TUGAS AKHIR – KI141502

ANALISIS KINERJA MYSQL CLUSTER DAN SQL-TO-NOSQL DATA ADAPTER (SNDA) UNTUK TEMU KEMBALI INFORMASI

MUHAMAD RIZKI PRAWIRAATMAJA
05111340000120

Dosen Pembimbing
Ir. Muchammad Husni, M.Kom.
Radityo Anggoro S.Kom, M.Sc

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

TUGAS AKHIR – KI141502

ANALISIS KINERJA MYSQL CLUSTER DAN SQL-TO-NOSQL DATA ADAPTER (SNDA) UNTUK TEMU KEMBALI INFORMASI

MUHAMAD RIZKI PRAWIRAATMAJA
05111340000120

Dosen Pembimbing I
Ir. Muchammad Husni, M.Kom.

Dosen Pembimbing II
Radityo Anggoro S.Kom, M.Sc

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

[*Halaman ini sengaja dikosongkan*]

FINAL PROJECT - KI141502

PERFORMANCE ANALYSIS OF MYSQL CLUSTER AND SQL-TO-NOSQL DATA ADAPTER (SNDA) FOR INFORMATION RETRIEVAL

MUHAMAD RIZKI PRAWIRAATMAJA
05111340000120

Supervisor I
Ir. Muchammad Husni, M.Kom.

Supervisor II
Radityo Anggoro S.Kom, M.Sc

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
Sepuluh Nopember Institute of Technology
Surabaya, 2018

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

LEMBAR PENGESAHAN

ANALISIS KINERJA MYSQL CLUSTER DAN SQL-TO-NOSQL DATA ADAPTER (SNDA) UNTUK TEMU KEMBALI INFORMASI

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
MUHAMAD RIZKI PRAWIRAATMAJA
NRP: 05111340000120

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Ir. Muchammad Husni, M. Kom
NIP. 196002211984031001 ..... (Pembimbing 1)

Radityo Anggoro S.Kom, M.Sc
NIP. 198410162008121002 ..... (Pembimbing 2)

SURABAYA
JULI, 2018

[*Halaman ini sengaja dikosongkan*]

ANALISIS KINERJA MYSQL CLUSTER DAN SQL-TO-NOSQL DATA ADAPTER (SNDA) UNTUK TEMU KEMBALI INFORMASI

Nama Mahasiswa : Muhamad Rizki Prawiraatmaja
NRP : 05111340000120
Jurusan : Teknik Informatika, FTIK ITS
Dosen Pembimbing 1 : Ir. Muchammad Husni, M. Kom.
Dosen Pembimbing 2 : Radityo Anggoro S.Kom, M.Sc

Abstrak

Perkembangan basis data dewasa ini semakin berkembang seiring dengan bertambahnya kuantitas data yang ada di jaringan internet saat ini. Basis data relasional (RDB) merupakan salah satu jenis basis data yang paling banyak digunakan saat ini. Tetapi, RDB masih dianggap belum mencukupi untuk menangani data yang ada saat ini, terutama saat menghadapi data yang besar atau yang sekarang dikenal sebagai big data. Sebagai alternatif, terdapat jenis basis data NoSQL yang dapat memproses data tersebut dalam kecepatan yang lebih tinggi. Namun begitu, NoSQL masih memiliki keterbatasan terutama dalam pengambilan kembali data tersebut, sehingga dibuatlah sebuah adapter yang disebut sebagai SQL-to-NoSQL Adapter (SNDA).

SNDA ini sendiri merupakan sebuah hasil riset kerja dari rekanan penulis sebagai bahan Tugas akhir dari rekanan. Proses kerja dari SNDA itu sendiri adalah dengan membuat sebuah sinkronisasi data antara RDB dengan NoSQL sehingga data akan berada di kedua basis data tersebut dan data dapat diambil kembali lebih cepat. Meskipun begitu, teknologi SNDA ini masih belum diuji lebih jauh, sehingga diperlukan sebuah perbandingan unjuk kerja dengan basis data yang mampu menangani big data

serta masih banyak digunakan saat ini. Pilihan basis data yang diambil untuk perbandingan unjuk kerja adalah MySQL Cluster yang masih banyak digunakan saati ini dan dapat disebut sebagai syarat minimal kelayakan basis data untuk pemrosesan big data.

Dari pengujian unjuk kerja dari kedua teknologi ini, diharapkan dapat menguji lebih jauh kelayakan SNDA itu sendiri mulai dari kecepatan proses query data, besar sumber daya yang digunakan, hingga kehandalan SNDA itu sendiri jika dibandingkan dengan MySQL Cluster.

Kata kunci: Basis data, MySQL Cluster, Data Adapter

PERFORMANCE ANALYSIS OF MYSQL CLUSTER AND SQL-TO-NOSQL DATA ADAPTER (SNDA) FOR INFORMATION RETRIEVAL

Student Name	: Muhamad Rizki Prawiraatmaja
Registration Number	: 0511134000120
Department	: Informatics Engineering, FTIK ITS
First Supervisor	: Ir. Muchammad Husni, M. Kom.
Second Supervisor	: Radityo Anggoro S.Kom, M.Sc

Abstract

The Development of database technology as of late grown as with the increase of quantity of data that exists in the present internet network. Currently relational database (RDB) are one of the most used database type. Even then, RDB still can't be considered enough to handle most data that exists right now, especially when handing data with enormous size such as big data. As an alternative, there exists another database type known as NoSQL that could process those kind data with higher speed. Even then, NoSQL still had lots of limitations especially in terms of data retrieval. And thus created an adapter known as SQL-to-NoSQL Adapter.

SNDA itself is a result of work from the writer's colleague as a topic of his final project. How SNDA works itself is by creating a data synchronization between RDB and NoSQL in order to make those data exists in those two database and makes data could be retrieved quicker. Despite the benefits, this SNDA technology itself is still not tested further, so there's a need to make a comparative performance with current database technology that can handle big data and still used widely today. Database chosen as comparison for is MySQL Cluster since it is still widely used and could be considered as standard minimum to big data processing.

It is to be expected from the performance test of these two technologies, it would be able to test further SNDA itself from it's query processing speed, resources needed, and also whether it could be comparable or even better if it's compared to MySQL Cluster.
Keyword(s): Database, MySQL Cluster, Data adapter

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Segala puji bagi Allah SWT yang telah melimpahkan rahmat dan anugerah-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul **"Analisis Perbandingan Kinerja MySQL Cluster dan SQL-to-NoSQL Data Adapter (SNDA) untuk Temu Kembali Informasi pada Big Data"**.

Buku tugas akhir ini disusun dengan harapan dapat memberikan pendalaman lebih jauh terhadap teknologi SNDA yang baru saja dikembangkan ini. Selain itu, penulis berharap dapat memberikan kontribusi positif bagi kampus Teknik Informatika ITS.

Dalam perancangan, pengerjaan, dan penyusunan tugas akhir ini, penulis banyak mendapatkan bantuan dari berbagai pihak. Penulis ingin mengucapkan terima kasih kepada:

1. Bapak Ir. Muchammad Husni, M. Kom. dan Bapak Radityo Anggoro S.Kom, M.Sc selaku dosen pembimbing penulis yang telah memberi ide, nasihat dan arahan sehingga penulis dapat menyelesaikan tugas akhir dengan tepat waktu.
2. Orang tua penulis Boni Benyamin dan Catarina Indah beserta kakak tersayang Anisa Wulandari yang telah memberikan dukungan moral, spiritual dan material serta senantiasa memberikan doa demi kelancaran dan kemudahan penulis dalam mengerjakan tugas akhir.
3. Teman-teman seperti Agung, Dimas Rahman, Rezky, dan mahasiswa angkatan 2013 lain yang saya banggakan.
4. Pihak-pihak lain yang tidak bisa penulis sebutkan satu-persatu.

Penulis menyadari masih ada kekurangan dalam penyusunan tugas akhir ini. Penulis mohon maaf atas kesalahan, kelalaian maupun kekurangan dalam penyusunan tugas akhir ini. Kritik dan

saran yang membangun dapat disampaikan sebagai bahan perbaikan ke depan.

Surabaya, Juli 2018

Penulis

DAFTAR ISI

DAFTAR GAMBAR

[*Halaman ini sengaja dikosongkan*]

DAFTAR TABEL

[*Halaman ini sengaja dikosongkan*]

DAFTAR KODE SUMBER

[*Halaman ini sengaja dikosongkan*]

BAB I
PENDAHULUAN

Pada bab ini dibahas hal-hal yang mendasari tugas akhir. Bahasan meliputi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika laporan tugas akhir.

1.1 Latar Belakang

Dewasa ini jumlah data yang dihasilkan dari tahun ke tahun terus mengalami peningkatan. Laporan terakhir menunjukkan, sebanyak 90% data yang ada di dunia dihasilkan hanya dalam waktu dua tahun saja [1]. Seluruh data tersebut diproses secara simultan dalam jumlah yang besar dan berlangsung secara cepat hingga *real-time*. Untuk itu, diperlukan sebuah basis data yang mampu mendukung kecepatan yang tinggi dan data-data yang besar tersebut.

Salah satu solusi yang sudah ada adalah MySQL *Cluster*. Dengan MySQL *Cluster*, basis data dapat memberikan ketersediaan yang tinggi, skalabilitas perangkat, serta peningkatan kinerja dikarenakan MySQL *Cluster* dirancang untuk berjalan pada lebih dari satu komputer dan prosesnya berjalan secara paralel [2], sehingga sesuai dengan kebutuhan *big data* tersebut. Tetapi, MySQL *Cluster* masih memiliki keterbatasan kecepatan dari SQL itu sendiri, sehingga masih diperlukan basis data yang lebih cepat dalam memproses *query* data dalam jumlah besar, seperti NoSQL. SQL-to-NoSQL Data Adapter (SNDA) merupakan salah satu solusi yang akan diuji sebagai alternatif dari MySQL *Cluster*, dimana SNDA memanfaatkan *query* SQL untuk diproses lebih lanjut di NoSQL untuk mendapatkan peningkatkan kecepatan [3].

SNDA sendiri masih sangat baru dan merupakan hasil dari Penelitian tahun 2017, sehingga masih banyak hal dari teknologi ini yang belum diuji lebih jauh, baik itu ketersediaan, skalabilitas, hingga kinerjanya itu sendiri. Untuk itu, tugas akhir ini dilakukan untuk mengukur lebih jauh kemampuan dari SNDA. Diharapkan,

solusi SNDA ini dapat menjadi alternatif terbaru dari metode penyimpanan basis data yang memanfaatkan keteraturan data dari SQL dan kecepatan pemrosesan dari NoSQL. Data yang digunakan adalah data kependudukan SIAK Magetan yang telah diseleksi untuk menyembunyikan bagian yang penting.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana fungsionalitas SNDA bekerja ketika menggunakan dataset yang berbeda?
2. Bagaimana hasil kinerja dari MySQL *Cluster* dan SNDA?

1.3 Batasan Masalah

Tugas Akhir ini memiliki beberapa batasan masalah antara lain:

1. Penelitian akan menggunakan 3 server untuk MySQL *Cluster* dan basis data berbasis SNDA.
2. Sistem operasi yang digunakan adalah Ubuntu Server 14.04 LTS.
3. Basis data berbasis SNDA terdiri atas Hadoop + HBase, Apache Phoenix, MySQL *Server*, serta *script* dari SNDA itu sendiri.
4. MySQL *Cluster* yang digunakan adalah versi 7.4.
5. Alat *benchmarking* yang digunakan adalah mysqlslap, Postman, dan JMeter.
6. Hal-hal yang menjadi tolak ukur uji dari tugas akhir ini adalah *query speed*, jumlah klien yang dapat ditampung, serta beban pada jaringan.

1.4 Tujuan Tugas Akhir

Tujuan dari pembuatan Tugas Akhir ini adalah mengetahui bagaimana kinerja dari SNDA dan MySQL *Cluster* yang merupakan salah satu pilihan basis data terdistribusi saat ini.

1.5 Manfaat Tugas Akhir

Tugas Akhir ini diharapkan dapat memberi pemahaman lebih jauh dari SNDA yang masih baru saja dibuat. Selain itu, diharapkan analisa ini dapat menambah studi ilmiah tentang perancangan basis data baik yang berbasis *clustering* ataupun yang menggunakan SNDA ini.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan tugas akhir ini adalah sebagai berikut:

1. Studi Literatur

   Tugas Akhir ini menggunakan dua literatur *paper* dan satu buku. *Paper* pertama yang digunakan adalah "*Processing Performance on Apache Pig, Apache Hive and MySQL Cluster*". *Paper* tersebut memberikan teknik-teknik yang dapat digunakan untuk melakukan *benchmarking* pada MySQL *Cluster*. *Paper* kedua yang digunakan adalah "*Schema Conversion Model of SQL Database to NoSQL*". *Paper* tersebut menjelaskan konsep serta desain yang digunakan seperti pada SNDA. Dan yang terakhir adalah sebuah buku dengan judul "*MySQL Clustering*" yang disediakan oleh pengembang MySQL sendiri sebagai acuan untuk perancangan MySQL *Cluster*.

2. Desain dan Perancangan Sistem

   Tahap desain dan perancangan merupakan tahap untuk menjelaskan bagaiman sistem SNDA dan MySQL *Cluster* akan dibangun beserta rancangan skenario dari pengujian. Pengujian akan dilakukan dalam tiga jenis skenario, yaitu dalam Pseudo MySQL *Cluster*, MySQL *Cluster* 3 server, serta SNDA 3 server.

3. Implementasi

   Pengerjaan seluruh sistem akan dilakukan pada 3 server yang sama secara bergantian, dengan dibangun diatas sistem operasi Ubuntu 14.04. Basis Data yang digunakan adalah MySQL

Cluster, MySQL server, HBase, beserta adapter atau manager yang digunakan dari kedua jenis sistem. Kakas bantu pendukung lain diantaranya, *Microsoft Office* untuk dokumentasi, dan *Microsoft Paint* sebagai pengolah *screenshoot*.

4. Uji Coba dan Evaluasi

   Dalam tahap ini, dilakukan pengujian parameter-parameter yang dimaksudkan sebelumnya seperti kecepatan *query*, besar sumber daya yang digunakan, hingga efisiensi data. Kakas uji yang digunakan adalah mysqlslap untuk MySQL *Cluster* dan Postman serta JMeter untuk SNDA.

1.7 Sistematika Laporan

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku tugas akhir terdiri atas beberapa bagian seperti berikut:

Bab I Pendahuluan

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan tugas akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan tugas akhir ini.

Bab III Desain dan Perancangan

Bab ini berisi tentang desain dan perancangan sistem SNDA beserta MySQL *Cluster*.

Bab IV Implementasi

Bab ini membahas implementasi dari desain dan skenario yang telah dibuat pada bab sebelumnya. Penjelasan berupa tahapan yang dilakukan dalam pengerjaaan kedua sistem tersebut.

Bab V Uji Coba dan Evaluasi

Bab ini membahas tahap-tahap uji coba yang dilakukan dengan kedua kakas uji yang telah dipilih yaitu mysqlslap, Postman, dan JMeter. Kemudian hasil uji coba dievaluasi untuk kinerja dari aplikasi yang dibangun.

Bab VI Kesimpulan dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan sistem ke depannya.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

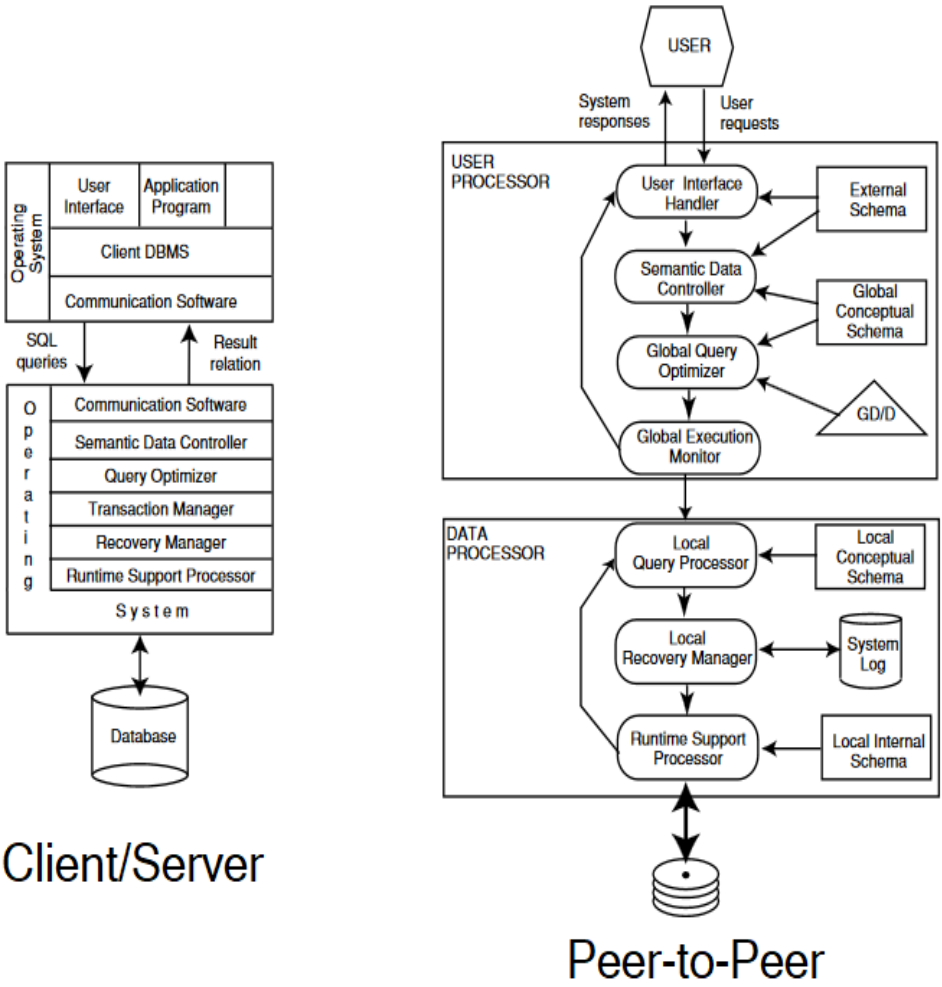
Bab ini menjelaskan mengenai dasar-dasar teori yang digunakan dalam pengerjaan tugas akhir dengan tujuan untuk memberikan gambaran secara umum terhadap sistem yang dibangun. Bab ini berisi penjelasan mengenai apa itu basis data terdistribusi beserta teknologi yang memanfaatkan konsep tersebut, MySQL *Cluster* dan SNDA yang merupakan ruang lingkup dari implementasi yang akan digunakan.

