PERBANDINGAN KINERJA CASSANDRA DAN MONGODB SEBAGAI *BACKEND* IOT DATA *STORAGE*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Adam Kukuh Kurniawan
NIM: 135150207111056

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

PERBANDINGAN KINERJA CASSANDRA DAN MONGODB SEBAGAI BACKEND IOT DATA STORAGE

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Adam Kukuh Kurniawan
NIM: 135150207111056

Skripsi ini telah diuji dan dinyatakan lulus pada
31 Juli 2018
Telah diperiksa dan disetujui oleh:

Pembimbing I	Pembimbing II
Eko Sakti Pramukantoro, S.Kom., M.Kom	Ir. Primantara Hari Trisnawan, M.Sc.
NIK. 201102 860805 1 001	NIP. 19680912 199403 1 002

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP. 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 6 Agustus 2018

Adam Kukuh Kurniawan

NIM: 135150207111056

KATA PENGANTAR

Assalamualaikum warahmatullahi wabarakatuh.

Puji syukur penulis panjatkan kehadirat Allah SWT yang telah memberikan rahmat dan hidayah-Nya sehingga laporan skripsi yang berjudul "PERBANDINGAN KINERJA *CASSANDRA* DAN *MONGODB* SEBAGAI *BACKEND IOT DATA STORAGE*" dapat terselesaikan.

Selama pengerjaan laporan skripsi penulis mendapat banyak pelajaran dan pengalaman baru. Penulis juga banyak mengaplikasikan ilmu yang didapat selama menduduki bangku perkuliahan ke dala laporan skripsi ini. Penulis menyadari bahwa laporan skripsi ini tidak akan berhasil tanpa dukungan, doa, dan bantuan dari beberapa pihak. Oleh karena itu, penulis ingin menyampaikan rasa erima kasih dan rasa hormat kepada :

1. Bapak Eko Sakti Pramukantoro, S.Kom.,M.Kom., dan bapak Ir. Primantara Hari Trisnawan, M.Sc., selaku dosen pembing skripsi penulis yang begitu sabar membimbing penulis juga memberikan arahan dengan sangat baik kepada penulis sehingga dapat menyelesaikan skripsi ini.
2. Kepada kedua orang tua saya yang tidak henti-hentinya memberikan doa dan dukungan selama proses menempuh pendidikan di Malang termasuk saat proses pengerjaan skripsi ini.
3. Bapak Wayan Firdaus Mahmudy, S.Si.,M.T., Ph.D., selaku Dekan Fakultas Ilmu Komputer
4. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D., selaku Ketua Jurusan Teknik Informatika
5. Bapak Agus Wahyu Widodo, S.T., M.Cs., selaku Ketua Program Studi Teknik Informatika
6. Ocki Bagus Setyawan, S.Kom., yang selalu meluangkan waktu untuk membantu dan berbagi ilmu sehingga laporan skripsi ini dapat terselesaikan
7. Ajeng Dearista Wulansari, S.T., yang selalu bersedia memberikan bantuan, saran, dan semangat selama proses penyelesaian skripsi ini.
8. Abdul Aziz Hadyansah yang selalu bersedia memberikan bantuan untuk menyelesaikan proses penyelesaian skripsi ini
9. Teman – Teman Kandang singa, Susiawan Hastomo Ajie dan M. Reza Mawardi yang selalu bersedia membantu selama proses penyelesaian skripsi ini.

Penulis mengakui bahwa dalam penyusunan skripsi ini masih terdapat banyak kekurangan, sehingga kritik yang membangun dan juga saran sangat penulis butuhkan. Akhir kata penulis berharap agar skripsi ini bisa membawa manfaat bagi semua pihak yang menggunakannya.

Wassalamualaikum warahmatullahi wabarakatuh.

Malang, 6 Agustus 2018

Penulis

adhamkukuh@gmail.com

ABSTRAK

Adam Kukuh Kurniawan, Perbandingan Kinerja *Cassandra* dan *MongoDB* sebagai *Backend IoT data storage*

Dosen Pembimbing : Eko Sakti P., S.Kom., M.Kom. dan Ir. Primantara H T., M.Sc

Solusi media penyimpanan untuk menyimpan data yang beragam adalah menggunakan NoSQL. Pada penelitian sebelumnya telah dikembangkan *framework* media penyimpanan data IoT untuk mengatasi permasalahan data IoT yang besar dan beragam dengan menggunakan NoSQL MongoDB dan GridFS sebagai media penyimpanan datanya. Akan tetapi, saat ini terdapat banyak NoSQL database dengan mekanisme implementasi dan karakteristik penyimpanan yang berbeda-beda. Hal tersebut yang membawa tantangan dalam pemilihan NoSQL database yang digunakan sebagai media penyimpanan data IoT. Pada penelitian ini akan diusulkan media penyimpanan data IoT dengan menggunakan NoSQL Cassandra karena mekanisme implementasi dan karakteristik penyimpanannya berbeda dengan NoSQL MongoDB. Pengujian dilakukan dari segi fungsionalitas pada Cassandra dalam menyimpan data dari node sensor, serta dari segi kinerja Cassandra dan MongoDB dalam melakukan operasi insert data string dan data gambar dengan menggunakan parameter *runtime*, throughput, CPU usage, memory usage dan *disk I/O*. Dari hasil pengujian fungsionalitas, Cassandra dapat menyimpan data yang heterogen dari node sensor. Untuk operasi insert data string, MongoDB memiliki nilai *runtime* paling baik sebesar 121,2 detik, *throughput* paling baik sebesar 1236,7 ops/detik, CPU usage paling baik dengan nilai 9,7 % dan *disk I/O* paling baik sebesar 479,8 Kb/detik. Sedangkan untuk operasi insert data file, Cassandra memiliki nilai *runtime* terbaik sebesar 86,4 detik, *throughput* terbaik sebesar 115,8 ops/detik, dan *disk i/o* terbaik sebesar 115792,4 KB/detik.

Kata kunci: *Internet of Things*, *NoSQL*, *Cassandra*, MongoDB

ABSTRACT

Adam Kukuh Kurniawan, Comparison of Cassandra and MongoDB Performance as Backend IoT data storage

Supervisors : Eko Sakti P., S.Kom., M.Kom. dan Ir. Primantara H T., M.Sc.

The storage solutions for keeping a variety of data is using NoSQL. In the previous research, IoT data storage framework has been developed to solve the problems of large and diverse IoT data by using NoSQL MongoDB and GridFS as the data storage media. But currently there are many NoSQL databases with different implementation mechanisms and storage characteristics. It brings challenges in the NoSQL databases selections that are used as IoT data storage media. This research proposes IoT data storage media using NoSQL Cassandra. The researcher chose NoSQL Cassandra because the implementation mechanism and characteristics of the sessions differ from MongoDB NoSQL. The test is performed in terms of functionality on Cassandra in storing data from sensor nodes, as well as in terms of performance of Cassandra and MongoDB in performing data text and file insertion using Runtime, Throughput, CPU Usage, Memory Usage and DISK I/O parameters. From the results of functionality testing, Cassandra can store heterogeneous data from sensor nodes. For data text insertion, MongoDB has Runtime, Throughput, CPU Usage and disk I/O values best compared to Cassandra (runtime 121,2 second, throughput at 1236,7 ops/s, CPU usage 9,7 % and disk I/O 479,8 KB/s). As for data file insertion, Cassandra has better Runtime, Throughput, and disk I/O values compared to MongoDB (runtime 86,4 second, throughput at 115,8 ops/s, and disk I/O at 115792,4 KB/s)

Keywords : Internet of Things, *NoSQL*, *Cassandra*, MongoDB, Data *Storage*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Perkembangan *Internet of Things* (IoT) kini diberbagai lini aspek kehidupan manusia, telah berkembang pesat seiring berjalannya waktu. Dengan demikian, sebuah perangkat yang terkoneksi melalui internet, dapat disebut sebagai sebuah *things* atau objek dalam sebuah lingkungan IoT. Adapun wujud *things* dapat berupa sensor, hingga peralatan rumah tangga. Tujuan utama dari IoT adalah agar tiap objek fisik dapat berkomunikasi satu sama lain sehingga dapat bertukar data yang disimpan dalam sebuah media penyimpanan untuk dilakukan proses komputasi sesuai dengan kebutuhan pengguna. Untuk melakukan pengumpulan data tersebut, media penyimpanan atau *storage* adalah hal yang sangat penting untuk dipertimbangkan kinerjanya, hingga mempengaruhi efektifitas dari penggunaannya. (Zafar. dkk., 2017).

Terdapat beberapa tantangan dalam membangun media penyimpanan data IoT, seperti *volume* data yang besar, bentuk dan format datanya yang beragam. Pada penelitian (Pramukantoro. dkk., 2017) telah dikembangkan sebuah *framework* untuk menyimpan dan mengakses data sensor dari suatu perangkat IoT ke dalam sistem basis data MongoDB dan GridFS. Skenario dalam menguji *framework* tersebut yang dilakukan oleh peneliti sebelumnya meliputi: *functional testing*, *scalability testing* dan *response time testing* dengan *IoT Apps*. Hasil yang didapat dari *functional* testing adalah *framework* tersebut dapat mengirim dan menyimpan data yang beragam dalam jumlah yang besar. Sedangkan hasil yang didapat dari *non functional testing* memberi kesimpulan bahwa *framework* yang dikembangkan dapat menerima 443 data /detik dari IGD (*Internet Gateway Device*), dapat mengirimkan 173 data/detik ke *IoT Apps* dan *response time* yang didapat dibawah 1 detik.

Pada penelitian (Pramukantoro, dkk., 2017) telah digunakan NoSQL MongoDB sebagai media penyimpanan datanya, akan tetapi pada saat ini telah terdapat lebih dari 225 jenis NoSQL *database*, dimana setiap tipe NoSQL *database* memiliki mekanisme implementasi, karakteristik penyimpanan, konfigurasi serta metode optimasi yang berbeda-beda. Hal inilah yang membawa lebih banyak tantangan dalam pemilihan NoSQL *database* yang akan digunakan sebagai media penyimpanan data (Enqingtang & Fan, 2016). Maka dari itu, NoSQL MongoDB yang digunakan pada penelitian (Pramukantoro, dkk., 2017) perlu dibandingkan dengan NoSQL *database* yang lain.

Pada penelitian ini, akan diusulkan media penyimpanan data IoT dengan menggunakan NoSQL Cassandra. NoSQL Cassandra dipilih karena pada hasil penelitian yang dilakukan (Kabakus & Kara, 2016) menunjukkan bahwa NoSQL Cassandra memiliki kinerja yang baik ditinjau dari parameter *memory usage* dalam melakukan operasi *insert*, operasi *read* dan operasi *delete*. Sehingga, pada penelitian ini akan membandingkan kinerja NoSQL Cassandra dan NoSQL MongoDB pada lingkungan *framework* yang telah dikembangkan oleh (Pramukantoro, dkk., 2017) dalam melakukan operasi *insert data*. Parameter uji yang digunakan pada penelitian

ini meliputi parameter *runtime, throughput, memory usage, CPU usage* dan *Disk I/O*. Selain itu, NoSQL Cassandra yang diusulkan pada penelitian ini akan diuji fungsionalitasnya dalam hal menyimpan dan menampilkan data yang beragam. Dari penelitian ini diharapkan dapat memberikan informasi mengenai kinerja NoSQL *database* yang diteliti sebagai pertimbangan bagi orang-orang yang ingin membangun sebuah sistem yang membutuhkan media penyimpanan dengan data yang beragam serta menjadikan hasil kinerja dari NoSQL Cassandra dan NoSQL MongoDB sebagai pembanding dan acuan untuk penelitian selanjutnya.

1.2 Rumusan masalah

Adapun rumusan masalah yang mendasari penulis dalam melakukan penelitian adalah:

1. Bagaimana membangun IoT data *storage* dengan NoSQL Cassandra?
2. Bagaimana menguji kinerja dari NoSQL Cassandra dan NoSQL MongoDB?
3. Bagaimana hasil perbandingan kinerja data *storage dengan* NoSQL Cassandra dan NoSQL MongoDB?

1.3 Tujuan

Sesuai dengan rumusan masalah yang telah dipaparkan diatas, tujuan dari penelitian ini adalah:

1. Membangun IoT data *storage* dengan menggunakan NoSQL Cassandra.
2. Menentukan cara untuk menguji kinerja dari NoSQL Cassandra dan NoSQL MongoDB.
3. Mengetahui hasil dari perbandingan kinerja NoSQL Cassandra dan NoSQL MongoDB sebagai *backend* IoT data *storage*.

1.4 Manfaat

Berdasarkan tujuan dan latar belakang diatas, manfaat yang diharapkan dari penelitian ini adalah:

1. Memberikan informasi mengenai kinerja *database* yang diteliti sebagai pertimbangan bagi orang-orang yang ingin membangun sebuah sistem yang membutuhkan media penyimpanan dengan data yang beragam.
2. Untuk menjadikan hasil kinerja dari NoSQL Cassandra dan MongoDB sebagai pembanding dan acuan untuk penelitian selanjutnya.

1.5 Batasan masalah

Berdasarkan latarbelakang dan rumusan masalah yang telah dipaparkan sebelumnya, penelitian ini memiliki beberapa batasan agar tidak terlalu meluas dan fokus untuk menjawab rumusan masalah. Berikut batasan masalah dalam penelitian ini :

1. Lingkungan penelitian yang digunakan adalah lingkungan penelitian yang dikembangkan oleh Pramukantoro *dkk*(2017).
2. Penelitian ini hanya mengganti *database* NoSQL MongoDB pada penelitian yang dikembangkan oleh Pramukantoro *dkk* (2017) dengan NoSQL Cassandra
3. Lingkungan pengujian kinerja *database* dilakukan pada *Virtual Private Server* yang ada di FILKOM dengan spesifikasi dengan spesifikasi *Virtual Private Server* yaitu RAM 4 GB, CPU 2 Core dan Har*disk* 40 GB.
4. Pengujian yang dilakukan pada penelitian ini lebih ditekankan pada sisi kinerja NoSQL databasenya.
5. Data yang digunakan berasal dari NodeMCU dan kamera.

1.6 Sistematika pembahasan

Bagian ini berisi struktur skripsi di mulai dari Bab Pendahuluan sampai Bab Pentup dan deskripsi singkat dari masing-masing bab. Diharapkan bagian ini dapat membantu memahami sistematika pembahasan isi dalam skripsi ini. Sistematika penulisan skripsi ini adalah sebagai berikut:

BAB I: PENDAHULUAN

Bab ini membahas tentang latarbelakang masalah mengapa penelitian ini dilakukan sehingga menghasilkan pertanyaan rumusan masalah, dan pernyataan tujuan dilakukannya penelitian ini. Selain itu, pada bab ini juga terdapat butir-butir pernyataan batasan masalah, dan sistematika pembasahan agar alur penelitian dapat disusun dengan sistematik.

BAB II: LANDASAN KEPUSTAKAAN

Bab ini membahas tentang kajian pustaka beserta dasar teori yang dibutuhkan untuk menunjang penelitian. Dalam landasan kepustakaan terdapat landasan teori dari berbagai sumber pustaka yang terkait. Kajian pustaka juga menjelaskan secara umum penelitian-penelitian terdahulu yang berhubungan dengan topik skripsi untuk mendapatkan literature pendukung.

BAB III: METODOLOGI

Bab ini membahas tentang langkah-langkah sistematik bagaimana penelitian ini dilakukan. Metodologi menjelaskan berbagai langkah yang umumnya digunakan pada saat penelitian, seperti studi literartur, analisis kebutuhan, perancangan lingkungan uji, pengujian dan pengambilan data, hasil dan pembahasan, serta kesimpulan dan saran.

BAB IV: PEMBANGUNAN LINGKUNGAN UJI

Bab ini membahas tentang kebutuhan – kebutuhan yang digunakan untuk membangun lingkungan pengujian, menjelaskan perancangan yang akan

digunakan untuk proses pengujian serta implementasi dari perancangan yang dibuat.

BAB V: PENGUJIAN DAN PENGAMBILAN DATA

Bab ini membahas tentang pengujian yang akan dilakukan berdasarkan perancangan skenario pengujian yang telah dirancang, serta menjelaskan cara pengambilan data dari parameter uji yang digunakan.

BAB VI: HASIL DAN PEMBAHASAN

Bab ini membahas tentang hasil pengujian berdasarkan skenario pengujian yang telah dibuat, serta berisi pembahasan hasil pengujian untuk menjawab pertanyaan yang ada pada rumusan masalah penelitian.

BAB VII: PENUTUP

Bab ini memuat kesimpulan yang diperoleh dari penelitian yang telah dilakukan untuk menjawab rumusan masalah serta memuat saran-saran untuk penelitian pengembangan selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Bab berikut membahas tentang kajian dari kumpulan literatur yang berisi uraian dan pembahasan tentang kajian pustaka terdahulu dan dasar teori sesuai dengan kaidah konsep IoT. Hasil dari kajian literatur tersebut dirangkum dan dijadikan dasar peneliti untuk melaksanakan penelitian.

2.1 Kajian Pustaka

Kajian pustaka dalam bab ini membahas tentang literatur dari penelitian terdahulu yang terkait dengan penelitian yang akan dilakukan. Kajian pustaka dilakukan untuk mengetahui perbedaan penelitian ini dengan yang sudah ada, serta untuk membantu kepustakaan dan refrensi dalam penelitian ini. Berikut pemaparan beberapa penelitian terkait pada Tabel 2.1.

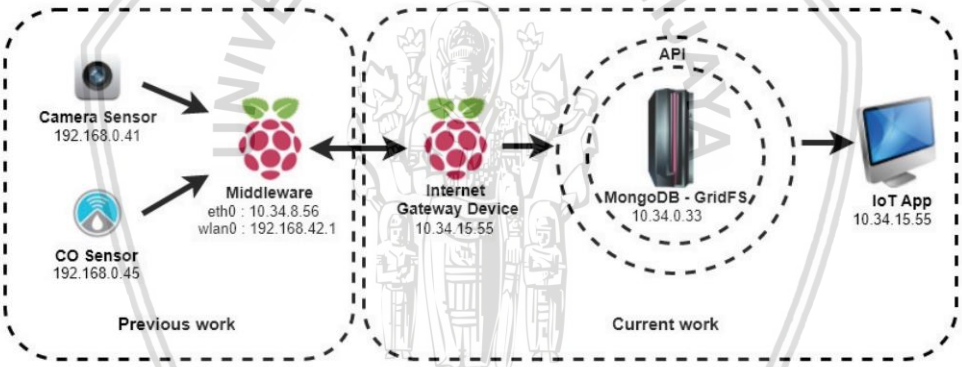
Tabel 2.1 Penelitian Terkait

No	Nama Penulis, Tahun, Judul	Perbedaan	
		Penelitian terdahulu	Rencana penelitian
1.	EnqingTang &Yushun Fan. (2016). *Performance Comparison between Five NoSQL Databases.*	- Data uji yang digunakan menggunakan data *dummy* dari *tools* YCSB - Parameter uji yang digunakan yaitu *throughput* dan *runtime*.	- Data uji yang akan digunakan berupa data *string* dan data gambar. - Parameter uji yang digunakan meliputi *runtime, throughput, memory usage, CPU usage* dan *disk I/O.*

2.	Jan Sipke van der Veen, Bram van der Waaj, Robert J.M. (2012). *Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual.*	- Parameter uji yang digunakan yaitu *throughput*.	- Menguji kinerja penyimpanan data dengan menggunakan 2 tipe *NoSQL database* (*Cassandra* dan *MongoDB*) - Data uji yang digunakan berupa data *string* dan data gamba - Parameter uji yang digunakan *meliputi runtime, throughput, memory usage, CPU usage* dan *disk I/O.*
3.	Pramukantoro, Eko. Dkk. (2017). *Topic Based IoT Data Storage Framework for Heterogeneous Sensor Data*.	- Membangun IoT *data storage* dengan menggunakan *MongoDB*.	- Membangun IoT *data storage* menggunakan *Cassandra*. - Menguji kinerja NoSQL *Cassandra* dan NoSQL *MongoDB* dalam melakukan operasi insert data. - Parameter uji yang digunakan *meliputi runtime, throughput, memory usage, CPU usage* dan *disk I/O.*

4	Abdullah Talha Kabakus & Resul Kara.(2016). A performance evaluation of in-memory databases	-Parameter uji yang digunakan yaitu memory usage	- Parameter uji yang digunakan *meliputi runtime, throughput, memory usage, CPU usage* dan *disk I/O.*

Pada penelitian Pramukantoro dkk tahun 2017, Telah dikembangankan sebuah *IoT data storage framework* untuk menyimpan data yang beragam dari *node sensor*. *Framework* tersebut menggunakan *MongoDB* dan *GridFS* sebagai media penyimpanan datanya. Adapun topologi sistem dapat dilihat pada Gambar 2.1.



