PENGEMBANGAN PLATFORM PENGOLAHAN DATA SENSOR INTERNET OF THINGS BERJENIS STREAMING DENGAN KOMPUTASI TERDISTRIBUSI MENGGUNAKAN SPARK STREAMING

SKRIPSI

Untuk memenuhi sebagian persyaratan   
memperoleh gelar Sarjana Komputer

Disusun oleh:

Fidia Rosianti

NIM: 135150201111033



PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2019

PERSETUJUAN

PENGEMBANGAN PLATFORM PENGOLAHAN DATA SENSOR INTERNET OF THINGS BERJENIS STREAMING DENGAN KOMPUTASI TERDISTRIBUSI MENGGUNAKAN SPARK STREAMING

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan

memperoleh gelar Sarjana Komputer

Disusun Oleh :

Fidia Rosianti

NIM: 135150201111033

Telah diperiksa dan disetujui oleh:

|  |  |
| --- | --- |
| Dosen Pembimbing I  Adhitya Bhawiyuga, S.Kom, M.S  NIK. 201405 890720 1 001 | Dosen Pembimbing 2  Kasyful Amron, S.T, M.Sc  NIP. 19750803 200312 1 003 |

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 referensi.

Apabila ternyata didalam 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, 1 Januari 2015



­

Fidia Rosianti

NIM: 135150201111033

PRAKATA

Bagian ini memuat pernyataan resmi untuk menyampaikan rasa terima kasih penulis kepada berbagai pihak yang telah membantu penyelesaian skripsi ini. Nama-nama penerima ucapan terima kasih sebaiknya dituliskan lengkap, termasuk gelar akademik, dan pihak-pihak yang tidak terkait dihindari untuk dituliskan. Bahasa yang digunakan seharusnya mengikuti kaidah bahasa Indonesia yang baku. Prakata boleh diakhiri dengan paragraf yang menyatakan bahwa penulis menerima kritik dan saran untuk pengembangan penelitian selanjutnya. Terakhir, prakata ditutup dengan mencantumkan kota dan tanggal penulisan prakata, lalu diikuti dengan kata “Penulis”.

Malang, 1 Januari 2015

Penulis

email@domain.com

ABSTRAK

Nama Mahasiswa, Judul Skripsi

Pembimbing: Nama Pembimbing 1 dan Nama Pembimbing 2

Bagian ini diisi dengan abstrak dalam Bahasa Indonesia. Abstrak adalah uraian singkat (umumnya 200-300 kata) yang merupakan intisari dari sebuah skripsi. Abstrak membantu pembaca untuk mendapatkan gambaran secara cepat dan akurat tentang isi dari sebuah skripsi. Melalui abstrak, pembaca juga dapat menentukan apakah akan membaca skripsi lebih lanjut. Oleh karena itu, abstrak sebaiknya memberikan gambaran yang padat tetapi tetap jelas dan akurat tentang (1) apa dan mengapa penelitian dikerjakan: sedikit latar belakang, pertanyaan atau masalah penelitan, dan/atau tujuan penelitian; (2) bagaimana penelitian dikerjakan: rancangan penelitian dan metodologi/metode dasar yang digunakan dalam penelitian; (3) hasil penting yang diperoleh: temuan utama, karakteristik artefak, atau hasil evaluasi artefak yang dibangun; (4) hasil pembahasan dan kesimpulan: hasil dari analisis dan pembahasan temuan atau evaluasi artefak yang dibangun, yang dikaitkan dengan pertanyaan/tujuan penelitian.

Yang harus dihindari dalam sebuah abstrak diantaranya (1) penjelasan latar belakang yang terlalu panjang; (2) sitasi ke pustaka lainnya; (3) kalimat yang tidak lengkap; (3) singkatan, jargon, atau istilah yang membingungkan pembaca, kecuali telah dijelaskan dengan baik; (4) gambar atau tabel; (5) angka-angka yang terlalu banyak.

Di akhir abstrak ditampilkan beberapa kata kunci (normalnya 5-7) untuk membantu pembaca memposisikan isi skripsi dengan area studi dan masalah penelitian. Kata kunci, beserta judul, nama penulis, dan abstrak biasanya dimasukkan dalam basis data perpustakaan. Kata kunci juga dapat diindeks dalam basis data sehingga dapat digunakan untuk proses pencarian tulisan ilmiah yang relevan. Oleh karena itu pemilihan kata kunci yang sesuai dengan area penelitian dan masalah penelitian cukup penting. Pemilihan kata kunci juga bisa didapatkan dari referensi yang dirujuk.

Kata kunci: abstrak, skripsi, intisari, kata kunci, artefak

ABSTRACT

Student Name, Skripsi Title

Supervisors: First Supervisor’s Name and Second Supervisor’s Name

The absract of your skripsi in English is written here.

DAFTAR ISI

[PERSETUJUAN ii](#_Toc26290484)

[PERNYATAAN ORISINALITAS iii](#_Toc26290485)

[PRAKATA iv](#_Toc26290486)

[ABSTRAK v](#_Toc26290487)

[ABSTRACT vi](#_Toc26290488)

[DAFTAR ISI vii](#_Toc26290489)

[DAFTAR TABEL x](#_Toc26290490)

[DAFTAR GAMBAR xi](#_Toc26290491)

[DAFTAR LAMPIRAN xii](#_Toc26290492)

[BAB 1 PENDAHULUAN 1](#_Toc26290493)

[1.1 Latar Belakang 1](#_Toc26290494)

[1.2 Rumusan Masalah 2](#_Toc26290495)

[1.3 Tujuan 2](#_Toc26290496)

[1.4 Manfaat 3](#_Toc26290497)

[1.5 Batasan Masalah 3](#_Toc26290498)

[1.6 Sistematika Pembahasan 3](#_Toc26290499)

[BAB 2 LANDASAN KEPUSTAKAAN 5](#_Toc26290500)

[2.1 Tinjauan Kepustakaan 5](#_Toc26290501)

[2.2 Internet of Things 10](#_Toc26290502)

[2.3 Apache Spark 11](#_Toc26290503)

[2.3.1 Pemrograman dengan RDD 13](#_Toc26290504)

[2.3.2 Spark Streaming 16](#_Toc26290505)

[2.4 Apache Kafka 19](#_Toc26290506)

[2.5 Apache Cassandra 19](#_Toc26290507)

[2.6 MQTT 20](#_Toc26290508)

[2.6.1 Broker 20](#_Toc26290509)