2.1 Basis Data Terdistribusi

Basis data terdistribusi adalah sekumpulan dari sejumlah basis data yang saling berkaitan secara logika yang tersebar lokasinya dan terhubung melalui sebuah jaringan komputer. Sedangkan sebuah sistem manajemen basis data terdistribusi (DBMS) merupakan sebuah sistem perangkat lunak yang memungkinkan manajemen basis data terdistribusi dapat dilakukan dan membuat distribusinya dapat dilihat secara langsung oleh penggunanya [4].

Basis data terdistribusi memiliki banyak jenis arsitektur mulai dari sistem *client/server* hingga terdistribusi sepenuhnya dalam sistem *peer-to-peer*, tetapi pada umumnya sebuah basis data terdistribusi memiliki dua buah antarmuka yang bertugas menghubungkan antar basis data yang ada serta berinteraksi dengan sistem operasi yang ditumpanginya. Untuk sistem *client/server*, pihak server adalah yang melakukan paling banyak proses dimana server menangani seluruh proses *query*, optimisasi, manajemen transaksi, hingga penyimpanan datanya. Sedangkan client adalah pihak yang bertanggung jawab di sisi aplikasi, antarmuka, serta melakukan proses *caching* data yang kemudian akan dimanajemen datanya oleh pengguna. Di lain sisi, sistem *peer-to-peer* tidak membedakan pihak *client* ataupun *server* dan bertanggung jawab atas proses hingga penyimpanan masing-

masing bagian data yang berada di basis data tersebut. Perbedaan masing-masing sistem dapat dilihat pada Gambar 2.1.



Gambar 2.1 Perbedaan sistem DBMS client/server dengan peer-to-peer

Untuk pemrosesan *query* data dan optimasinya, semuanya tetap dilakukan dengan menggunakan bahasa SQL yang memang sudah umum digunakan pada basis data. Optimasi pada basis data terdistribusi selain pada proses *query* lokal juga dilakukan untuk melakukan lokalisasi data serta *global query* sehingga setiap basis data dapat terhubung satu sama lain.

2.2 MySQL *Cluster*

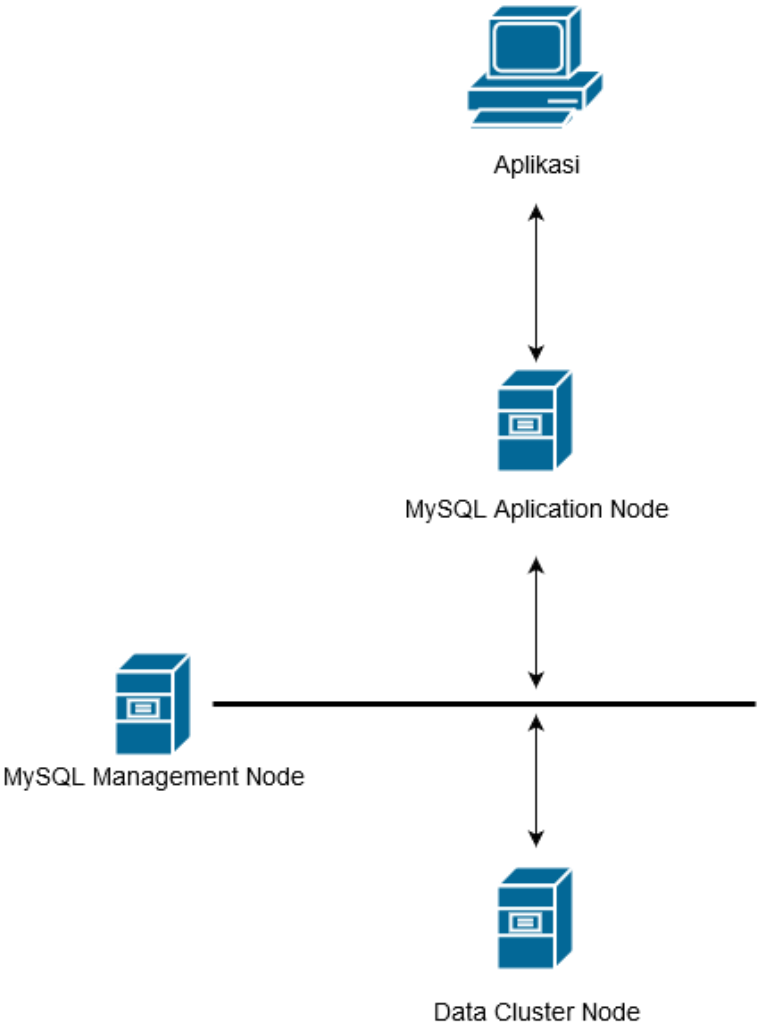
MySQL *Cluster* adalah sebuah teknologi yang digunakan oleh sebuah server MySQL untuk menyediakan fitur *sharding* serta *clustering* pada sebuah lingkungan basis data. Secara teknis, MySQL *Cluster* menjanjikan ketersediaan data sebesar 99.999%. Secara desain, MySQL *Cluster* dirancang untuk arsitektur *node* yang terdistribusi tanpa ada sama sekali kerusakan pada data. Sebuah basis data MySQL yang menggunakan MySQL *Cluster* terdiri atas sejumlah *node* yang tersebar pada berbagai perangkat untuk menjamin sistem dapat bekerja, meskipun terdapat sebuah *node* yang mengalami suatu masalah seperti saat terjadi kegagalan jaringan [5].

Desain dari MySQL *Cluster* yang dibuat untuk berjalan secara *real-time* memungkinkan basis data menjalankan jutaan operasi data dalam waktu milidetik. Ditambah dengan dukungan penyimpanan data baik dalam HDD maupun memori, *sharding* data secara otomatis, serta dapat menambah node dalam sekejap membuatnya dapat digunakan pada proses *big data.*

Meskipun begitu, kecepatan tersebut hanya bisa diraih dengan syarat kompleksitas *query* yang digunakan tidak terlalu rumit. Jika *query* yang dilakukan merupakan sebuah proses yang kompleks, waktu yang dihabiskan bisa mencapai beberapa detik. Jika datanya semakin besar, maka proses yang membutuhkan beberapa detik tersebut dapat menjadi jauh lebih lama lagi [3].

Pada sistem ini, jumlah server yang digunakan adalah tiga server, satu server untuk MySQL *application* server dan *Cluster Management* server, dan dua server untuk *node cluster data*. Versi

MySQL *Cluster* yang digunakan adalah 7.4. Penggambaran secara umum dari sistem MySQL Cluster dapat dilihat pada Gambar 2.2.

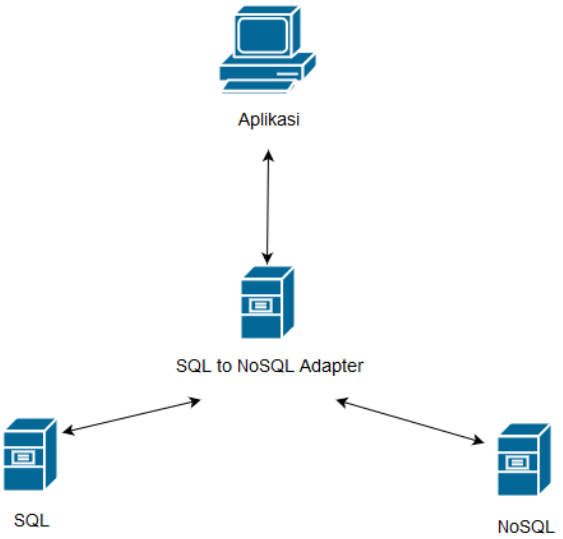


Gambar 2.2 Desain sistem MySQL Cluster secara umum

2.3 SQL-to-NoSQL Data Adapter (SNDA)

SQL-to-NoSQL Data Adapter (SNDA) merupakan sebuah sistem yang dibuat oleh Husni et al. dan merupakan salah satu hasil dari Penelitian tahun 2017. Sistem yang dibangun adalah sinkonisasi dari sebuah basis data SQL dengan basis data NoSQL. SNDA itu sendiri menjadi penghubung utama antara kedua jenis basis data tersebut dengan melakukan transfomasi data relasional yang ada pada SQL menjadi baris NoSQL [6].

Proses kerja SNDA dimulai dengan melakukan inisialisasi data pada NoSQL. SNDA akan membuat tabel terlebih dahulu pada NoSQL. Apabila tabel sudah dibuat, SNDA akan melakukan proses konversi data dari SQL menjadi format .csv yang kemudian dieksekusi ke dalam NoSQL. Data yang berada di NoSQL tersebut kemudian akan mengalami transformasi jika data pada SQL mengalami perubahan, semisal terjadi proses *query insert*, *update*, ataupun *delete*. Semua ini dimungkinan jika NoSQL memiliki API yang dapat melakukan konversi data tersebut.



Gambar 2.3 Topologi dasar dari sistem SNDA

Untuk basis data jenis SQL, *Database Management System* (DBMS) yang digunakan adalah MySQL. Sedangkan untuk basis data jenis NoSQL, DBMS yang digunakan adalah Apache HBase. HBase sendiri merupakan sebuah basis data yang menyimpan data-datanya pada sebuah peta multidimensional yang tersortir, membuatnya berbeda dari sistem NoSQL yang biasanya [7]. Topologi dasar dari SNDA dapat dilihat pada Gambar 2.3.

2.4 Mysqlslap

Mysqlslap adalah aplikasi diagnostik yang dikeluarkan oleh Oracle untuk mensimulasikan *client load* pada sebuah server MySQL. Simulasi pada mysqlslap bekerja dalam tiga tahap, dimulai dengan membuat skema, tabel, serta data yang dispesifikasikan oleh penggunanya pada basis data yang diujikan. Kemudian mysqlslap memulai simulasi *client load* dalam beberapa tahap berdasarkan jumlah *client*. Setelah simulasi selesai proses pembersihan basis data dimulai mulai dari pemutusan jaringan hingga penghapusan tabel jika diperlukan. Mysqlslap digunakan disini sebagai alat uji pada setiap skenario MySQL *Cluster*.

2.5 Postman

Postman adalah sekumpulan perangkat lunak yang berfungsi sebagai sebuah REST *Client*. Dengan Postman, pengguna dapat melakukan interaksi dengan antarmuka aplikasi menggunakan metode HTTP seperti POST, GET, PUT, hingga DELETE. Postman memungkinkan pengguna melakukan pengujian terhadap aplikasi yang sedang dikerjakan hingga menyimpan hasil respon dari pengujian tersebut. Pada Postman juga tersedia fitur monitoring sehingga pengguna dapat mengetahui seberapa responsif aplikasi yang dibangun. Disini, Postman akan digunakan sebagai alat uji pada sistem SNDA yang dibangun dengan memanfaatkan baris perintah SQL yang dijalankan melalui PHP.

2.6 JMeter

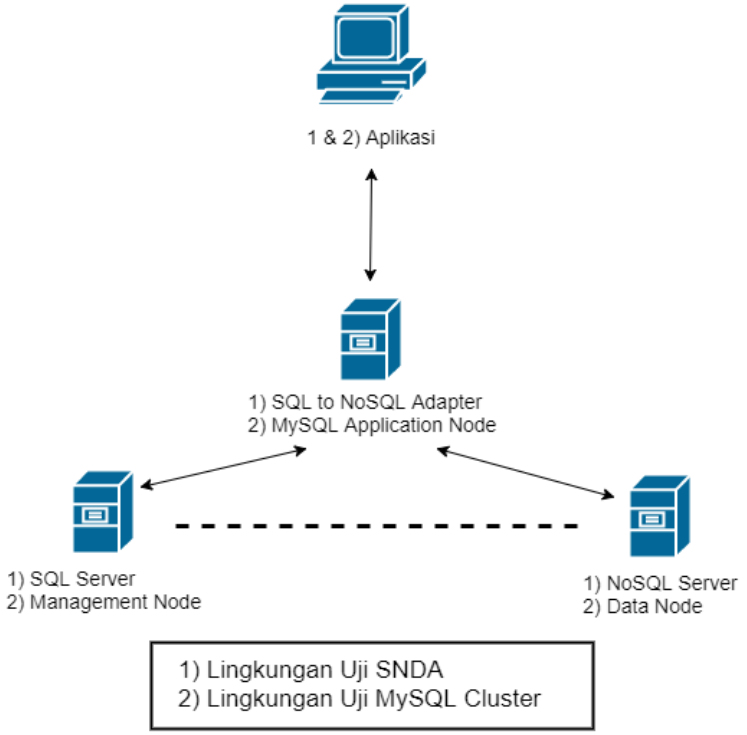
JMeter adalah sebuah proyek dari Apache yang dapat digunakan untuk melakukan pengujian *load* dari sebuah aplikasi *web*. Sepeerti Postman, JMeter mampu melakukan interaksi dengan antarmuka aplikasi lewat metode HTTP yang ada. Akan tetapi, JMeter mempunyai kemampuan untuk melakukan pengujian *per-thread* sehingga sebuah halaman dapat diuji secara bersamaan dalam jumlah klien lebih dari satu.

[*Halaman ini sengaja dikosongkan*]

BAB III
DESAIN DAN PERANCANGAN

Bab desain dan perancangan berisi tentang perancangan dan pembuatan sistem yang akan dibangun. Perancangan yang dijelaskan pada bab ini meliputi perancangan sistem secara umum, perancangan arsitektur sistem, dan perancangan basis data.

3.1 Desain dan Perancangan Sistem Secara Umum



Gambar 3.1 Desain lingkungan pengujian secara umum

Arsitektur yang dibangun akan digunakan untuk dua buah sistem sekaligus, yaitu satu sistem SQL-to-NoSQL Data Adapter (SNDA) yang dibuat untuk sinkronisasi data dari basis data SQL

dan NoSQL, serta satu sistem MySQL *Cluster* yang terdiri atas *management node*, *data node*, serta MySQL *apllication node* itu sendiri. Untuk SQL pada SNDA, digunakan DBMS MySQL. Sedangkan untuk NoSQL, DBMS yang diunakan adalah Apache HBase. Jumlah server yang digunakan untuk pengujian ini berjumlah tiga server, yang digunakan secara bergantian. Pada sistem SNDA, satu server akan digunakan untuk MySQL server, satu server untuk server HBase, dan satu server untuk SNDA itu sendiri. Secara keseluruhan, skema pengujian akan berlangsung pada lingkungan sistem seperti yang dapat dilihat pada Gambar 3.1.

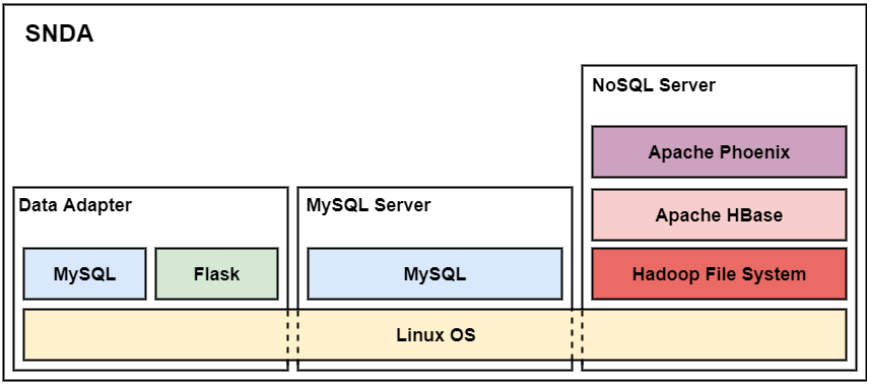
3.2 Desain Sistem SNDA

Sistem SNDA terdiri atas data adapter, SQL Server, dan NoSQL Server. Data adapter merupakan penghubung antara aplikasi dengan basis data. Data adapter berfungsi sebagai penerima permintaan dari aplikasi dan melakukan transformasi data dari MySQL ke Apache HBase. SQL Server digunakan untuk menyimpan semua data aplikasi yang bersifat relasional. Sedangkan NoSQL Server digunakan untuk menyimpan data dalam format yang berbeda dengan menggunakan basis data NoSQL.

Data adapter terdiri atas dua komponen utama, yaitu DB Adapter dan DB Converter. DB Adapter berfungsi untuk menerima seluruh permintaan dari aplikasi seperti pengajuan *query*, pengambilan data, pembaharuan data, serta penghapusan data. Semua permintaan tersebut dimungkinkan dengan menyediakan API berkeluaran JSON sebagai antarmukanya. Sedangkan DB Converter berfungsi untuk melakukan transformasi data dari basis data SQL menjadi basis data NoSQL. Hal ini dikarenakan secara *default* permintaan perubahan data akan diarahkan ke MySQL dan permintaan akses data serta permintaan lainnya dari aplikasi akan diarahkan ke HBase. Transformasi data ini dilakukan dengan bantuan Apache Phoenix yang berguna menerjemahkan *query* SQL menjadi rangkaian perintah HBase, yang kemudian dieksekusi ke

HBase Server. Seluruh proses ini diimplementasikan menggunakan Python sebagai bahasa pemrogramannya dan Flask sebagai kerangka kerja mikro yang digunakan untuk pemenuhan fiturnya.

Untuk SQL Server, digunakan MySQL sebagai DBMS SQL yang digunakan. Sedangkan HBase dipilih sebagai aplikasi yang digunakan untuk NoSQL Server. Pemasangan HBase memerlukan Apache Hadoop yang merupakan aplikasi *big data* dikarenakan HBase berjalan di atas Apache Hadoop. Pemisahan basis data SQL dengan NoSQL ini bertujuan untuk memberikan kemampuan pada aplikasi untuk dapat diakses oleh banyak pengguna. Arsitektur keseluruhan dari sistem SNDA dapat dilihat pada Gambar 3.2.



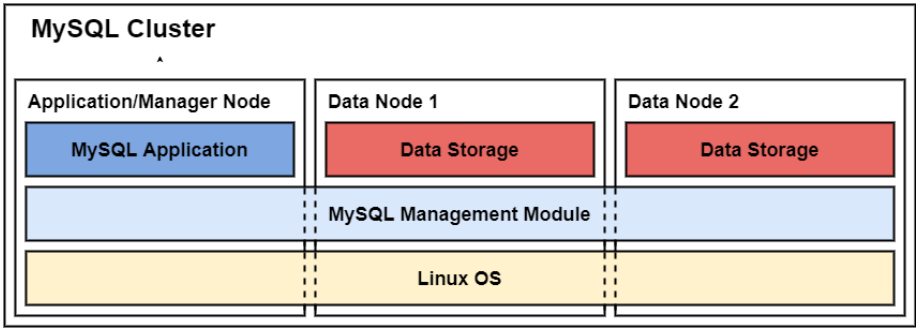
Gambar 3.2 Desain Arsitektur SNDA

3.3 Desain Sistem MySQL *Cluster*

Sistem MySQL *Cluster* terdiri atas *application node*, *management node*, dan *data node* yang dapat berjumlah lebih dari satu. *Application node* merupakan *node* yang menghubungkan *data node* yang ada dengan aplikasi yang sedang dibangun. *Application node* yang paling umum digunakan adalah MySQL Server, tetapi aplikasi atau bahasa pemrograman lain seperti Java hingga Node.js bisa menjadi *application node* dengan pustaka yang disediakan. *Management node* adalah *node* yang mengendalikan

informasi dari susunan *cluster* yang dibangun serta mengendalikan *node* lain, sehingga harus dijalankan terlebih dahulu sebelum menjalankan *node* lain. Sedangkan *data node* adalah proses basis data yang menyimpan data pada *cluster* serta menjawab permintaan dari proses *query* bersama-sama dengan *data node* yang lain jika ada.