Gambar 2.1 Topologi Sistem

Sumber: (Pramukantoro *dkk*., 2017)

Untuk pengujian yang dilakukan pada penelitian oleh Pramukantoro dkk 2017 adalah pengujian skalabilitas *API webservice*, *response time* pada IoT *Application* dan pengujian fungsional pada sistem yang dibuat. Pada penelitian ini, akan dibangun IoT data *storage* dengan menggunakan NoSQL Cassandra sebagai media penyimpanannya. Kemudian akan dibandingkan kinerja dari *data storage* menggunakan NoSQL Cassandra, dengan *data storage* menggunakan NoSQL MongoDB yang digunakan pada penelitian Pramukantoro dkk tahun 2017 dalam melakukan operasi *insert* data.

2.2 Dasar Teori

Berikut adalah pemaparan dasar teori yang mendukung dalam memahami penelitian ini, antara lain:

2.2.1 *Database*

Database atau markas data adalah sekumpulan data. Data ialah fakta yang dapat diketahui, mampu direkam dan mempunyai makna tersembunyi atau tersirat jika dipetakan dengan baik. Suatu *database* pada umumnya menggambarkan beberapa objek *real* di dunia nyata dan wujudnya digunakan untuk tujuan tertentu oleh satu atau sekelompok pengguna. Untuk mengimplementasikan dan menjaga sebuah *database* secara otomatis, pengguna dapat menggunakan sebuah *Database Management System* (*DBMS*). *DBMS* sendiri merupakan sebuah perangkat lunak yang berfungsi untuk mempermudah pengguna dalam manajemen sebuah *database*.

Misalkan terdapat sebuah tabel kumpulan data, yang terdiri nama, nomor telepon, dan alamat dari keluarga kita. Kumpulan data tersebut pada umumnya dapat kita simpan pada sebuah buku alamat yang memiliki indeks atau mungkin disimpan pada perangkat keras pada komputer pribadi dan perangkat lunak seperti *Microsoft Access* atau *Excel*. Kumpulan data dengan arti tersembunyi tersebut dapat disebut sebagai *database*, sedangkan perangkat lunak seperti *Microsoft Access* dan *Excel* dapat disebut sebagai *DBMS* (Elmasri and Navathe, 2011).

2.2.2 Not Only SQL (*NoSQL*)

Database Not Only SQL (NoSQL) adalah jenis terbaru dari *database* non-relasional. Pada perkembangannya mendapat perhatian dari pengembang dan perusahaan dengan pesat. *NoSQL Database* mampu menangani hampir seluruh kebutuhan penyimpanan dan akses data pada sebuah sistem perusahaan/ *web application* yang beragam secara *realiable* dan efisien. *NoSQL Database* memiliki fitur tambahan yang membedakannya dengan *Relational Database Management Systems (RDBMS)*, seperti tidak memerlukan skema *database* statis atau tetap pada definisinya dan mempunyai skalabilitas horizontal yang mudah daam pengaturannya.

NoSQL Database dibentuk dan dibangun untuk memfasilitasi kebutuhan seperti skalibilitas *shared-nothing* horizontal yang dibuktikan pada pendistribusian data melalui peladen yang beragam. Dengan demikian, pengembangan *NoSQL Database* dapat dilakukan dengan skalabilitas (penambahan beban) yang mudah dan dapat melakukan distribusi data secara efisien. Dan pada kondisi jumlah peladen yang semakin meningkat, sistem tersebut justru mampu melayani *request data* secara lebih efisien. Seluruh permintaan dikerjakan dengan cara paralel, sehingga dapat menghasilkan nilai *Throughput* lebih tinggi serta waktu eksekusi *query* yang lebih rendah (Abramova, Bernardino and Furtado, 2014).

berdasarkan model penyimpanannya, terdapat 4 tipe *NoSQL Database,* yaitu:

1. *Key-Value based*, yaitu data disimpan menyerupai sebuah kunci dengan nilai yang saling berpasangan, yang dimaksudkan adalah pada tipe ini terdapat 2 bagian, yakni sebuah *string* yang menunjukkan sebuah kunci (*key*) dan data aktual berupa nilai (*value*), sehingga akan membentuk sebuah pasangan kunci berisi (*key-value*). Adapun contoh *database* yang menggunakan konsep *key-value* based adalah *DynamoDB* dan *Aerospike*.

2. *Column-oriented based*, yakni data disimpan pada beberapa kolom. Walau menggunakan konsep kolom, akan tetapi tipe tersebut tidak sama dengan konsep relasional kolom *database* yang ada. Data disimpan melainkan pada sebuah arsitektur distribusi yang besar. Pada tipe ini, setiap *key* terkoneksi antara satu atau lebih atribut (kolom). Adapun contoh *database* yang mengadopsi konsep *Column-oriented* adalah *HBase* dan *Cassandra*.
3. *Document-store based*, yakni data disimpan dalam pada sebuah dokumen. Dokumen yang digunakan pada *document-store database* menyerupai *record* pada bentuk *relational database*. Format dokumen yang disimpan dapat berupa *PDF*, *XML*, *JSON* dan lain sebagainya. Dokumen tersebut dapat terdiri dari *key-value, unique key* digunakan untuk menggambarkan dokumen. *Key* dapat berbentuk *string* sederhana atau merujuk pada sebuah direktori, *URI* atau *path*. adapun contoh *database* yang mengadopsi konsep *document-store* adalah *CouchDB* dan *MongoDB*.
4. *Graph based*, merupakan konsep penyimpanan data yang diwujudkan berupa sebuah *graph*. Suatu *graph* terdiri dari *node* dan *edge*. *Node* merepresentasikan suatu objek dan *edge* merepresentasikan suatu hubungan antar objek. Setiap *node* memiliki *direct pointer* atau titik yang dituju, titik tersebut adalah *node* yang berdekatan. *Database Graph* mempunyai fitur *schema-less* serta penyimpanan yang cukup efisien untuk data semi terstruktur. Adapun contoh *database* yang mengadopsi konsep *graph* adalah *Neo4j*.
5. *Object oriented based*, yaitu adalah data yang disimpan dan digambarkan sebagai suatu objek. Tipe tersebut menggabungkan antara konsep *Object Oriented Programming (OOP),* dengan konsep dasar *database*. Objek data dapat disimpan dengan fitur yang menyerupai dengan *OOP* seperti *encapsulation, polymorphism* dan *inheritance*. Sebuah c*lass* dapat disama artikan dengan sebuah tabel. Setiap objek memiliki suatu *identifier* yang berfungsi untuk membedakan objek satu yang lainnya. Adapun contoh *database* yang mengadopsi konsep ini adalah *db4o* (Nayak, Poriya and Poojary, 2013).

2.2.3 Internet of Things (*IoT*)

Berdasarkan analisis dari *McKinsey Global Institute*, *Internet of Things* atau IoT adalah suatu konsep konfigurasi perangkat beserta pengaturannya, yang memungkinkan pengguna untuk melakukan koneksi pada mesin, peralatan, dan objek fisik lainnya, dengan sensor yang dapat terhubung pada jaringan dan aktuator untuk mendapatkan nilai data dan mengelola kinerjanya secara mandiri, sehingga setiap mesin dapat bekerjasama, bahkan dapat berperilaku berdasarkan informasi terbaru yang diperoleh secara mandiri. IoT merupakan sebuah konsep interkoneksi yang unik antara *embedded computing devices* dalam infrastruktur internet yang telah ada. Terdapat sebuah artikel penelitian mengenai *Internet of Things in 2020* menjelaskan bahwa *Internet of Things* merupakan sebuah kondisi ketika suatu benda memiliki jati diri identitas, mampu beroperasi secara independen, serta mampu berhubungan dengan sosial, lingkungan, bahkan pengguna (Atzori, Iera, & Morabito, 2010).

Maka demikian, dapat diambil konkusinya bahwa konsep *Internet of Things (IoT)* dapat memungkinkan pengguna untuk membangun sebuah hubungan mesin antar mesin, sehingga semua mesin tersebut mamput berinteraksi dan beroperasi secara mandiri sesuai dengan nilai yang didapat dan dikalkulasi dengan mandiri. Tujuannya agar manusia mampu berkomunikasi dengan alat atau mesin secara mudah, bahkan agar benda pun dapat berkomunikasi dengan benda lainnya. Adapun mekanisme kerja dari *Internet of Things* cukup mudah. Setiap benda harus mempunyai sebuah *IP Address* atau alamat IP. *IP Address* merupakan suatu identitas pada jaringan yang membuat benda tersebut memiliki identitas unik yang dapat dibedakan dengan yang lain, sehingga benda tersebut dapat dipanggil dan dapat diberi perintah dari benda lain yang ada pada jaringan yang sama.

2.2.4 Web Service

Sesuai pernyatakan Microsoft pada tahun 2000, bahwa *web service* adalah tahap ketiga dari tahapan evolusi ASP (*Application Service Provider*), yang mana ditahapan pertama difokuskan pada ketersediaan aplikasi *desktop,* sedangkan pada tahapan kedua dimaksudkan pada penyediaan aplikasi berbasis *client-server*. Pada tahapan ketiga, komponen perangkat pembangun atau *building blocks software* disediakan sebagai *service* dan dapat diakses melalui jaringan internet untuk kemudian diintegrasikan dengan aplikasi lain. Menurut Kreger (2001) *web service* dapat diartikan sebagai suatu antar muka (*interface*) yang merepresentasikan sekumpulan atau banyak operasi yang dapat diakses via jaringan, contohnya adalah internet, dalam format XML.

Sependapat dengan Kreger, Manes (2001) menyimpulkan bahwa *web service* adalah sepotong atau sebagian informasi atau proses yang kemampuannya dan fiturya dapat diakses oleh siapapun, kapanpun, dengan piranti apapun, serta tidak tergantung pada sistem operasi atau bahasa pemrograman yang digunakan. *Web service* dapat dibangun menggunakan bahasa pemrograman apapun dan dapat diwujudkan pada *platform* manapun. Pengguna dapat membuat suatu *web service* pada Windows 2000 dan mengoperasikannya pada Windows, Linux, Unix, Mac, PalmOS dan WinCE (Hamids, 2000). Hal tersebut sangat memungkinkan, karena *web service* mampu berhubungan dengan sebuah standar format data yang bersifat universal, yakni XML dengan menggunakan protokol SOAP. Karena *web service* menggunakan format data XML, maka *web service* pun secara langsung bersifat *multi-tier* dari XML, sehingga sangat mungkin untuk berintegrasi antar *web service* atau aplikasi (Microsoft, 2001).

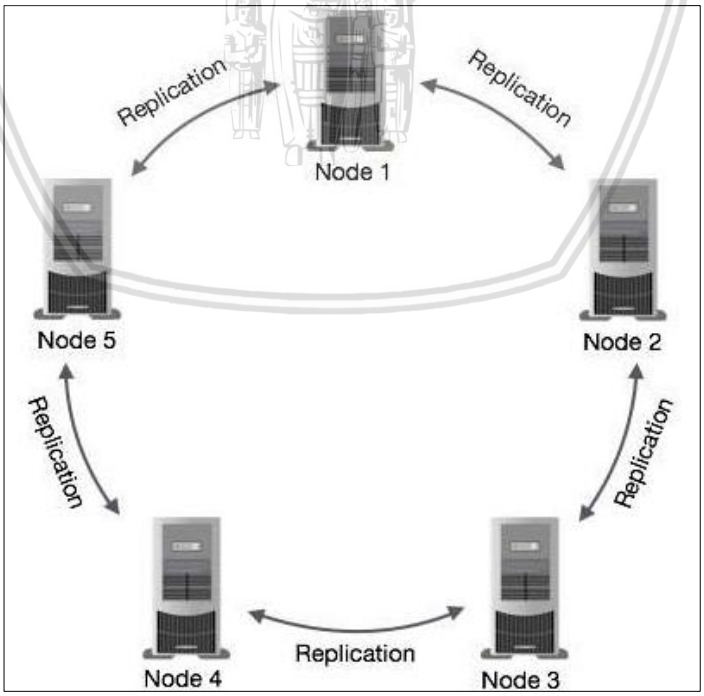
Berdasarkan pernyataan Meiyanto (2001), pada sistem *multi-tier*, aplikasi maupun dokumen XML dapat dikirim ke pihak lain dan atau diolah oleh pihak tersebut. Dalam sistem ini, memungkinkan untuk sebuah aplikasi dapat mengambil nilai data dari suatu sumber, tanpa perlu tahu bahwa data tersebut dihasilkan melalui proses pengolahan kalkulasi oleh sistem atau perangkat lain, sehingga memungkinkan untuk terjadi integrasi data maupun aplikasi yang dapat disebut juga dengan A2A *(application to application*). Kreger (2001) mengungkapkan, bahwa model dari suatu *web service* berdasarkan oleh hasil interaksi antar 3 peran

komponen dalam web service, yaitu: *service provider*, *service registry* dan *service requestor/consumer*. Hubungan yang terjadi antara ketiga komponen tersebut pun melakukan operasi *publish*, *find* dan *bind*. *Service provider* menyediakan layanan yang dapat diakses lewat jaringan komputer, misalnya internet. Lalu , *service provider* mendefinisikan layanan yang dibangun dan melakukan operasi *publish service description* pada *service registry* atau secara langsung ke *service consumer*. Adapun *Service requestor/consumer* melakukan operasi *find* agar dapat menerima dan mengakses nilai data *service description* secara lokal maupun melalui *service registry*. *Service description* yang didapat kemudian digunakan untuk melakukan operasi *bind service provider* dan berkomunikasi dengan implementasi *web service* yang akan digunakan.

2.2.5 *Apache Cassandra*

Apache Cassandra merupakan sistem *database* secara desentralisasi yang terdistribusi. Pada awalnya dikembangkan sebagai *write-efficient database* untuk mekanisme fitur pesan masuk pada *platform Facebook*. Perangkat lunak ini akhirnya diluncurkan sebagai sebuah proyek perangkat lunak berbasis *open-source* yang saat ini dikelola oleh *Apache Software Foundation*.

Mekanisme cara kerja *Cassandra* sendiri mengimplementasikan bentuk arsitektur mirip dengan *Dynamo*, dan model data serta mesin penyimpanan *Big-Table* yang semakin dioptimalkan dengan tujuan agar *Throughput* penulisan data dapat mencapai nilai tinggi, serta pengguna dapat melakukan input query baris dengan jumlah kolom yang besar dalam skema yang dinamis (Kuhlenkamp, Klems and Röss, 2014).



Gambar 2.2 Arsitektur *Apache Cassandra*
Sumber: (Tutorialspoint, 2017)

Pada Gambar 2.2 tersebut adalah gambaran umum dari arsitektur *Cassandra*. *Cassandra* menggunakan yang disebut sebagai *Gossip Protocol* pada sisi *background* agar dapat berkomunikasi antara *node* satu dengan *node* yang lain, serta agar dapat mendeteksi setiap *node* yang mengalami kecacatan/ *down* pada *cluster*. Selain itu, *Gossip Protocol* juga digunakan untuk mengirimkan hasil duplikasi data antara setiap *node* yang terdaftar sebagai *seeds* dalam sebuah arsitektur *Cassandra*. Pada gambar 2.1, disebutkan yang berperan sebagai *seeds* adalah Node 1, Node 2, Node 3, Node 4 dan Node 5.

2.2.6 *MongoDB*

MongoDB merupakan suatu *database* bersifat *Open Source*, dengan perfurma tinggi. *Database* ini memiliki konsep manajemen berorietasi dokumen, yang dibangun menggunakan bahasa pemrograman C++.

Database Berorientasi Dokumen sendiri merupakan sebuah program komputer yang didesain untuk menyimpan, mengambil dan mengelola data yang berorientasi format dokumen. *Database* berorientasi Dokumen sendiri adalah salah satu dari kategori *database* yang diketahui dengan julukan terkenal *NoSQL*. *NoSQL* merupakan akronim dari *Not Only SQL*, yang berarti suatu sistem basis data yang tidak perlu menggunakan perintah *query SQL* (*Structure Query Language*) untuk menjalankan proses manipulasi data. *MongoDB* merupakan basis data non-relasional, hal ini membuat *MongoDB* dapat melakukan komputasi sangat cepat saat menjalankan proses manipulasi data dibandingkan dengan sistem basis data relasional (RDBMS). Karena *MongoDB* berbasis dokumen, sehingga tidak mempunyai struktur yang teratur seperti tabel pada umumnya. Adapun kelebihan *MongoDB* dibandingkan *database* lainnya adalah, dapat melakukan proses *searching* lebih cepat, tidak perlu mendesaain struktur tabel karena *MongoDB* akan secara otomatis membuatkannya, yang diperlukan adalah melakukan *insert* saja, mempercepat proses CRUD (*create, read, update, delete*), sehingga tidak heran jika banyak digunakan *website* dengan kompleksitas data yang cukup besar (Chodorow & Dirolf, 2010).

Berdasarka pernyataan Seguin (2012, p4), *MongoDB* merupakan *data storage* yang mana setiap tabelnya tidak harus memiliki relasi dengan tabel lain. *MongoDB* berisi yang disebuut *collection*, dan setiap *collection* terdiri dari *documents*. Pada tiap *documents* terdiri dari *fields*. Dan di tiap *collections* dapat dilakukan *indexing*, sehingga meningkatkan kinerja proses *sorting data*. Berikut adalah 6 konsep yang harus di pahami dalam *MongoDB*:

1. *MongoDB* mempunyai konsep sama dengan jeins *data storage* lainnya seperti *MySQL* ataupun *Oracle*. *MongoDB* tidak boleh memiliki dua *data storage* atau lebih, karena masing *data storage* bertindak sebagai "*high level containers*".
2. Suatu *data storage* MongoDB tidak boleh memiliki 2 *collection* atau lebih. Sebuah *collection* mempunyai banyak kemiripan dengan tabel tradisional di *data storage* seperti *MySQL*. Karenanya, sebuah *collection* dan tabel tradisional dalam hal tersebut dapat di anggap sama.

3. Suatu *data storage* tidak boleh memiliki 2 *documents* atau lebih. Karena sebuah *documents* dapat dianggap sama seperti sebuah *row* layaknya pada tabel tradisional.
4. 1 *documents* terdiri dari satu *fields* dan atau *columns*, atau lebih.
5. *Indexes* pada *MongoDB* seperti layaknya *indexes* di *Relational Data storage Management System* (RDSMS).
6. *Cursors MongoDB* di gunakan untuk proses meminta atau memanggil data.