[2.6.2 Client 20](#_Toc26290510)

[BAB 3 METODOLOGI PENELITIAN 21](#_Toc26290511)

[3.1 Tipe Penelitian 21](#_Toc26290512)

[3.2 Studi Literatur 22](#_Toc26290513)

[3.3 Analisis Kebutuhan 22](#_Toc26290514)

[3.4 Perancangan Sistem 24](#_Toc26290515)

[3.4.1 Perancangan Data Integration 25](#_Toc26290516)

[3.4.2 Perancangan Data Processing 26](#_Toc26290517)

[3.4.3 Perancangan Data Storage 26](#_Toc26290518)

[3.4.4 Perancangan Visualiasi 26](#_Toc26290519)

[3.4.5 Perancangan Pengujian 27](#_Toc26290520)

[3.5 Implementasi 27](#_Toc26290521)

[3.6 Pengujian dan Analisis 29](#_Toc26290522)

[3.6.1 Pengujian Fungsional 29](#_Toc26290523)

[3.6.2 Pengujian Non-fungsional 30](#_Toc26290524)

[3.6.3 Analisis Hasil Pengujian 30](#_Toc26290525)

[3.7 Kesimpulan 30](#_Toc26290526)

[BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN 31](#_Toc26290527)

[4.1 Deskripsi Umum Sistem 31](#_Toc26290528)

[4.2 Batasan Perancangan dan Implementasi 32](#_Toc26290529)

[4.3 Analisis Kebutuhan 32](#_Toc26290530)

[4.3.1 Kebutuhan Fungsional 33](#_Toc26290531)

[4.3.2 Kebutuhan Non-Fungsional 33](#_Toc26290532)

[4.4 Perancangan 35](#_Toc26290533)

[4.4.1 Perancangan Komponen Integrasi Data 35](#_Toc26290534)

[4.4.2 Perancangan Komponen Pengolahan Data 35](#_Toc26290535)

[4.4.3 Perancangan Komponen Penyimpanan Data 35](#_Toc26290536)

[4.4.4 Perancangan Visualisasi 35](#_Toc26290537)

[4.4.5 Perancangan Pengujian 35](#_Toc26290538)

[BAB 5 IMPLEMENTASI 36](#_Toc26290539)

[BAB 6 PENGUJIAN DAN ANALISIS 37](#_Toc26290540)

[BAB 7 PENUTUP 38](#_Toc26290541)

[7.1 Kesimpulan 38](#_Toc26290542)

[7.2 Saran 38](#_Toc26290543)

[DAFTAR REFERENSI 39](#_Toc26290544)

DAFTAR TABEL

[Tabel ‎2.1 Pembentukan bilangan random untuk Indeks Masa Tubuh (IMT) 9](#_Toc494110309)

[Tabel ‎2.2 Contoh tabel 2 10](#_Toc494110310)

DAFTAR GAMBAR

[Gambar ‎2.1 Pengaruh nilai K terhadap akurasi 12](#_Toc494110311)

DAFTAR LAMPIRAN

[LAMPIRAN A PERSYARATAN FISIK DAN TATA LETAK 29](#_Toc496081034)

[A.1 Kertas 29](#_Toc496081035)

[A.2 Margin 29](#_Toc496081036)

[A.3 Jenis dan Ukuran Huruf 29](#_Toc496081037)

[A.4 Spasi 29](#_Toc496081038)

[A.5 Kepala Bab dan Subbab 29](#_Toc496081039)

[A.6 Nomor Halaman 30](#_Toc496081040)

[LAMPIRAN B PENGGUNAAN BAHASA 31](#_Toc496081041)

# PENDAHULUAN

Pada bab ini akan dijelaskan gambaran mengenai topik penelitian yang akan disajikan. Pada bab pendahuluan memuat latar belakang penelitian, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika penelitian.

## Latar Belakang

Dewasa ini, Internet of things (IoT) menjadi salah satu inovasi di bidang industri. Kementrian Ekonomi, Perdagangan dan Industri Jepang menyatakan bahwa IoT akan mengisi data-driven society dimana data digital yang dikumpulkan oleh objek IoT akan menambah nilai dan manfaat bagi organisasi (2015 disitasi dalam Yasumoto, 2017). Menurut McKinsey (2015), potensi ekonomi dari pengaruh IoT adalah sekitar 11,1 triliun dolar per tahun pada tahun 2025. IoT dapat menyediakan solusi dan metode dalam melakukan efisiensi dan menurunkan cost disertai peningkatan kualitas pada bidang industri. Smart agriculture menjadi salah satu penggunaan teknologi IoT yang akan berkembang dan memberikan dampak besar bagi produktivitas pertanian. Permintaan produksi makanan akan meningkat terus seiring berkembangnya populasi global saat ini. IoT menyediakan solusi untuk precision crop monitoring dan diagnosis penyakit pada tanaman yang menggunakan berbagai sumber data. Berbagai jenis sensor seperti kelembapan, suhu dan pH ditempatkan di beberapa tempat disesuaikan dengan topografi dan kebutuhan pada area. Sensor tersebut dibangun untuk melakukan sensing di sebuah area untuk ymelakukan monitor fenomena yang spesifik dan mengumpulkan data (Farhan et al, 2018).

Sensor menghasilkan data dengan jumlah yang sangat besar dan kecepatan yang sangat tinggi pula. Data IoT menggambarkan sebuah tantangan aspek velocity pada big data yang dihasilkan secara terus menerus dan berasal dari banyak sumber dengan periode berbeda. Data streaming IoT akan memberikan manfaat yang besar jika dilakukan pengolahan data. Pengolahan data menjadi komponen penting pada IoT untuk menggali nilai dari data tersebut. Beberapa pengolahan data lebih bernilai jika data tersebut segera diproses setelah data dihasilkan. Oleh karena itu stream processing menjadi hal yang penting dalam aplikasi IoT.

Komputasi data streaming menyediakan pengolahan data dengan throughput yang tinggi dan latency yang rendah dengan didukung arsitektur pengolahan paralel secara distribusi untuk menggali potensi dari data tersebut (Vananthi & Khadir, 2017). Pemanfaatan data streaming telah merubah paradigma dalam penggunaan arsitektur secara terdistribusi karena arsitektur yang berbasis centralized akan menyebabkan delay yang tinggi dalam penyediaan layanan (Yasumoto et al, 2017). Tetapi pada sistem terdistribusi dan jaringan, selalu ada celah gangguan seperti matinya server. Sehingga dibutuhkan platform yang mampu menangani pengolahan data stream IoT dengan skalabilitas yang tinggi dan fault-tolerant (Ounacer et al, 2014).