Management node merupakan bagian pertama yang harus dijalankan pada sebuah sistem MySQL *Cluster*. *Management node* berfungsi sebagai pengelola sistem dengan menyediakan konfigurasi dari lingkungan sistem itu sendiri, menjalankan atau mematikan *data node*, melakukan proses *backup*, hingga data mengenai struktur dari sistem tersebut. *Management node* juga menyimpan histori dari sistem tersebut dalam bentuk *log*, mulai dari identitas *node* hingga alamat dari *node* tersebut, sehingga setiap kali terdapat *node* yang ingin terhubung dengan sistem, maka *node* tersebut perlu menghubungi *management node* terlebih dahulu untuk mengetahui susunan dari sistem. Setelah *node* ini berjalan barulah *application node* dan *data node* yang berada dalam *cluster* dapat berinteraksi satu sama lain. Rancangan arsitektur dari MySQL *Cluster* dapat dilihat pada Gambar 3.3.



Gambar 3.3 Desain Arsitektur MySQL Cluster

3.4 Proses Kerja SNDA

Sistem SNDA dimulai dengan proses transformasi dan sinkronisasi. Pertama-tama, sinkronisasi dimulai secara manual

yang kemudian sistem akan melakukan pengecekan apakah sistem sudah pernah disinkronisasi sebelumnya. Jika belum pernah, maka dimulailah proses inisialisasi, yaitu duplikasi basis data SQL menjadi basis data NoSQL dengan memanfaatkan MySQL *dumping* dalam bentuk .csv yang kemudian dibuatkan tabel yang baru pada basis data NoSQL. Data hasil *dump* tadi kemudian ditransformasi dengan bantuan Apache Phoenix. Tetapi, jika sistem sudah pernah melakukan sinkronisasi sebelumnya, maka proses langsung berjalan ke proses transformasi dengan terlebih dahulu membaca *log query* yang dihasilkan dari proses sebelumnya untuk dibandingkan proses sinkronisasi sebelumnya. Perubahan yang belum dilakukan kemudian disimpan ke dalam NoSQL dengan memanfaatkan Apache Phoenix. Seluruh proses tersebut disediakan melalui sebuah antarmuka yang juga bertugas untuk mengarahkan *query* sesuai dengan tujuannya, yaitu NoSQL jika melakukan pengambilan data dan SQL jika melakukan perubahan data.

3.5 Proses Kerja MySQL *Cluster*

Sistem MySQL *Cluster* dimulai dengan menjalankan *management node* di salah satu server. Jika sistem dijalankan untuk pertama kalinya, sistem harus memberikan lokasi dimana konfigurasi sistem ditempatkan. *File* konfigurasi tersebut berupa spesifikasi dari alamat masing-masing node beserta *folder* yang digunakan oleh MySQL *Cluster*. Pengguna juga dapat mengatur jumlah replika data yang diperlukan hingga batas memori yang dapat digunakan oleh masing-masing *node*. Kemudian, *management node* akan mengkondisikan sistem berdasarkan konfigurasi tersebut. Setelah *management node* selesai melakukan pengaturan, barulah semua *data node* yang berada dalam lingkungan sistem dapat dijalankan. Nantinya, setiap eksekusi *query* yang dilakukan oleh *application node* akan dikirimkan kepada setiap *data node* yang ada untuk diproses secara bersama-sama.

3.6 Skenario Pengujian SNDA dan MySQL *Cluster*

Pengujian akan dilakukan dalam 3 buah skenario, dengan 1 skenario yang sesuai dengan arsitektur dari SNDA sendiri dan 2 buah skenario pada MySQL *Cluster*. Skenario untuk SNDA melibatkan tiga buah server sesuai dengan fungsinya masing-masing, yaitu Data Adapter, MySQL Server, serta server basis data NoSQL yang terdiri atas Hadoop, HBase, dan Apache Phoenix. Pengujian akan dilakukan pada sebuah *client* yang akan mengakses Data Adapter sebagai antarmuka utama dari SNDA, karena pada SNDA sendiri sudah terdapat skenario untuk melakukan proses CRUD itu sendiri. Antarmuka tersebut dapat diakses menggunakan Postman dengan metode POST dan GET. Kemudian, sistem akan diuji lebih lanjut menggunakan JMeter untuk mengetahui kemampuan akses dari sistem yang telah dibuat.

Untuk MySQL *Cluster*, pengujian akan dilakukan pada 2 buah skenario, dimana satu skenario dilakukan pada sebuah pseudo clustering dalam satu server dan skenario lainnya pada 3 server terpisah dengan satu server menjadi *management* dan *application node* serta dua buah server untuk *data node*. Hal ini dilakukan untuk mengetahui seberapa besar efek latensi atau jeda waktu pada pemrosesan *query* dan data serta memastikan bahwa data benar-benar terdistribusi pada lebih dari satu *data node*. Pengujian akan dilakukan menggunakan *mysqlslap* yang merupakan alat diagnostik dari MySQL untuk melakukan simulasi *stress test* dari klien yang dilakukan secara bertahap. Hasilnya akan ditampilkan pada akhir pengujian dan sumberdaya yang digunakan pada masing-masing *node* akan ditampilkan pada laporan yang dihasilkan oleh *management node*.

BAB IV
IMPLEMENTASI

Pada bab ini diuraikan mengenai implementasi dari sistem SNDA dan MySQL *Cluster* yang telah dibahas pada Bab III yang meliputi rancangan dari kedua sistem. Pembahasan dilakukan secara menyeluruh untuk setiap komponen dari SNDA dan MySQL *Cluster*.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengujian awal dari kedua sistem dilakukan secara *cloud* menggunakan jasa penyewaan server dari Scaleway dengan jenis server VC1M dengan pilihan lokasi server di Paris, Perancis. Spesifikasi yang didapat dari *nbench* menunjukkan bahwa setiap server yang disewa memiliki spesifikasi yang sama yaitu 4 buah *core* dari Intel(R) Atom(TM) CPU C2750 @ 2.40GHz dengan memory 4.0 GB serta jaringan yang dibatasi pada kecepatan sekitar 200Mbit/s. Ketiga server tersebut berada di jangkauan alamat IP publik 202.47.xxx.xxx. Untuk memudahkan penjelasan, ketiga server akan didefinisikan sebagai 202.47.1.1 hingga 202.47.1.3. Perangkat yang digunakan pada saat melakukan implementasi dapat dilihat pada ***Tabel 4.1***.

Tabel 4.1 Perangkat Lunak yang Digunakan

No.	Perangkat Lunak	Versi	Keterangan
1	Ubuntu	14.04	Sistem Operasi
2	Nano	2.2.6	*Text Editor*
3	Python	2.7.6	Bahasa Pemrograman
4	Flask	0.12.2	Pustaka Python
5	Virtual Environment	15.1.0	Lingkungan Kerja Python
6	Apache Hadoop	2.7.3	Sistem Berkas *Big Data*

7	Apache HBase	1.2.4	Basis Data NoSQL untuk SNDA
8	Apache Phoenix	4.10	Translator
9	MySQL	5.5	Basis Data SQL untuk SNDA
10	Postman	5.5.0	REST *client*
11	JMeter	4.0	*Load Testing Tool*
12	SQLyog Community Edition	12.09	Antarmuka MySQL
13	MySQL Cluster	7.4	Basis Data SQL Terdistribusi
14	Mysqlslap	4.5.7	Alat uji MySQL berbasis *stress test*
15	Microsoft Word	2016	Pengolah Data

4.2 Pengaturan Awal Server

Karena semua server berjalan pada lingkungan jaringan internet publik, diperlukan beberapa perlindungan untuk mencegah server diretas oleh pihak yang tidak diinginkan. Hal pertama yang dilakukan adalah dengan mengatur SSH agar tidak terhubung dengan *port* 22. Terdapat pilihan tambahan dengan memanfaatkan SSH *tunneling* untuk mengakses setiap server serta memblokir seluruh port selain *port* yang digunakan untuk SSH, tetapi karena implementasi dari SNDA menggunakan banyak *port* dengan *range* yang berbeda-beda, pilihan untuk memblokir seluruh *port* tidak diambil. Untuk mengatasi masalah ini, dipilihlah tindakan tambahan dengan membatasi akses server untuk hanya bisa diakses melalui ketiga server tersebut atau melalui VPN yang terenkripsi. VPN yang digunakan disini adalah VPN pribadi yang telah dipasang sebelum penulisan dimulai. Langkah-langkah yang diambil secara detail adalah sebagai berikut:

1. Buatlah user sudo non-*root* bernama hduser pada masing-masing server dengan menggunakan perintah *adduser --gecos "" hduser*.

2. Buat satu usergroup baru dengan nama hadoop menggunakan perintah *addgroup hadoop.*
3. Tambahkan user tersebut pada grup sudo dan usergroup baru tadi dengan menjalankan perintah *adduser –ingroup hadoop hduser* dan *adduser hduser sudo.*
4. Sunting konfigurasi dari ssh yang terletak pada */etc/ssh/sshd_config*. Rincian perubahan dan tambahan yang dilakukan dapat dilihat pada Kode Sumber 4.1.
5. Tambahkan alamat *host* yang diperbolehkan dan diblokir oleh SSH pada */etc/hosts.allow* dan */etc/hosts.deny*. Tambahan yang dilakukan pada berkas dapat dilihat pada Kode Sumber 4.2.

```
1  Port 21976
2  PermitRootLogin no
3  X11Forwarding no
4  #PasswordAuthentication no
5
6  ###Tambahkan pada bagian paling bawah###
7  UseDNS no
8  AllowGroups sudo hduser
```

Kode Sumber 4.1 Konfigurasi SSH pada Berkas sshd_config

```
1  #/etc/hosts.allow
2  sshd: 202.47.1.1
3  sshd: 202.47.1.2
4  sshd: 202.47.1.3
5  sshd: 163.172.xxx.xxx
6
7  #/etc/hosts.deny
8  sshd: ALL EXCEPT LOCAL
```

Kode Sumber 4.2 Konfigurasi dari Berkas *hosts.allow* dan *hosts.deny*

4.3 Rincian Implementasi Sistem SNDA

Seperti yang telah disebutkan pada bab sebelumnya, sistem SNDA terdiri atas server-server basis data beserta data adapternya. Lebih jauh lagi, server basis data yang akan disinkronisasikan adalah basis data NoSQL dan SQL, kemudian untuk data adapternya terdiri atas DB Adapter dan DB Converter. Sistem basis data SQL yang digunakan adalah MySQL Server dan untuk NoSQL adalah Apache HBase. Kedua basis data akan ditempatkan secara terpisah sedangkan DB Adapter dan DB Converter akan dijalankan pada satu server yang sama. Pada dua subbab pertama akan menjelaskan implementasi dari masing-masing basis data yang digunakan dan dua subbab berikutnya untuk masing-masing komponen dari data adapter.

4.3.1 Instalasi Server Basis Data SQL

Untuk pengerjaan sistem SNDA ini hanya menggunakan satu buah server terpisah untuk basis data SQL. Meskipun begitu, tetap diperlukan konfigurasi tambahan setelah proses pemasangan MySQL pada server agar data adapter dapat terhubung dengan basis data tersebut. Selain hal tersebut, untuk proses instalasi MySQL tidak ada perbedaan signifikan.

Alamat IP publik telah disediakan oleh pihak penyedia server untuk bisa terhubung dengan server lain, tetapi aksesnya perlu dibatasi nanti agar hanya data adapter yang dapat mengakses server MySQL. Alamat IP yang digunakan untuk basis data SQL adalah 202.47.1.1.

Sesuai dengan panduan pengerjaan, server MySQL perlu mencatat seluruh *log query* yang dieksekusi karena data tersebut yang akan dibaca nantinya oleh data adapter untuk proses sinkronisasi. Untuk itu, konfigurasi dari MySQL perlu diubah dari hanya mencatat log *error* menjadi dapat mencatat segala *query* yang dieksekusi ke MySQL. Untuk memungkinkan proses tersebut, sunting berkas */etc/mysql/log_query.log* dan tambahkan baris *log = /var/log/mysql/log_query.log*. Selain itu, kita juga perlu mengatur server MySQL untuk dapat diakses oleh IP tertentu sehingga hanya data adapter, server lain, serta VPN yang dapat

mengakses server MySQL. Perintah untuk menjalankan hal tersebut dijalankan seperti mengeksekusi *query* di MySQL. Pertama-tama, masuk ke server mysql sebagai user *root*. Kemudian, lakukan langkah-langkah menambahkan user beserta hak aksesnya sebagai berikut:

1. Masuk ke MySQL menggunakan username dan password *default*. Jalankan perintah *mysql –u root –p,* kemudian tekan enter.
2. Buat pengguna baru untuk semua alamat dengan nama ‘atma’ dan password ‘a’ dengan menjalankan perintah *CREATE USER ‘atma’@’%’ IDENTIFIED BY ‘a’*;
3. Berikan hak akses kepada pengguna baru tersebut dengan membatasi IP yang diperbolehkan mengakses. Hal tersebut dapat dicapai dengan mengulai perintah *GRANT ALL PRIVILEGS ON *.* TO ‘atma’@’<alamat_ip>’ IDENTIFIED BY ‘a’*;
4. Muat ulang konfigurasi yang telah diubah dengan menjalankan perinth *FLUSH PRIVILEGES*;

4.3.2 Instalasi Server Basis Data NoSQL

Untuk basis data NoSQL digunakan Apache HBase yang berjalan di atas sistem berkas Hadoop. Server HBase dipasang terpisah dengan server basis data MySQL. Sebelum melakukan instalasi HBase, Hadoop harus terpasang terlebih dahulu.

4.3.2.1 Instalasi Hadoop

Pada bagian ini akan dijelaskan tahapan dalam pemasangan sistem berkas Hadoop. Instalasi dilakukan menggunakan user non-*root* yang telah dibuat sebelumnya Langkah-langkahnya adalah sebagai berikut:

1. Pastikan bahwa server yang digunakan telah terpasang Java. Jika belum, pasang Java Developer Kit (JDK) dengan menuliskan perintah *sudo apt-get install default-jdk.*

2. Hadoop juga memerlukan SSH untuk melakukan manajemen node-nya. Lakukan instalasi SSH dengan menjalankan perintah *sudo apt-get install ssh.*
3. Hadoop dalam menggunakan SSH untuk mengakses *node*-nya biasanya mengharuskan penggunanya untuk menggunakan kata sandi. Tetapi persyaratan ini dapat ditiadakan dengan membuat dan mengatur sertifikat SSH untuk menggunakan kata kunci yang kosong dengan menjalankan perintah *ssh-keygen –t rsa –P* ‘’.
4. Setelah itu tambahkan sertifikat tersebut ke daftar kunci yang ada di sistem operasi sehingga Hadoop dapat menggunakan ssh tanpa permintaan password. Ketik perintah *cat $HOME/ .ssh/id_rsa.pub >> $HOME/ .ssh/authorized_key*s.
5. Unduh brkas instalasi Hadoop dengan perintah *wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.3/hadoop-2.7.3.tar.gz*.
6. Lakukan ekstraksi berkas Hadoop dengan perintah *tar -xvzf hadoop-2.7.3.tar.gz*.
7. Pindahkan semua isi dari berkas hadoop ke direktori */usr/local/hadoop.*
8. Selanjutnya adalah melakukan konfigurasi Hadoop. Yang pertama dilakukan adalah menyunting konfigurasi dari bashrc dengan menjalankan perintah *nano ~/.bashrc*. Konfigurasi yang ditambahkan dapat dilihat pada Kode Sumber 4.3.
9. Sunting isi dari variabel JAVA_HOME pada berkas hadoop-env.sh agar Hadoop bisa mendapatkan nilai JAVA_HOME ketika dibutuhkan. Jalankan perintah *nano /usr/local/hadoop/hadoop-env.sh* untuk merubahnya. Jangan lupa tambahkan nomor port SSH yang kita ubah sebelumnya pada variabel HADOOP_SSH_OPTS. Isi konfigurasi dari variabel ini dapat dilihat pada Kode Sumber 4.4.