2.2.7 Cloud Computing

Cloud computing merupakan suatu model komputasi terkonfigurasi, dimana sumber daya seperti *processor/* tenaga komputasi, media penyompanan, jaringan, dan perangkat lunak menjadi abstrak atau *virtual*, serta dihadirkan sebagai sebuah layanan pada jaringan internet menggunakan pola akses *remote* atau jarak jauh (Purbo, 2011). Badan Nasional Standar dan Teknologi Amerika Serikat atau NIST (*National Institute of Standarts and Technology*), memberikan sebuah pengertian tentang *cloud computing*, yaitu sebuah model untuk menghadirkan kenyamanan, akses jaringan *on-demand* untuk dimanfaatkan secara bersama dari suatu sumber daya komputasi yang terkonfigurasi (misalnya, jaringan, peladen, media penyimpanan, aplikasi, dan layanan) yang dengan secara cepat dapat diberikan dan dirilis dengan manajemen yang dapat dikerjakan secara minimal atau mudah (Mell & Grance, 2011).

Cloud computing sendiri terdiri dari 3 model layanan yaitu :

a) *Software as a Service* (SaaS)
b) *Platform as a Service* (PaaS)
c) *Infrastructure as a Service* (IaaS) (Purbo, 2011)

Demikian juga dengan model pemetaan *cloud computing*, menurut definisi dari NIST (Mell & Grance, 2011), terdiri dari 4 model, yaitu:

a) Private Cloud
b) Community Cloud
c) Public Cloud
d) Hybrid Cloud

2.2.8 Apache JMeter

Apache JMeter merupakan suatu perangkat lunak berbasis *open-source*, yang dikembangkan oleh *Apache Software Foundation*. *JMeter* berwujud sebuah aplikasi perangkat lunak berbasis *Java* yang dibangun untuk melakukan uji fungsional dari suatu sistem serta mengukur kinerjanya.

Sejatinya, aplikasi ini dibangun untuk tujuan pengujian *web application*. Namun saat ini, pengembangannya sudah disebarluaskan agar bermanfaat bagi siapapun yang ingin melakukan uji fungsi lain pada berbagai sistem. Sampai hari ini,

terdapat banyak pengembang dari pihak ketiga yang ikut berkolaborasi untuk membangun *plugin* tambahan yang dapat digunakan secara mudah untuk menguji kinerja dari suatu sistem/ aplikasi menggunakan *JMeter* ini (*JMeter*, 2017).

2.2.9 Parameter Pengujian

Pada penelitian ini menggunakan beberapa parameter pengujian untuk menguji dari kinerja NoSQL Cassandra dan NoSQL MongoDB. Adapun parameter pengujian yang digunakan antara lain:

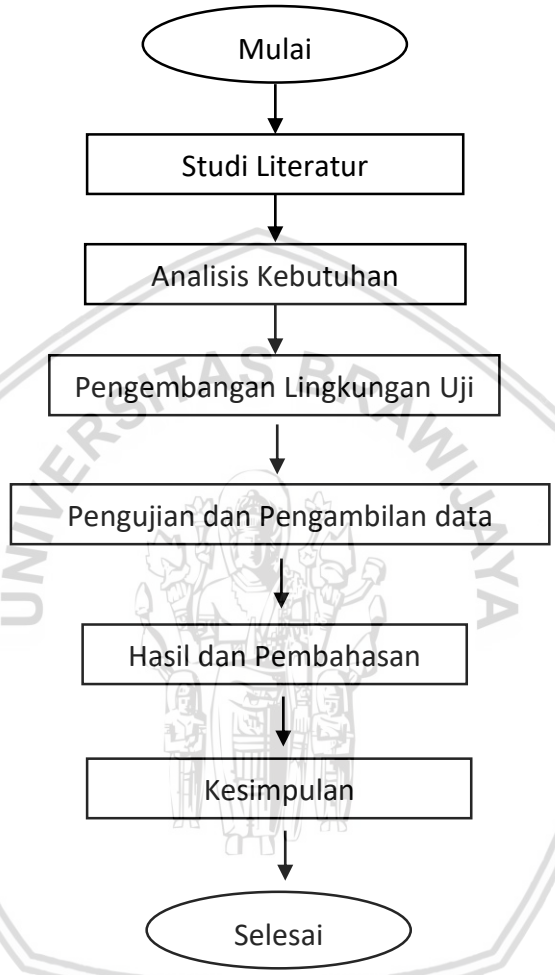
- *runtime* atau *execution time* merupakan parameter waktu yang digunakan *database* untuk menyelesaikan operasi query yang dijalankan. Satuan dari *runtime* atau *execution time* adalah sekon(s).
- *throughput* merupakan parameter kecepatan yang diperlukan *database* dalam menyelesaikan operasi *query* yang dijalankan dalam waktu tertentu. Untuk menghitung *throughput* dapat digunakan perhitungan sebagai berikut:

$$\textit{throughput}\ (\text{ops/sec}) = \frac{Jumlah\ Operasi\ Query\ (Ops)}{Waktu\ Eksekusi\ Query\ (s)}$$

- *memory usage* merupakan parameter jumlah penggunaan kapasitas memory (RAM) yang digunakan oleh *database server* ketika mengeksekusi suatu operasi transaksi data, misal operasi insert.
- *CPU usage* merupakan parameter jumlah persentase *processor* (CPU) yang digunakan oleh *database server* ketika mengeksekusi suatu operasi transaksi data, missal operasi insert.
- *disk I/O* merupakan parameter untuk mengukur kecepatan har*disk* dalam menulis atau membaca data.

BAB 3 METODOLOGI

Pada bab metodologi penellitian ini akan dibahas tentang studi literatur, analisis kebutuhan pembangunan lingkungan uji, pengujian dan pengambilan data, pembahasan dan hasil, kesimpulan dan saran. Untuk langkah-langkah nya di tunjukkan pada gambar berikut:



Gambar 3.1 Diagram Alir Metodologi Penelitian

3.1 Studi Literatur

Dalam melaksanakan penelitian ini, peneliti melakukan studi literature untuk mempelajari berbagai teori yang mendukung penelitian ini. Refrensi penunjang dalam penelitian ini berasal dari beberapa jurnal penelitian yang membahas tentang performa NoSQL *database.* Selain itu, peneliti juga mendapatkan refrensi melalui buku-buku elektronik (*ebook*) dan situs website yang menjelaskan beberapa teori yang berkaitan dengan penelitian ini.

3.2 Analisis Kebutuhan

Analisis kebutuhan memiliki suatu tujuan untuk menganalisa kebutuhan apa saja yang digunakan pada penelitian ini. Pada penelitian ini, analisis kebutuhan dibagi menjadi analisis kebutuhan fungsional dan analisis kebutuhan non fungsional.

3.2.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan kebutuhan yang berisi tentang proses apa saja dan hasil apa yang didapat pada saat membangun Cassandra sebagai media penyimpanan data. Pada penelitian ini terdapat beberapa kebutuhan fungsional yang diharapkan sebagai berikut:

1. API Web service dapat menyimpan data dari sensor DHT11 dan sensor kamera yang telah disubscribe oleh *Internet Gateway Device* (IGD) ke dalam data *storage* dengan menggunakan NoSQL Cassandra
2. NoSQL Cassandra dapat menyimpan data dari sensor DHT11 dan sensor kamera
3. IoT Apps dapat menampilkan data yang disimpan pada NoSQL Cassandra Berdasarkan topic yang diinputkan.

3.2.2 Kebutuhan Non Fungsional

Kebutuhan non-fungsional merupakan kebutuhan yang diperlukan untuk membangun lingkungan pengujian. Pada penelitian ini, kebutuhan non-fungsional dibagi menjadi dua yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak yang dapat disebutkan sebagai berikut:

3.2.2.1 Kebutuhan perangkat keras

Kebutuhan perangkat keras digunakan untuk menunjang lingkungan pengujian dari segi perangkat keras. Dibawah ini merupakan perangkat keras yang digunakan pada penelitian ini, yaitu sebagai berikut:

1. Laptop
2. Server
3. Raspberry Pi
4. Sensor DHT11
5. Sensor Kamera

3.2.2.2 Kebutuhan perangkat Lunak

Kebutuhan perangkat lunak digunakan untuk menunjang proses pengujian dari sedi perangkat lunak. Dibawah ini merupakan perangkat lunak yang digunakan pada penelitian ini, yaitu sebagai berikut:

1. Putty

2. Robomongo
3. Python
4. Apache JMeter
5. Brackets
6. CQLSH
7. IoT Apps
8. cassandra-client

3.3 Pembangunan Lingkungan Uji

Pembangunan lingkungan uji merupakan tahap untuk menyiapkan berbagai perancangan yang digunakan untuk proses pengujian. Perancangan – perancangan tersebut dapat dibagi menjadi perancangan topologi, perancangan database, perancangan utilitas pengujian, perancangan skenario pengujian dan data uji yang digunakan.

3.4 Pengujian dan Pengambilan Data

Pengujian dilakukan untuk mengetahui apakah NoSQL Cassandra yang diusulkan dapat bekerja sesuai dengan fungsionalitas yang dirancang. Selain untuk mengetahui fungsionalitas dari NoSQL Cassandra yang diusulkan, pengujian dilakukan untuk mengumpulkan data informasi terkait dengan nilai dari parameter: *runtime*, *throughput*, *memory usage*, *CPU usage*, dan *disk I/O* pada saat melakukan operasi insert data dengan variasi jumlah data yang berbeda-beda. Hasil dari pengujian parameter tersebut digunakan untuk mengukur kinerja dari NoSQL Cassandra dan NoSQL MongoDB sebagai media penyimpanan dari perangkat IoT. Saat berlangsungnya pengujian ini juga dilaksanakan pengambilan informasi dari parameter uji berupa: *memory usage*, *CPU usage* dan *disk I/O* menggunakan JMeter.

3.5 Hasil dan Pembahasan

Setelah dilakukan proses pengujian baik pengujian fungsional maupun pengujian kinerja database, akan dilakukan pembahasan pada hasil pengujian yang didapat. Pembahasan dilakukan untuk mengetahui apakah NoSQL Cassandra yang diusulkan dapat berjalan sesuai dengan fungsionalitas yang diharapkan serta mengetahui kinerja dari NoSQL Cassandra dan NoSLQ MongoDB pada saat melakukan operasi insert data dengan melihat dan membandingkan *runtime, throughput, CPU usage, memory usage* dan *disk I/O*.

3.6 Kesimpulan dan Saran

Pengambilan kesimpulan dan saran dilakukan setelah semua tahapan pengujian selesai dilakukan. Kesimpulan dibuat untuk memberikan jawaban terhadap rumusan masalah yang telah dibuat. Peneliti akan memberikan saran yang

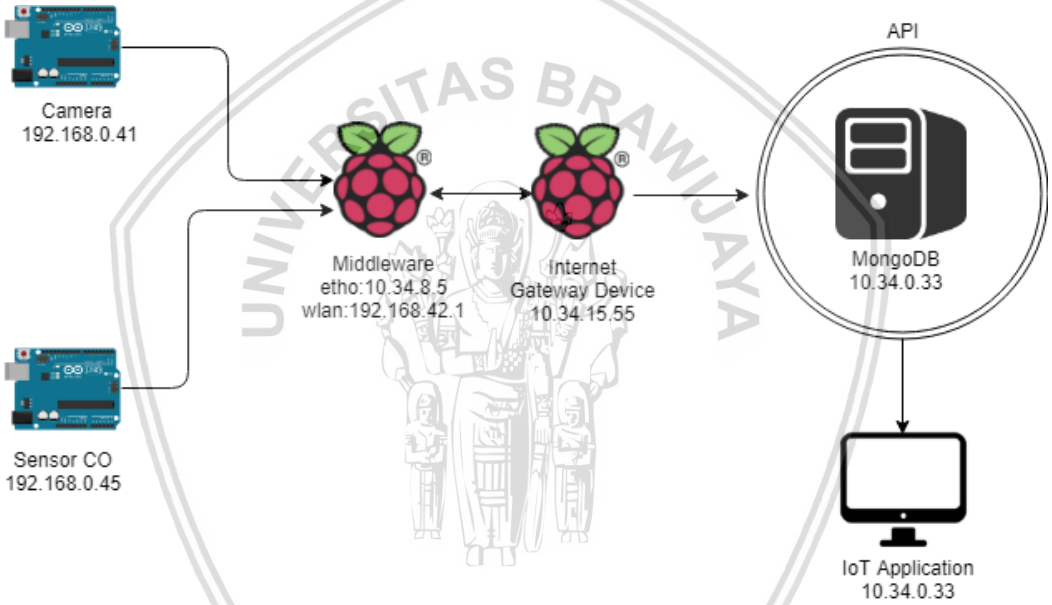
dapat dijadikan rujukan untuk memperbaiki atau mengembangkan penelitian ini maupun penelitian sejenis.

BAB 4 PEMBANGUNAN LINGKUNGAN UJI

Pembangunan lingkungan uji merupakan tahap yang digunakan penulis untuk menyiapkan berbagai kebutuhan yang diperlukan untuk melakukan penelitian. Penelitian ini dilakukan di Lab Jaringan Komputer, Fakultas Ilmu Komputer.

4.1 Deskripsi Umum Sistem

Pada penelitian sebelumnya (Pramukantoro. Dkk., 2017) telah dikembangkan sebuah *framework* untuk menyimpan dan mengakses data sensor dari suatu perangkat IoT ke dalam sistem basis data MongoDB . Data yang disimpan berasal dari sensor CO dan sensor kamera. Media penyimpanan yang digunakan pada penelitian tersebut menggunakan MongoDB dan GridFS. Untuk topologi framework yang dikembangkan dapat dilihat pada Gambar 4.1.



Gambar 4.1 Topologi penelitian terdahulu

Data yang dihasilkan dari topologi diatas berasal dari sensor camera dan sensor CO. Kedua sensor tersebut mempublish data menuju ke middleware. Middleware tersebut menggunakan metode publish/subscribe untuk pertukaran datanya. Untuk menerima data dari middleware tersebut harus ada subscriber dimana pada penelitian yang dikembangkan sebelumnya disebut dengan *Internet Gateway Device* (IGD). Selanjutnya IGD (*Internet Gateway Device*) mensubscribe topic dari middleware untuk mendapatkan data dari sensor CO dan sensor camera untuk selanjutnya disimpan pada data storage MongoDB dan GridFS. Untuk memudahkan proses *get* dan *post* data, dibangun sebuah API *Web service*. Untuk melihat data yang disimpan dalam data storage, dibangun sebuah IoT App dengan menggunakan *websocket*. Pada penelitian ini, akan berfokus pada kinerja database NoSQL Cassandra yang diusulkan oleh peneliti dengan NoSQL MongoDB yang

digunakan pada penelitian pramukantoro (2017) dalam melakukan operasi insert data ditinjau dari parameter *runtime*, throughput, CPU usage, memory usage dan *disk* I/O. Selain itu, NoSQL Cassandra yang diusulkan oleh peneliti akan diuji fungsionalitasnya dalam menyimpan dan menampilkan data dari node sensor.

4.2 Analisis Kebutuhan

Dalam melakukan penelitian tentang perbandingan kinerja Cassandra dan MongoDB sebagai backend IoT data storage, dibutuhkan kebutuhan yang dapat menunjang terbentuknya lingkungan pengujian. Terdapat dua kebutuhan untuk menunjang terbentuknya lingkungan pengujian yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak.

4.2.1 Kebutuhan Perangkat Keras

Pada bagian kebutuhan perangkat keras ini menjabarkan berbagai macam perangkat keras yang digunakan pada penelitian. Adapun spesifikasi Kebutuhan perangkat keras yang digunakan adalah sebagai berikut:

a. Laptop:
   - *OS* : *Windows 10 Pro* (64-*bit*)
   - Prosesor : Intel® Core ™ i5-6400U
   - *RAM* : 4 GB
   - Jumlah : 1
   - Fungsi : Untuk meremote database server dan client
     Untuk menjalankan kode pengujian dan JMeter

b. Raspberry pi 2:
   - *OS* : Raspbian GNU/Linux
   - Prosesor : ARMv7 Processor rev 4 (v7l)
   - *RAM* : 1 GB
   - Jumlah : 1
   - Fungsi : Untuk menjalankan *Middleware*
     Untuk menjalankan *Internet Gateway Device*

c. Database server:
   - *OS* : Ubuntu Server 16.04
   - Prosesor : Westmere E56xx/L56xx/X56xx (Nehalem-C)
   - *RAM* : 4 GB
   - Har*disk* : 40 GB
   - Jumlah : 1
   - Fungsi : Sebagai media penyimpanan data

d. Client:
   - *OS* : Ubuntu Server 16.04
   - Prosesor : Westmere E56xx/L56xx/X56xx (Nehalem-C)
   - *RAM* : 4 GB
   - Har*disk* : 30 GB
   - Jumlah : 1
   - Fungsi : Untuk menjalankan kode pengujian insert data

e. Sensor Kamera:
   - Fungsi : Untuk mempublish data dari sensor kamera
f. Sensor DHT11:
   - Fungsi : Untuk mempublish data dari sensor DHT11

4.2.2 Kebutuhan Perangkat Lunak

Pada bagian kebutuhan perangkat lunak ini menjelaskan berbagai macam perangkat lunak yang digunakan pada penelitian. Kebutuhan perangkat lunak pada penelitian ini dijelaskan pada Tabel 4.2.

Tabel 4.2 Kebutuhan Perangkat Lunak

Komponen	**Keterangan**
Putty	Software remote konsol atau terminal yang digunakan untuk me-remote koneksi computer melalui port SSH
Robomongo / Robo 3T	Merupakan sebuah software yang digunakan untuk mengatur, management dan mengembangkan database MongoDB. Software ini menyediakan GUI dan bersifat open source.
Python	Digunakan sebagai bahasa pemrograman dalam membuat service untuk melakukan POST dan GET data ke database, subscriber untuk melalukan subscribe data dari middleware dan API Cassandra untuk melakukan pengujian kinerja database
CQLSH	Cassandra Query Language Shell digunakan untuk berinteraksi dengan database Cassandra. CQLSH digunakan untuk membuat keyspace dan table, memasukkan dan mengquery data yang ada pada database Cassandra.
Brackets	Bertindak sebagai editor source code yang support berbagai macam bahasa termasuk python.
JMeter	Bertindak sebagai tools yang digunakan untuk pengambilan resource metric CPU , Memory dan *disk* i/o pada database server.
IoT Apps	Untuk menampilkan data sensor yang berada pada data *storage*.
cassandra-client	*Tools* yang digunakan untuk melihat tabel penyimpanan data pada NoSQL Cassandra yang berbasis GUI.

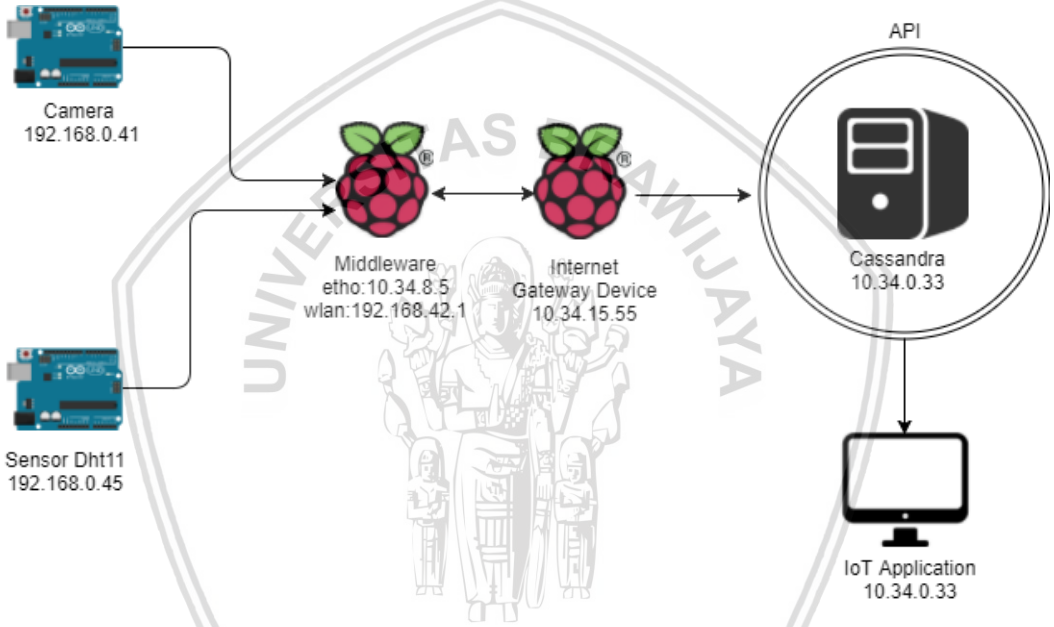
4.3 Pembangunan Lingkungan Pengujian

Pada tahap ini dilakukan perancangan yang digunakan untuk proses pengujian. Perancangan – perancangan tersebut dapat dibagi menjadi perancangan

topologi, perancangan database, perancangan utilitas pengujian, perancangan skenario pengujian dan data uji yang digunakan.