Terdapat beberapa sistem terdistribusi yang digunakan dalam pengolahan data streaming. Apache Spark menjadi salah satu *engine* pengolahan *big data* terbaik karena kapabilitas kinerja yang tinggi, infrastruktur yang kaya dan komunitas yang luas (Chintapali et al, 2016) (Zaharia et al, 2016). Spark Streaming dapat menangani masalah *node failure* dengan baik. Spark Streaming melakukan pemulihan tanpa kehilangan *message*. Spark Streaming menyimpan keseluruhan *processing state* pada *micro-batch* dan di distribusikan ke *worker node* lainnya sehingga tidak ada *message* yang hilang (Lopez et al, 2016). Serta, pemulihan Spark Streaming jika terdapat worker yang mati lebih stabil jika ada peningkatan jumlah data (Qian et al, 2016). Karena Spark Streaming menggunakan pendekatan micro-batch, hal tersebut menyebabkan *latency* yang tinggi, tetapi mempunyai *throughput* yang lebih tinggi. Dalam mengolah data streaming, tidak hanya berfokus pada Spark Streaming. Tetapi membutuhkan sistem yang mendukung Spark Streaming untuk menggali value dari data streaming untuk aplikasi IoT. Berdasarkan kebutuhan dan tantangan yang dihadapi, hal tersebut menentukan perbedaan pada prinsip desain dan optimasi ketika mendesain solusi pengolahan data streaming untuk aplikasi IoT (Ge et al, 2016).

Dari pembahasan sebelumnya, maka penelitian ini mengembangkan platform Spark Streaming yang mampu mengolah data sensor IoT berjenis streaming. Sumber data dapat berasal dari berbagai macam sensor, sehingga dibutuhkan sistem messaging yang mampu menampung data streaming sebelum diolah. Data tersebut diolah Spark Streaming secara terdistribusi untuk menghasilkan value dengan periode tertentu. Hasil pengolahannya ditentukan oleh operasi output pada Spark Streaming. Dengan operasi tersebut dapat menentukan langkah selanjutnya seperti menyimpan ke sistem penyimpanan eksternal dan visualisasi.

Berdasarkan latar belakang tersebut, subbab 1.2 menyajikan rumusan masalah penelitian. Subbab 1.3 dan 1.4 menyajikan tujuan dan manfaat penelitian. Subbab 1.5 menyajikan berbagai batasan masalah penelitian. Dan subbab 1.6 menyajikan sistematika penulisan dokumen penelitian.

## Rumusan Masalah

Berdasarkan latar belakang di atas, diperoleh rumusan masalah yang menjadi pedoman dalam penelitian ini adalah sebagai berikut:

Bagaimana menerapkan Spark Streaming dalam pengolahan data sensor IoT berjenis streaming dengan komputasi terdistribusi ?

Bagaimana kinerja Spark Streaming dalam mengolah data sensor IoT berjenis streaming berdasarkan *latency*, *throughput*, *fault tolerance* dan penggunaan CPU serta memori ?

## Tujuan

Tujuan dari penelitian ini untuk mendesain dan mengimplementasikan pengembangan platform Spark Streaming dalam melakukan pengolahan data streaming IoT secara terdistribusi. Penanganan data dengan kecepatan yang tinggi, membutuhkan sistem yang mendukung Spark Streaming. Pengembangan platform berdasarkan kebutuhan dan model referensi IoT. Sehingga pengolahan data dapat diperoleh secara realtime dan tepat.

## Manfaat

Manfaat penelitian ini adalah sebagai berikut:

Penelitian ini dapat membantu pengembangan platform yang berkaitan dengan pengolahan data IoT berjenis menggunakan Spark Streaming.

Penelitian ini menjadi sarana referensi untuk mengembangkan dan melanjutkan penelitian yang terkait dengan skripsi ini.

## Batasan Masalah

Agar permasalahan yang dirumuskan dapat lebih terfokus, maka terdapat batasan-batasan dalam lingkup penelitian. Batasan masalah dalam penelitian ini adalah sebagai berikut:

Penelitian ini berfokus pada pengembangan paltform pengolahan data streaming menggunakan Spark Streaming, sehingga data untuk menguji platform menggunakan data *dummy* yang dibuat secara acak.

Penelitian tidak melibatkan perangkat fisik seperti sensor.

Implementasi sistem dilakukan pada emulator.

## Sistematika Pembahasan

Dalam penyusunan dokumen penelitian pengolahan data sensor IoT berjenis streaming ini, terdapat struktur penulisan beserta deskripsi singkat dengan susunan sebagai berikut:

**BAB 1** **PENDAHULUAN**

Bab ini berisi latar belakang penelitian pengolahan data sensor IoT berjenis streaming yang menjelaskan tentang permasalahan yang diangkat dan menjelaskan tujuan dilakukannya penelitian, identifikasi masalah, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah dan sistematika dokumen penelitian.

**BAB 2** **LANDASAN KEPUSTAKAAN**

Bab ini menyajikan rangkuman referensi baik berupa jurnal internasional maupun buku, penelitian sebelumnya yang berkaitan dengan pengolahan data sensor IoT berjenis streaming, serta dasar teori yang mendukung penelitian ini.

**BAB 3 METODOLOGI PENELITIAN**

Bab ini menjelaskan metode yang digunakan penulis untuk menyelesaikan masalah pengolahan data sensor IoT berjenis streaming, perancangan sistem dan pengujian sistem. Perancangan sistem dijelaskan berdasarkan studi literatur dan penelitian sebelumnya pada bab 2. Pengujian sistem dilakukan berdasarkan asumsi teknologi yang digunakan.

**BAB 4** **ANALISIS KEBUTUHAN DAN** **PERANCANGAN**

Bab ini menyajikan gambaran penelitian secara mendetail berdasarkan skenario penelitian. Perancangan sistem pengolahan data sensor IoT berjenis streaming dilakukan berdasarkan landasan literasi pada bab 2 dan metodologi penelitian pada bab 3.

**BAB 5 IMPLEMENTASI**

Bab ini menyajikan implementasi teknis dari sistem pengolahan data sensor IoT berjenis streaming berdasarkan perancangan yang berada pada bab 4. Penjabaran implementasi ini menyajikan potongan kode program dan penjelasan fungsinya.