10. Sunting isi dari berkas */usr/local/hadoop/core-site.xml*. *Core-site.xml* adalah konfigurasi awal Hadoop saat memulai proses. Jalankan *nano /usr/local/hadoop/hadoop-env.sh* untuk memulai penyuntingan. Isi konfigurasi dapat dilihat di Kode Sumber 4.5
11. Salin dan ubah nama berkas dari */usr/local/ hadoop/etc/hadoop/mapred-site.xml.template* menjadi *mapred-site.xml*. Kemudian tambahkan konfigurasi seperti yang tertulis di Kode Sumber 4.6.
12. Sunting berkas */usr/local/ hadoop/etc/hadoop/hdfs-site.xml*. *Hdfs-site.xml* memiliki fungsi untuk mengatur setiap *host* dari *cluster* yang digunakan. Sebelumnya menyunting berkas tersebut, tambahkan dua direktori baru untuk *namenode dan datanode* Hadoop dengan menjalankan perintah *mkdir –p /usr/local/hadoop-store/hdfs/namenode* dan *mkdir –p /usr/local/hadoop-store/hdfs/datanode*. Setelah itu, ubah *hdfs-site.xml* seperti dengan yang tertulis di Kode Sumber 4.7.
13. Sebelum dijalankan, sistem berkas Hadoop perlu di format untuk dapat digunakan. Jalankan perintah *hadoop namenode –format.* Hadoop siap dijalankan dengan menjalankan *start-all.sh* yang terletak di */usr/local/hadoop/bin*. Tampilan UI dapat diakses dari port 50070 di localhost.

```
1  #HADOOP VARIABLES START
2  export JAVA_HOME=/usr/lib/jvm/java-7-open-
   jdk-amd64
3  export HADOOP_INSTALL=/usr/local/hadoop
4  export PATH=$PATH:$HADOOP_INSTALL/bin
5  export PATH=$PATH:$HADOOP_INSTALL/sbin
6  export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
7  export HADOOP_COMMON_HOME=$HADOOP_INSTALL
8  export HADOOP_HDFS_HOME=$HADOOP_INSTALL
9  export YARN_HOME=$HADOOP_INSTALL
10 export HADOOP_COMMON_LIB_NATIVE_DIR=
   $HADOOP_INSTALL/
   lib/native
11 export HADOOP_OPTS="-
   Djava.library.path=$HADOOP_INSTALL/lib"
12 #HADOOP VARIABLES END
```

Kode Sumber 4.3 Konfigurasi dari Berkas .bashrc

```
1  #Cari baris ini
2  export JAVA_HOME=${JAVA_HOME}
3
4  #Ubah menjadi baris ini
5  export JAVA_HOME=/usr/lib/jvm/java-7-
   openjdk-amd64
6
7  #Cari baris seperti ini atau tambahkan jika
   tidak ditemukan
8  export HADOOP_SSH_OPTS="-p 21976"
```

Kode Sumber 4.4 Konfigurasi dari Berkas hadoop-env.sh

```
1  <configuration>
2   <property>
3    <name>hadoop.tmp.dir</name>
4    <value>/app/hadoop/tmp</value>
5    <description>A base for other temporary
     directories.</description>
6   </property>
7
8   <property>
9    <name>fs.default.name</name>
10   <value>hdfs://localhost:54310</value>
11   <description>The name of the default
     file system.  A URI whose
12   scheme and authority determine the
     FileSystem implementation.  The
13   uri's scheme determines the config
     property (fs.SCHEME.impl) naming
14   the FileSystem implementation class.
     The uri's authority is used to
15   determine the host, port, etc. for a
     filesystem.</description>
16  </property>
17 </configuration>
```

Kode Sumber 4.5 Konfigurasi dari Berkas core-site.xml

```
1  <configuration>
2   <property>
3    <name>mapred.job.tracker</name>
4    <value>localhost:54311</value>
5    <description>The host and port that the
MapReduce job tracker runs
6    at.  If "local", then jobs are run in-
process as a single map
7    and reduce task.
8    </description>
9   </property>
10 </configuration>
```

Kode Sumber 4.6 Konfigursi dari Berkas mapred-site.xml

```
1  <configuration>
2   <property>
3    <name>dfs.replication</name>
4    <value>1</value>
5    <description>Default block replication.
6    The actual number of replications can be
specified when the file is created.
7    The default is used if replication is not
specified in create time.
8    </description>
9   </property>
10  <property>
11    <name>dfs.namenode.name.dir</name>
12    <value>
file:/usr/local/hadoop_store/hdfs/namenode
</value>
13  </property>
```

```
14  <property>
15  <name>dfs.datanode.data.dir</name>
16  <value>
    file:/usr/local/hadoop_store/hdfs/datanode
    </value>
17  </property>
18  </configuration>
```

Kode Sumber 4.7 Konfigurasi dari Berkas hdfs-site.xml

4.3.2.2 Instalasi Apache HBase & Apache Phoenix

Setelah instalasi sistem berkas Hadoop selesai, proses instalasi Apache HBase dan Apache Phoenix dapat dimulai. Versi Apache HBase yang digunakan adalah 1.2.4, sedangkan Apache Phoenix menggunakan versi 4.10.0. Selanjutnya pada subbab ini akan menjelaskan mengenai cara pemasangan HBase dan Apache Phoenix.

1. Unduh paket Apache HBase dan Apache Phoenix dengan menjalankan perintah *wget https://archive.apache.org/dist/hbase/1.2.4/hbase-1.2.4-bin.tar.gz* dan *wget https://archive.apache.org/dist/phoenix/apache-phoenix-4.10.0-HBase-1.2/bin/apache-phoenix-4.10.0-HBase-1.2-bin.tar.gz*.
2. Ekstrak kedua berkas tersebut dengan menjalankan *tar –xvzf hbase-1.2.4-bin.tar.gz* dan *tar –xvzf apache-phoenix-4.10.0-HBase-1.2-bin.tar.gz*.
3. Buat direktori HBase dengan menjalankan perintah *mkdir usr/local/hbase*. Pindahkan seluruh hasil ekstrasi dari berkas *hbase-1.2.4-bin.tar.gz* ke direktori tersebut.
4. Atur path java dan port SSH dari berkas hbase.sh. Sunting berkas tersebut dengan menjalankan perintah *nano*

/usr/local/hbase/conf/hbase-env.sh dan gunakan konfigurasi yang tertulis seperti pada Kode Sumber 4.4.

5. Tambahkan path HBASE_HOME pada berkas .bashrc. Jalankan perintah *nano ~/.bashrc* lalu tambahkan konfigurasi yang serupa dengan Kode Sumber 4.3.
6. Selanjutnya adalah mengubah berkas konfigurasi pada berkas hbase-site.xml untuk mengatur direktori tempat HBase akan menyimpan data.
7. Pindahkan berkas *phoenix-4.10.0-server-hbase-1.2.jar* dari hasil ekstraksi ke */usr/local/hbase/lib*.
8. Mulai HBase dengan masuk ke dalam direktori */usr/local/hbase/bin* kemudian jalankan berkas *start-hbase.sh*. Untuk mengetahui apakah HBase telah berhasil berjalan, jalankan perintah *hbase shell*.

4.3.3 Instalasi Data Adapter

Data adapter adalah bagian dari sistem yang menghubungkan aplikasi dengan kedua basis data. Data Adapter merupakan penerima, pengeksekusi seluruh *query* CRUD yang diambil dari basis data SQL maupun NoSQL. Semua proses tersebut dilakukan dengan pembuatan API yang dibangun menggunakan Flask di atas Python. Setiap data yang dikirim menggunakan *form-data* HTML dan menghasilkan keluaran JSON.

Untuk memfasilitasi fungsi adapter tersebut, dibuatlah antar muka dengan rute-rute tertentu sesuai dengan tujuan yang diinginkan. Fungsionalitas antarmuka yang dibangun terdiri atas rute-rute yang telah dideskripsikan pada Tabel 4.2.

Tabel 4.2 Penjelasan Rute dari Antarmuka

No	Rute	Masukan	Luaran	Proses
1	GET /sinkron	-	status, pesan	Sinkronisasi basis data MySQL ke HBase

2	POST /insert_penduduk	data_penduduk	status, pesan	Memasukkan data penduduk ke tabel kependudukan
3	POST /delete_penduduk by_nik	NIK	status, pesan	Menghapus satu baris data penduduk di tabel kependudukan
4	GET /select_all_penduduk	-	alamat_host, jenis_db, jumlah_baris, data_penduduk	Mengambil semua data pada tabel kependudukan
5	GET /select_penduduk_by_nik/<nik>	NIK	alamat_host, jenis_db, jumlah_baris, data_penduduk	Mengambil salah satu data penduduk berdasarkan NIK
6	POST /update_penduduk_by_nik	NIK, data_penduduk	status, pesan	Mengubah atau memperbaharui data pada tabel kependudukan

4.4 Rincian Implementasi Sistem MySQL *Cluster*

Pada bab sebelumnya, kita sudah membahas komponen dari MySQL *Cluster*, yaitu *management node, application node,* serta *data node*. *Management node* sebagai pengatur utama *cluster* akan diimplementasikan terlebih dahulu bersamaan dengan *application node*. *Data node* akan dikonfigurasikan sesuai dengan apa yang didefinisikan oleh *management node* sebelumnya. Pada tugas akhir ini, implementasi dari *management node* dan *application node* akan selalu diletakkan dalam satu server yang sama, sehingga tidak akan ada jeda waktu dalam komunikasi antara kedua *node* yang disebabkan oleh koneksi jaringan. Sedangkan dua buah *node* harus diletakkan pada server yang terpisah untuk skenario kedua agar dapat mensimulasikan proses basis data yang terdistribusi. Versi basis data MySQL *Cluster* yang digunakan adalah versi 7.4 dan untuk *application node*, versi MySQL server yang digunakan adalah 5.6. Terdapat subbab yang akan menjelaskan implementasi dari *management node* dan *application node* serta kedua *data node*. Setiap server disini melalui proses pengaturan awal server yang sama dengan SNDA untuk menjaga keamanan server.

Sebelum menjelaskan tahapan instalasi dari setiap *node*, terdapat beberapa langkah instalasi yang dilakukan dengan tahapan yang sama pada seluruh server. Seluruh instalasi dilakukan menggunakan user sudo non-*root* yang telah dibuat sebelum memulai tahapan ini. Beberapa tahapan tersebut adalah sebagai berikut:

1. Pasang dependensi yang dibutuhkan oleh MySQL *cluster*. Dependensi tersebut dapat dipasang dengan menggunakan perintah *sudo apt-get install libaio1*.
2. Unduh paket instalasi MySQL *Cluster* pada setiap server. Jalankan perintah *wget https://downloads.mysql.com/archives/get/file/mysql-cluster-gpl-7.4.16-debian7-x86_64.deb* untuk semua server.

3. Pasang paket instalasi tersebut pada setiap server. Untuk melakukannya, gunakan perintah *sudo dpkg -i mysql-cluster-gpl-7.4.16-debian7-x86_64.deb*. Hasil instalasi bisa dilihat pada direktori */opt/mysql/server-5.6/*.

4.4.1 Instalasi *Management Node dan Application Node*

Pada bagian ini akan dijelaskan tahapan dalam pemasangan pemasangan komponen *management node* dan *application node* pada sistem MySQL *Cluster*. Tahapan ini dilakukan agar proses sinkronisasi dapat berjalan dengan tepat dan data bisa terdistribusi ke seluruh *data node*. Langkah-langkah yang diperlukan adalah sebagai berikut:

1. Pastikan terdapat sebuah direktori yang bisa digunakan untuk menyimpan berkas konfigurasi dari MySQL *Cluster*. Direktori tersebut dibuat menggunakan perintah *sudo mkdir /var/lib/mysql-cluster*.
2. Karena file konfigurasi dari MySQL *Cluster* tidak ditujukan untuk *node* yang berjalan pada satu server, kita perlu membuat berkas konfigurasi sendiri. Jalankan perintah *sudo touch /var/lib/mysql-cluster/config.ini* lalu sunting berkas tersebut dengan menggunakan perintah *sudo nano /var/lib/mysql-cluster/config.ini*. Tambahkan konfigurasi untuk *management node* seperti yang tertulis pada **Kode Sumber 4.8**.
3. Pada tahap ini, *management node* siap dijalankan. Gunakan perintah *sudo /opt/mysql/server-5.6/bin/ndb_mgmd -f /var/lib/mysql-cluster/config.ini* untuk memulai *node* tersebut.
4. Agar dapat dijalankan secara otomatis setiap kali server dinyalakan, atur berkas rc.local untuk menjalankan perintah di atas setiap kali *booting*. Cara tersebut dapat dilakukan dengan menjalankan perintah *sudo systemctl enable rc-local.service* untuk memastikan fitur tersebut aktif. Kemudian tambahkan perintah di atas sebelum baris "exit" dengan mengubah berkas melalui *sudo nano /etc/rc.local.*

```
[ndb_mgmd]
hostname=202.47.1.1
datadir=/var/lib/mysql-cluster

[ndbd]
hostname=202.47.1.2
datadir=/usr/local/mysql/data

[ndbd]
hostname=202.47.1.3
datadir=/usr/local/mysql/data

[mysqld]
hostname=202.47.1.1
```

Kode Sumber 4.8 Konfigurasi pada Berkas config.ini

5. Atur *application node* untuk menggunakan MySQL *Cluster* sebagai sistem penyimpan basis datanya. Ubah berkas konfigurasi MySQL Server dengan menjalankan *sudo nano /etc/my.cnf*. Kemudian ubah tulisan dibawah baris yang bertuliskan "[mysqld]" menjadi *"ndbcluster"*.
6. Setelah *data node* sudah terhubung seluruhnya, kita dapat membuat basis data *default* untuk *application node* dengan menjalankan perintah *sudo /opt/mysql/server-5.6/scripts/mysql_install_db --user=hduser*.
7. Agar *application node* dapat berjalan, kita akan menggunakan berkas pemulaian dari */opt/mysql/server-5.6/support-files/mysql.server*. Duplikasikan berkas tersebut dengan menggunakan perintah *sudo cp /opt/mysql/server-5.6/support-files/mysql.server /etc/init.d/mysqld*. Jalankan juga *sudo systemctl enable mysqld.service* agar *application node* dapat berjalan secara otomatis setiap kali *booting*.

8. Jika ingin mengakses *application node* langsung dari server tersebut, kita dapat menggunakan *client* dari */opt/mysql/server-5.6*. Hubungkan *client* tersebut dengan perintah *sudo ln -s /opt/mysql/server-5.6/bin/mysql /usr/bin/* dan *client* dapat langsung diakses dengan cukup menuliskan *mysql* setelahnya.

4.4.2 Instalasi *Data Node*

Pada bagian ini akan dijelaskan tahapan dalam pemasangan pemasangan *data node* pada sistem MySQL *Cluster*. *Data node* adalah lokasi dimana bagian dari data akan disimpan dan *engine* dari basis data yang melakukan proses *query* berada. Tahapan ini dilakukan setelah tahapan instalasi *management node* dan sebelum *application node*, tetapi karena instalasi dilakukan pada server yang berbeda dari kedua *node* tersebut, penjelasan dari instalasi *data node* ditempatkan terpisah dari kedua *node* tersebut. Tahapan ini harus dilakukan di kedua *data node* yang ada. Langkah-langkah yang perlu dilakukan adalah:

1. Ubah konfigurasi dari MySQL pada server yang terletak di */etc/my.cnf*. Jalankan perintah *sudo nano /etc/my.cnf* kemudian setelah bagian *[mysql_cluster]* tambahkan baris "*ndb-connectstring=202.47.1.1*" tanpa tanda kutip.
2. Atur lokasi dimana data-data yang akan diterima oleh *data node* akan disimpan. Karena pada spesifikasi dari config.ini mengarahkan data ke direktori */usr/local/mysql/data*, kita perlu membuat direktori tersebut dengan perintah *sudo mkdir -p /usr/local/mysql/data*.
3. Jalankan *data node* untuk pertama kali menggunakan perintah *sudo /opt/mysql/server-5.6/bin/ndbd*. Jika berhasil, *data node* akan terhubung ke *manager node*.
4. Agar dapat dijalankan secara otomatis setiap kali server dinyalakan, atur berkas rc.local untuk menjalankan perintah di atas setiap kali *booting*. Cara tersebut dapat dilakukan dengan menjalankan perintah *sudo systemctl*

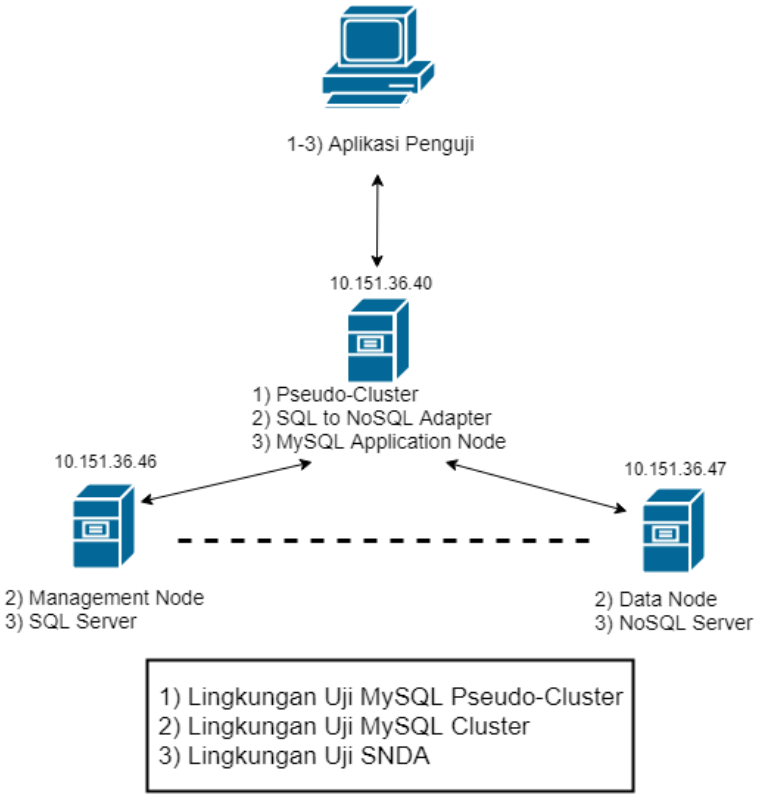
enable rc-local.service untuk memastikan fitur tersebut aktif. Kemudian tambahkan perintah di atas sebelum baris "exit" dengan mengubah berkas melalui *sudo nano /etc/rc.local*.

5. Jalankan seluruh tahap-tahap sebelumnya dari subbab ini pada *data node* kedua.

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan menjelaskan hasil dari beberapa pengujian yang telah dilakukan terhadap kedua sistem yang telah dikerjakan sebelumnya beserta hasil uji coba dan analisa dari sistem MySQL Cluster dan SNDA.

5.1 Lingkungan Uji Coba



Gambar 5.1 Desain Umum Dari Lingkungan Uji Coba Sistem

Lingkungan pengujian untuk ketiga skenario dilakukan pada tiga buah server seperti yang diperlihatkan pada Gambar 5.1. Ketiga server mempunyai peranan berbeda untuk setiap skenario yang diujikan. Sedangkan untuk skenario kedua, digunakan tiga buah server yang terdiri atas *application node*, *management node*, dan *data node* dari MySQL Cluster. Skenario ketiga adalah skema pengujian dari sistem SNDA yang terdiri atas server basis data MySQL, server basis data NoSQL, serta satu buah server data adapter seperti . Seluruh Pengujian skenario dilakukan secara terpisah pada sebuah komputer. Proses pengujian ketiga skenario tersebut dilakukan di Laboratorium Arsitektur dan Jaringan Komputer gedung Informatika ITS. Spesifikasi server dan komputer yang digunakan diuraikan lebih jauh pada Tabel 5.1 ***Spesifikasi Server Pertama*** hingga Tabel 5.4 ***Spesifikasi Komputer Penguji***.