4.3.1 Topologi

Untuk pengujian fungsionalitas NoSQL Cassandra dalam menyimpan dan menampilkan data dari node sensor, maka perlu dibuat sebuah topologi sistem. Topologi sistem yang dibuat menggunakan topologi sistem yang dikembangkan pada penelitian (Pramukantoro, dkk,. 2017). Topologi sistem ini menggambarkan secara keseluruhan bagaimana data sensor dipublish sampai disimpan pada data *storage* dengan NoSQL Cassandra. Untuk topologi sistem yang digunakan dapat dilihat pada Gambar 4.2

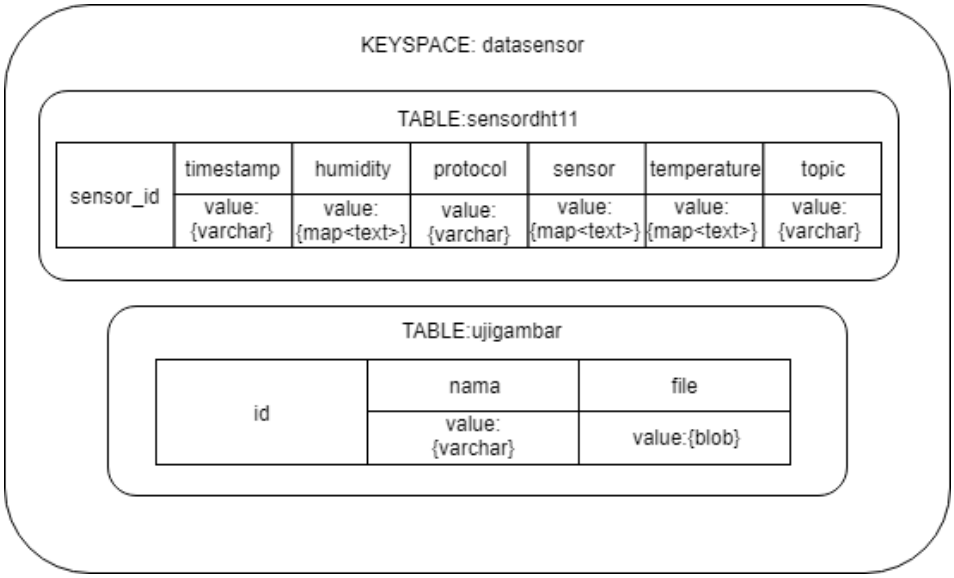


Gambar 4.2 Topologi penelitian saat ini

Perbedaan topologi pada Gambar 4.2 dan Gambar 4.1 hanya sebatas pada data *storage* dan sensor yang digunakan. Untuk data *storage* yang digunakan pada topologi Gambar 4.2 menggunakan NoSQL Cassandra yang diusulkan oleh peneliti dan sensor yang digunakan adalah sensor DHT11 yang dimana pada Gambar 4.1 menggunakan sensor CO.

4.3.2 Perancangan Database NoSQL Cassandra

Pada perancangan database NoSQL Cassandra yang berbasis pada column-oriented tentunya memerlukan penyesuaian terhadap data yang akan disimpan. Pada penelitian ini digunakan satu keyspace. Keyspace adalah sebuah tempat yang digunakan untuk menampung satu atau lebih column families. Column families dapat dianalogikan sebagai tabel pada relational database. Seperti relational database, keyspace juga mempunyai nama dan sekumpulan attribute. Untuk penjelasan dari perancangan NoSQL Cassandra dapat dilihat pada Gambar 4.3



Gambar 4.3 *Schema* Database NoSQL Cassandra

Pada Gambar 4.3 terdapat dua buah tabel yaitu Table sensordht11 dan Tabel ujigambar. Pada Tabel sensordht11 digunakan untuk menyimpan data dari sensor DHT11, sedangkan Tabel ujigambar digunakan untuk menyimpan data dari sensor kamera. Pada tabel sensordht11 memiliki sensor_id yang digunakan sebagai partition key, sedangkan timestamp, humidty, protocol, sensor, temperature dan topic digunakan untuk menyimpan value yang dihasilkan oleh sensor dht11. Pada tabel ujigambar memiliki id yang digunakan sebagai partition key, sedangkan nama, dan file digunakan untuk menyimpan value yang dihasilkan dari sensor kamera.

4.3.3 Perancangan Utilitas Pengujian

Perancangan utilitas pengujian adalah pengujian yang dilakukan menggunakan kode program yang dibuat oleh peneliti untuk menjalankan proses pengujian sesuai dengan spesifikasi dan target dari pengujian. Adapun *psudocode* program yang dibuat untuk proses pengujian yaitu:

- *Psudocode webservice*

  Webservice pada penelitian ini digunakan untuk menyimpan dan menampilkan data dari sensor DHT11 dan sensor kamera yang telah disubscribe oleh *Internet Gateway Device* (IGD). Untuk *psudocode webservice* dapat dilihat pada Tabel 4.1.

Tabel 4.1 *Psudocode webservice*

```
1  @app2.route('/api/postdht11',methods=['POST'])
2  FUNCTION postdht11():
3          data <- request.get_json()
4          a <- str(uuid.uuid4())
5          dict_sensor <- {'module':data['sensor']['module'],
   'tipe':data['sensor']['tipe'],'ip':data['sensor']['ip']}
6          dict_humi <- {'value':data['humidity']
   ['value'],'unit':data['humidity']['unit']}
7          dict_temp <- {'value':data['temperature']['value'],
   'unit':data['temperature']['unit']}
```

```
 8          b <- data['protocol']
 9          e <- data['topic']
10          l <- str(data['timestamp'])
11          cluster <- Cluster(['10.34.0.33'])
12          session <- cluster.connect('datasensor')
            session.execute("INSERT INTO sensordht11
(sensor_id,timestamp,humidity,protocol,
13 sensor,temperature,topic)
          VALUES (%s,%s,%s,%s,%s,%s,%s)",
14 (a,l,dict_humi,b,dict_sensor,dict_temp,e))
15          RETURN "DATA SENSOR DHT11 BERHASIL DI POST"
16 ENDFUNCTION
17 @app2.route('/api/postgambar',methods=['POST'])
18 FUNCTION postgambar():
19          data <- request.get_json()
20          id <- str(uuid.uuid4())
21          c <- data['Name']
22          g <- data['Data']
23          masuk <- pickle.loads(g)
24          t <- base64.b64encode(masuk)
25          q <- base64.b64decode(t)
26          i <- bytearray(q)
27          cluster <- Cluster(['10.34.0.33'])
28          session <- cluster.connect('datasensor')
            session.execute("INSERT INTO
29 ujigmbr(id,file,nama)VALUES(%s,%s,%s)",(id,i,c))
30          RETURN "DATA SENSOR KAMERA BERHASIL DI POST"
31 ENDFUNCTION
32 @app2.route('/api/gambar/<string:name>',methods=['GET'])
33 FUNCTION gambar(name):
34          id <- name
35          cluster <- Cluster(['10.34.0.33'])
36          session <- cluster.connect('datasensor')
            query <- session.prepare('SELECT file FROM ujigmbr
37 WHERE "id"=?')
38          x <- session.execute(query,(id,))
39          for y in x:
40                  a <- base64.b64encode(y.file)
41          ENDFOR
42          RETURN render_template ("list.html", gambar=a)
```

Pada *psudocode* Tabel 4.1 tersebut terdapat fungsi postdht11, postgambar dan gambar. Fungsi dari postdht11 pada baris ke 2 sampai ke 15 adalah untuk menyimpan data sensor DHT11 yang telah disubscribe oleh *Internet Gateway Device* (IGD) ke dalam NoSQL Cassandra. Untuk Fungsi dari postgambar pada baris ke18 sampai ke30 digunakan untuk menyimpan data sensor kamera yang telah disubscribe oleh *Internet Gateway Device* (IGD) ke

dalam NoSQL Cassandra. Sedangkan fungsi gambar pada baris ke33 sampai ke42 digunakan untuk menampilkan data gambar yang ada pada NoSQL Cassandra ke web browser. Untuk kode selengkapnya dapat dilihat pada bagian lampiran dengan nama service.py.

- *Psudocode websocket*

  Websocket yang dirancang pada penelitian ini digunakan untuk menampilkan data yang disimpan pada data *storage* NoSQL Cassandra pada IoT Apps. Untuk psudocode websocket yang di rancang dapat dilihat pada Tabel 4.2.

Tabel 4.2 *Psudocode websocket*

```
1  FUNCTION dataReceived(self, data):
2          OUTPUT data
3          msg=""
4          IF data=="WEB":
5              nama["WEB"]=self
6          ELSEIF data=="/datagambar":
7              cluster <- Cluster(['10.34.0.33'])
8              session <- cluster.connect('datasensor')
               query <- session.execute("SELECT * FROM
9  ujigmbr")
10             for document in query :
11                 a <- base64.b64encode(document.file)


                   msg+=("</br>ID : "+str(document.id)+"
   "+"</br>Name:<a target='_blank'
   href='http://10.34.0.33:5000/api/gambar/"+str(document.id)
12 +"'>"+str(document.nama)+"</a></br>")
13                  transport.write(msg)
14             ENDFOR
15         ELSEIF data=="/datadht11":
16             cluster <- Cluster(['10.34.0.33'])
17             session <- cluster.connect('datasensor')
18             OUTPUT data
               query <- session.execute("SELECT * FROM
19 sensordht11")
20             for document in query:
                  msg+=("</br>ID :
   "+str(document.sensor_id)+" "+str(document.humidity)+"
21 "+str(document.protocol)+"</br>")
22             ENDFOR
```

Psudocode websocket diatas digunakan untuk menampilkan data yang disimpan pada data storage NoSQL Cassandra pada IoT Apps dengan cara menginputkan topik /datagambar atau topic /datadht11. Untuk kode selengkapnya dapat dilihat pada bagian lampiran pada laporan ini dengan nama websocket_cassandra.py.

- *Psudocode subscriber / Internet Gateway Device*

  Subscriber pada penelitian ini digunakan untuk mensubscribe topik yang ada pada middleware, dalam hal ini berhubungan dengan broker. Middleware yang digunakan pada penelitian pramukantoro (2017) menggunakan metode publish subscribe untuk pertukaran pesan/datanya. Untuk *psudocode subscriber* dapat dilihat pada Tabel 4.3 berikut ini.

Tabel 4.3 *Psudocode subscriber*

```
1  FUNCTION on_message(mqttc,obj,msg):
2          IF msg.topic=='office/roomA14':
3                  headers <- {"Content-type":"application/json"}
4                  pesan <- msg.payload
5                  koneksi.request("POST","/api/postdht11",pesan,headers)
6                  response <- koneksi.getresponse()
7                  OUTPUT (response.read())
8          ELSEIF msg.topic=='office/gambarA13':
9                  headers <- {"Content-type":"application/json"}
10                 pesan_dua <- msg.payload
11 koneksi.request("POST","/api/postgambar",pesan_dua,headers)
12                 response <- koneksi.getresponse()
13                 OUTPUT (response.read())
14         ENDIF
15 ENDFUNCTION
```

Pada baris ke2 sampai ke7 merupakan proses subscribe data yang dipublish oleh sensor DHT11 dengan nama topic office/roomA14. Setelah itu, payload yg berhasil disubscribe ditampung pada variable pesan. Setelah itu, payload akan dikirimkan ke *webservice* yang selanjutnya untuk di simpan pada data *storage*. Pada baris ke8 sampai ke13 merupakan proses subscribe data yang dipublish oleh sensor kamera dengan nama topic office/gambarA13. Setelah itu, payload yang berhasil disubscribe ditampung pada variable pesan_dua. Setelah itu, payload akan dikirimkan ke *webservice* yang selanjutnya untuk disimpan pada data *storage.* Untuk kode selengkapnya dapat dilihat pada bagian lampiran dengan nama sub1.py.

- *Psudocode insert* data *string* Cassandra

  Psudocode insert data *string* Cassandra digunakan untuk pengujian kinerja NoSQL Cassandra dalam melakukan *insert* data dengan menggunakan data *string*. Untuk *psudocode insert* data *string* Cassandra dapat dilihat pada Tabel 4.4.

Tabel 4.4 Psudocode *insert* data *string* Cassandra

```
1   x <- 1
2           awal <- time.time()
            insert_user <- session.prepare("INSERT INTO sensordht11
(sensor_id,timestamp,humidity,protocol,sensor,
3   temperature,topic) VALUES(?,?,?,?,?,?,?)")
4           for operation in range(0,x):
5               a <- str(uuid.uuid4())
                dict_sensor <- {'module':'dht11','tipe':'espn8266',
6   'index':'8456747','ip':'192.168.42.50'}
7               dict_humidity <- {'value':'36.00','unit':'%'}
8               dict_temperature <- {'value':'25.00','unit':'celcius'}
9               i <- "1524105130"
10              j <- "office/roomA13"
11              k <- "mqtt"
                session.execute(insert_user,[a,i,dict_humidity,k,
12  dict_sensor,dict_temperature,j])
13          ENDFOR
14          akhir <- time.time()
15          total <- akhir-awal
16          tps <- x/total
17          OUTPUT "TPS yg dihasilkan"+ str(tps)
18          OUTPUT "runtime"+str(total)
```

Psudocode pada Tabel 4.4 digunakan untuk menguji kinerja Cassandra dalam melakukan operasi *insert* data dengan menggunakan data *string*. Hasil yang didapat dari menjalankan *psudocode* tersebut adalah nilai *throughput* dan *runtime*. Untuk kode selengkapnya dapat dilihat pada bagian lampiran dengan nama cassandra_insert_text.py.

- *Psudocode insert* data gambar Cassandra

  Psudocode insert data gambar Cassandra digunakan untuk pengujian kinerja NoSQL Cassandra dalam melakukan insert data dengan menggunakan data gambar. Untuk psudocode *insert* data gambar Cassandra dapat dilihat pada Tabel 4.5.

Tabel 4.5 *Psudocode insert* data *file* Cassandra

```
1   with open("baru.jpg","rb") as gambar:
2                   x <- 100000
3                   h <- gambar.read()
4                   awal <- time.time()
                    insert_user <- session.prepare("INSERT INTO
5   imagetes(id,file,nama) VALUES (?,?,?)")
6                   for operation in range (0,x):
7                           a <- str(uuid.uuid4())
8                           l <- base64.b64encode(h)
9                           d <- base64.b64decode(l)
```

```
10                          b <- bytearray(d)
11                          c <- "image1.jpg"
12                          session.execute(insert_user,[a,b,c])
13                  ENDFOR
14                  akhir <- time.time()
15                  total <- akhir - awal
16                  tps <- x / total
17                  OUTPUT ("TPS yang dihasilkan"+ str(tps))
18                  OUTPUT ("runtime"+str(total))
```

Psudocode pada Tabel 4.5 digunakan untuk menguji kinerja Cassandra dalam melakukan operasi *insert* data dengan menggunakan data gambar. Hasil yang didapat dari menjalankan *psudocode* tersebut adalah nilai *throughput* dan *runtime*. Untuk kode selengkapnya dapat dilihat pada bagian lampiran dengan nama cassandra_insert_file.py.

- *Psudocode insert* data *string* MongoDB

  Psudocode insert data *string* MongoDB digunakan untuk pengujian kinerja NoSQL MongoDB dalam melakukan insert data dengan menggunakan data *string*. Untuk *psudocode insert* data *string* MongoDB dapat dilihat pada Tabel 4.6.

Tabel 4.6 *Psudocode insert* data *string* MongoDB

```
1   x <- 1
2           awal <- time.time()
3           for operation in range(0,x):
4                   db.sensor.insert_one({
5                   "protocol" : "mqtt",
6                           "timestamp" : "1524105130",
7                           "topic" : "office/roomA13",
8                               "sensortipe" : "esp8266",
9                               "sensorindex" : "8456747",
10                              "sensorip" : "192.168.42.245",
11                              "sesnormodule" : "dht22",
12                              "humidityvalue" : "20",
13                              "humidityunit" : "%",
14                              "temperaturevalue" : "30",
15                              "temperatureunit" : "celcius"})
16          ENDFOR
17          akhir <- time.time()
18          total <- akhir-awal
19          tps <- x /total
20          OUTPUT "TPS"+ str(tps)
21          OUTPUT "runtime"+ str(total)
```

Psudocode pada Tabel 4.6 digunakan untuk menguji kinerja NoSQL MongoDB dalam melakukan operasi *insert* data dengan menggunakan data *string*. Hasil yang didapat dari menjalankan *psudocode* tersebut adalah nilai *throughput* dan *runtime*. Untuk kode selengkapnya dapat dilihat pada bagian lampiran dengan nama mongodb_insert_text.py.

- *Psudocode insert* data gambar MongoDB

  Psudocode insert data gambar MongoDB digunakan untuk pengujian kinerja NoSQL MongoDB dalam melakukan *insert* data dengan menggunakan data gambar. Untuk *psudocode insert* data *file* MongoDB dapat dilihat pada Tabel 4.7.

Tabel 4.7 *Psudocode insert* data *file* MongoDB

```
1  with open("baru.jpg","rb") as gambar:
2          file <- gambar.read()
3          x <- 10000
4          name <- "image.jpg"
5          awal <- time.time()
6          for operation in range(0,x):
7                  fs.put(file,filename <- name)
8          ENDFOR
9          akhir <- time.time()
10         total <- akhir - awal
11         tps <- x / total
12         OUTPUT ("TPS"+ str(tps))
13         OUTPUT ("runtime"+str(total))
```

Psudocode pada Tabel 4.7 digunakan untuk menguji kinerja NoSQL MongoDB dalam melakukan operasi *insert* data dengan menggunakan data gambar. Hasil yang didapat dari menjalankan *psudocode* tersebut adalah nilai *throughput* dan *runtime*. Untuk kode selengkapnya dapat dilihat pada bagian lampiran dengan nama mongodb_insert_file.py.