**BAB 6 PENGUJIAN DAN PEMBAHASAN**

Bab ini membahas hasil pengujian sistem pengolahan data sensor IoT berjenis streaming. Pengujian dilakukan berdasarkan perancangan skenario pada bab 4.

**BAB 7 PENUTUP**

Bab ini menguraikan kesimpulan penelitian sistem pengolahan data sensor IoT berjenis streaming berdasarkan hasil dari pengujian pada bab 6. Bab ini juga menjelaskan saran untuk penelitian selanjutnya yang serupa dapat digunakan dan dikembangkan.

# LANDASAN KEPUSTAKAAN

Berdasarkan uraian permasalahan penelitian, dijelaskan bahwa penelitian ini menjadi salah satu alternatif dalam mengembangkan platform untuk mengolah data sensor IoT berjenis streaming secara terdistribusi menggunakan Spark Streaming. Untuk mencapai tujuan penelitian, digambarkan arsitektur pada platform yang dibangun untuk pengolahan data sensor IoT berjenis streaming. Pengembangan platform berdasarkan studi penggunaan Spark Streaming. Platform terdiri dari 4 komponen sebagai berikut.

**Integrasi data**: Apache Kafka digunakan untuk mengumpulkan dan menyalurkan data dari beberapa sumber ke pengolahan data. Untuk melakukan pengujian sistem, data streaming dibuat menggunakan data *dummy* dan dikirim menggunakan protokol MQTT. Apache Kafka juga menampung hasil pengolahan dari dari Spark Streaming sebelum ditampilkan melalui aplikasi website.

**Pengolahan data**: Spark Streaming menjadi data processor karena kemampuan untuk mengolah data streaming. Arsitektur Apache Spark dibuat terdistribusi sehingga dapat mengoptimasi kinerja *worker*.

**Penyimpanan data**: Apache Cassandra digunakan untuk penyimpanan raw data dan hasil pengolahan.

**Visualisasi:** Hasil pengolahan akan ditampilkan melalui aplikasi website secara real time.

Pada bab ini menjelaskan kajian pustaka dan dasar teori yang berkaitan dengan penelitian. Kajian pustaka membahas mengenai penelitian yang pernah dilakukan sebelumnya. Selain itu dijelaskan dasar-dasar teori yang berkaitan dengan penelitian. Dasar teori yang akan dibahas adalah Internet of Things, Spark Streaming, Apache Kafka, Apache Cassandra dan MQTT.

## Tinjauan Kepustakaan

Pada sub bab ini menjelaskan referensi penelitian yang berkaitan dengan penelitian yang akan dilakukan. Referensi penelitian didapatkan dari jurnal dan buku yang berkaitan dengan pengolahan data streaming menggunakan Spark Streaming. Terdapat beberapa referensi penelitian yang sebelumnya pernah dilakukan dan menjelaskan keterkaitan penelitian tersebut dengan penelitian ini.

Penelitian dilakukan oleh S. Ounacer, M. A. Talhaoui, S. Ardchir, A. Daif dan M. Azouazi (2017), “A New Architecture for Real Time Data Stream Processsing” yang menyajikan sebuah gambaran arsitektur baru untuk pengolahan data stream dan penerapannya menggunakan teknologi pengolahan data stream yang sudah ada. Penelitian ini mengangkat masalah arsitektur dan teknologi yang digunakan dalam pengolahan data stream secara *real-time*. Karena pada saat penelitian ini berlangsung, sistem big data menggunakan teknologi Hadoop. Hadoop menggunakan konsep MapReduce dalam pengolahan datanya. Hadoop mempunyai kinerja proposional untuk kompleksitas data yang besar. MapReduce merupakan model pemrograman yang digunakan mengolah sejumlah data yang mempunyai ukuran tetap dan tidak cocok untuk pengolahan data stream. Karena data stream mempunyai ukuran, kecepatan dan kompleksitas yang tinggi. Sehingga membawa tantangan yang berbeda.

Penelitian ini menyajikan gambaran arsitektur baru untuk pengolahan data streaming. Arsitektur tersebut didesain berdasarkan kekurangan dan kelebihan dari arsitektur Lambda dan Kappa. Arsitektur ini dapat mengumpulkan, mengorganisasikan, mengintegrasi, memproses, menganalisis, menyimpan dan menyediakan visualisasi data stream dengan latency yang rendah. Terdapat 5 layer pada arsitektur baru yaitu *integration layer*, *filtering layer*, *real time processing layer*, *storage layer* dan *presentation layer*. Meskipun demikian, arsitektur tersebut tidak bersifat mengikat. Pemilihan tersebut dipilih yang paling cocok dan efisien berdasarkan kebutuhan dan kemudahan analis atau developer dalam menyelesaikan masalah. Arsitektur sebelumnya terdapat 3 layer saja yaitu penyimpanan, pengolahan dan analisis. Arsitektur baru ini bekerja berbeda. Data stream dapat berasal dari berbagai macam objek, sehingga dibutuhkan *integration layer*. Pada penelitian ini, *integration layer* menggunakan Apache Kafka. Layer ini berguna untuk mengumpulkan data dengan berbagai format dan kecepatan. Pada *filtering layer* berguna untuk menghilangkan *field* yang tidak diinginkan sebelum diproses pada *real time processsing layer*. Untuk menyaring data stream dapat menggunakan beberapa algoritma seperti *sliding window*, *load shedding* dan *synopsis data model*. Setelah data siap untuk diolah, data stream diproses secara real time dan dengan latency yang rendah. Pada real time processing layer menggunakan Apache Storm untuk mengaplikasikan beberapa fungsionalitias dan machine learning. Setelah data diolah, data di simpan kedalam storage layer yang menggunakan HBase dan membuat visualisasi untuk menampilkan hasil secara real time.

Arsitektur Lambda dan Kappa hanya fokus pada masalah kinerja dengan menyeimbangkan throughput dan latency dibandingkan dengan masalah kualitas data dan hasil analisis data. Arsitektur baru menyediakan beberapa layer tambahan sehingga memberikan solusi yang meningkatkan efisiensi dalam mengolah data streaming. Dalam mengolah data stream menggunakan Storm yang merupakan sistem komputasi real time yang bersifat *fault-tolerant*. Storm dapat secara cepat melakukan *compile* dan menyebarkan komputasi *real time* ke cluster dan memprosesnya. Pada layer tersebut, diintegrasikan distribusi algoritma *machine learning*.