Tabel 5.1 Spesifikasi Server Pertama

Server 1	
Komponen	**Spesifikasi**
Jenis	Komputer Fisik
Alamat IP	10.151.36.40
Processor	Intel(R) Core(TM) i3-4150 CPU @ 3.50GHz (4 CPUs)
Memori	RAM 8,5 GB
Penyimpanan	931,6 GB
Grafis	NVidia GeForce GT 705
Sistem Operasi	Ubuntu Desktop 14.04.5 LTS
Perangkat Lunak	Skenario 1:
	1. MySQL Cluster Manager
	2. MySQL Data Node Manager
	3. MySQL Server
	Skenario 2:
	1. MySQL Server
	Skenario 3:

	1. MySQL Server
	2. Flask 0.12.1
	3. Python 2.7.6

Tabel 5.2 Spesifikasi Server Kedua

Server 2	
Komponen	**Spesifikasi**
Jenis	Komputer Fisik
Alamat IP	10.151.36.46
Processor	Intel(R) Core(TM)2 Duo-E7300 @ 2.66GHz (2 CPUs)
Memori	RAM 2,14 GB
Penyimpanan	250GB
Grafis	Intel(R) G31
Sistem Operasi	Ubuntu Desktop 14.04.5 LTS
Perangkat Lunak	Skenario 2:
	1. MySQL Cluster Manager
	Skenario 3:
	1. MySQL Server

Tabel 5.3 Spesifikasi Server Ketiga

Server 3	
Komponen	**Spesifikasi**
Jenis	Komputer Fisik
Alamat IP	10.151.36.47
Processor	Intel(R) Core(TM)2 Duo CPU E7300 @ 2.66GHz (2 CPUs)
Memori	RAM 2,14 GB
Penyimpanan	298GB
Grafis	Intel(R) G31
Sistem Operasi	Ubuntu Desktop 14.04.5 LTS
	Skenario 2:

	1. MySQL Data Node Manager
	Skenario 3:
	1. MySQL Server
	2. Flask 0.12.1
	3. Python 2.7.6
	4. Apache Hadoop 2.7.3
	5. Apache HBase 1.2.4
	6. Apache Phoenix 4.10.0

Tabel 5.4 Spesifikasi Komputer Penguji

Komputer Penguji	
Komponen	**Spesifikasi**
Jenis	Komputer Fisik
Processor	AMD A10-5750M APU with Radeon(tm) HD Graphics @ 2.50 GHz
Memori	RAM 7.2 GB
Penyimpanan	928,6 GB
Grafis	AMD Radeon HD 8650G
Sistem Operasi	Windows 10
Perangkat Lunak	1. Postman 6.1.3
	2. mysqlslap 4.5.7
	3. Microsoft Word 2016
	4. SQLyog 12.09
	5. JMeter 4.0

5.2 Dataset Uji Coba

Untuk tugas akhir ini, *data set* yang digunakan adalah data percontohan yang menyerupai data kependudukan di salah satu daerah di Jawa Timur. Data-data ini sudah dimanipulasi dan disederhanakan untuk menyesuaikan dengan kebutuhan dari tugas

akhir ini. Seluruh data disimpan dalam format .csv dan .sql sesuai dengan kebutuhan skenario masing-masing. *Data set* terdiri atas 2 tabel dengan rincian:

1. Tabel Data Penduduk:
   a. Baris : 43.659
   b. Kolom :
      1) nik (integer)
      2) nama_lgkp (varchar)
      3) jenis_klmn (integer)
      4) tempat_lhr (varchar)
      5) tgl_lhr (date)
      6) no_kk (int)
2. Tabel Data Keluarga:
   a. Baris : 11.644
   b. Kolom :
      1) no_kk (integer)
      2) nama_kep (varchar)
      3) alamat (varchar)

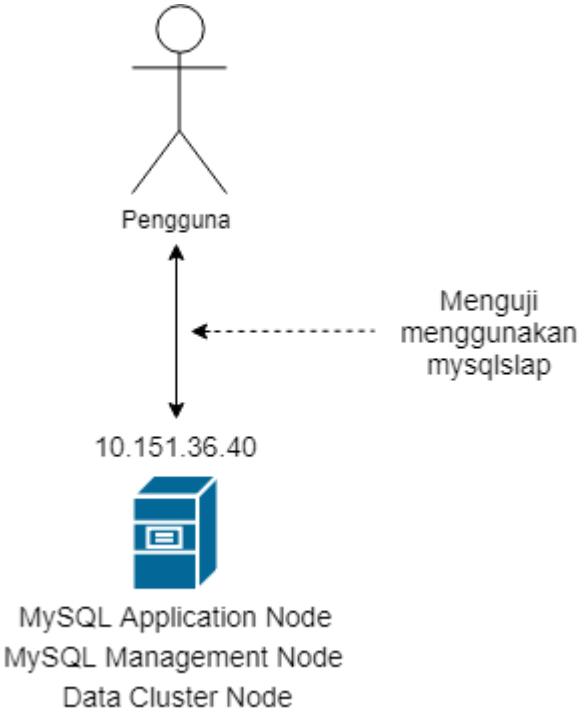
5.3 Skenario Uji Coba

Uji coba terhadap kedua sistem dilakukan dalam tiga buah skenario untuk mengetahui kemampuan dari kedua sistem yang telah dibuat, terutama SNDA jika dibandingkan dengan sistem basis data terdistribusi yang sudah ada seperti MySQL Cluster. Pengujian performa dilakukan dengan melakukan proses yang umum dilakukan pada basis data yaitu CRUD (*Create*, *Read*, *Update*, *Delete*). Pengujian tersebut dilakukan dengan bantuan tiga buah perangkat lunak yaitu mysqlslap, Postman, dan JMeter. Seluruh pengujian menggunakan dataset yang telah disiapkan sebelumnya.

Hasil dari setiap uji coba disimpan dalam bentuk grafik dan tabel. Nilai ukur yang digunakan pada pengujian meliputi Penggunaan CPU, memory, dan waktu yang diperlukan untuk memproses seluruh permintaan data oleh klien. Perbandingan jumlah klien digunakan untuk mengukur kemampuan setiap sistem

dalam menangani penggunaan sehari-hari. Jumlah perbandingan klien yang diuji adalah 50, 100, 150, 200, 250, dan 300 klien. Untuk rincian setiap skenarionya akan dijelaskan pada subbab-subbab berikutnya.

5.3.1 Skenario Uji Coba MySQL Cluster



Gambar 5.2 Desain Arsitektur Uji Coba Skenario 1

Pengujian pada MySQL Cluster meliputi pengujian pada satu server *pseudo-cluster* dan tiga buah server yang merupakan syarat minimal dari sebuah MySQL Cluster. Kedua skenario yang dibuat meliputi perbedaan dari posisi *node* yang ada pada sistem MySQL Cluster, yaitu *management node*, *data node*, dan

application node. Untuk uji coba performa CRUD, digunakan mysqlslap agar sistem dapat diuji kehandalannya dalam menangani lebih dari 1 pengguna. Perhitungan performa diambil dari rata-rata nilai *management node* dan *data node* dimana mayoritas proses dilakukan. Pengujian dengan mysqlslap dilakukan menggunakan perintah seperti berikut, dengan nilai *concurrency* divariasikan sesuai dengan jumlah klien yang akan diujikan:

- Pengujian SELECT

```
mysqlap --host=10.151.36.40 user=”atma” 
-password=”zsefvd123” --create-
schema=data_penduduk --query="SELECT * 
FROM kependudukan;" --concurrency=50 --
iterations=1
```

- Pengujian INSERT

```
mysqlap --host=10.151.36.40 user=”atma” 
-password=”zsefvd123” --create-
schema=data_penduduk --query="INSERT 
INTO kependudukan 
(NIK,NAMA_LGKP,JENIS_KLMN,TEMPAT_LHR,
TGL LHR,NOMOR KK) VALUES 
('5113511351135113','ATMA','1','JAKARTA
','1994-10-26','5113511351135113')" --
concurrency=50 --iterations=1
```

- Pengujian UPDATE

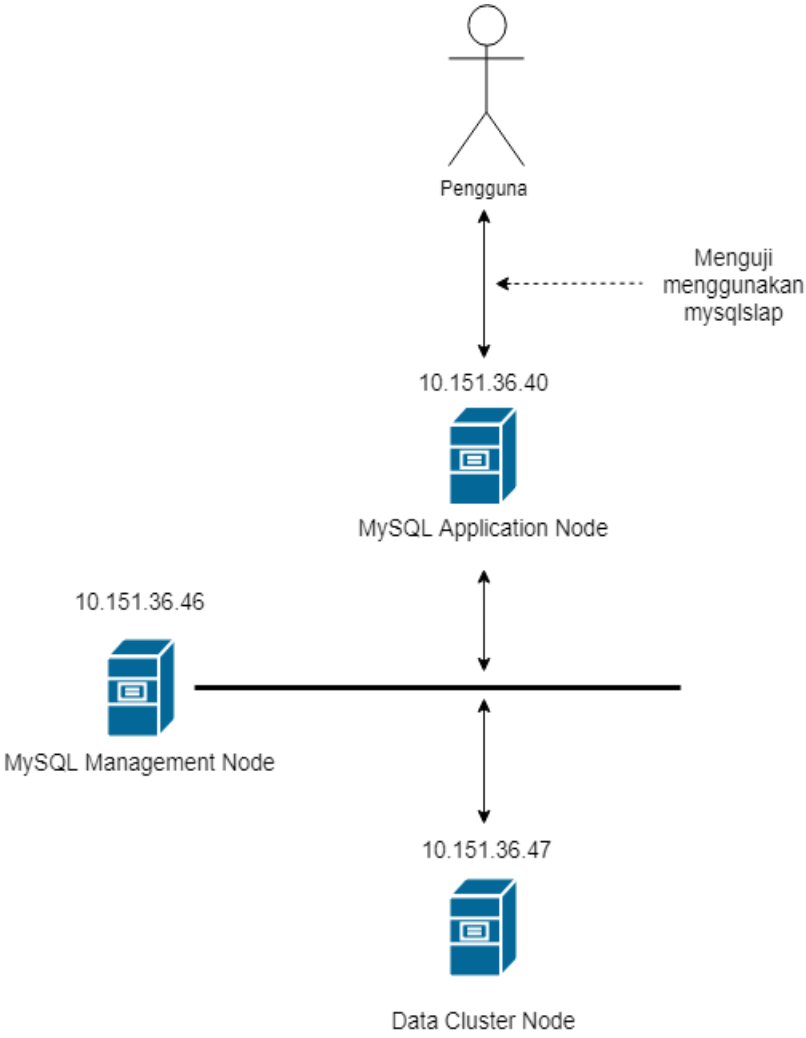
```
mysqlap --host=10.151.36.40 --
user=”atma” -password=”zsefvd123” --
create-schema=data_penduduk --
query="UPDATE kependudukan SET 
NOMOR_KK='3115311531153115' WHERE 
NIK='5113511351135113'" --
concurrency=50 --iterations=1
```

- Pengujian DELETE

```
mysqlap --host=10.151.36.40 --
user=”atma” –password=”zsefvd123” --
create-schema=data_penduduk
 --query="DELETE FROM kependudukan
WHERE NIK=’5113511351135113’ "--
concurrency=50 --iterations=1
```

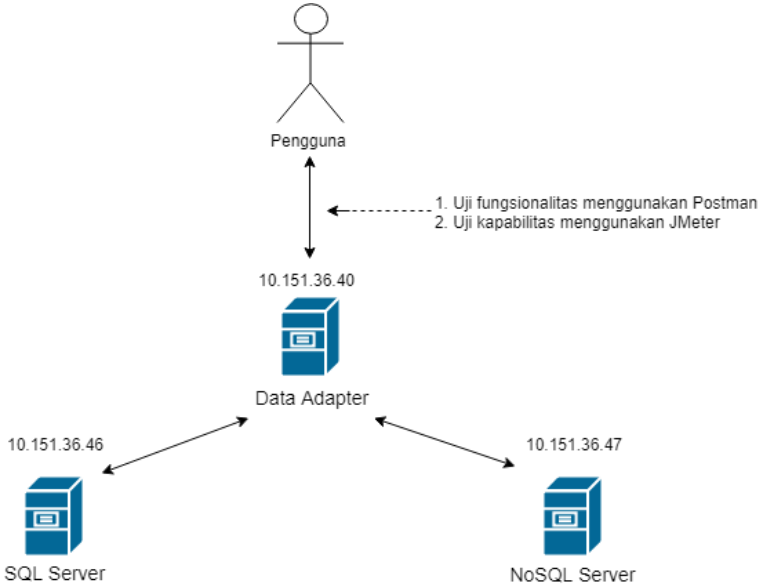
Skenario pertama adalah pengujian pada sistem MySQL Cluster yang dipasang pada sebuah server. Sistem ini tidak dapat dibilang sebagai sebuah *cluster* karena seluruh *node* terletak pada satu server yang sama. Tetapi, sistem ini memiliki fungsionalitas yang sama seperti sebuah MySQL Cluster pada umumnya sehingga sistem ini dapat disebut sebagai *pseudo-cluster* untuk menguji sistem pada kecepatan optimalnya. Arsitektur dari sistem ini dijelaskan pada Gambar 5.2.

Skenario kedua adalah pengujian pada sistem MySQL Cluster yang dipasang pada tiga server terpisah. Sistem ini merupakan bentuk paling sederhana dari sebuah sistem MySQL Cluster dimana *management*, *data*, dan *application node* terletak pada server masing-masing. Pengujian dengan skenario ini dilakukan untuk melihat performa sebenarnya pada sebuah sistem MySQL Cluster. Arsitektur untuk skenario ini dapat dilihat pada Gambar 5.3.



Gambar 5.3 Desain Arsitektur Uji Coba Skenario 2

5.3.2 Skenario Uji Coba SNDA



Gambar 5.4 Desain Arsitektur Uji Coba Skenario 3

Uji coba pada sistem SNDA dilakukan dalam dua metode pada satu skenario arsitektur, yaitu uji fungsionalitas dan performa CRUD pada tiga server. Uji fungsionalitas dilakukan untuk memastikan fitur-fitur yang telah dibuat berjalan dengan seharusnya. Sedangkan uji performa CRUD dilakukan dimana hasilnya akan digunakan untuk komparasi dengan hasil kedua skenario sistem MySQL Cluster. Seluruh uji coba dilakukan pada arsitektur sesuai dengan Gambar 5.4.

Uji fungsionalitas terdiri atas pengujian inisialisasi basis data dan data adapter. Inisialisasi basis data adalah proses sinkronisasi MySQL dengan HBase dimana tabel basis data SQL akan diekspor ke dalam berkas dengan format .csv yang kemudian dieksekusi ke basis data NoSQL. Pengujian dilakukan dengan memastikan apakah data berhasil tersinkronisasi dengan baik pada NoSQL. Sedangkan data adapter adalah antarmuka yang

menghubungkan kedua basis data dengan seluruh proses yang terjadi pada sistem mulai dari transformasi data, pengambilan data, hingga proses *query* CRUD, dimana setiap perubahan data diarahkan ke basis data SQL dan pengambilan data ke NoSQL. Pengujian fungsionalitas pada bagian ini dilakukan dengan memastikan bahwa seluruh fitur tersebut dapat berjalan dengan fitur yang dimaksud. Kecepatan sinkronisasi tabel, *query* INSERT, UPDATE, dan DELETE juga dihitung disini. Semua perhitungan kecepatan tersebut berdasarkan 1000 baris data yang dilakukan. Pengujian fungsionalitas dilakukan menggunakan Postman. Seluruh isi uji coba fungsionalitas tersebut dapat dilihat pada Tabel 5.5.

Untuk uji performa pada sistem SNDA, sistem akan diuji melakukan seluruh proses *query* CRUD menggunakan JMeter. Rute yang digunakan untuk pengujian CRUD adalah "select_all_penduduk","insert_penduduk","update_penduduk_by_nik", dan "delete_penduduk_by_nik". Khusus untuk "select_all_penduduk", *query* akan diproses iakses melalui basis data NoSQL, sedangkan untuk yang lainnya akan diproses melalui SQL. Pengujian ini akan menghitung seluruh nilai ukur yang telah disebutkan sebelumnya, yaitu penggunaan CPU, *memory*, serta waktu yang diperlukan untuk memproses seluruh permintaan data oleh klien. Perbandingan jumlah klien yang diuji dimungkinkan untuk dilakukan dengan menggunakan fitur *threading* dan pengacakan data angka dari JMeter.

Tabel 5.5 Uji Fungsionalitas dan Hasil yang Diinginkan

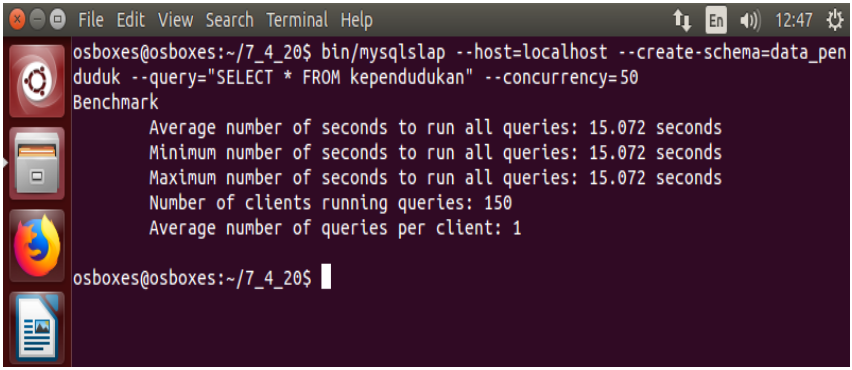
No	Rute	Uji Coba	Hasil yang Diinginkan
1	GET /sinkron	Melakukan proses sinkronisasi basis data SQL dan NoSQL	Status sinkronisasi berhasil, terdapat log sinkronisasi, isi data pada kedua basis data sama.

2	POST /insert_penduduk	Menambahkan satu data penduduk baru dari tabel kependudukan	Terdapat penambahan data pada tabel kependudukan
3	POST /delete_penduduk by_nik	Menghapus satu data penduduk dari tabel kependudukan	Data yang dimaksud pada tabel kependudukan terhapus
4	GET /select_all_penduduk	Mengambil semua baris data dari tabel kependudukan	Semua data dari tabel kependudukan terambil
5	GET /select_penduduk_by_nik/<nik>	Mengambil satu baris data dari tabel kependudukan	data yang dimaksud dari tabel kependudukan terambil
6	POST /update_penduduk_by_nik	Menyunting satu baris data dari tabel kependudukan	status, pesan

5.4 Hasil Uji Coba

Pada subbab ini akan menjelaskan hasil pengujian yang telah dilakukan dengan semua skenario yang telah dijelaskan pada subbab sebelumnya. Hasil Pengujian akan diuraikan tiap sistem lalu dibandingkan dengan semua skenario yang telah diuji. Seluruh hasil diberikan dengan asumsi tidak terjadi kegagalan pada proses pengujiannya.