4.3.4 Perancangan Skenario Pengujian.

Perancangan skenario pengujian yang akan dilakukan terdiri dari pengujian fungsional dan pengujian kinerja database. Pengujian fungsional bertujuan untuk mengetahui apakah NoSQL Cassandra yang diusulkan dapat menyimpan dan menampilkan data dari sensor DHT11 dan sensor kamera. Pengujian kinerja database bertujuan untuk mengetahui kinerja dari Cassandra dan MongoDB dalam melakukan operasi insert data ditinjau dari parameter *runtime, throughput, CPU usage, memory usage* dan *disk i/o*. Untuk skenario pengujian fungsional dan pengujian kinerja database akan dijelaskan sebagai berikut:

- Pengujian Fungsional

  Skenario pengujian fungsional akan dilampirkan pada Tabel 4.7 berikut ini:

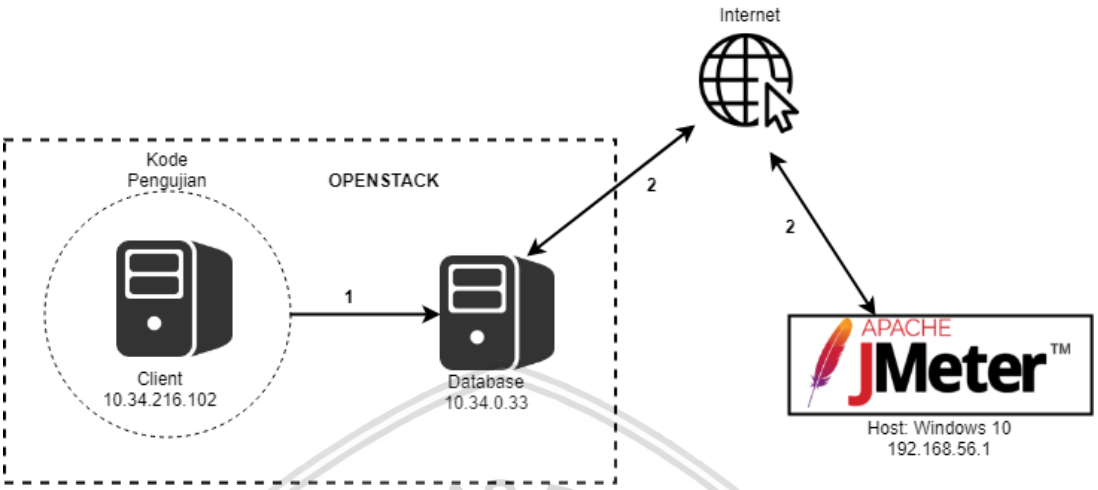
Tabel 4.8 Pengujian fungsional

NO	Deskripsi Pengujian	Skenario	Hasil yang Diharapkan
1	API Webservice dapat menyimpan data sensor dht11 dan sensor kamera dari *Internet Gateway Device* ke database Cassandra	1.Middleware sudah berjalan 2.Pengguna menjalankan kode service.py 3.Pengguna menjalankan kode sub1.py	Kode sub1.py menampilkan pesan berupa " DATA SENSOR DHT11 BERHASIL DI POST" dan" DATA SENSOR KAMERA BERHASIL DI POST"
2	Database Cassandra Dapat menyimpan data dari sensor DHT11 dan sensor kamera	1.Pengguna membuka tools Cassandra Client Untuk melihat data yang disimpan 2.Pengguna melihat data pada tabel penyimpanan data	Pengguna dapat melihat data sensor DHT11 dan sensor kamera pada tabel penyimpanan database Cassandra
3	IoT apps dapat menampilkan data yang disimpan pada database Cassandra berdasarkan topic, topic /datagambar, /datadht11 melalui protocol websocket	1.Pengguna menjalankan kode service 2.Pengguna menjalankan kode websocket 3.Pengguna membuka dan menjalankan IoT App Dengan alamat IP:10.34.0.33 4.Pengguna menginputkan topic yang akan ditampilkan pada IoT App	IoT Apps dapat menampilkan data berdasarkan topic yang diinputkan

- Pengujian Kinerja Database:

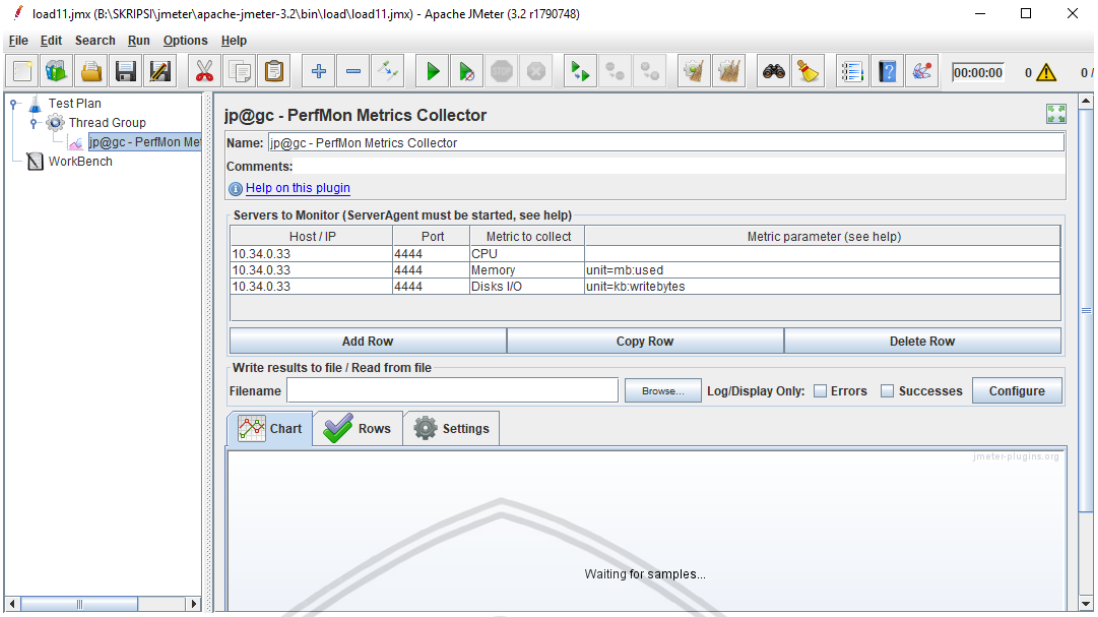
Pengujian kinerja database dilakukan dengan meninjau parameter uji berupa *runtime, throughput, memory usage, CPU usage* dan *disk I/O* dalam melakukan operasi *insert* data terhadap NoSQL Cassandra dan NoSQL MongoDB. Data yang digunakan pada saat melakukan operasi *insert* berupa data *string* dan data gambar

dengan variasi jumlah data antara 10.000, 30.000, 50.000, 70.000, 100.000, 120.000, 150.000 untuk data *string* serta 1000, 3000, 5000, 7000, 10.000 untuk data gambar. Adapun topologi pengujian kinerja database dapat dilihat pada Gambar 4.4



Gambar 4.4 Topologi pengujian kinerja database

Pada topologi diatas, utilitas kode pengujian dijalankan pada sisi client dengan IP: 10.34.216.102. Pada nomor 1 merupakan proses dari utilitas kode pengujian tersebut adalah melakukan operasi insert data sebanyak jumlah operasi yang ditentukan sesuai dengan perancangan skenario pengujian kinerja database yang dibuat terhadap database NoSQL Cassandra dan NoSQL MongoDB yang menggunakan IP: 10.34.0.33. Selama proses utilitas kode pengujian berjalan, dilakukan pengambilan data metrics CPU Usage, Memory Usage dan *disk* i/o terhadap database server menggunakan JMeter yang dijalankan pada laptop dengan IP: 192.168.56.1 yang ditunjukkan pada nomor 2. Untuk konfigurasi JMeter yang digunakan untuk pengambilan data metrics CPU usage, memory usage dan *disk* I/O dapat dilihat pada Gambar 4.5.



Gambar 4.5 Konfigurasi JMeter

Untuk menjalankan JMeter dengan plugisn Perfmon pada Gambar 4.5 tersebut, terlebih dahulu mengisi kolom Host/ip, port, metric to collect, metric parameter. Kolom host/ip diisi dengan ip dari server target yang akan didapatkan resource metric nya. Isi dari kolom port sudah otomatis terisi dengan angka 4444 dikarenakan ServerAgent yang dijalankan pada server target menggunakan port 4444. Kolom metric to collect diisi dengan metric apa saja yang ingin didapatkan pada server target, dalam penelitian ini menggunakan metric memory usage, CPU usage dan *Disk* i/o. Kolom metric parameter diisi dengan parameter tambahan dari metric yang dipilih, dalam metric yang dipilih ini menggunakan parameter unit=mb:used pada metric memory usage dan unit=kb:writebytes pada metric *disk* I/O.

4.3.5 Data uji

Data uji digunakan untuk menguji kinerja dari database Cassandra dan MongoDB dalam melakukan operasi insert data. Dalam pengujian database ini, peneliti menggunakan data uji berupa data string dan data gambar. Ukuran data gambar yang digunakan sebagai data uji sebesar 250 KB. Alasan peneliti menggunakan data uji tersebut dikarenakan data yang disimpan pada database penelitian pramukantoro (2017) menggunakan data string yang berasal dari sensor DHT11 dan data gambar yang berasal dari sensor kamera. Untuk data uji pada penelitian ini dapat dilihat pada Gambar 4.6 dan Gambar 4.7.


```
{"sensor":{"module":"dht11",
"tipe":"esp8266",
"index":8456747,
"ip":"192.168.42.50"},
"protocol":"mqtt",
"topic":"office\/roomA13",
"humidity":{"value":"36.0","unit":"%"},
"timestamp":1528320394,
"temperature":{"value":"25.0","unit":"celcius"}}
```


Gambar 4.6 Data String

Gambar 4.7 Data Gambar

4.4 Implementasi

Pada sub bab ini akan dilakukan implementasi dari perancangan database NoSQL Cassandra. Berikut adalah langkah-langkah yang dilakukan dalam proses implementasi database NoSQl Cassandra:

1. Membuat keyspace

   Untuk implementasi pembuatan keyspace dapat dilihat pada Gambar 4.7

```
cqlsh> CREATE KEYSPACE datasensor
   ... WITH replication = {'class': 'SimpleStrategy',
   ... 'replication_factor':1};
```

Gambar 4.8 Implementasi pembuatan keyspace

   Dalam membuat keyspace, kita harus menentukan terlebih dahulu nama keyspace yang akan digunakan. Setelah itu, menentukan Class Strategy yang digunakan yang akan digunakan pada keyspace tersebut, dalam proses ini menggunakan SimpleStrategy. dikarenakan pada penelitian ini hanya menggunakan 1 buah node saja. Setelah itu menentukan jumlah replication_factor, dalam proses ini menggunakan 1 replication factor dikarenakan hanya menggunakan 1 buah node dan data tidak direplikasi.

2. Membuat tabel dan menentukan tipe data

   Sebelum membuat tabel, terlebih dahulu menggunakan keyspace yang telah dibuat dengan mengetikkan command “USE datasensor” pada *Cassandra Query Language*. Pada tahap ini, akan dibuat dua tabel yaitu tabel untuk menyimpan data dari sensor DHT11 dan sensor kamera. Untuk implementasi pembuatan tabel dapat dilihat pada Gambar 4.9 dan Gambar 4.10.

```
cqlsh> USE datasensor ;
cqlsh:datasensor> CREATE TABLE sensordht11 (sensor_id varchar primary key,
             ... timestamp varchar,
             ... humidity map<text,text>,
             ... protocol varchar,
             ... sensor map<text,text>,
             ... temperature map<text,text>,
             ... topic varchar);
```

Gambar 4.9 Implementasi pembuatan tabel sensordht11

Pada Gambar 4.9 telah dilakukan implementasi pembuatan tabel untuk menyimpan data dari sensor DHT11. Pada tabel tersebut terdapat kolom sensor_id dengan tipe data varchar yang digunakan sebagai primary key dari setiap baris, kolom timestamp dengan tipe data varchar yang digunakan untuk menyimpan value timestamp dari sensor DHT11, kolom humidity dengan menggunakan collection map dengan tipe data text yang digunakan untuk menyimpan value humidity dari sensor DHT11, kolom protocol dengan tipe data varchar yang digunakan untuk menyimpan value protocol dari sensor DHT11, kolom sensor dengan menggunakan collection map dengan tipe data text yang digunakan untuk menyimpan value sensor dari sensor DHT11, kolom temperature dengan menggunakan collection map dengan tipe data text yang digunakan untuk menyimpan value temperature dari sensor DHT11 dan kolom topic dengan tipe data varchar yang digunakan untuk menyimpan value topic dari sensor DHT11.

```
cqlsh:datasensor> CREATE TABLE sensorkamera (id varchar primary key,
             ... file blob,
             ... nama varchar);
```

Gambar 4.10 Implementasi pembuatan tabel sensorkamera

Pada Gambar 4.10 telah dilakukan implementasi pembuatan tabel untuk menyimpan data dari sensor kamera. Pada tabel tersebut terdapat kolom id dengan tipe data varchar yang digunakan sebagai primary key dari setiap baris, kolom file dengan tipe data blob yang digunakan untuk menyimpan gambar yang berasal dari sensor kamera dan kolom nama dengan tipe data varchar yang digunakan untuk menyimpan nama file dari gambar yang dihasilkan oleh sensor kamera.

BAB 5 PENGUJIAN DAN PENGAMBILAN DATA

5.1 Pengujian Fungsional

Pengujian Fungsional dilakukan untuk mengetahui bahwa NoSQL Cassandra yang diusulkan pada penelitian ini dapat berjalan sesuai dengan fungsionalitas yang dirancang pada perancangan skenario pengujian. Berikut cara yang dilakukan dalam proses pengujian fungsional yang dibutuhkan.

5.1.1 Fungsional No 1:

Pada pengujian fungsional ke 1 yaitu API Webservice dapat mengirim data sensor DHT11 dari *Internet Gateway Device* (IGD) ke database NoSQL Cassandra. Untuk menjalankan fungsional 1, terlebih dahulu dijalankan kode service.py. Untuk menjalankan kode service.py dapat dilihat pada Gambar 5.1.

```
ubuntu@ega:~/skripsi_cassandra$ python service.py
 * Running on http://192.168.100.10:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 289-325-841
```

Gambar 5.1 Menjalankan kode service.py

Kode service.py digunakan sebagai service untuk melakukan post dan get data yang ada pada database NoSQL Cassandra.

Setelah menjalankan kode service.py, kemudian dijalankan kode sub1.py yang digunakan untuk mensubscribe data yang berasal dari middleware. Untuk menjalankan kode sub1.py dapat dilihat pada Gambar 5.2.

```
pi@TheMiddleware:~/sendHTTP $ python sub1.py
DATA SENSOR KAMERA BERHASIL DI POST
DATA SENSOR KAMERA BERHASIL DI POST
DATA SENSOR KAMERA BERHASIL DI POST
DATA SENSOR DHT11 BERHASIL DI POST
DATA SENSOR KAMERA BERHASIL DI POST
DATA SENSOR DHT11 BERHASIL DI POST
```

Gambar 5.2 Menjalankan kode sub1.py

Pada Gambar 5.2 menunjukkan bahwa kode sub1.py berhasil mensubscribe data yang berasal dari middleware dan menyimpan datanya melalui kode service.py yang sudah dijalankan sebelumnya. Untuk kode sub1.py dan service.py selengkapnya dapat dilihat pada bagian lampiran di laporan ini.

5.1.2 Fungsional No 2:

Pada pengujian fungsional ke 2 yaitu database Cassandra Dapat menyimpan data dari sensor DHT11 dan sensor kamera. Untuk menjalankan fungsional 2 dapat dilakukan dengan menjalankan tools cassandra-client. Tools tersebut digunakan untuk melihat data yang tersimpan pada database Cassandra dengan menggunakan

GUI. Untuk hasil data sensor DHT11 dan sensor kamera yang tersimpan pada database NoSQL Cassandra dapat dilihat pada Gambar 5.3 dan Gambar 5.4 dengan menggunakan tools cassandra-client.

#	sensor_id ▲	timestamp	humidity	protocol	sensor	temperature	topic
4	'057f0960-f...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'
2	'2c1f4da9-a...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'
7	'2d127e57-f...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'
9	'4f1168ab-8...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'
3	'5d1f56bc-3...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'
5	'73815e84-...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'
8	'7df50e73-d...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'
0	'b16414c9-f...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'
10	'c12f8861-e...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'
1	'c2020689-...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'
6	'd4f238f6-fc...	'0'	{'unit':'%','value':'38.0'}	'mqtt'	{'ip':'192.168.42.50','module':'...	{'unit':'celcius','value':'23.0'}	'office/roomA13'

Gambar 5.3 Tabel penyimpanan data sensor DHT11

Pada Gambar 5.3 terdapat kolom timestamp, humidity, protocol, sensor, temperature, dan topic. Kolom tersebut menyimpan value yang dihasilkan oleh sensor DHT11. Kolom sensor_id bertindak sebagai primary key setiap baris yang ada pada tabel penyimpanan data sensor DHT11 tersebut.

#	id	file	nama ▼
0	'e3db3d1e-...	0xffd8ffe163fa4578696600004d4d002a00000008000a0100000400000001000000...	'images0.jpg'
1	'69713eee-...	0xffd8ffe163fa4578696600004d4d002a00000008000a0100000400000001000000...	'images0.jpg'
2	'cfe066a0-1...	0xffd8ffe163fa4578696600004d4d002a00000008000a0100000400000001000000...	'images0.jpg'
3	'2e6ff748-a...	0xffd8ffe163fa4578696600004d4d002a00000008000a0100000400000001000000...	'images0.jpg'
4	'9aaae82e-...	0xffd8ffe163fa4578696600004d4d002a00000008000a0100000400000001000000...	'images0.jpg'
5	'4b3c6659-...	0xffd8ffe163fa4578696600004d4d002a00000008000a0100000400000001000000...	'images0.jpg'
6	'e96dd4a9-...	0xffd8ffe163fa4578696600004d4d002a00000008000a0100000400000001000000...	'images0.jpg'
7	'b6653a45-...	0xffd8ffe163fa4578696600004d4d002a00000008000a0100000400000001000000...	'images0.jpg'

Gambar 5.4 Tabel penyimpanan data sensor kamera

Pada Gambar 5.4 terdapat kolom file dan nama. Kolom tersebut menyimpan value yang dihasilkan oleh sensor kamera. Kolom id bertindak sebagai primary key setiap baris yang ada pada tabel penyimpanan data sensor kamera tersebut.

5.1.3 Fungsional No 3:

Pada pengujian fungsional ke 3 yaitu IoT app dapat menampilkan data yang disimpan pada database Cassandra berdasarkan topic yang diinputkan, topic /datagambar, /datadht11 melalui protokol *websocket*. Untuk menjalankan fungsional 3, terlebih dahulu menjalankan kode service.py. Untuk menjalankan kode service.py dapat dilihat pada Gambar 5.5.

```
ubuntu@ega:~/skripsi_cassandra$ python service.py
 * Running on http://192.168.100.10:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 289-325-841
```

Gambar 5.5 Menjalankan kode service.py

Kode service.py digunakan sebagai service untuk melakukan post dan get data yang ada pada database NoSQL Cassandra.