Penelitian tersebut memiliki keterkaitan dengan penelitian yang akan dilakukan, yaitu gambaran arsitektur yang digunakan sebagai referensi dalam mengolah data streaming IoT. Arsitektur dan teknologi yang digunakan menyesuaikan dengan kebutuhan peneliti dalam mengembangkan platform pengolahan data streaming. Penelitian yang akan dilakukan mempunyai 4 layer yang terdiri dari integration layer, processing layer, storage layer dan visualisasi. Dengan integration layer sama menggunakan Apache Kafka, tetapi ditambahkan dengan integrasi dari MQTT. Processing layer menggunakan Spark Streaming, storage layer menggunakan Apache Cassandra dan visualisasi menggunakan website. Pada penelitian sebelumnya, belum dijelaskan visualiasi yang digunakan.

Penelitian selanjutnya dilakukan oleh O. Carvalho, E. Roloff, P. O. A. Navaux (2018) dengan judul “*A Distributed Stream Processing based Architecture for IoT Smart Grids Monitoring*”, penelitian ini mengusulkan sebuah arsitektur untuk data IoT dengan kombinasi data sensor dan sistem pengolahan stream secara terdistribusi khususnya untuk *smart grid*. Meskipun konektivitas sensor akan menghasilkan keuntungan yang besar, tetapi menghasilkan tantangan yang besar pula dalam mengolahnya. Karena data yang dihasilkan oleh sensor IoT memiliki kecepatan berbeda dan membutuhkan jaminan bahwa data akan segera diproses sehingga menghasilkan *values* dalam waktu tertentu. *Smart grid* akan mengizinkan konsumen menerima *feedback* secara *near real-time* tentang konsumsi energi dan biaya. Sehingga dapat menghemat milyaran dolar untuk konsumer dan generator, berdasarkan perkiraan terbaru. Karena data diambil dari jutaan pengguna, dibutuhkan pengolahan dan aliran informasi untuk smart grids yang mempunyai skalabilitas tinggi. Layanan ini menyediakan kemampuan yang reliabel, terdistribusi dan *redundant* dengan skala global.

Gambaran arsitektur dibangun berdasarkan penelitian *state-of-the-art* pada pengolahan stream secara terdistribusi. Arsitektur dapat mempertahankan karakteristik throughput yang tinggi dan latency yang rendah dengan tetap memperhatikan skalabilitas dan ketersediaan. Terdapat 3 bagian yaitu messaging layer, processing layer dan volatile layer. Layer tersebut menyediakan sebuah arsitektur reliabel yang dibangun untuk skalabilitas yang tinggi, sehingga dapat mengolah jumlah data yang sangat besar dengan throughput yang tinggi dan latency yang rendah. Messaging layer digunakan untuk menjembatani sumber data dengan layer selanjutnya. Apache Kafka digunakan pada *messaging layer* karena mempunyai kemampuan untuk mendukung skalabilitas yang besar dan *throuhgput* yang tinggi, dengan *fault tolerant* yang baik. Processing layer digunakan untuk mengolah data ketika data tersebut datang. Untuk menggali value dari data tersebut dapat menggunakan fungsionalitas dari data processor. Spark Streaming digunakan pada processing kayer karena kemampuan untuk menyediakan pengolahan dengan throughput yang tinggi dan memiliki semantik exactly-once dengan pemrograman API yang sederhana. Setelah data diolah, pengguna menentukan operasi output dapat dikembalikan ke messaging layer atau volatile layer. Operasi output disesuaikan dengan kebutuhan dan tujuan penelitian. Volatile layer digunakan untuk cache dari hasil pengolahan dari processing layer ke sistem eksternal. Hal tersebut berguna untuk menyimpan keseluruhan data stream menjadikan sebagai data histori. Redis digunakan pada volatile layer karena dapat melakukan query dengan sederhana dan tidak membutuhkan penyimpanan data yang permanen, karena model tersebut sesuai dengan kebutuhan arsitektur.

Data diambil dari disk dan memasukannya ke topik Kafka. Pada sistem Kafka, data akan dibuat data partisi dan mengirimkannya ke node Spark Streaming. Untuk proses pengolahan data, Spark Streaming menggunakan I node Spark master dan 8 node Spark slave. Pengolahan data dimulai dengan membaca input data dari Apache Kafka kemudian melakukan *filtering* untuk menghilangkan field yang tidak diinginkan untuk proses pengolahan data. Setelah data siap untuk diolah, menentukan *aggregate windowing* untuk mengaplikasikan algoritma prediksi. Proses tersebut digunakan untuk menghitung rata-rata untuk setiap *time slice* berdasarkan *aggregate windowing*, secara berurutan sistem akan melakukan *update* median ke iterasi berikutnya. Hasil rata-rata akan digunakan untuk menghitung tahapan *load prediction*. Hasilnya akan disimpan di Redis dan dapat digunakan untuk melakukan *query* eksternal ke AMI. Selain itu hasilnya dapat dikembalikan ke topik Kafka yang berbeda untuk melakukan *query* lainnya. Untuk melihat kinerja Spark Streaming, terdapat 2 parameter yang digunakan yaitu latency dan throughput. Ketika ukuran batch dibesarkan, sistem mempunyai waktu yang cukup dalam menentukan schedule dan mengolah data dalam pipeline, sehingga tidak menimbulkan delay yang tinggi. Tetapi tidak berpengaruh secara signifikan untuk parameter throughput. Meskipun batch size dibesarkan, kinerja Spark Streaming tetap stabil.

Penelitian tersebut memiliki keterkaitan dengan arsitektur yang digunakan. Tetapi teknologi penyimpanan berbeda dengan rencana penelitian. Jika penelitian sebelumnya menggunakan Redis sebagai tempat penyimpanan sementara, rencana penelitian menggunakan Apache Cassandra yang bersifat permanen. Selain itu, penelitian sebelumnya tidak menjelaskan proses visualiasi untuk menampilkan hasil yang didapat. Rencana penelitian menggunakan website sebagai sarana untuk menampilkan hasil secara real time. Untuk menguji Spark Streaming, rencana penelitian menambahkan parameter penggunaan memory, cpu dan disk.