5.4.1 Hasil Uji Coba MySQL Cluster



Gambar 5.5 Tampilan Terminal MySQL Cluster Ketika Pengujian Berlangsung

Pengujian dilakukan pada sistem yang telah diisi data sebelumnya. Pengujian dilakukan oleh pengguna menggunakan mysqlslap. Kecepatan respon didapat dari waktu rata-rata yang dibutuhkan untuk menjalankan satu *query*. Contoh tampilan pengujian dapat dilihat pada Gambar 5.5. Hasil Pengujian CRUD untuk sistem MySQL Cluster dapat dilihat pada Tabel 5.6 untuk sistem 1 server dan Tabel 5.7.

Tabel 5.6 Hasil Kecepatan Respon untuk Uji Coba CRUD pada MySQL Cluster 1 Server

No	Jenis *Query*	Kecepatan Respon (detik)					
		50	**100**	**150**	**200**	**250**	**300**
1	SELECT	17,3	40,1	64,2	X	X	X
2	INSERT	0,7	2,5	3,1	5,2	6,8	7,6

3	UPDATE	1,1	2,4	3,9	5,9	X	X
4	DELETE	1,2	2,8	3,6	5,8	X	X

(X) = Terjadi setidaknya satu *error* saat pengujian klien

Tabel 5.7 Hasil Kecepatan Respon untuk Uji Coba CRUD pada MySQL Cluster 3 Server

No	Jenis *Query*	Kecepatan Respon (detik)					
		50	**100**	**150**	**200**	**250**	**300**
1	SELECT	38,7	89,1	152,5	229,7	337,1	X
2	INSERT	0,2	0,3	0,5	3,2	6,1	X
3	UPDATE	0,8	2,1	3,4	3,2	4,9	6,4
4	DELETE	1,1	1,0	1,5	4,8	7,2	X

(X) = Terjadi setidaknya satu *error* saat pengujian klien

Seperti yang dapat dilihat, terdapat beberapa pengujian yang tidak memiliki nilai kecepatan respon. Itu dikarenakan terdapat nilai *error* pada saat pengujian yang menyebabkan hasil menjadi tidak valid. Pada sistem 1 server, kegagalan terjadi pada SELECT 200, 250, dan 300 klien; UPDATE 250 dan 300 klien; serta DELETE 250 dan 300 klien. Sedangkan untuk sistem 3 server terjadi pada SELECT 300 server, INSERT 300 Server, dan DELETE 300 Server. Perbedaan kejadian *error* antara kedua sistem ini disebabkan oleh perbedaan pembagian proses dimana pada sistem 3 server setiap proses diserahkan ke masing-masing *node* sesuai fungsinya.

Untuk hasil performa dari pengujian sistem MySQL Cluster, data hasilnya hanya diambil dari sistem 3 server sebagai sistem percontohan dari MySQL Cluster. Data yang diambil adalah hasil rata-rata dari *node management* dan *data node* sebagai *node* yang melakukan mayoritas proses *query*. Hasil pengujian performa dapat dilihat pada Tabel 5.8.

Tabel 5.8 Hasil Uji Performa dari Sistem MySQL Cluster dengan Perbandingan Banyak Klien yang Mengakses

No	Jenis *Query*	Banyak Klien	Memori (MB)	Rata-Rata Prosesor (%)
1	SELECT	50	1678	22,4
2		100	1679	22,6
3		150	1681	23,3
4		200	1682	23,7
5		250	1683	24,1
6		300	1685	22,4
7	INSERT	50	1672	0,7
8		100	1674	0,7
9		150	1675	0,8
10		200	1677	0,9
11		250	1679	1,3
12		300	1680	1,3
13	UPDATE	50	1679	43,9
14		100	1679	82,7
15		150	1679	85,0
16		200	1679	100,2
17		250	1679	100,7
18		300	1679	100,5
19	DELETE	50	1672	0,6
20		100	1674	0,8
21		150	1675	0,9
22		200	1676	1,1
23		250	1677	1,2
24		300	1679	1,2

Untuk hasil performa, hasil perhitungan memori yang digunakan cenderung stabil hanya dengan sedikit perbedaan penggunaan memori yang cenderung meningkat kecuali untuk hasil UPDATE dimana penggunaan memori konstan pada angka

yang sama. Penggunaan prosesor cenderung meningkat dengan pengecualian pada 300 hasil klien yang disebabkan oleh *error* pada proses *query*. Perlu diperhatikan bahwa penggunaan prosesor pada *query* UPDATE melebihi batas 100% yang menunjukkan bahwa proses tersebut adalah yang proses yang paling banyak menggunakan daya pada pengujian ini.

5.4.2 Hasil Uji Coba SNDA

Pengujian pada sistem SNDA dimulai dengan uji coba fungsionalitas dengan aplikasi Postman. Hal-hal yang dilihat dari pengujian ini adalah keluaran dari terminal ketika melakukan setiap rute yang telah dibuat dengan hasil yang diinginkan sesuai apa yang dituliskan pada Tabel 5.5. Contoh keluaran pada terminal dapat dilihat pada Gambar 5.6 dimana rute yang diambil adalah sinkronisasi. Untuk seluruh hasil uji coba kecepatan sinkronisasi dapat dilihat di Tabel 5.9 dan hasil tes fungsionalitas di Tabel 5.10.

```
sinkron terakhir:  0
waktu sekarang:  2018-07-03 19:41:56.563111
time stamp:  2018-07-03 19:41:56
sql:  INSERT INTO log_sinkronisasi (waktu, ip_src, ip_dst, status) VALUES (
'2018-07-03 19:41:56', '10.151.36.46', '10.151.36.47', '0')
Belum pernah sinkronisasi
sql:  SELECT * FROM kependudukan
file_output:  file/csv/fetch_all_kependudukan.csv
Berhasil, tabel  kependudukan  berhasil di dump ke fetch_all_ kependudukan
.csv
Data berhasil di export!
Inisiasi data awal..
Mentransformasi data ke HBase..
Tabel berhasil dibuat di HBase
Tabel kependudukan berhasil diimport ke HBase...
Proses inisialisasi selesai...
Durasi sinkronisasi : 34.0823340416 seconds
Done
EXIT!

202.46.129.90 - - [03/Jul/2018 19:42:25] "GET / HTTP/1.1" 200 -
```

Gambar 5.6 Keluaran pada Terminal Ketika Melakukan Salah Satu Rute Data Adapter (Sinkronisasi)

Tabel 5.9 Kecepatan Sinkronisasi Data Adapter

No	Jenis Sinkronisasi/Transformasi	Rata-rata Memori (MB)	Rata-Rata Prosesor (%)	Kecepatan Respon (Detik)
1	Sinkronisasi 2 Tabel	1421	59,6	17,0
2	Transformasi INSERT	1456	60,8	7,3
3	Transformasi UPDATE	1425	55,5	6,5
4	Transformasi DELETE	1519	79,7	12,2

Tabel 5.10 Hasil Uji Coba Fungsionalitas

No	Rute	Uji Coba	Hasil
1	GET /sinkron	Melakukan proses sinkronisasi basis data SQL dan NoSQL	Berhasil
2	POST /insert_penduduk	Menambahkan satu data penduduk baru dari tabel kependudukan	Berhasil
3	POST /delete_penduduk by_nik	Menghapus satu data penduduk dari tabel kependudukan	Berhasil
4	GET /select_all_penduduk	Mengambil semua baris data	Berhasil dengan jumlah baris data yang dikurangi hingga

		dari tabel kependudukan	tinggal sekitar 25% dari jumlah awal
5	GET /select_penduduk_by_nik/<nik>	Mengambil satu baris data dari tabel kependudukan	Berhasil
6	POST /update_penduduk_by_nik	Menyunting satu baris data dari tabel kependudukan	Berhasil

Semua skenario yang diberikan oleh Tabel 5.5 menunjukkan bahwa seluruh rute berhasil dilakukan. Akan tetapi, untuk rute "/select_all_penduduk", jumlah baris harus dikurangi hingga tinggal 25% dari jumlah data awal. Ini menunjukkan bahwa data menggunakan lebih banyak sumber daya dari yang bisa diberikan oleh server. Walau demikian, hasil kecepatan *query* akan tetap diuji dengan dengan dataset yang sama.

Pencatatan performa pada sistem SNDA juga dimulai dengan penghitungan kecepatan respon dari setiap *query* CRUD yang ada pada rute data adapter, yaitu *select, insert, update,* dan *delete*. Seluruh pengujian dilakukan sebanyak sepuluh kali untuk memastikan deviasi nilai. Seperti yang telah disebutkan sebelumnya, semua *query* selain SELECT diarahkan ke SQL. Hasil pengujian CRUD dapat dilihat pada ***Tabel 5.11*** dengan hasil pertama menunjukkan jangkauan hasil dari sepuluh kali pengujian dan hasil berikutnya menunjukkan rata-rata dari sepuluh kali pengujian.

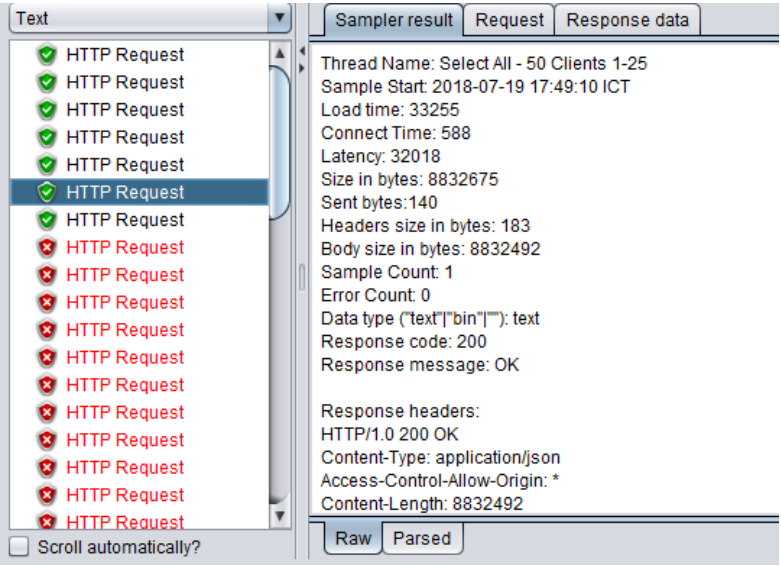
Tabel 5.11 Hasil Kecepatan Respon untuk Uji Coba CRUD pada Sistem SNDA

No	**Jenis *Query***	**Kecepatan Respon (detik)**					
		50	**100**	**150**	**200**	**250**	**300**
1	SELECT	X	X	X	X	X	X
2	INSERT	0,2	1,2	3,2	6,1	9,7	14,6

3	UPDATE	0,4	1,5	3,4	6,2	9,8	15,9
4	DELETE	0,2	0,4	0,7	1,0	1,5	1,8

(X) = Terjadi setidaknya satu *error* saat pengujian klien

Selain hasil SELECT, hasil *query* CRUD dari SNDA menyerupai hasil SQL pada umumnya, dengan *delay* yang terjadi karena jaringan. Hal ini disebabkan basis data yang digunakan adalah MySQL dengan konfigurasi asalnya. Untuk SELECT sendiri, *error* biasanya mulai terjadi pada pengujian klien ke-5 hingga ke-7. Hal ini tidak dapat dihindari karena sistem yang dibangun memang dibuat menggunakan fitur transformasi data NoSQL tersebut yang memang memakan sumber daya yang banyak. Deviasi nilai tersebut berdasarkan sepuluh kali pengujian adalah ±0,5-5,4%. Tampilan sebelum error dapat dilihat pada Gambar 5.7.



Gambar 5.7 Keluaran Antarmuka Sebelum Mengalami Error

Untuk hasil pengujian performa, data diambil dari masing-masing *server* yang bertanggung jawab atas *query* yang diprosesnya. Jadi, data INSERT, UPDATE, dan DELETE berasal dari server SQL dan SELECT dari NoSQL. Hasil pengujian performa dapat dilihat pada Tabel 5.12.

Tabel 5.12 Hasil Uji Performa dari Sistem SNDA dengan Perbandingan Banyak Klien yang Mengakses

No	Jenis *Query*	Banyak Klien	Rata-rata Memori (MB)	Rata-Rata Prosesor (%)
1	SELECT	50	1685	98,2
2		100	1687	100,5
3		150	1687	100,4
4		200	1687	100,6
5		250	1687	100,7
6		300	1687	100,7
7	INSERT	50	667	25,2
8		100	670	35,6
9		150	673	41,1
10		200	675	42,2
11		250	681	43,8
12		300	685	46,5
13	UPDATE	50	629	27,4
14		100	633	38,2
15		150	636	44,2
16		200	640	46,9
17		250	649	47,8
18		300	662	49,1
19	DELETE	50	670	1,6
20		100	672	1,8
21		150	673	1,7
22		200	674	2,0
23		250	674	2,3
24		300	674	3,7

Seperti pada hasil waktu respon, sistem SNDA menunjukkan penggunaan sumberdaya yang serupa dengan MySQL dengan konfigurasi asalnya untuk *query* INSERT, UPDATE dan DELETE. Penggunaan memori dan nilai prosesor yang fluktuatif dengan nilai relatif rendah mencerminkan basis data yang diujikan. Untuk SELECT, penggunaan sumberdaya jauh lebih tinggi dibandingkan dengan proses *query* lainnya dengan nilai penggunaan memori sekitar 1685 MB hingga 1687 MB dan penggunaan prosesor di kisaran 98,2% hingga 100,7%. Salah satu gambar hasil pengujian dapat dilihat di Gambar 5.8.

```
kueri:  INSERT INTO kependudukan (NIK, NAMA_LGKP, JENIS_KLMN, TEMPAT_LHR, T
GL_LHR, NOMOR_KK) VALUES (3520065132526390,'Atma',1,'JAKARTA','1994-10-26',
'5113511351135113')
Durasi waktu :  0.0326509475708
10.151.36.109 - - [19/Jul/2018 18:21:52] "POST /insert_penduduk HTTP/1.1" 2
00 -
jumlah kolom:  6
kueri:  INSERT INTO kependudukan (NIK, NAMA_LGKP, JENIS_KLMN, TEMPAT_LHR, T
GL_LHR, NOMOR_KK) VALUES (3520023127401355,'Atma',1,'JAKARTA','1994-10-26',
'5113511351135113')
Durasi waktu :  0.0326390266418
10.151.36.109 - - [19/Jul/2018 18:21:52] "POST /insert_penduduk HTTP/1.1" 2
00 -
jumlah kolom:  6
kueri:  INSERT INTO kependudukan (NIK, NAMA_LGKP, JENIS_KLMN, TEMPAT_LHR, T
GL_LHR, NOMOR_KK) VALUES (3520073165710535,'Atma',1,'JAKARTA','1994-10-26',
'5113511351135113')
Durasi waktu :  0.0326051712036
10.151.36.109 - - [19/Jul/2018 18:21:52] "POST /insert_penduduk HTTP/1.1" 2
00 -
jumlah kolom:  6
kueri:  INSERT INTO kependudukan (NIK, NAMA_LGKP, JENIS_KLMN, TEMPAT_LHR, T
GL_LHR, NOMOR_KK) VALUES (3520089910744466,'Atma',1,'JAKARTA','1994-10-26',
'5113511351135113')
Durasi waktu :  0.0325989723206
10.151.36.109 - - [19/Jul/2018 18:21:52] "POST /insert_penduduk HTTP/1.1" 2
00 -
```

Gambar 5.8 Tampilan Terminal saat Uji INSERT ke-9

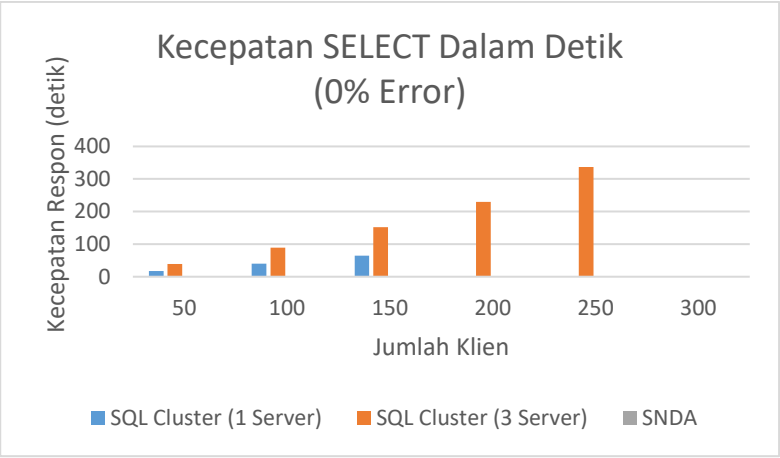
5.4.3 Hasil Uji Coba MySQL Cluster dengan SNDA

Pada subbab ini, seluruh data yang telah dikumpulkan akan diuji kembali dengan menggabungkan hasil dari setiap skenario yang ada. Parameter perbandingan yang digunakan adalah sama seperti yang telah disebutkan sebelumnya, yaitu kecepatan respon,

penggunaan memori, dan rata-rata persentase prosesor yang digunakan. Seluruh skenario akan dibandingkan hasilnya pada perbandingan kecepatan respon. Untuk penggunaan memori dan rata-rata persentase prosesor yang digunakan, hanya skenario 2 dan 3 yang diuji sebagai basis data terdistribusi yang merupakan sistem terdistribusi yang sebenarnya.

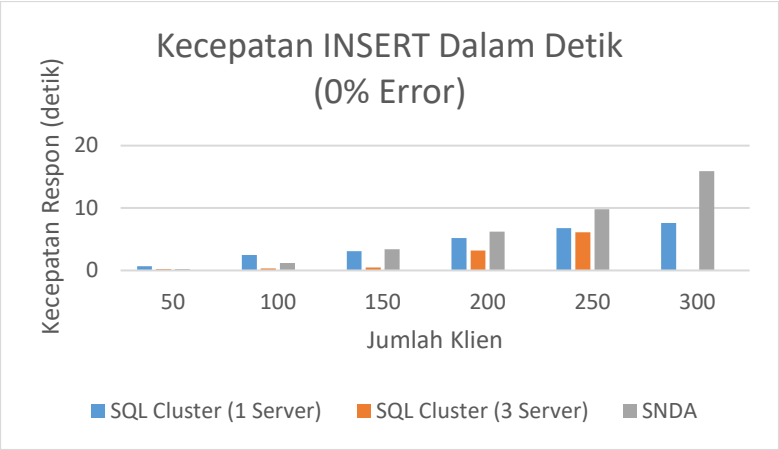
5.4.3.1 Kecepatan Respon MySQL Cluster dan SNDA

Pengujian pertama yang dilakukan adalah dengan membandingkan data SELECT dari ketiga skenario yang telah dikerjakan. Untuk perbandingan kecepatan respon SELECT, hasil uji coba dapat dilihat pada Gambar 5.9.