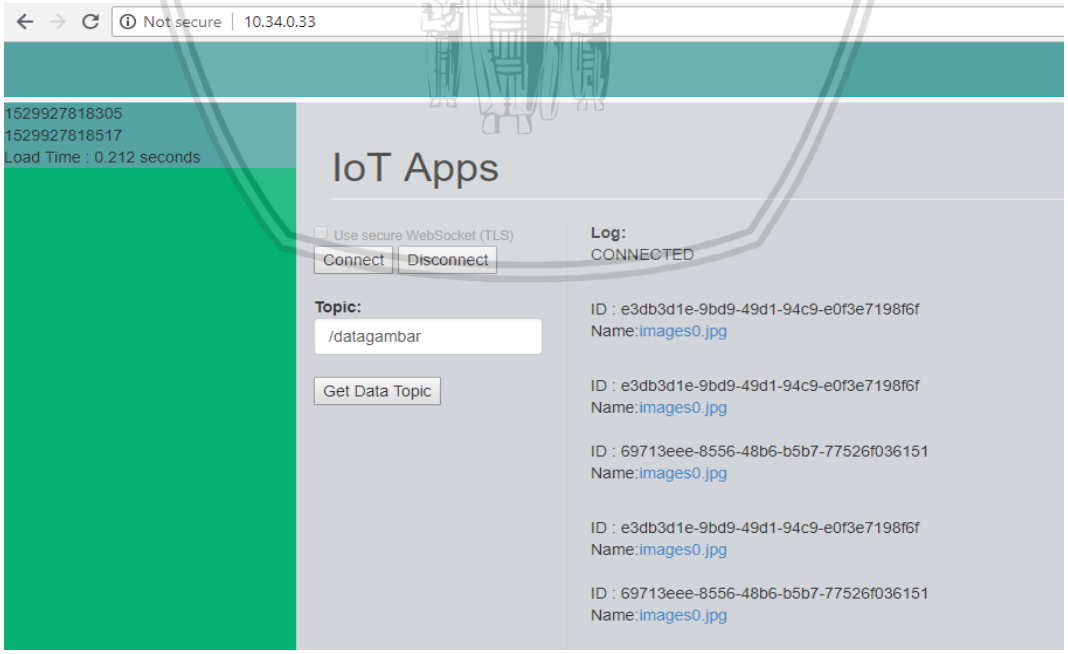
Setelah itu, dijalankan kode websocket_cassandra.py yang digunakan untuk menampilkan data berdasarkan topic yang diinputkan pada IoT Apps. Untuk menjalankan kode websocket_cassandra.py dapat dilihat pada Gambar 5.6.

```
ubuntu@ega:~/skripsi_cassandra$ python websocket_cassandra.py
```

Gambar 5.6 Menjalankan websocket_cassandra.py

Kode websocket_cassandra.py digunakan untuk menampilkan data yang disimpan pada data *storage.*

Setelah menjalankan kode websocket_cassandra.py, selanjutnya menjalankan Iot Apps pada web browser dengan mengetikkan IP:10.34.0.33. Hasil dari menjalankan IoT Apps dapat dilihat pada Gambar 5.7 dan Gambar 5.8. Untuk kode websocket_cassandra.py selengkapnya dapat dilihat pada bagian lampiran di laporan ini.



Gambar 5.7 Menampilkan data gambar

Pada Gambar 5.7 menunjukkan hasil get data sensor kamera dengan menginputkan topic /datagambar. Hasil yang ditampilkan berupa ID dan Name yang dimana ID

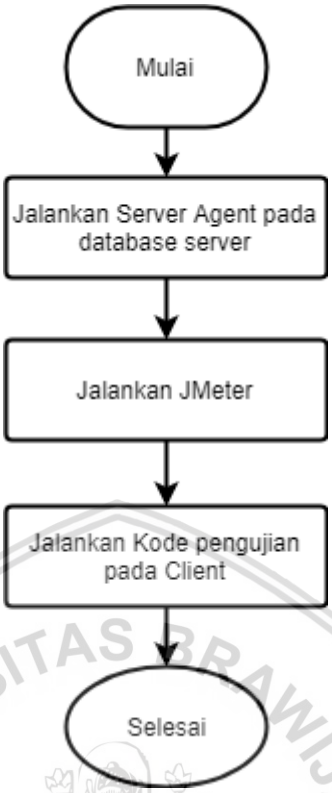
tersebut menunjukkan primary key dari data sedangkan Name merupakan nama file dari data yang berada pada database Cassandra.

Gambar 5.8 Menampilkan data dht11

Pada Gambar 5.8 menunjukkan hasil get data sensor DHT11 dengan menginputkan topic /datadht11. Hasil yang ditampilkan berupa ID, humidity dan protocol dimana ID tersebut menunjukkan primary key dari data, humidity merupakan isi dari data humidity dan protocol merupakan isi dari data protocol yang berada pada database Cassandra.

5.2 Pengujian Kinerja Database

Pengujian kinerja NoSQL Cassandra dan NoSQL MongoDB dalam melakukan operasi insert data *string* dan data gambar dilakukan dengan menjalankan utilitas kode pengujian yang sebelumnya telah dirancang. Pada saat pengujian insert data berlangsung, akan dilakukan pengambilan data runtime, throughput, CPU usage, memory usage dan *disk* I/O menggunakan apache JMeter dengan menambahkan *plugins PerfMon* (*Servers Performance Metric*). Agar JMeter dapat mengambil resource metric dari server database, *ServerAgent* harus dijalankan terlebih dahulu pada server database tersebut. Untuk alur dilakukannya pengujian kinerja database dalam melakukan operasi insert data dapat dilihat pada Gambar 5.9.



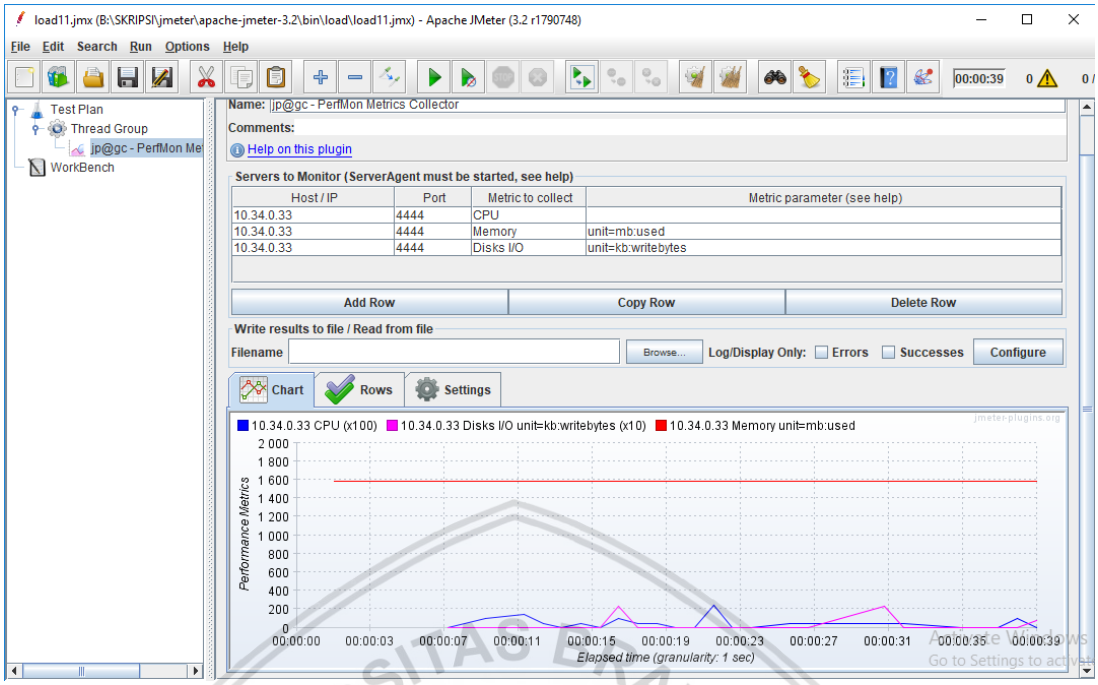
Gambar 5.9 Alur Pengujian Kinerja Database

5.2.1 Jalankan Server Agent

Pada proses ini menjalankan ServerAgent pada database server. ServerAgent dijalankan dengan tujuan agar JMeter dapat melakukan binding terhadap server target yang ingin didapatkan resource metricnya. Dalam pengujian kinerja database, resource metric yang ingin didapatkan berupa memory usage, CPU usage dan *disk* I/O.

5.2.2 Jalankan JMeter

Setelah menjalankan ServerAgent pada server target, langkah selanjutnya adalah menjalankan tools JMeter dengan tambahan *plugins PerfMon (Servers Performance Monitoring)*. Plugins ini bekerja dengan cara mengumpulkan resource metrics dari server target. Proses pengambilan data CPU usage, memory usage dan *disk* I/O dapat dilihat pada Gambar 5.10.



Gambar 5.10 Proses pengambilan metrics pada server target

5.2.3 Jalankan Kode Pengujian

Setelah menjalankan JMeter dengan plugin perfmon, selanjutnya menjalankan kode pengujian yang telah dirancang pada perancangan utilitas pengujian. Kode pengujian dijalankan untuk mendapatkan nilai *runtime* dan throughput pada saat melakukan operasi insert data pada database yang diuji. Untuk pengambilan data *runtime* dan throughput dapat dilihat pada Gambar 5.11.

```
tuti@trusty64:~/pengujian_nosql$ python cassandra_insert_text.py
TPS yg dihasilkan336.079089046
runtime0.297549009323
```

Gambar 5.11 Pengambilan data *runtime* dan throughput

Pada Gambar 5.11 dilakukan pengambilan nilai dari parameter *runtime* dan throughput dengan menjalankan kode pengujian yang telah dirancang pada perancangan utilitas pengujian.

BAB 6 HASIL DAN PEMBAHASAN

Pada bab ini dijelaskan hasil dari pengujian yang telah dilakukan dalam penelitian ini. Mendeskripsikan bagaimana database Cassandra bekerja dalam hal menyimpan dan menampilkan data berupa sensor dht11 dan data file berupa gambar serta kinerja database Cassandra dan MongoDB dalam melakukan operasi insert ditinjau dari parameter *Runtime, Throughput, CPU Usage, Memory Usage* dan *Disk I/O.*

6.1 Pengujian Fungsionalitas

Hasil dari pengujian fungsionalitas yang telah dilakukan akan dipaparkan pada Tabel 6.1.

Tabel 6.1 Hasil Pengujian Fungsional

No	Deskripsi Pengujian	Skenario	Hasil yang Diharapkan	Hasil
1	API Webservice dapat mengirim data sensor DHT11 dan sensor kamera dari *Internet Gateway Device* ke database Cassandra	1.*Middleware* sudah berjalan 2.Pengguna menjalankan kode sub1.py 3.Pengguna menjalankan kode service.py	Kode sub1.py menampilkan pesan data sensor DHT11 berhasil di POST dan sensor kamera berhasil di POST	BERHASIL
2	Database Cassandra Dapat menyimpan data dari sensor DHT11 dan sensor kamera	1.Pengguna membuka tools Cassandra Client Untuk melihat data yang disimpan 2.Pengguna melihat data pada tabel penyimpanan data	Pengguna dapat melihat data sensor DHT11 dan sensor kamera pada tabel penyimpanan database Cassandra	BERHASIL

3	IoT app dapat menampilkan data yang disimpan pada database Cassandra berdasarkan topic, topic /datagambar, /datadht11 melalui protocol websocket	1.Pengguna menjalankan kode service 2.Pengguna menjalankan kode websocket 3.Pengguna membuka dan menjalankan IoT App Dengan alamat IP:10.34.0.33 4.Pengguna menginputkan topic yang akan ditampilkan pada IoT App	IoT Apps dapat menampilkan data berdasarkan topic yang diinputkan	BERHASIL

6.2 Pengujian Kinerja Database

Dari pengujian kinerja Database NoSQL Cassandra dan NoSQL MongoDB dalam melakukan operasi insert data, didapatkan data berupa *runtime, throughput, memory usage, CPU usage* dan *disk I/O*. Data yang didapatkan telah diolah dan disajikan dalam bentuk tabel dan dalam bentuk whisker plot.

6.2.1 *runtime* Cassandra dan MongoDB

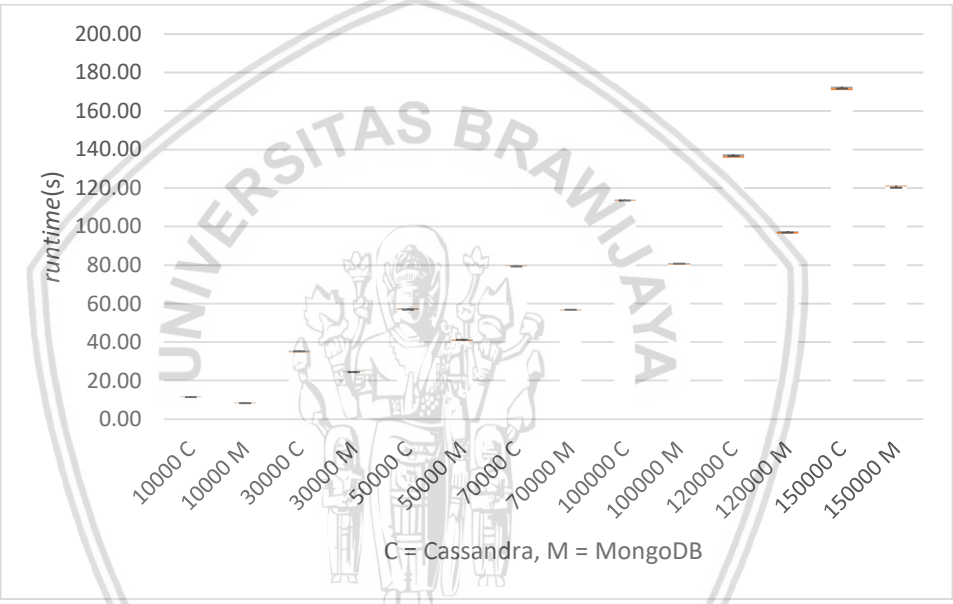
Hasil pengujian parameter *runtime* Cassandra dan MongoDB dalam melakukan operasi insert data dengan data string dapat dilihat pada Tabel 6.2.

Tabel 6.2 Hasil pengujian *runtime* dengan data *string*

Jumlah Variasi Data	Q1	Median	Q3	Max	Min
10000 C	11.5	11.6	11.8	12.4	11.1
10000 M	8.2	8.5	8.5	8.7	8.0
30000 C	34.7	35.1	35.5	38.8	34.3
30000 M	24.8	24.8	25.0	25.0	24.2
50000 C	56.8	57.2	57.6	57.7	56.0
50000 M	40.5	41.1	41.4	41.8	40.4
70000 C	79.4	79.8	80.0	82.1	78.8

70000 M	56.5	56.9	57.0	57.2	56.4
100000 C	113.4	113.8	114.1	114.8	112.9
100000 M	80.3	80.7	81.0	81.8	80.0
120000 C	135.8	136.6	137.6	139.2	134.8
120000 M	96.4	97.2	97.5	97.7	95.7
150000 C	170.8	171.9	172.7	174.3	169.9
150000 M	120.8	121.2	121.3	121.6	119.7

Data yang ada pada Tabel 6.2 disajikan dalam bentuk Whisker plot agar lebih mudah dibaca serta terlihat perbandingannya. Hasil dari *runtime* yang di dapat selama sepuluh kali pengujian operasi insert data dengan data *string* dapat dilihat pada Gambar 6.1



Gambar 6.1 Hasil *runtime* dengan data *string*

Pada Gambar 6.1 menunjukkan hasil perhitungan parameter *runtime* dalam melakukan operasi insert data menggunakan data string. Dari hasil yang didapatkan bahwa, semakin tinggi jumlah variasi data yang digunakan semakin naik *runtime* yang didapatkan. Pada variasi jumlah data 10000, MongoDB mendapatkan nilai *runtime* sebesar 8.5 second dibandingkan dengan Cassandra yang mendapatkan nilai 11,6 detik. Hingga variasi jumlah data mencapai 150000, MongoDB masih mendapatkan nilai runtime tercepat yaitu sebesar 121,2 detik dibandingkan dengan Cassandra yang hanya mendapatkan nilai runtime sebesar 171,9 detik.

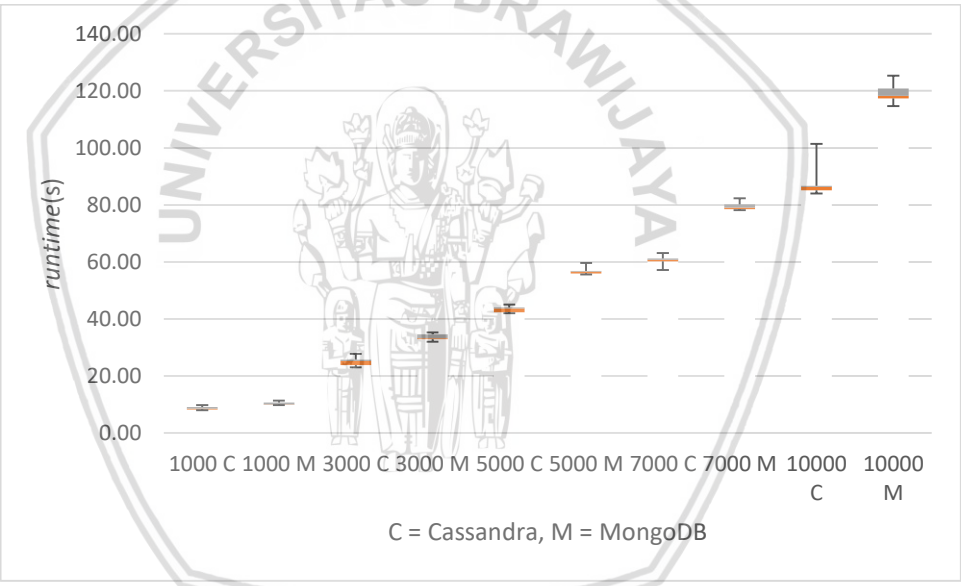
Hasil pengujian parameter *runtime* Cassandra dan MongoDB dalam melakukan operasi insert data dengan data gambar dapat dilihat pada Tabel 6.3.

Tabel 6.3 Hasil pengujian *runtime* dengan data gambar

Jumlah Variasi Data	Q1	Median	Q3	Max	Min
1000 C	8.2	8.5	9.1	9.8	8.0

1000 M	9.9	10.0	10.7	11.3	9.8
3000 C	23.8	25.3	25.7	27.8	23.1
3000 M	32.9	33.2	34.6	35.3	32.0
5000 C	42.3	43.6	44.0	45.0	42.0
5000 M	55.9	56.6	56.8	59.7	55.6
7000 C	60.2	60.7	61.2	63.1	57.1
7000 M	78.5	79.1	80.2	82.3	78.1
10000 C	85.1	86.4	86.7	101.4	84.0
10000 M	117.4	118.2	120.9	125.4	114.7

Data yang ada pada Tabel 6.3 disajikan dalam bentuk Whisker plot agar lebih mudah dibaca serta terlihat perbandingannya. Hasil dari *runtime* yang di dapat selama sepuluh kali pengujian operasi insert data dengan data gambar dapat dilihat pada Gambar 6.2



Gambar 6.2 Hasil *runtime* dengan data gambar

Pada Gambar 6.2 menunjukkan hasil perhitungan parameter *runtime* dalam melakukan operasi insert data menggunakan data gambar. Pada variasi jumlah data 1000, Cassandra memiliki nilai runtime sebesar 8.5 second lebih cepat dari MongoDB yang hanya mendapatkan nilai runtime sebesar 10 detik. Hingga variasi jumlah data mencapai 10000, Cassandra memiliki nilai runtime sebesar 86,4 detik lebih cepat dari MongoDB yang hanya mendapatkan nilai runtime sebesar 118,2 detik.

Dari hasil runtime yang sudah dibahas, dapat disimpulkan bahwa MongoDB memiliki nilai runtime lebih cepat dari pada Cassandra apabila tranksaksi data yang dipertukarkan tidak terlalu besar dalam hal ini datanya berupa data string dengan nilai *runtime* akhir sebesar 121,2 detik. Sebaliknya, apabila transaksi data yang dipertukarkan ukuran nya besar dalam hal ini datanya berupa data gambar, maka

Cassandra yang memiliki nilai runtime lebih cepat dari pada MongoDB dengan nilai runtime akhir sebesar 86,4 detik.