Penelitian selanjutnya dilakukan oleh Y. Nait Malek, A. Kharbouch, H. El Khoukhi, M. Bakhouya, V. De Florio, D. El Ouadghiri, S. Latre, C. Blondia(2017) dengan judul “*On the use of IoT and Big Data Technologies for Real-time Monitoring and Data Processing*”, penelitian ini mengusulkan sebuah platform yang mengkombinasikan teknik IoT dengan teknologi Big Data untuk monitoring data secara terus menerus dan real time serta mengolahnya. Saat ini, koneksi dari benda-benda ke Internet yang disebut IoT dapat membangun aplikasi kecerdasan dan layanan-layanan lainnya. Variasi dari berbagai jenis benda seperti RFID tags, sensor, aktuator, ponsel dan lainnya saling terhubung dan berkerja sama untuk meraih tujuan. Data yang dihasilkan pun sangatlah banyak. Pertumbuhan data streaming membentuk yang dinamakan fenomena “Big Data”. Saat ini, beberapa teknologi IoT dan Big data telah dibangun untuk mengizinkan dalam pengumpulan, pengolahan dan analisis data streaming yang berjumlah sangat besar. Tujuan dilakukan pengolahan secara realtime untuk mengantisipasi perubahan dan menyesuaikan kebutuhan pengguna. Ada beberapa IoT platform dapat dengan mudah diaplikasikan. Akan tetapi, aplikasi IoT saat seperti Thingspeak, dapat digunakan monitoring data dan analisis tanpa mempunyai jaminan pada transmisi dan pengolahan. Penggunaan teknologi pengolahan real time seperti Storm dan S4, dapat dibangun untuk mengolah data dengan cepat sehingga mendapatkan nilai dari data tersebut secara realtime. Penelitian ini bertujuan untuk menyediakan sebuah platform yang mampu mengolah data sensor dengan skala besar untuk membangun *context-aware application* dan layanan. Pembuatan platform berdasarkan kajian penggunaan kombinasi teknologi Kaa dan Storm kedalam sebuah platform holistik. Sebuah purwarupa dari platform tersebut dibangun untuk membantu di bidang kesehatan.

Arsitektur yang dibangun berguna untuk pengolahan data streaming dari sensor. Peneliti menggunakan teknologi open source seperti Apache Storm, Kaa dan Apache Flume. Platform terdiri dari 3 layer antara lain adalah akuisisi data, pengolahan data, penyimpanan data dan visualisasi. Akuisisi data menggunakan platform Kaa untuk mengumpul data dari berbagai macam sensor. Platform ini berguna sebagai middleware yang berguna untuk membangun, mengatur dan mengintegrasi berbagai macam perangkat IoT. Kaa mempunyai kemampuan skalabilitas secara horizontal, fault-tolerant dan menyediakan beberapa fitur IoT. Pengolahan data menggunakan Storm. Storm mempunyai kemampuan untuk mengolah data secara realtime, mudah dan mendukung beberapa bahasa pemrograman. Penyimpanan data menggunakan MongoDB dan aplikasi website sebagai visualisasi. Terdapat 3 komponen utama dalam membangun dan menguji layanan yaitu alat pendeteksi, perangkat pengolahan dan infrastruktur pengolahan/penyimpanan. Alat pendeteksi menggunakan Arduino yang berhubungan dengan sensor dan aktuator. Untuk komunikasi antara Arduino dan aplikasi Kaa menggunakan protokol MQTT. Peneliti menggunakan sebuah perangkat Raspberry untuk melakukan pengolahan jangka pendek seperti agregasi data. Ini berguna untuk menggabungkan aplikasi Kaa sehingga mendapatkan data sensor dari program Arduino. Kemudian data tersebut dikirimkan ke cluster Storm. Aplikasi pada Storm berdasarkan topologi pengolahan yaitu spout/bolt. Untuk mengirimkan data dari aplikasi Kaa ke aplikasi Storm, Apache Flume digunakan. Apache Flume digunakan untuk mengambil, agregasi dan memindahkan data yang berjumlah besar. Teknologi itu mempunyai arsitektur yang mudah dan fleksibel untuk aliran data streaming. Infrastruktur cluster digunakan untuk pengolahan data dan penyimpanan. Pada penelitian ini menggunakan sebuah cluster yang terdiri dari 6 node.

Tujuan dari penelitian berdasarkan studi kasus kesehatan adalah untuk memeriksa dan analisis organ vital dari penghuni bangunan saat terkena karbondioksida dan bieffluent. Percobaan dilakukan menggunakan alat medikal seperti detak jantung dan oximetry, karbondioksida pada udara, sensor kelembapan dan sensor suhu. Penelitian ini menguraikan bagaimana kekurangan ventilasi pada bangunan dapat berdampak pada kinerja dan kesehatan penghuni bangunan. Untuk menguji korelasi antara konsentrasi karbondioksida dan oksigen di darah, terdapat 2 komponen utama yaitu aplikasi Kaa dan aplikasi Storm. Aplikasi Kaa berguna untuk mengambil data dari 2 entitas berbeda yaitu lingkungan (petugas keamanan) dan penghuni bangunan. Aplikasi Kaa berguna untuk mengumpulkan data lingkungan dan organ vital penghuni. Kemudian data tersebut dikirimkan ke cluster pengolahan. Aplikasi Storm berguna untuk mengolaha data dengan algoritma yang sesuai dengan kebutuhan. Topologi Storm mempunyai 2 spout yang membaca data dari Flume kemudian mengirimkannya ke bolt sebagai sebuah stream. Kemudian dilakukan monitoring variasi konsentrasi level karbondioksida dan oksigen. Jika level karbondioksida lebih besar dari oksigen, kinerja tubuh akan menurun.

Penelitian tersebut memiliki keterkaitan dengan arsitektur yang digunakan. Penelitian tersebut menjelaskan komponen layer yang masing-masing komponen mempunyai tugas berdasarkan tujuan penelitian. Layer yang digunakan sama dengan penelitian yang akan dilakukan. Tetapi menggunakan teknologi yang berbeda. Komponen akuisis data, rencana penelitian menggunakan Kafka. Kemudian komponen pengolahan data menggunakan Spark Streaming. Dan komponen penyimpanan data menggunakan Cassandra. Penelitian tersebut tidak dilakukan pengujian sistem. Sehingga tidak diketahui kinerja Storm dalam pengolahan datanya.