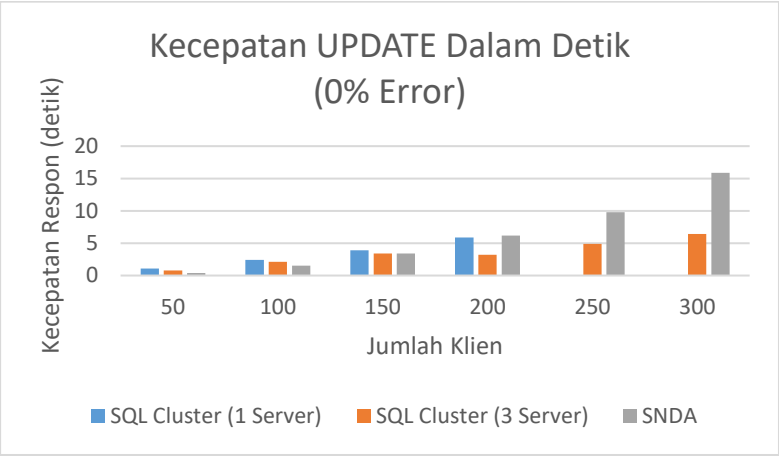
Gambar 5.9 Grafik Kecepatan Respon SELECT dari Ketiga Skenario

Pengujian berikutnya adalah dengan membandingkan data INSERT dari ketiga skenario yang telah dikerjakan. Hasil uji coba perbandingan kecepatan respon INSERT dapat dilihat pada Gambar 5.10.

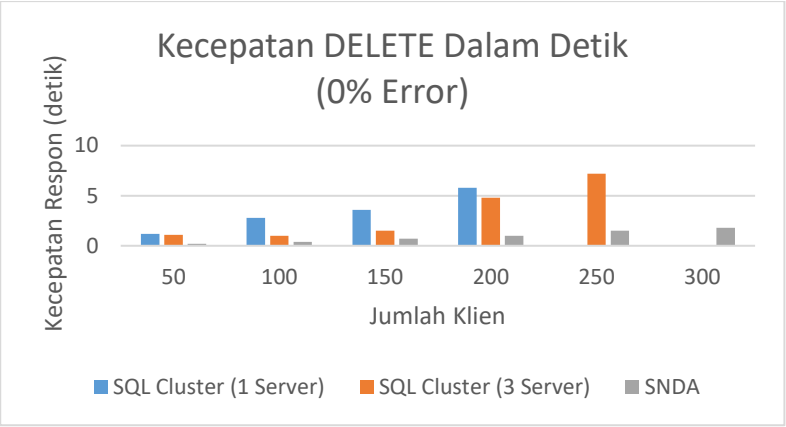


Gambar 5.10 Grafik Kecepatan Respon INSERT dari Ketiga Skenario

Pengujian selanjutnya adalah dengan membandingkan data UPDATE dari ketiga skenario yang telah dikerjakan. Hasil uji coba perbandingan kecepatan respon UPDATE dapat dilihat pada Gambar 5.11.



Gambar 5.11 Grafik Kecepatan Respon UPDATE dari Ketiga Skenario



Gambar 5.12 Grafik Kecepatan Respon DELETE dari Ketiga Skenario

Dan pengujian yang terakhir adalah perbandingan data kecepatan respon DELETE dari ketiga skenario yang telah dikerjakan. Hasil uji coba perbandingan kecepatan respon DELETE dapat dilihat pada Gambar 5.12.

Dari seluruh hasil percobaan, sistem SNDA menampilkan performa yang kompetitif dibandingkan dengan MySQL Cluster dimana tidak terjadi *error* sama sekali pada proses *query* INSERT, UPDATE, dan DELETE pada sistem SNDA. Waktu kecepatan respon yang diberikan oleh SNDA juga kurang lebih sama atau lebih lambat sedikit daripada MySQL Cluster. Untuk waktu kecepatan respon DELETE, SNDA bahkan lebih cepat dari kedua skenario MySQL Cluster yang telah dibuat. Sistem SNDA hanya mengalami masalah serius pada *query* SELECT dimana waktu yang diperlukan terlampau lambat dan mengalami *error* dengan jumlah klien 300 walau datanya sudah dikurangi hingga tinggal 25% dari besar data sebenarnya.

Setiap skenario memiliki rentang waktu kecepatan respon sebagai berikut:

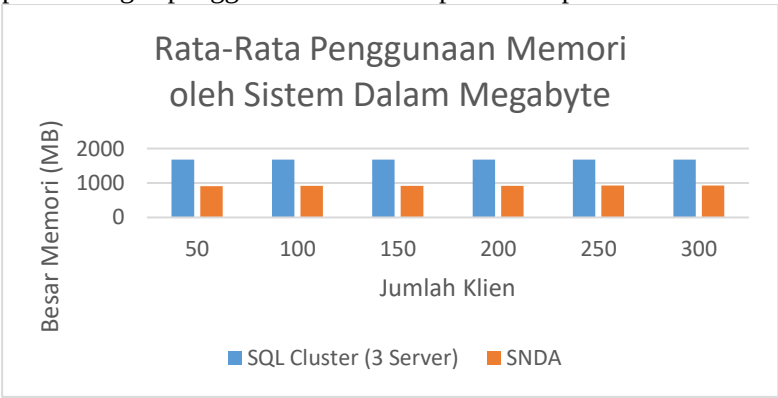
1. MySQL Cluster 1 Server
    - SELECT: 17,3 detik – 64,2 detik (150 klien)

- INSERT: 0,7 detik – 7,6 detik (300 klien)
- UPDATE: 1,1 detik – 5,9 detik (200 klien)
- DELETE: 1,2 detik – 5,8 detik (200 klien)

2. MySQL Cluster 3 Server
   - SELECT: 38,7 detik – 337,1 detik (250 klien)
   - INSERT: 0,2 detik – 6,1 detik (250 klien)
   - UPDATE: 0,8 detik – 6,4 detik (300 klien)
   - DELETE: 1,1 detik – 7,2 detik (250 klien)
3. SNDA
   - SELECT: 45,6 detik – 397,1 detik (250 klien)
   - INSERT: 0,2 detik – 14,6 detik (300 klien)
   - UPDATE: 0,4 detik – 15,9 detik (300 klien)
   - DELETE: 0,2 detik – 1,8 detik (300 klien)

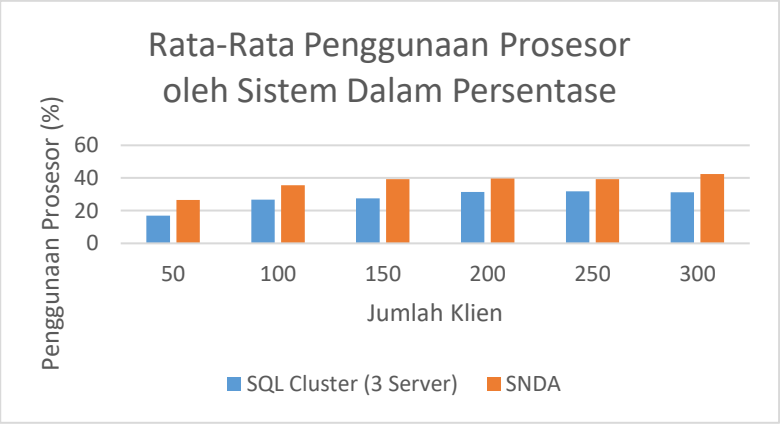
5.4.3.2 Penggunaan Sumberdaya MySQL Cluster dan SNDA

Pengujian kedua yang dilakukan adalah dengan membandingkan data penggunaan memori dan prosesor dari skenario 2 milik MySQL Cluster dan skenario 3 milik SNDA. Semua nilai penggunaan memori dan prosesor dirata-ratakan dari jumlah *query* yang diujikan. Pengujian pertama adalah untuk penggunaan memori dari kedua sistem. Hasil uji coba perbandingan penggunaan memori dapat dilihat pada Gambar 5.13.



Gambar 5.13 Grafik Rata-Rata Penggunaan Memori dari MySQL Cluster dan SNDA

Pengujian berikutnya dan yang terakhir adalah perbandingan data rata-rata penggunaan prosesor dari skenario 2 milik MySQL Cluster dan skenario 3 milik SNDA. Hasil uji coba perbandingan rata-rata penggunaan prosesor dapat dilihat pada Gambar 5.14.



Gambar 5.14 Grafik Perbandingan Rata-Rata Penggunaan Memori dari MySQL Cluster dan SNDA

Terlepas dari penggunaan memori yang cukup besar dari *query* SELECT, sistem SNDA secara keseluruhan menggunakan besar memori yang lebih sedikit dibandingkan dengan MySQL Cluster. Meskipun begitu, penggunaan prosesor pada sistem SNDA lebih tinggi dibandingkan dengan MySQL Cluster disebabkan sistem SNDA itu sendiri berjalan di atas bahasa pemrograman Python untuk transformasi dan pengambilan data sebelum dikonversi menjadi SQL dibandingkan dengan MySQL Cluster yang berjalan langsung dalam aplikasi untuk setiap prosesnya.

Untuk rentang tertinggi dari penggunaan memori dari sistem MySQL Cluster dengan SNDA kurang lebih sama disebabkan oleh *query* SELECT yang menggunakan memori yang hampir sama dengan banyak *query* dari MySQL Cluster. MySQL Cluster mempunyai rentang 1672 MB hingga 1685 MB, sedangkan SNDA

mempunyai rentang 631 MB hingga 1687 MB untuk penggunaan memori. Sedangkan untuk rata-rata penggunaan prosesor, MySQL Cluster menggunakan 0,6% hingga 100,7% sumber daya tersebut dibandingkan dengan 1,6% hingga 72,4% untuk penggunaan prosesor oleh SNDA.

5.5 Evaluasi Uji Coba

Berdasarkan hasil uji coba, semua sistem dapat berjalan sesuai dengan skenario pada sub bab 5.3 dengan sedikit kompensasi untuk salah satu skenario fungsionalnya. Dari pengujian performa kecepatan respon, SNDA mampu menyaingi kecepatan MySQL Cluster dalam hampir semua *proses* CRUD. Namun, untuk proses pengambilan data atau SELECT, SNDA masih terlampau jauh lebih lambat dari MySQL Cluster. Proses pengambilan data juga menghabiskan memori yang sama banyak dengan MySQL Cluster dengan hasil yang lebih lambat. Di sisi lain, SNDA menggunakan lebih sedikit memori secara keseluruhan dan mampu menghadapi lebih banyak klien dalam satu waktu dibandingkan dengan MySQL Cluster.

[*Halaman ini sengaja dikosongkan*]

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah. Selain itu juga terdapat saran yang ditujukan untuk pengujian dan pengembangan lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan dapat diambil beberapa kesimpulan sebagai berikut:

1. Menurut hasil pengujian fungsional, SNDA berhasil diadaptasi ke dataset yang berbeda, dalam tugas akhir ini yaitu dataset kependudukan.
2. Berdasarkan hasil kecepatan respon, SNDA mempunyai rentang kecepatan:
    a. 0,2-14,6 detik hingga 300 klien untuk INSERT.
    b. 0,4-15,9 detik hingga 300 klien untuk UPDATE.
    c. 0,2-1,8 hingga 300 klien untuk DELETE.

    SNDA mengalami *error* pada *query* SELECT mulai dari klien ke-7. Sedangkan MySQL *Cluster* memiliki rentang kecepatan:
    a. 38,7-337,1 detik hingga 250 klien untuk SELECT .
    b. 0,2-6,1 detik hingga 250 klien untuk INSERT.
    c. 0,8-6,4 detik hingga 300 klien untuk UPDATE.
    d. 1,0-7,2 hingga 250 klien untuk DELETE.
3. Untuk hasil performa, rentang penggunaan sumberdaya SNDA adalah:
    a. 1678-1685MB memori dan 98,2-100,7% prosesor untuk SELECT.

b. 667-685MB memori dan 25,2-46,5% prosesor untuk INSERT.
c. 629-662MB memori dan 27,4-49,1% prosesor untuk UPDATE.
d. serta 670-674MB memori dan 1,6-3,7% prosesor untuk DELETE.

Sedangkan MySQL *Cluster* memiliki rentang penggunaan sumberdaya:

a. 1685-1687MB memori dan 22,4-24,1% prosesor untuk SELECT.
b. 1672-1679MB memori dan 25,2-46,5% prosesor untuk INSERT.
c. 629-662MB memori dan 27,4-49,1% prosesor untuk UPDATE.
d. 670-674MB memori dan 1,6-3,7% prosesor untuk DELETE.

6.2 Saran

Saran yang diberikan untuk pengujian dan pengembangan lebih lanjut terhadap sistem SNDA adalah sebagai berikut:

1. Optimasi antarmuka perlu dikembangkan lagi agar adaptasi setiap dataset yang ada lebih mudah tanpa perlu merubah kode sumber.
2. Masalah pada pengambilan data harus bisa diatasi terlebih dahulu sebelum sistem SNDA dapat dikembangkan lebih jauh lagi terutama pada sisi data adapternya.
3. Adaptasi ulang SNDA ke dalam bahasa pemrograman lain dapat dipertimbangkan untuk mengatasi penggunaan prosesor yang berlebih.

DAFTAR PUSTAKA

[1] Å. Dragland and SINTEF, "Big Data, for better or worse: 90% of world's data generated over last two years," *ScienceDaily*. [Online]. Available: https://www.sciencedaily.com/releases/2013/05/130522085217.htm. [Accessed: 27-Apr-2017].

[2] A. Davies and H. Fisk, *MySQL Clustering*. Sams Publishing, 2006.

[3] A. Fuad, A. Erwin, and H. P. Ipung, "Processing performance on Apache Pig, Apache Hive and MySQL cluster," in *Proceedings of International Conference on Information, Communication Technology and System (ICTS) 2014*, 2014, pp. 297–302.

[4] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems, Third Edition*. New York, NY: Springer New York, 2011.

[5] Oracle, *MySQL Cluster Evaluation Guide. A MySQL Technical White Paper*. 2016.

[6] M. Husni, S. Djanali, H. T. Ciptaningtyas, and I. G. N. Adi Wicaksana, "Improved Information Retrieval Performance on SQL Database Using Data Adapter," 2017.

[7] G. Zhao, Q. Lin, L. Li, and Z. Li, "Schema Conversion Model of SQL Database to NoSQL," in *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 2014, pp. 355–362.

[*Halaman ini sengaja dikosongkan*]

