganya memiliki grafik yang semakin meningkat setiap diberikan tambahan beban *threads.* Namun, pada setiap kenaikannya memiliki perbedaan perbandingan masing-masing. Pada Gambar 4.4 menunjukkan nilai *response time* terbaik dengan nilai 52,11 ms pada beban 8 *threads*.

Nilai *response time* MySQL Cluster *default* lebih baik dibandingkan MySQL Cluster *load balancing* saat diberikan beban 8 *threads* namun selanjutnya ketika diberikan beban 16 hingga 512, MySQL Cluster *load balancing* memiliki nilai yang lebih baik dibandingkan nilai *response time* MySQL Cluster *default*. Hal ini terjadi karena pada MySQL Cluster *load balancing* jumlah *node* yang dijadikan *cluster* saling berbagi beban kerja sehingga waktu tanggap atau *respon time* dan dilakukan lebih cepat. Bahkan sama seperti sebelumnya pada MySQL Cluster *default* nilai pada beban 512 *threads* tidak ada nilai *response time* karena pada saat pengujian pada beban tersebut *server* sudah tidak mampu melayani transaksi data dengan kata lain *server overload*.

V. *PENUTUP*

A. Kesimpulan

Setelah dilakukan konfigurasi, pengujian dan analisa data dapat disimpulkan beberapa hal yang dapat digunakan sebagai perbaikan dan pengembangan selanjutnya yaitu :

1. Berdasarkan tabel perbandingan TPS pada *simple-mode*, nilai TPS MySQL Cluster dengan *load balancing* memiliki nilai lebih baik yaitu 72771,83 *transaction per seconds* pada beban *threads* 256 sedangkan MySQL Cluster *default* memiliki nilai 44212,57 pada beban *threads* yang sama.
2. Berdasarkan tabel perbandingan *Response Time* pada *simple-mode*, pada saat beban kerja tinggi

waktu tanggap MySQL Cluster dengan *load balancing* memiliki nilai lebih baik yaitu 13,30 ms pada beban *threads* 512 sedangkan MySQL Cluster *default* pada saat beban yang sama tidak mampu melakukan transaksi data pada beban kerja yang tinggi.

3. Berdasarkan tabel perbandingan TPS pada *complex-mode*, nilai TPS MySQL Cluster dengan *load balancing* memiliki nilai lebih baik yaitu 446,60 *transaction per seconds* pada beban *threads* 256 sedangkan MySQL Cluster *default* memiliki nilai 334,01 pada beban *threads* yang sama.

4. Berdasarkan tabel perbandingan *Response Time* pada *complex-mode*, pada saat beban kerja tinggi waktu tanggap MySQL Cluster dengan *load balancing* memiliki nilai lebih baik yaitu 2033,51 ms pada beban *threads* 512 sedangkan MySQL Cluster *default* pada saat beban yang sama tidak mampu melakukan transaksi data pada beban kerja yang tinggi.

5. Jumlah TPS (*Transaction per Second*) pada *complex-mode* memiliki nilai yang lebih kecil daripada *simple-mode* dikarenakan pada *complex-mode* melibatkan *query* yang kompleks yaitu SELECT, UPDATE, DELETE dan INSERT sedangkan pada saat *simple-mode* hanya melibatkan *query* SELECT. Begitu juga nilai *Response Time* yang memiliki nilai waktu tanggap yang besar karena pengaruh hal yang sama.

6. Berdasarkan hasil pengujian dan tabel data, menunjukkan bahwa MySQL Cluster *load balancing* dapat meningkatkan kinerja *server* dibuktikan dengan nilai hasil pengujian yang mempunyai perbandingan yang besar saat beban kerja *threads* yang besar.

7. Dengan *load balancing* dapat mengatasi ketidakmampuan *server* saat menerima beban kerja yang besar, dibuktikan dengan MySQL Cluster *load balancing* dapat melakukan transaksi data pada beban 512 *threads* sedangkan MySQL Cluster *default* tidak mampu karena *server overload*.

B. Saran

Pada penelitian ini tidak lepas dari berbagai macam kekurangan dan kesalahan baik dari perancangan sistem maupun pengujian yang penulis buat, maka dari itu agar sistem dapat menjadi lebih baik maka dapat dikembangakan lebih sempurna, saran dari penulis antara lain sebagai berikut :

1. Perlu dilakukannya penambahan jumlah *server* di setiap *node*-nya agar dapat lebih jelas hasil penelitian yang dilakukan pada MySQL Server.

2. Dapat menggunakan algoritma penjadwalan *load balancing* yang lebih banyak untuk melihat perbandingan pada penggunaan algoritma penjadwalan *load balancing* tersebut.

DAFTAR PUSTAKA

[1] Rijayana, I. (2005). Teknologi Load Balancing Untuk Mengatasi Beban Server. *Seminar* Nasional Aplikasi Teknologi Informasi 2005 (SNATI 2005). ISBN: 979-756-061-6

[2] Lukitasari, D. (2010). Server Load Balancing. JURNAL GENERIC. Vol.5:2

[3] Towidjojo,Rendra. (2013). MikroTik kung Fu Kitab 1. Yogyakarta. Penerbit: Jasakom.

[4] Ellrod, C. (2010). *Load Balancing – Round Robin*. http://blogs.citrix.com/2010/09/03/load-balancing-round-robin/

[5] Kopytov, Alexey. (2009). Scriptable database and system performance benchmark, https://github.com/akopytov/sysbench.

[6] Jadhav, S. S. (2009). Advanced Computer Architecture and Computing. Technical Publications Pune, India.

[7] Coronel, Carlos, Steven Morris, and Peter Rob. (2012) Database Systems: Design, Implementation, and Management, 10th Edition. Cengage Learning, Stamford.

[8] Inmon, William H. (2005). Building the Data Warehouse, Wiley Publishing, Indianapolis.

VI. BIODATA PENULIS

Yosep Dwi Irawan, lahir di Banyuwangi 3 September 1997. Menempuh pendidikan dasar di SDN 1 Karangsari pada tahun 2009 dilanjutkan dengan pendidikan tingkat menengah di SMPN 1 Sempu pada 2012 dan pendidikan tingkat atas di SMKN 1 Glagah Banyuwangi lulus pada tahun 2015. Penulis memulai pendidikan di Institut Teknologi Nasional Malang pada tahun 2015 di Jurusan Elektro Fakultas Teknik Industri peminatan Teknik Komputer. Penulis aktif menjadi asisten Laboratorium Jaringan Komputer Teknik Elektro S-1 ITN Malang.

Email: mydwiirawan@gmail.com