## Internet of Things

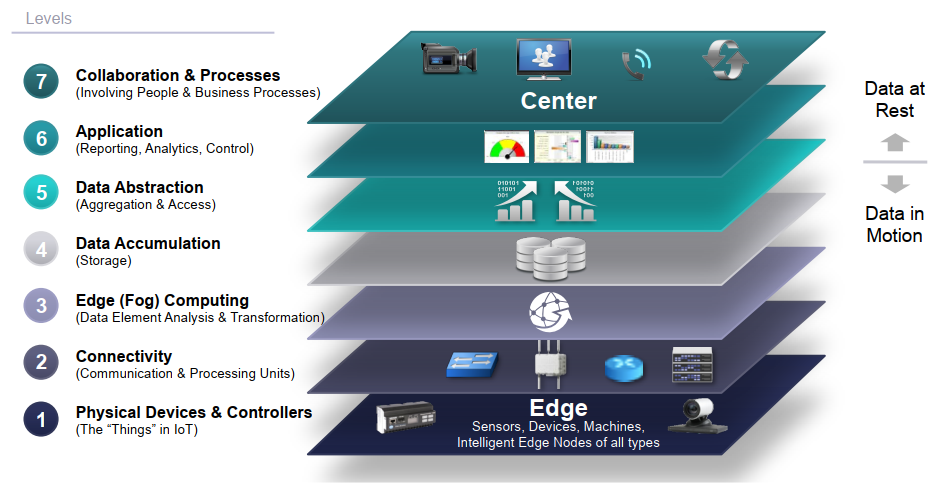
Internet of things (IoT) merupakan paradigma yang berkembang di bidang telekomunikasi wireless. Ide dasar dari konsep ini adalah berbagai macam “things” atau objek seperti Radio-Frequency Identification (RFID), tags, sensor, mobile phone dan lain-lain yang saling berinteraksi dan bekerja sama satu dengan yang lainnya untuk mencapai tujuan (Atzori, et. al., 2010). Paradigma ini membuka inovasi-inovasi baru yang akan membangun jenis interaksi antar objek dan manusia. IoT mendukung integrasi, transfer dan analitik dari data yang dihasilkan oleh *smart device* seperti sensor (Buyya et al, 2017).

IoT meningkatkan koneksi antara manusia dan objek serta volume data yang dihasilkan sangatlah besar. Sensor menghasilkan data streaming yang selalu dihasilkan terus menerus dengan berbagai kecepatan. Sebuah data streaming secara berurutan dan berpasangan (S,T) dimana :

* S adalah sebuah rangkaian *tuple*.
* T adalah sebuah rangkaian *event-time*.

Dengan berkembangnya aplikasi IoT, jumlah objek dan data IoT akan berkembang cepat. IoT data *streaming* membawa *volume*, *velocity* dan *complexity* yang berbeda (Namiot, 2015). Peningkatan jumlah *device* dan aplikasi mengkombinasikan jangkauan dan kekuatan pada internet yang memungkinkan interaksi kecerdasan yang baru antar entitas. Data sensor yang terus-menerus dihasilkan, membutuhkan pengolahan data. Sehingga dibutuhkan arsitektur yang mampu menangani aplikasi IoT.

Arsitektur jaringan, komputasi, aplikasi dan manajemen data pada IoT memerlukan komunikasi dan model pemrosesan yang berbeda. Saat ini, belum ada standar dalam memahami atau mendeskripsikan model tersebut untuk IoT. Cisco (2014) memaparkan IoT *reference model* yang berfungsi menyediakan definisi dan deskripsi yang jelas yang dapat diaplikasikan secara akurat untuk elemen dan fungsi aplikasi dalam sistem IoT. IoT *reference model* terdiri dari 7 level yaitu *physical devices* and *controllers*, *connectivity*, *edge computing*, *data accumulation*, *data abstraction*, *application*, *collaboration* and *processes*. Model ini mendeskripsikan garis besar untuk memahami setiap kebutuhan dan potensial dari IoT. Tetapi tidak membatasi cakupan atau lokalitas komponen itu sendiri pada setiap level, tergantung pada situasi. (Cisco, 2014).



Gambar 2.1 IoT Reference Model

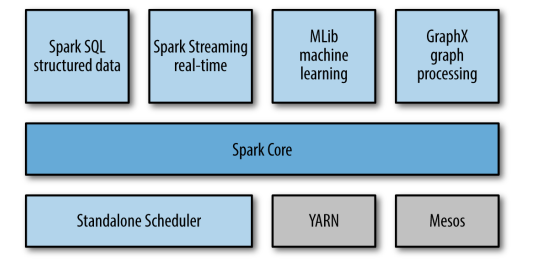
Sumber: Cisco (2016)

Dalam penggunaan aplikasi IoT, tidak terlepas dengan penggalian *value* yang dihasilkan dari data *streaming* IoT. Pada layer 6, proses analisis dibutuhkan untuk memperoleh *value* yang diinginkan. Dibutuhkan *engine* yang mampu menangani data *streaming* IoT dengan menyediakan aplikasi dengan *throughput* yang tinggi dan *latency* yang rendah. Kemampuan skalabilitas dan *fault-tolerance* juga diperlukan untuk menangani *workload* dan kesalahan secara efisien (de Assuncao, et. al., 2017). Terdapat beberapa engine yang dapat digunakan untuk pengolahan data *streaming* antara lain Apache Spark, Apache Storm, Apache Flink, Apache Samza, Apache Apex dan Google Cloud Dataflow (Carvalho et al, 2018).

## Apache Spark

Apache Spark merupakan sebuah sistem komputasi terdistribusi yang cepat dan *general-purpose*. Pada sisi kecepatan, Spark mengembangkan model MapReduce untuk efisiensi mendukung beberapa jenis komputasi, termasuk interaktif *query* dan pengolahan *stream*. Salah satu fitur utama Spark dalam hal kecepatan adalah kemampuan untuk menjalankan komputasi di memori, tetapi sistem lebih efisien dibandingkan MapReduce untuk komputasi kompleks yang berjalan di disk.

Di sisi umum, Spark didesain untuk mencakup sebuah *workload* dengan ukuran luas yang sebelumnya memerlukan sistem terdistribusi, termasuk aplikasi *batch*, algoritma iteratif, interaktif *query* dan *streaming*. Dengan mendukung *workload* seperti di *engine* yang sama, Spark membuatnya lebih mudah dan terjangkau karena mengkombinasikan jenis pengolahan yang berbeda, yang sering dibutuhkan pada data analisis. Apache Spark didesain mempunyai akses yang tinggi, dengan menyediakan API untuk Python, Java, Scala dan SQL dan juga library built-in yang beragam.