LAMPIRAN A
KODE SUMBER

Kode Sumber A.1 Antarmuka untuk Sistem SNDA

```
1  import MySQLdb
2  import phoenixdb
3  from flask import Flask, request
4  from flask import jsonify, Response
5  import mysql_kueri
6  import json, urllib2
7  import c_db
8  import time
9  from ConfigParser import SafeConfigParser
10 from flask_cors import CORS, cross_origin
11
12 app = Flask(__name__)
13 CORS(app)
14
15 data_mysql = c_db.getConfMysqlDb()
16 data_hbase = c_db.getConfHbaseDb()
17 data_sync_db = c_db.getConfSyncLogDb()
   data_ssh = c_db.getSshAccess()
```

```
18
19  def connect_db_mysql(data):
20        host = data['host']
21        username = data['username']
22        password = data['password']
          db_name = data['db_name']
23
24        db = MySQLdb.connect(host=host, user=username,
25  passwd=password, db=db_name)
26        return db
27
28  def connect_db_hbase(data):
29        database_url =
30  'http://{}:8765/'.format(data_hbase['host'])
31        conn = phoenixdb.connect(database_url, autocommit=True)
32
33        return conn
34
35  def insert_into(table, data):
36        db = connect_db_mysql(data_mysql)
37        cursor = db.cursor()
38
39        kueri = mysql_kueri.insert_into_kependudukan(table,
40  data)
          print "kueri: ", kueri
```

```
41          cursor.execute(kueri)
42          db.commit()
43          db.close()
44
            if kueri:
45                  return 1
46          else:
47                  return 0
48
49    def hapus_kependudukan(table, data):
50          db = connect_db_mysql(data_mysql)
51          cursor = db.cursor()
52
53          kueri = mysql_kueri.delete_kependudukan_by_nik(table,
      data)
54          print "kueri: ", kueri
55          result = cursor.execute(kueri)
56          db.commit()
57          db.close()
58
59          if result:
60                  return 1
61          else:
62                  return 0
63
```

```
64  def query_db(query, args=(), one=False, db=0):
65        # 1 : MySQL
66        # 0 : HBase
67        if (db==1):
68              db = connect_db_mysql(data_mysql)
69        else :
70              db = connect_db_hbase(data_hbase)
71
          cursor = db.cursor()
72        cursor.execute(query, args)
73        result = cursor.fetchall()
74        db.close()
75        return (result[0] if result else None) if one else
result
76
77  def query_db_insert_update(query, db=0):
78        # 1 : MySQL
79        # 0 : HBase
80        if (db==1):
81              db = connect_db_mysql(data_mysql)
82        else :
83              db = connect_db_hbase(data_hbase)
84
85        cursor = db.cursor()
86        result = cursor.execute(query)
```

```
87          db.commit()
88          db.close()
89
90          if result:
91                  return 1
92          else :
                    return 0
93
94  def select_all_kependudukan_mysql():
95          daftar_pend = []
            data = query_db("SELECT * FROM
    kependudukan",'',False,0)
96          data_numrows = len(data)
97          for row in data :
98                  pend_tmp = {
99                          'NIK': row[0],
100                         'NAMA_LGKP': row[1],
101                         'JENIS_KLMN': row[2],
102                         'TEMPAT_LHR': row[3],
103                         'TGL_LHR': row[4],
104                         'NOMOR_KK': row[5]
105                 }
106                 daftar_pend.append(pend_tmp)
107
            result = {
```

```
108                 "Lokasi Host" : "10.151.36.46",
109                 "Jenis Database" : "MySQL",
110                 "Banyak Data" : data_numrows,
111                 "Data Penduduk" : daftar_pend
112                 }
113         return result
114
115 def select_all_kependudukan_hbase():
116         database_url =
117 'http://{}:8765/'.format(data_hbase['host'])
118         conn = phoenixdb.connect(database_url, autocommit=True)
119         cursor = conn.cursor()
120         cursor.execute("SELECT * FROM kependudukan")
121         daftar_pend = []
122         data = cursor.fetchall()
123         for row in data :
124               pend_tmp = {
125                     'NIK': row[0],
126                     'NAMA_LGKP': row[1],
127                     'JENIS_KLMN': row[2],
128                     'TEMPAT_LHR': row[3],
129                     'TGL_LHR': row[4],
                    'NOMOR_KK': row[5]
```

```
130                 }
131                 daftar_pend.append(pend_tmp)
132
133         result = {
134                 "Lokasi Host" : "10.151.36.47",
135                 "Jenis Database" : "HBase",
136                 "Data Penduduk" : daftar_pend
137                 }
138         return result
139
140 def getLastSync(data):
141         host = data['host']
142         username = data['username']
143         password = data['password']
144         db_name = data['db_name']
145
146         db = MySQLdb.connect(host=host, user=username,
147 passwd=password, db=db_name)
148         cursor = db.cursor()
149
150         sql = "SELECT * FROM log_sinkronisasi ORDER BY waktu
151 DESC LIMIT 1"
152         try :
153                 cursor.execute(sql)
                row = cursor.fetchone()
```

```
154             return row
155       except :
156             print "Error, unable to fetch Last Sync data"
157             return 0
158
159 ##-------------------------------------------------------------
160 --------------------------##
161 @app.route('/insert_penduduk', methods=['POST'])
162 def insert_penduduk():
163       start_time = time.time()
164       if request.method == 'POST' :
165             data = [ {
166                         'nama': 'NIK',
167                         'type': 'int',
168                         'value': request.form['NIK']
169                   },
170                   {
171                         'nama': 'NAMA_LGKP',
172                         'type': 'varchar',
173                         'value': request.form['NAMA_LGKP']
174                   },
175                   {
176                         'nama': 'JENIS_KLMN',
177                         'type': 'int',
                          'value': request.form['JENIS_KLMN']
```

```
178                     },
179                     {
180                           'nama': 'TEMPAT_LHR',
                              'type': 'varchar',
181                           'value': request.form['TEMPAT_LHR']
182                     },
183                     {
184                           'nama': 'TGL_LHR',
185                           'type': 'varchar',
186                           'value': request.form['TGL_LHR']
                        },
                        {
187                           'nama': 'NOMOR_KK',
188                           'type': 'varchar',
189                           'value': request.form['NOMOR_KK']
                        },
190               ]
191
                  do_insert = insert_into('kependudukan', data)
192               if do_insert :
193                     msg = 'Successed to INSERT'
                  else :
194                     msg = "Failed to INSERT"
195
196               duration = time.time() - start_time
```

```
197             print "Durasi waktu : ", duration
198             result = {
199                   "Status" : do_insert,
200                   "Message" : msg
201             }
202
                return jsonify(result)
203
204 @app.route("/delete_penduduk_by_nik", methods=['POST'])
205 def delete_penduduk_by_nik():
206       start_time = time.time()
207       do_delete = ''
208       msg = ''
209
210       if request.method == 'POST':
211             data = { 'name': 'NIK',
212                   'type' : 'int',
213                   'value' : request.form['NIK']
214             }
                try :
215                   do_delete =
216 hapus_kependudukan('kependudukan', data)
217                   if do_delete :
218                         msg = "Succeed to DELETE"
```

```
219                     else:
220                             msg = "Failed to DELETE"
221             except Exception, e:
222                             print str(e)
223
224         duration = time.time() - start_time
225         print "Durasi waktu : ", duration
226
227         result = {
228                 "Status" : do_delete,
229                 "Message" : msg
230         }
231
232         return jsonify(result)
233
234 @app.route('/select_all_penduduk')
235 def select_all_penduduk():
236         start_time = time.time()
237         print "[log] Select all from table kependudukan"
238         # Cek status Sinkronisasi
239         last = getLastSync(data_sync_db)
240
241         # 1 : MySQL
242         # 0 : HBase
```

```
                print "Proses sinkronisasi sedang TIDAK
243 BERLANGSUNG"
244             print "Mengambil data dari HBase.."
245             get_from = 0
246             db_data = data_hbase
247       else:
248             print "Proses sinkronisasi sedang BERLANGSUNG"
                print "Mengambil data dari MySQL.."
249             get_from = 1
250             db_data = data_mysql
251
252       daftar_pend = []
253       data = query_db("SELECT * FROM
254 kependudukan",[],False,get_from)
255       data_numrows = len(data)
256       for row in data :
257             pend_tmp = {
258                   'NIK': row[0],
259                   'NAMA_LGKP': row[1],
260                   'JENIS_KLMN': row[2],
261                   'TEMPAT_LHR': row[3],
262                   'TGL_LHR': row[4],
263                   'NOMOR_KK': row[5]
264             }
              daftar_pend.append(pend_tmp)
```

```
265
266         duration = time.time() - start_time
267         print "Durasi waktu : ", duration
268
269         result = {
270               "Lokasi Host" : db_data['host'],
271               "Jenis Database" : db_data['name'],
272               "Jumlah Baris" : data_numrows,
273               "Data Penduduk" : daftar_pend
274               }
275
276         return Response(json.dumps(result, encoding='latin1'),
277 mimetype='application/json')
278
279 @app.route('/select_penduduk_by_nik/<nik>')
    def select_penduduk_by_nik(nik):
280         start_time = time.time()
281         print "[log] Select penduduk by NIK"
282
            last = getLastSync(data_sync_db)
283
            if (last[4]==1):
284               print "Proses sinkronisasi sedang TIDAK
285 BERLANGSUNG"
286               print "Mengambil data dari HBase.."
```

```
287                 get_from = 0
                    db_data = data_hbase
288           else:
289                 print "Proses sinkronisasi sedang BERLANGSUNG"
290                 print "Mengambil data dari MySQL.."
291                 get_from = 1
292                 db_data = data_mysql
293
294           kueri = "SELECT * FROM kependudukan WHERE NIK =
295     {0}".format(nik)
296           data = query_db(kueri,[],True,get_from)
297
298           if (data==None) :
299                 pend = "Data penduduk tidak ada!"
300           else:
301                 pend = {
                          'NIK': data[0],
                          'NAMA_LGKP': data[1],
302                       'JENIS_KLMN': data[2],
303                       'TEMPAT_LHR': data[3],
304                       'TGL_LHR': data[4],
305                       'NOMOR_KK': data[5]
306                 }

307           duration = time.time() - start_time
```

```
308         print "Durasi waktu : ", duration
309
310         result = {
311               "Lokasi Host" : db_data['host'],
                  "Jenis Database" : db_data['name'],
312               "Data Penduduk" : pend
313         }
314
315         return jsonify(result)
316
317   @app.route('/update_penduduk_by_nik', methods=['POST'])
318   def update_penduduk_by_nik():
319         start_time = time.time()
320         print "[log] UPDATE kependudukan data by NIK"
321         # Cek status Sinkronisasi
322
323         if request.method == 'POST':
324               nik = request.form['NIK']
325               nama_lgkp = request.form['NAMA_LGKP']
326               jenis_klmn = request.form['JENIS_KLMN']
                  tempat_lhr = request.form['TEMPAT_LHR']
                  tgl_lhr = request.form['TGL_LHR']
                  nomor_kk = request.form['NOMOR_KK']

                  kueri = ("UPDATE kependudukan "
```

```
                    "SET NAMA_LGKP='{0}', "
                    "JENIS_KLMN={1}, "
                    "TEMPAT_LHR='{2}', "
                    "TGL_LHR='{3}', "
                    "NOMOR KK='{4}' "
                    "WHERE NIK={5}").format(nama_lgkp,
jenis_klmn, tempat_lhr, tgl_lhr, nomor_kk, nik)

            print kueri
            do_update = query_db_insert_update(kueri,1)

            if do_update :
                  msg = 'Succeed to UPDATE table
kependudukan'
            else :
                  msg = "Failed to UPDATE table kependudukan"

            duration = time.time() - start_time
            print "Durasi waktu : ", duration

            result = {
                  "Status" : do_update,
                  "Message" : msg
            }
```

```
        return jsonify(result)

#-------------------------------------------------------------
-------------------------#
@app.route("/sinkron")
def sinkron():
        response = urllib2.urlopen('http://10.151.36.47:5001')
        print "[log] Melakukan sinkronisasi .."
        data = json.load(response)
        # print data
        return jsonify(data)

#-------------------------------------------------------------
-------------------------#
@app.route('/select_all_mysql')
def select_all_mysql():
        result = select_all_kependudukan_mysql()
        return jsonify(result)

@app.route("/select_all_hbase")
def select_all_hbase():
        result = select_all_kependudukan_hbase()
        return jsonify(result)

@app.route("/")
```

```
def hello():
      return "Pengujian SNDA"

#-------------------------------------------------------------
-------------------------#

if __name__ == "__main__":
      app.run(host='0.0.0.0')
      app.config['JSON_AS_ASCII'] = False
```

LAMPIRAN B
TABEL

Tabel B.1 Seluruh Hasil Pengujian Kecepatan Respon SNDA

Nomor Pengujian	**Jenis *Query***	**Kecepatan Respon (detik)**					
		50	**100**	**150**	**200**	**250**	**300**
1	SELECT	X	X	X	X	X	X
	INSERT	0,2	1,3	3,4	6,4	9,9	15,3
	UPDATE	0,4	1,5	3,4	6,4	10,0	16,0
	DELETE	0,2	0,4	0,7	1,0	1,5	1,8
2	SELECT	X	X	X	X	X	X
	INSERT	0,1	0,8	2,9	5,1	9,7	13,8
	UPDATE	0,2	1,0	2,8	6,2	8,9	14,9
	DELETE	0,1	0,3	0,6	0,9	1,4	1,9
3	SELECT	X	X	X	X	X	X
	INSERT	0,3	1,4	3,5	6,4	9,5	15,4
	UPDATE	0,5	1,5	3,6	6,3	9,9	15,9
	DELETE	0,3	0,5	0,7	1,1	1,5	1,6
4	SELECT	X	X	X	X	X	X
	INSERT	0,2	1,3	3,3	6,2	9,7	15,1
	UPDATE	0,5	1,6	3,5	6,2	9,7	15,7
	DELETE	0,3	0,4	0,7	1,1	1,6	1,9
5	SELECT	X	X	X	X	X	X
	INSERT	0,2	1,0	3,0	6,0	9,6	14,6
	UPDATE	0,4	1,7	3,3	6,2	9,8	15,8
	DELETE	0,3	0,4	0,8	1,0	1,5	1,8
6	SELECT	X	X	X	X	X	X
	INSERT	0,3	1,5	3,7	5,9	10,2	14,3
	UPDATE	0,4	1,9	3,5	6,5	10,5	16,2
	DELETE	0,4	0,6	0,9	1,0	1,5	1,8
7	SELECT	X	X	X	X	X	X
	INSERT	0,1	0,9	3,2	6,4	9,7	15,0
	UPDATE	0,5	1,6	3,7	6,8	10,7	16,5
	DELETE	0,2	0,4	0,9	1,1	1,5	1,7

8	SELECT	X	X	X	X	X	X
	INSERT	0,2	1,1	3,3	6,1	9,7	14,8
	UPDATE	0,5	1,4	3,3	6,3	9,9	15,9
	DELETE	0,2	0,3	0,8	1,0	1,6	1,8
9	SELECT	X	X	X	X	X	X
	INSERT	0,3	1,3	3,1	6,2	9,8	14,5
	UPDATE	0,4	1,5	3,5	6,4	9,8	15,9
	DELETE	0,3	0,4	0,7	1,0	1,5	1,8
10	SELECT	X	X	X	X	X	X
	INSERT	0,2	1,6	3,4	6,5	9,6	14,2
	UPDATE	0,3	1,6	3,3	6,3	9,6	15,8
	DELETE	0,3	0,4	0,7	1,1	1,6	1,8

(X) = Terjadi setidaknya satu *error* saat pengujian klien

Tabel B.2 Seluruh Hasil Pengujian Performa SNDA

Nomor Pengujian	**Jenis *Query***	**Banyak Klien**	**Rata-rata Memori (MB)**	**Rata-Rata Prosesor (%)**
1	SELECT	50	1685	98,3
		100	1687	100,5
		150	1687	100,4
		200	1687	100,6
		250	1687	100,7
		300	1687	100,7
	INSERT	50	667	25,1
		100	672	35,2
		150	673	41,5
		200	675	44,3
		250	681	45,8
		300	685	48,7
	UPDATE	50	638	28,2
		100	640	40,4
		150	645	45,2
		200	648	47,9

		250	655	48,8
		300	668	50,1
	DELETE	50	671	1,6
		100	672	1,8
		150	672	1,7
		200	673	2,0
		250	674	2,3
		300	674	3,7
2	SELECT	50	1684	98,4
		100	1687	100,6
		150	1687	100,4
		200	1687	100,6
		250	1687	100,7
		300	1687	100,7
	INSERT	50	668	25,3
		100	671	37,8
		150	673	43,1
		200	675	40,8
		250	681	41,8
		300	685	43,2
	UPDATE	50	615	26,4
		100	618	35,7
		150	622	42,2
		200	628	43,9
		250	640	46,8
		300	652	48,1
	DELETE	50	668	1,9
		100	669	2,1
		150	672	2,1
		200	673	2,2
		250	673	2,4
		300	674	3,9
3	SELECT	50	1686	98,5
		100	1687	100,5
		150	1687	100,4
		200	1687	100,7

		250	1687	100,7
		300	1687	100,7
	INSERT	50	664	25,8
		100	670	35,3
		150	673	40,5
		200	675	41,2
		250	681	43,5
		300	685	47,7
	UPDATE	50	631	27,5
		100	635	39,4
		150	636	44,7
		200	640	46,4
		250	650	47,2
		300	664	49,5
	DELETE	50	671	1,3
		100	671	1,5
		150	673	1,6
		200	673	1,8
		250	674	2,1
		300	674	3,4
4	SELECT	50	1685	98,5
		100	1687	100,6
		150	1687	100,5
		200	1687	100,6
		250	1687	100,7
		300	1687	100,7
	INSERT	50	665	25,0
		100	669	35,5
		150	673	40,7
		200	675	42,7
		250	681	43,3
		300	685	44,6
	UPDATE	50	632	27,3
		100	630	37,9
		150	636	43,2
		200	639	47,9

		250	648	43,8
		300	661	48,1
	DELETE	50	670	1,6
		100	672	1,8
		150	674	1,7
		200	673	2,0
		250	674	2,3
		300	674	3,7
5	SELECT	50	1685	98,2
		100	1687	100,5
		150	1687	100,4
		200	1687	100,7
		250	1687	100,7
		300	1687	100,7
	INSERT	50	668	24,8
		100	668	33,6
		150	673	41,5
		200	675	42,5
		250	681	43,6
		300	685	44,5
	UPDATE	50	628	27,4
		100	634	38,6
		150	636	44,4
		200	643	47,9
		250	650	48,8
		300	662	49,5
	DELETE	50	669	1,5
		100	670	1,9
		150	672	1,6
		200	673	1,9
		250	674	2,4
		300	674	3,6
6	SELECT	50	1686	98,0
		100	1687	100,5
		150	1687	100,4
		200	1687	100,6

		250	1687	100,7
		300	1687	100,7
	INSERT	50	665	25,3
		100	670	36,1
		150	673	41,4
		200	675	43,2
		250	681	43,7
		300	685	47,5
	UPDATE	50	626	28,0
		100	633	38,1
		150	636	42,3
		200	641	44,5
		250	649	45,8
		300	662	48,1
	DELETE	50	670	1,7
		100	673	1,8
		150	674	1,9
		200	673	2,1
		250	673	2,2
		300	674	3,5
7	SELECT	50	1685	98,0
		100	1687	100,5
		150	1687	100,5
		200	1687	100,6
		250	1687	100,7
		300	1687	100,7
	INSERT	50	667	24,9
		100	671	35,6
		150	673	41,1
		200	675	41,2
		250	681	43,8
		300	685	46,5
	UPDATE	50	630	26,8
		100	633	38,2
		150	636	44,1
		200	640	46,8

		250	649	48,1
		300	662	49,3
	DELETE	50	670	1,5
		100	672	1,9
		150	673	1,6
		200	674	2,1
		250	674	2,2
		300	674	3,4
8	SELECT	50	1686	98,2
		100	1687	100,6
		150	1687	100,4
		200	1687	100,6
		250	1687	100,7
		300	1687	100,7
	INSERT	50	667	25,0
		100	670	35,8
		150	673	41,3
		200	675	42,5
		250	681	43,8
		300	685	46,8
	UPDATE	50	630	27,4
		100	634	38,1
		150	636	43,2
		200	640	45,9
		250	650	46,8
		300	663	48,1
	DELETE	50	670	1,5
		100	671	1,7
		150	674	1,7
		200	673	2,0
		250	674	2,4
		300	674	3,7
9	SELECT	50	1685	98,3
		100	1687	100,5
		150	1687	100,4
		200	1687	100,6

		250	1687	100,7
		300	1687	100,7
	INSERT	50	668	25,1
		100	670	35,6
		150	673	40,6
		200	675	42,4
		250	681	43,8
		300	685	46,9
	UPDATE	50	629	27,7
		100	635	38,2
		150	636	44,2
		200	637	46,9
		250	645	47,8
		300	659	49,1
	DELETE	50	670	1,6
		100	673	1,9
		150	673	1,7
		200	674	2,0
		250	674	2,3
		300	674	3,6
10	SELECT	50	1685	98,5
		100	1687	100,5
		150	1687	100,4
		200	1687	100,6
		250	1687	100,7
		300	1687	100,7
	INSERT	50	667	25,2
		100	670	35,3
		150	673	40,8
		200	675	42,3
		250	681	43,8
		300	685	46,1
	UPDATE	50	629	27,8
		100	633	38,2
		150	636	44,0
		200	642	46,6

		250	647	47,5
		300	660	48,8
	DELETE	50	670	1,6
		100	672	1,9
		150	673	1,7
		200	674	2,0
		250	674	2,3
		300	674	3,7

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

Muhamad Rizki Prawiraatmaja, lahir pada 26 Oktober 1994 di Jakarta. Penulis menempuh pendidikan mulai dari SD Pertiwi (2001-2007), SMPN 1 Bogor (2007-2010), SMAN 1 Bogor (2010-2013), dan S1 Teknik Informatika ITS (2013-2018). Memiliki beberapa hobi antara lain membaca buku dan mendengarkan musik. Penulis pernah menjadi asisten dosen di mata kuliah Manajemen Basis Data. Selain menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain Staff Departemen Dana dan Usaha pada tahun ke-2 dan Staff Ahli Departemen Dana dan Usaha Keluarga Muslim Informatika pada tahun ke-3.

Kritik dan saran sangat diharapkan guna peningkatan kualitas dan penulisan selanjutnya. Untuk itu, silahkan kirim kritik dan saran ke: rizkiprawira26@gmail.com.