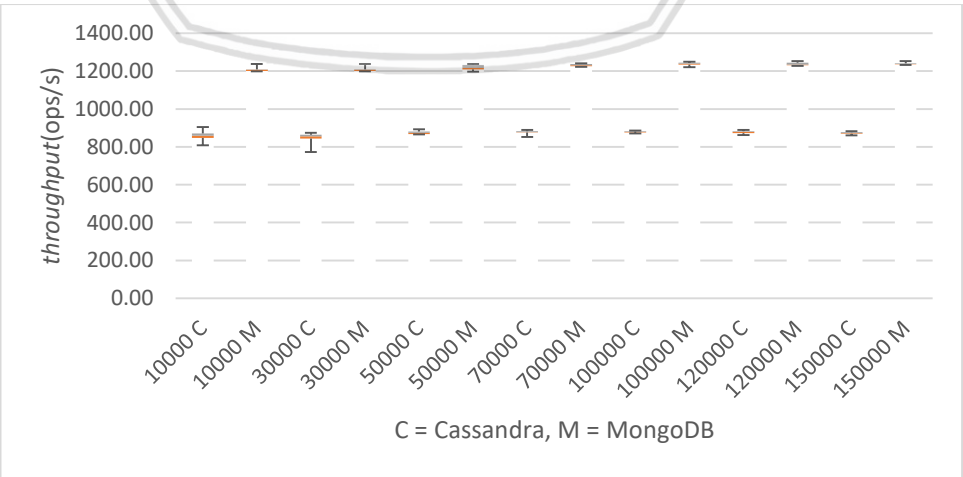
6.2.2 Throughput Cassandra dan MongoDB

Hasil pengujian parameter throughput Cassandra dan MongoDB dalam melakukan operasi insert data dengan data string dapat dilihat pada Tabel 6.4.

Tabel 6.4 Hasil pengujian *throughput* dengan data *string*

Jumlah Variasi Data	Q1	Median	Q3	Max	Min
10000 C	847.1	858.9	871.1	904.0	808.8
10000 M	1200.6	1208.2	1209.1	1238.8	1198.8
30000 C	844.6	855.5	864.0	873.9	772.3
30000 M	1200.6	1208.2	1209.1	1238.8	1198.8
50000 C	868.3	873.9	880.8	892.3	865.2
50000 M	1208.4	1217.4	1232.9	1238.1	1196.8
70000 C	875.5	877.5	882.0	888.8	852.6
70000 M	1226.9	1230.7	1237.9	1241.4	1223.5
100000 C	876.6	879.0	882.1	886.0	871.3
100000 M	1233.8	1239.3	1245.4	1249.5	1221.9
120000 C	872.1	878.7	883.6	890.0	862.1
120000 M	1230.6	1234.9	1244.6	1253.5	1228.1
150000 C	868.8	872.5	878.0	883.0	860.7
150000 M	1236.7	1238.1	1241.8	1253.5	1233.4

Data yang ada pada Tabel 6.4 disajikan dalam bentuk Whisker plot agar lebih mudah dibaca serta terlihat perbandingannya. Hasil dari throughput yang di dapat selama sepuluh kali pengujian operasi insert data dengan data string dapat dilihat pada Gambar 6.3



Gambar 6.3 Hasil throughput dengan data string

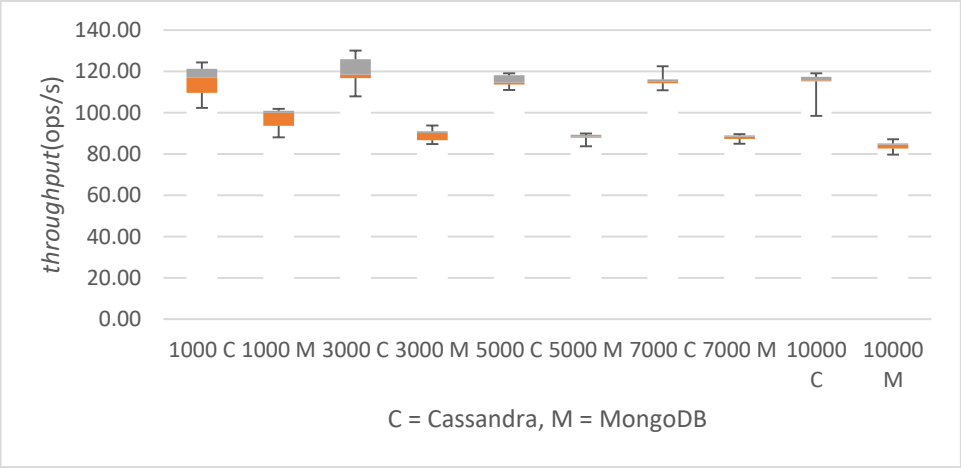
Pada Gambar 6.3 menunjukkan hasil perhitungan parameter throughput dalam melakukan operasi insert data menggunakan data string. Dari hasil yang didapatkan bahwa, semakin tinggi jumlah variasi data yang digunakan semakin naik throughput yang didapatkan. Nilai throughput yang dihasilkan pada Gambar 6.3 di pengaruhi dengan besar kecilnya *runtime* yang dihasilkan pada Gambar 6.1. Pada variasi jumlah data sebesar 10000, hasil throughput yang didapat oleh MongoDB sebesar 1208.2 ops/s dibandingkan dengan Cassandra yang hanya mendapat throughput sebesar 858.9 ops/s. Nilai throughput tertinggi pada MongoDB didapatkan ketika jumlah variasi data memasuki 150000 data dengan nilai throughput sebesar 1236.7 ops/s. Sedangkan untuk Cassandra, nilai throughput tertinggi didapatkan ketika jumlah variasi data memasuki 100000 dengan nilai throughput sebesar 879 ops/s.

Hasil pengujian parameter throughput Cassandra dan MongoDB dalam melakukan operasi insert data dengan data gambar dapat dilihat pada Tabel 6.4.

Tabel 6.5 Hasil *throughput* dengan data gambar

Jumlah Variasi Data	Q1	Median	Q3	Max	Min
1000 C	109.7	117.1	121.3	124.4	102.4
1000 M	93.7	100.0	101.1	102.0	88.2
3000 C	116.8	118.5	125.9	130.1	108.0
3000 M	86.7	90.3	91.1	93.9	84.9
5000 C	113.7	114.6	118.2	119.1	111.1
5000 M	88.0	88.4	89.4	90.0	83.8
7000 C	114.4	115.3	116.3	122.5	111.0
7000 M	87.3	88.5	89.1	89.6	85.1
10000 C	115.3	115.8	117.5	119.1	98.6
10000 M	82.7	84.6	85.2	87.2	79.8

Data yang ada pada Tabel 6.5 disajikan dalam bentuk Whisker plot agar lebih mudah dibaca serta terlihat perbandingannya. Hasil dari throughput yang di dapat selama sepuluh kali pengujian operasi insert data dengan data gambar dapat dilihat pada Gambar 6.4.



Gambar 6.4 Hasil *throughput* dengan data gambar

Pada Gambar 6.4 menunjukkan hasil perhitungan parameter throughput dalam melakukan operasi insert data menggunakan data gambar. Dari hasil yang didapatkan bahwa, semakin tinggi jumlah variasi data yang digunakan semakin naik nilai throughput yang dihasilkan. Nilai throughput yang dihasilkan pada Gambar 6.4 di pengaruhi dengan besar kecilnya *runtime* yang dihasilkan pada Gambar 6.2. Pada variasi jumlah data 1000, Cassandra mendapatkan nilai throughput sebesar 117.1 ops/s lebih cepat dibandingkan dengan MongoDB yang hanya mendapatkan nilai throughput sebesar 100 ops/s. Nilai throughput tertinggi pada Cassandra didapatkan ketika jumlah variasi data sebesar 3000 dengan nilai throughput yang didapat 118.5 ops/s. Untuk nilai throughput tertinggi pada MongoDB didapatkan ketika jumlah variasi data sebesar 1000 dengan nilai throughput yang didapatkan sebesar 100 ops/s. Semakin naik jumlah variasi data, nilai dari throughput pada MongoDB semakin menurun dengan nilai akhir sebesar 84.6 ops/s.

Dari hasil throughput yang sudah dibahas, dapat disimpulkan bahwa MongoDB memiliki nilai throughout lebih cepat dari pada Cassandra apabila tranksaksi data yang dipertukarkan tidak terlalu besar dalam hal ini datanya berupa data string dengan nilai *throughput* akhir sebesar 1236.7 ops/s. Sebaliknya, apabila transaksi data yang dipertukarkan ukuran nya besar dalam hal ini datanya berupa data gambar, maka Cassandra yang memiliki nilai throughput lebih cepat dari pada MongoDB dengan nilai throughput akhir sebesar 115.8 ops/s.

6.2.3 Memory Usage Cassandra dan MongoDB

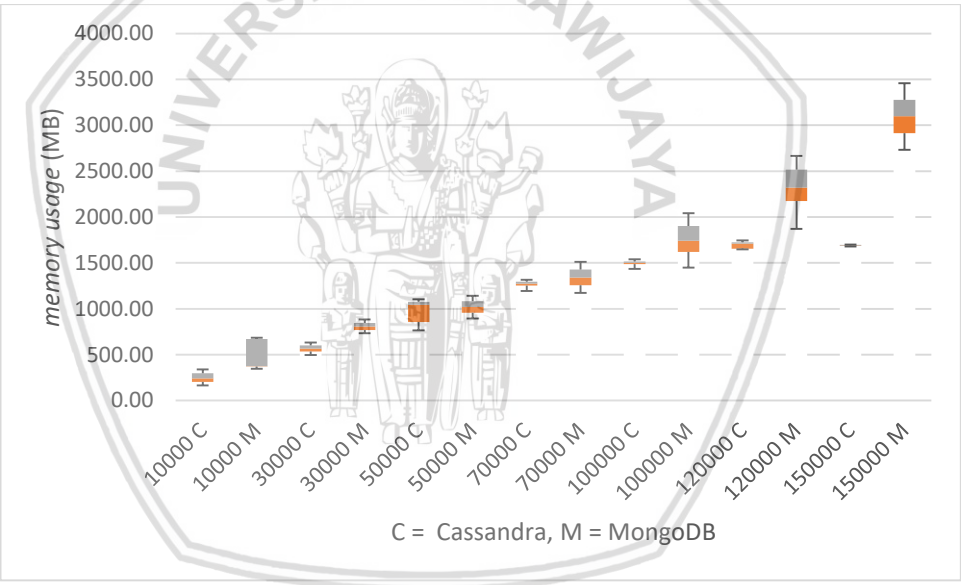
Hasil pengujian parameter memory usage Cassandra dan MongoDB dalam melakukan operasi insert data dengan data string dapat dilihat pada Tabel 6.5.

Tabel 6.6 Hasil pengujian *memory usage* dengan data *string*

Jumlah Variasi Data	Q1	Median	Q3	Max	Min
10000 C	204.2	241.9	295.7	337.9	163.6
10000 M	369.0	372.3	670.2	684.8	347.0
30000 C	532.7	565.3	599.4	633.6	497.7

30000 M	769.2	807.8	844.5	882.4	733.4
50000 C	854.3	1043.6	1073.9	1103.0	763.5
50000 M	956.7	1019.8	1082.4	1142.9	895.0
70000 C	1254.1	1273.4	1295.4	1315.5	1195.3
70000 M	1256.2	1341.9	1426.5	1512.9	1171.7
100000 C	1487.7	1500.3	1514.9	1539.4	1434.1
100000 M	1618.5	1739.8	1902.2	2039.9	1446.8
120000 C	1655.5	1709.4	1722.8	1746.4	1647.9
120000 M	2175.3	2321.6	2516.0	2665.4	1871.9
150000 C	1684.8	1690.4	1695.5	1700.9	1678.1
150000 M	2914.0	3098.1	3277.6	3458.8	2732.5

Data yang ada pada Tabel 6.6 disajikan dalam bentuk Whisker plot agar lebih mudah dibaca serta terlihat perbandingannya. Hasil dari memory usage yang di dapat selama sepuluh kali pengujian operasi insert data dengan data string dapat dilihat pada Gambar 6.5.



Gambar 6.5 Hasil memory usage dengan data string

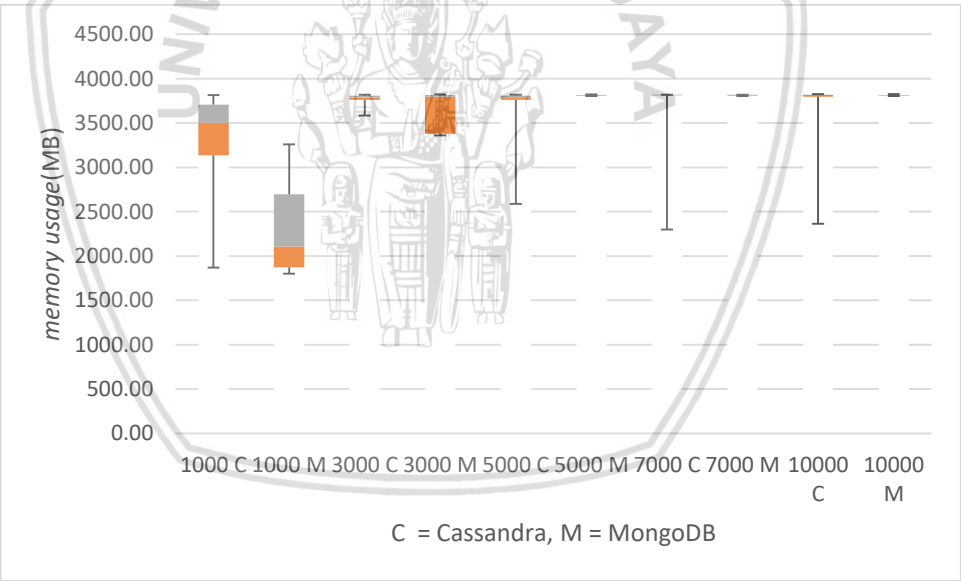
Pada Gambar 6.5 menunjukkan hasil perhitungan parameter memory usage dalam melakukan operasi insert data menggunakan data string. Dari hasil yang didapatkan bahwa, semakin tinggi jumlah variasi data yang digunakan semakin naik penggunaan memory yang dihasilkan. Pada jumlah variasi data sebesar 10000, penggunaan memory pada Cassandra sebesar 241.9 MB lebih sedikit dibanding dengan penggunaan memory pada MongoDB sebesar 372.3 MB. Pada saat jumlah variasi data mencapai 150000, penggunaan memory pada Cassandra masih jauh efficient yaitu sebesar 1690.4 MBdibandingkan dengan MongoDB yang penggunaan memory nya mencapai 3098.1 MB.

Hasil pengujian parameter memory usage Cassandra dan MongoDB dalam melakukan operasi insert data dengan data gambar dapat dilihat pada Tabel 6.6.

Tabel 6.7 Hasil pengujian *memory usage* dengan data gambar

Jumlah Variasi Data	Q1	Median	Q3	Max	Min
1000 C	3136.8	3504.3	3706.9	3814.1	1869.9
1000 M	1873.3	2106.3	2693.7	3259.5	1799.2
3000 C	3760.0	3783.7	3808.1	3816.9	3584.0
3000 M	3378.1	3795.4	3815.5	3821.2	3359.0
5000 C	3760.0	3783.5	3806.1	3816.9	2586.2
5000 M	3807.2	3808.3	3812.5	3823.0	3804.2
7000 C	3809.8	3814.9	3818.0	3819.1	2297.1
7000 M	3807.0	3814.3	3815.3	3816.6	3802.5
10000 C	3797.8	3812.8	3817.2	3824.8	2363.2
10000 M	3807.5	3810.9	3812.9	3823.8	3803.9

Data yang ada pada Tabel 6.7 disajikan dalam bentuk Whisker plot agar lebih mudah dibaca serta terlihat perbandingannya. Hasil dari memory usage yang di dapat selama sepuluh kali pengujian operasi insert data dengan data gambar dapat dilihat pada Gambar 6.6.



Gambar 6.6 Hasil memory usage dengan data gambar

Pada Gambar 6.6 menunjukkan hasil perhitungan parameter memory usage dalam melakukan operasi insert data menggunakan data gambar. Pada saat jumlah variasi data sebesar 1000 data, Cassandra mengkonsumsi memory lebih banyak sebesar 3504.3 MB dibandingkan dengan MongoDB yang hanya mengkonsumsi memory sebesar 2106.3MB. Penggunaan memory pada Cassandra meningkat dari pada saat jumlah variasi data memasuki 3000 data yakni sebesar 3783.7 MB. Kemudian memasuki jumlah variasi data berikutnya, penggunaan memory dari Cassandra semakin menurun sampai batas akhir pada variasi jumlah data sebesar 3812.8 MB. Untuk penggunaan memory pada MongoDB, meningkat pada saat

jumlah variasi data memasuki 3000 data yakni sebesar 3795.4 MB dan mengalami peningkatan sebesar 3808.3 MB pada saat jumlah variasi data sebesar 5000 data. Penggunaan memory pada MongoDB mengalami ketidaktabilan sampai batas akhri variasi jumlah data sebesar 10000 dengan penggunaan memory sebesar 3810.9 MB.

Dari pembahasan parameter memory usage, dapat disimpulkan bahwa semakin besar transaksi data yang dilakukan makan semakin besar jumlah konsumsi memory yang digunakan. Untuk transaksi data dengan menggunakan data string, Cassandra memiliki penggunaan memory lebih efficient dari MongoDB dengan penggunaan memory pada jumlah variasi akhir mencapai 1690.4 MB. Sedangkan untuk transaksi data dengan menggunakan data gambar, MongoDB memiliki penggunaan memory yang lebih efisient dibandingkan dengan Cassandra dengan penggunaan memory pada jumlah variasi akhir mencapai 3810.9 MB.

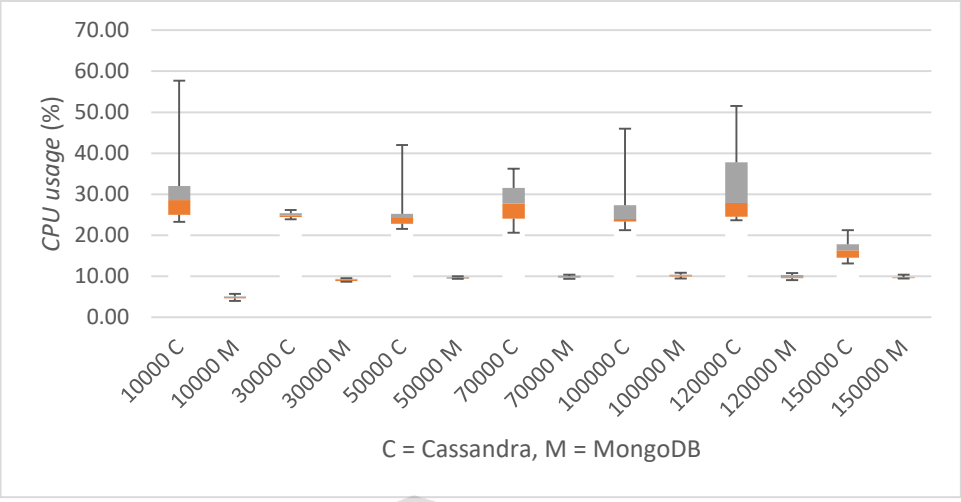
6.2.4 CPU Usage Cassandra dan MongoDB

Hasil pengujian parameter CPU usage Cassandra dan MongoDB dalam melakukan operasi insert data dengan data *string* dapat dilihat pada Tabel 6.7.

Tabel 6.8 Hasil pengujian *CPU usage* data *string*

Jumlah Variasi Data	Q1	Median	Q3	Max	Min
10000 C	25.0	28.6	32.0	57.7	23.3
10000 M	4.6	4.7	5.1	5.7	4.0
30000 C	24.4	24.7	25.3	26.2	23.9
30000 M	8.8	9.2	9.3	9.5	8.7
50000 C	22.8	24.2	25.2	42.0	21.5
50000 M	9.4	9.4	9.8	10.0	9.3
70000 C	24.0	27.7	31.6	36.2	20.6
70000 M	9.6	9.7	10.2	10.4	9.4
100000 C	23.3	24.0	27.3	46.0	21.2
100000 M	9.9	10.2	10.4	10.8	9.4
120000 C	24.5	27.9	37.8	51.5	23.7
120000 M	9.6	9.7	10.3	10.7	9.1
150000 C	14.5	16.3	17.8	21.2	13.1
150000 M	9.6	9.7	9.7	10.4	9.4

Data yang ada pada Tabel 6.8 disajikan dalam bentuk Whisker plot agar lebih mudah dibaca serta terlihat perbandingannya. Hasil dari *CPU usage* yang di dapat selama sepuluh kali pengujian operasi insert data dengan data *string* dapat dilihat pada Gambar 6.7.



Gambar 6.7 Hasil CPU Usage dengan data string

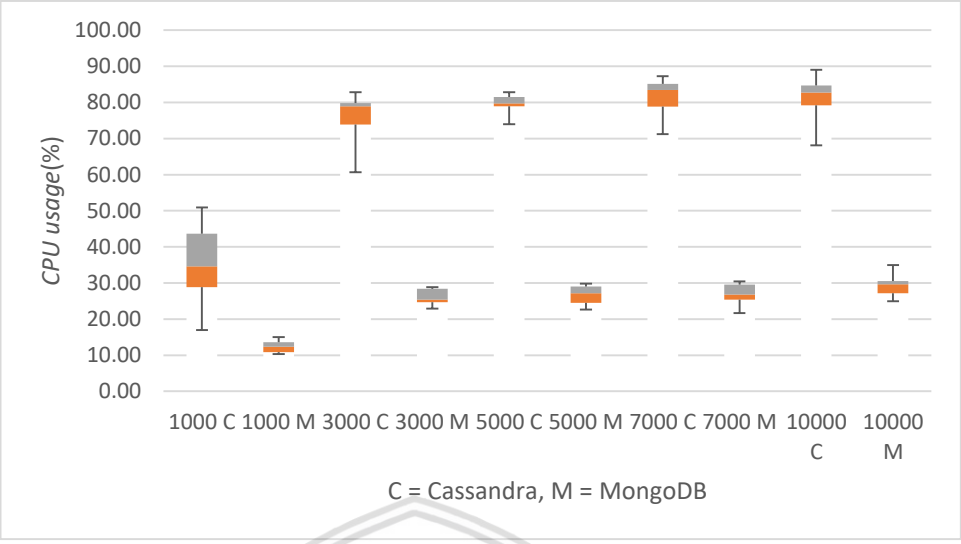
Pada Gambar 6.7 menunjukkan hasil perhitungan parameter penggunaan CPU dalam melakukan operasi insert data menggunakan data *string*. Dari hasil pengujian yang dilakukan, MongoDB memiliki penggunaan CPU paling efficient dibandingkan dengan Cassandra. Rentang penggunaan CPU pada saat melakukan operasi insert data antar 4% sampai 10% dibandingkan dengan Cassandra yang penggunaan nya lebih besar dengn rentang antara 16% sampai 28%.

Hasil pengujian parameter CPU usage Cassandra dan MongoDB dalam melakukan operasi insert data dengan data gambar dapat dilihat pada Tabel 6.8.

Tabel 6.9 Hasil pengujian *CPU usage* data gambar

Jumlah Variasi Data	Q1	Median	Q3	Max	Min
1000 C	28.9	34.6	43.6	50.9	17.0
1000 M	10.9	12.4	13.6	15.0	10.3
3000 C	73.8	78.9	79.8	82.8	60.7
3000 M	24.6	25.4	28.4	28.9	22.9
5000 C	78.9	79.8	81.5	82.8	74.0
5000 M	24.5	27.1	29.0	29.8	22.6
7000 C	78.8	83.4	85.1	87.2	71.2
7000 M	25.4	26.8	29.6	30.4	21.7
10000 C	79.2	82.8	84.7	89.1	68.2
10000 M	27.1	29.7	30.5	35.0	25.0

Data yang ada pada Tabel 6.9 disajikan dalam bentuk Whisker plot agar lebih mudah dibaca serta terlihat perbandingannya. Hasil dari CPU usage yang di dapat selama sepuluh kali pengujian operasi insert data dengan data gambar dapat dilihat pada Gambar 6.8.



Gambar 6.8 Hasil CPU Usage dengan data gambar

Pada Gambar 6.8 menunjukkan hasil perhitungan parameter penggunaan CPU dalam melakukan operasi insert data menggunakan data gambar. Dari hasil yang diapatkan, penggunaan CPU pada MongoDB lebih efisient dibandingkan dengan Cassandra. Rentang penggunaan CPU pada MongoDB yaitu antara 12% sampai dengan 29%. Sedangkan untuk Cassandra, rentang penggunaan CPU berkisar antara 34% sampai 83 %.

Dari pembahasan parameter CPU usage, dapat disimpulkan bahwa peningkatan penggunaan CPU dipengaruhi oleh besar kecilnya transaksi data yang dilakukan. Dalam melakukan operasi insert data string, MongoDB memiliki penggunaan CPU paling efisient dibanding dengan Cassandra dengan rentang antara 4% sampai 10%. Sedangkan dalam melakukan operasi insert dengan menggunakan data gambar, MongoDB memiliki penggunaan CPU paling efisient dibanding dengan Cassandra dengan rentang antara 12% sampai dengan 29%.

6.2.5 *disk I/O* Cassandra dan MongoDB

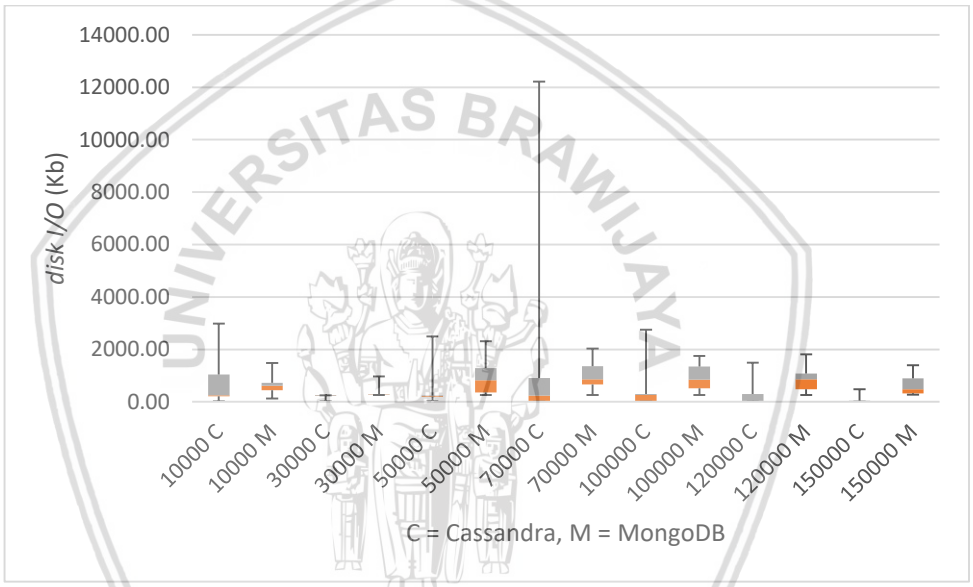
Hasil pengujian parameter *disk* i/o Cassandra dan MongoDB dalam melakukan operasi insert data dengan data *string* dapat dilihat pada Tabel 6.9.

Tabel 6.10 Hasil pengujian *disk* I/O dengan data *string*

Jumlah Variasi Data	Q1	Median	Q3	Max	Min
10000 C	209.6	245.3	1046.7	2984.9	12.0
10000 M	448.5	613.5	725.7	1483.0	123.0
30000 C	219.7	227.9	243.3	252.0	15.3
30000 M	269.9	275.2	279.8	970.4	264.8
50000 C	206.1	221.9	235.5	2495.3	11.5
50000 M	362.7	821.5	1286.2	2308.0	266.0
70000 C	12.2	237.0	908.1	12220.4	6.0
70000 M	668.0	857.3	1361.6	2032.8	265.0

100000 C	18.2	258.8	301.6	2755.6	5.6
100000 M	518.2	843.1	1352.5	1745.4	265.9
120000 C	5.8	10.8	300.0	1499.1	5.0
120000 M	475.0	853.7	1074.5	1815.6	267.3
150000 C	4.5	9.7	42.6	480.7	2.1
150000 M	316.6	479.8	894.4	1396.5	267.8

Data yang ada pada Tabel 6.10 disajikan dalam bentuk Whisker plot agar lebih mudah dibaca serta terlihat perbandingannya. Hasil dari *disk* I/O yang di dapat selama sepuluh kali pengujian operasi insert data dengan data *string* dapat dilihat pada Gambar 6.9



Gambar 6.9 Hasil *disk* I/O menggunakan data string

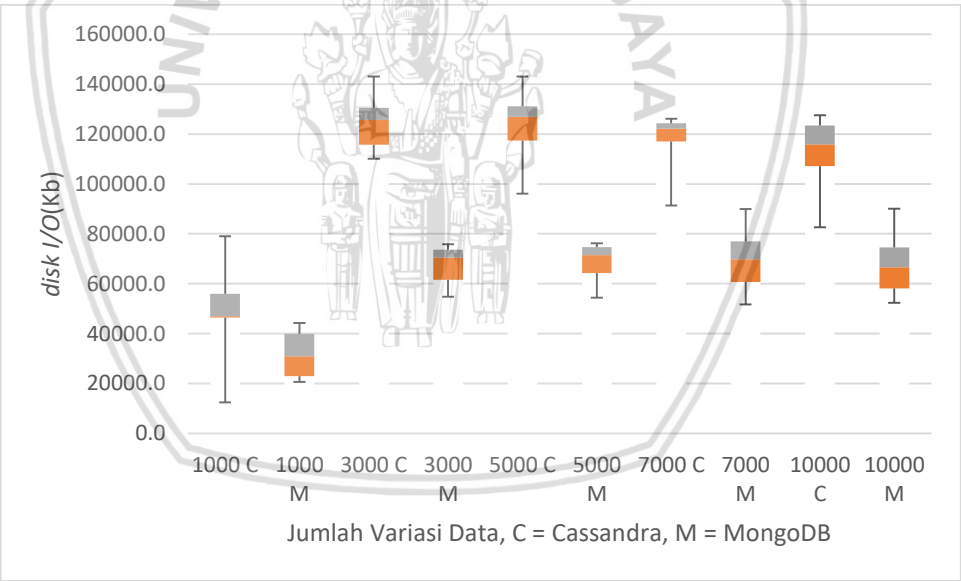
Pada Gambar 6.9 menunjukkan hasil parameter *disk* I/O yang didapat ketika melakukan operasi insert data menggunakan data string. Dari percobaan dalam melakukan operasi insert data dengan menggunakan data string, MongoDB memiliki nilai disk I/O yang lebih dibandingkan dengan Casandra. Pada saat jumlah variasi data berjumlah 10000 data, MongoDB mendapatkan nilai disk I/O sebesar 613.5 Kb dibandingkan dengan Cassandra yang hanya mendapatkan nilai disk I/O sebesar 245.3. Artinya kecepatan hardisk dalam menulis datanya lebih cepat menggunakan MongoDB dari pada menggunakan Cassandra pada saat data yang digunakan tidak terlalu besar ukurannya.

Hasil pengujian parameter disk I/O Cassandra dan MongoDB dalam melakukan operasi insert data dengan data gambar dapat dilihat pada Tabel 6.10.

Tabel 6.11 Hasil pengujian *disk* I/O data gambar

Jumlah Variasi Data	Q1	Median	Q3	Max	Min
1000 C	46468.1	46887.4	55951.1	79068.7	12393.3
1000 M	22910.9	30941.2	39848.2	44303.0	20658.8
3000 C	115716.9	125563.5	130434.2	143044.2	110035.5
3000 M	61593.9	70469.8	73600.3	75855.7	54820.2
5000 C	117376.9	126953.0	131187.3	143044.2	96139.4
5000 M	64274.6	71500.0	74659.6	76165.3	54359.6
7000 C	117035.7	122126.2	124344.3	126113.7	91331.2
7000 M	60638.4	69768.5	76988.1	89982.0	51680.0
10000 C	107172.3	115792.4	123420.2	127484.7	82628.6
10000 M	58130.6	66596.3	74590.3	90004.9	52362.9

Data yang ada pada Tabel 6.11 disajikan dalam bentuk Whisker plot agar lebih mudah dibaca serta terlihat perbandingannya. Hasil dari throughput yang di dapat selama sepuluh kali pengujian operasi insert data dengan data gambar dapat dilihat pada Gambar 6.10



Gambar 6.10 Hasil *DISK* I/O menggunakan data gambar

Pada Gambar 6.10 menunjukkan hasil parameter *disk* I/O yang didapat ketika melakukan operasi insert data menggunakan data gambar. Dari percobaan dalam melakukan operasi insert data dengan menggunakan data gambar, Cassandra memiliki nilai disk I/O yang lebih tinggi dibandingkan dengan MongoDB. Pada saat jumlah variasi data berjumlah 1000 data, Cassandra mendapatkan nilai disk I/O sebesar 46887.4 Kb dibandingkan dengan MongoDB yang hanya mendapatkan nilai disk I/O sebesar 30941.2 Kb. Artinya kecepatan hardisk dalam menulis datanya lebih cepat menggunakan Cassandra dari pada menggunakan MongoDB pada saat data yang digunakan ukurannya besar.

Dari hasil pembahasan parameter disk I/O, dapat disimpulkan bahwa kecepatan hardisk dalam menulis data dipengaruhi oleh besar kecil dari ukuran data yang di transaksikan. Pada saat melakukan operasi insert data dengan data string, nilai dari disk I/O terbaik didapatkan pada MongoDB dengan nilai 857.3 Kb. Sedangkan pada saat melakukan operasi insert data dengan data gambar, nilai dari disk I/O terbaik didapatkan pada Cassandra dengan nilai 126953 KB.

BAB 7 PENUTUP

7.1 Kesimpulan

- IoT data storage dengan NoSQL Cassandra dibangun pada topologi yang dikembangkan pada penelitian pramukantoro (2017). Pada topologi tersebut hanya mengganti data storage yang sebelumnya menggunakan NoSQL MongoDB dengan menggunakan NoSQL Cassandra. Dari hasil pengujian fungsionalitas yang dilakukan, NoSQL Cassandra dapat menyimpan data yang beragam dari node sensor pada topologi yang digunakan.
- Pengujian kinerja NoSQL Cassandra dan NoSQL MongoDB dilakukan dengan meninjau parameter *runtime, throughput, CPU usage, memory usage* dan *disk I/O* dalam operasi *insert* data. Data yang digunakan pada saat proses *insert* menggunakan data *string* dan data gambar.
- ***runtime:*** ditinjau dari parameter *runtime*, pada saat melakukan operasi *insert* menggunakan data *string*, NoSQL MongoDB lebih unggul dibandingkan dengan NoSQL Cassandra dengan perolehan runtime akhir sebesar 121.2 second. Sedangkan pada saat melakukan operasi insert dengan data gambar, NoSQL Cassandra lebih unggul dibandingkan dengan MongoDB dengan nilai *runtime* akhir sebesar 86.4 second

  throughput: ditinjau dari parameter *throughput*, NoSQL MongoDB memiliki nilai throughput lebih cepat dari pada NoSQL Cassandra pada saat melakukan operasi insert menggunakan data *string* dengan nilai *throughput* akhir sebesar 1236.7 ops/s. Sedangkan, pada saat melakukan operasi insert dengan data gambar, Cassandra yang memiliki nilai throughput lebih cepat dari pada MongoDB dengan nilai throughput akhir sebesar 115.8 ops/s.

  memory usage: ditinjau dari parameter *memory usage,* NoSQL Cassandra memiliki penggunaan memory yang efisien pada saat melakukan operasi insert menggunakan data string dengan nilai *memory usage* akhir sebesar 1690.4 MB. Sedangkan , pada saat melakukan operasi insert dengan data gambar, MongoDB memiliki penggunaan memory yang efisien dengan nilai *memory usage* akhir sebesar 3810.9 MB.

  CPU usage: ditinjau dari parameter *CPU usage*, NoSQL MongoDB memiliki penggunaan CPU yang efisien pada saat melaukan operasi insert data dengan menggunakan data *string* dengan rentang antara 4% sampai 10%. Sedangkan pada saat melakukan operasi insert dengan data gambar, NoSQL MongoDB memiliki penggunaan CPU yang efisien antara 12% sampai dengan 29%.

  Disk I/O: ditinjau dari parameter *disk I/O*, kecepatan hardisk dalam menulis data dipengaruhi oleh besar kecil dari ukuran data yang di transaksikan. Pada saat melakukan operasi insert data dengan data *string*, nilai dari disk I/O terbaik didapatkan pada NoSQL MongoDB dengan nilai 857.3 Kb.

Sedangkan pada saat melakukan operasi insert data dengan data gambar, nilai dari disk I/O terbaik didapatkan pada NoSQL Cassandra dengan nilai 126953 KB.

Kesimpulan yang dapat diambil adalah pada saat data yang dipertukarkan ukurannya besar, maka NoSQL Cassandra menjadi solusi untuk media penyimpanan datanya. Akan tetapi jika data yang dipertukarkan ukurannya kecil, maka NoSQL MongoDB menjadi solusi sebagai media penyimpanan datanya.

7.2 Saran

- Penelitian selanjutnya dapat dikembangkan dengan menggunakan variasi bentuk data yang berbeda – beda.
- Penelitian selanjutnya dapat membandingkan kinerja NoSQL Cassandra dan MongoDB dengan NoSQL database yang lain, salah satunya Hbase.
- Pengujian kinerja dapat dilakukan dengan menambah parameter uji yang lain seperti tingkat keamanan database.

DAFTAR PUSTAKA

Abramova, V., Bernardino, J. and Furtado, P. (2014) 'Testing Cloud Benchmark Scalability with Cassandra', *2014 IEEE World Congress on Services*, pp. 434–441. doi: 10.1109/SERVICES.2014.81.

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. Elsevier.

Chodorow, K., & Dirolf, M. (2010). *MongoDB*: The Definitive Guide (1st ed.). O'Reilly Media.

Elmasri, R. and Navathe, S. B. . (2011) *Fundamentals of Database Systems*. 6th edn. Addison-Wesley.

Enqingtang, & Yushun Fan. (2017). Performance Comparison between five noSQL *database*.

JMeter, A. (2017) *Apache JMeter - Apache JMeter™*, *Online*. Available at: http://jmeter.apache.org/ (Accessed: 5 March 2017).

Kuhlenkamp, J., Klems, M. and Röss, O. (2014) 'Benchmarking Scalability and Elasticity of Distributed Database Systems', *Proc. VLDB Endow.*, 7(12), pp. 1219–1230. doi: 10.14778/2732977.2732995.

Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, United States Department of Commerce

Nayak, A., Poriya, A. and Poojary, D. (2013) 'Type of NOSQL Databases and its Comparison with Relational Databases', *International Journal of Applied Information Systems*, 5(4), pp. 16–19.

Pramukantoro, Eko. Dkk. (2017). Topic Based *IoT* Data Storage Framework For Heterogeneous Sensor Data.

Purbo, O. W. (2011). Petunjuk Praktis Cloud Computing Menggunakan Open Source. Yogyakarta: CV Andi Offset.

Seguin, K. (2012). The Little *MongoDB* Book. San Fransisco: Git Hub Inc.

Zafar, R. *dkk.* (2017) 'Big Data: The NoSQL and RDBMS review', *ICICTM 2016 - Proceedings of the 1st International Conference on Information and Communication Technology*, (May), pp. 120–126. doi: 10.1109/ICICTM.2016.7890788.