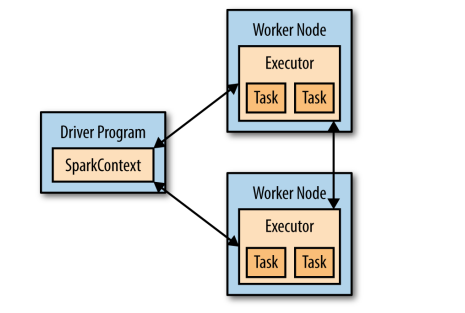
Gambar 2.2 Komponen Apache Spark

Sumber: Karau et al (2015)

Projek Spark terdiri dari beberapa komponen yang terintegrasi. Pada core, Spark merupakan sebuah engine komputasi yang bertanggung jawab untuk *scheduling*, *distributing* dan *monitoring* aplikasi terdiri dari banyak *task* komputasi yang dijalankan di banyak mesin worker atau sebuah cluster komputasi. Spark core terdiri dari fungsionalitas dasar pada Spark, termasuk komponen untuk *task scheduling*, manajemen memor, *fault recovery*, interaksi dengan sistem penyimpanan dan lainnya. (Karau et al, 2015). Spark memperkenalkan abstraksi yang disebut *Resilient Distributed Datasets* (RDD) yang dapat menjalankan komputasi pada memori. RDD bersifat immutable dan merupakan kumpulan partisi record, yang menyediakan sebuah pemrograman *interface* untuk komputasi seperti *map*, *filter* dan *join* (Assuncao, 2017).

Spark didesain secara efisien untuk *scale up* dari satu sampai ribuan node untuk komputasi. Untuk memaksimalkan fleksibilitas, Spark dapat berjalan diatas *cluster manager* seperti Hadoop YARN, Apache Mesos dan Spark itu sendiri yang disebut *Standalone Scheduler* (Karau, 2015). Berikut adalah komponen yang berada pada Spark cluster:

* *Driver* merupakan program client yang mendefinisikan SparkContext. SparkContext menggambarkan sebuah koneksi ke sebuah cluster komputasi (Karau, 2015). Driver menghubungkan antara cluster manager dan permintaan resource yang digunakan untuk mengeksekusi job pada distribusi komputasi.
* *Cluster manager/Spark master* merupakan komponen spark yang mengatur dan mengalokasikan sistem *resource* untuk *job* Spark. Selanjutnya, master mengkoordinasi dan mengetahui status hidup dari node pada cluster. Hal tersebut memungkinkan untuk mengeksekusi job yang dilakuka worker node dan menampilkan status berbagai job yang berjalan pada worker node.
* *Spark worker* merupakan komponen yang mengeksekusi komputasi yang diajukan oleh driver (Gupta, 2015). Untuk menjalankan berbagai operasi, driver mengatur sebuah jumlah node yang disebut executor.



Gambar 2.3 Komponen untuk sitribusi executor di Spark.

Sumber: Karau et al (2015)

Setiap aplikasi Spark terdiri dari program driver yang memulai berbagai operasi paralel pada cluster. Program driver terdiri dari main function dan mendefinisikan distribusi dataset pada cluster kemudian mengaplikasikan operasi ke dataset. Program driver mengakses Spark melalui objek SparkContext yang mempresentasikan koneksi ke cluster. Jika terdapat objek SparkContext maka digunakan untuk membuat RDD. Untuk menjalankan operasi, program driver secara khusus mengatur jumlah node yang disebut executor (Karau, 2015).

### Pemrograman dengan RDD

Abstraksi pada Spark yang bekerja pada data adalah RDD. Sebuah RDD merupakan sebuah koleksi elemen yang terdistribusi. Setiap RDD dibagi menjadi beberapa partisi, yang bisa jadi di komputasi di node yang berbeda pada cluster. Pada Spark semua perkerjaan dinyatakan dengan membuat RDD baru, transformasi RDD yang baru atau memanggil operasi pada RDD untuk komputasi sebuah hasil. Spark secara otomatis mendistribusikan data yang berada pada RDD antar cluster dan operasi dijalankan secara paralel.

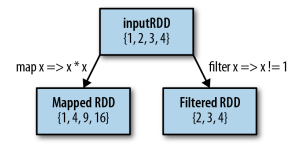
Pengguna membuat RDD dengan 2 cara yaitu berasal dari sistem eksternal atau dengan mendistribusikan sebuah koleksi objek pada program driver. Setelah RDD dibuat, RDD menyediakan 2 tipe operasi yaitu *transformations* dan *actions*. Operasi transformations menciptakan sebuah RDD baru dari sebelumnya. Sedangkan operasi actions mengkomputasi sebuah hasil berdasarkan sebuah RDD dan mengembalikannya ke program driver atau menyimpannya ke sebuah sistem penyimpanan eksternal. Operasi transformations dan actions berbeda karena ada perbedaan cara Spark mengkomputasi RDD. Spark melakukan komputasi bersifat lazy. Karena Spark tidak langsung mengkomputasi setelah RDD dibuat, tetapi Spark melakukan komputasi jika menggunakan operasi actions. Sehingga lebih efisien dan tidak membuang resource yang ada.

Setiap program Spark bekerja seperti berikut:

* Membuat beberapa inputan RDD dari data eksternal atau mendistribusikan sebuah koleksi objek pada program driver.
* Melakukan transformasi untuk mendefinisikan RDD baru menggunakan operasi transformations.
* Memeriksa apakah RDD memerlukan persist().
* Memberikan operasi actions untuk memulai komputasi paralel yang dioptimasi dan dieksekusi oleh Spark.

#### Transformations

Transformations adalah operasi pada RDD yang mengembalikan sebuah RDD baru. Tranformations bersifat “lazy”, karena operasi tersebut akan dieksekusi jika terdapat operasi action. Spark menjaga track operasi yang dijalankan RDD yang disebut dengan lineage graph. Itu digunakan sebagai informasi untuk mengkomputasi setiap RDD yang diinginkan dan recover jika terjadi hilangnya data.



Gambar 2.4 Function map dan filter dari sebuah input RDD

Sumber: Karau et al (2015)

Ada 2 transformations yag paling umum dipakai yaitu map() dan filter(). Transformations map() mengambil function dan menerapkan ke setiap elemen pada RDD dengan menghasilkan nilai baru pada setiap elemen RDD. Transformations filter() mengambil function dan mengembalikan sebuah RDD yang hanya memenuhi function filter().

**Tabel 2.1 RDD transformations pada sebuah RDD.**

|  |  |  |
| --- | --- | --- |
| **No.** | **Nama Function** | **Penjelasan** |
| 1. | map( ) | Menerapkan sebuah fungsi untuk setiap elemen pada RDD dan mengembalikan sebuah RDD baru sebagai hasil. |
| 2. | flatMap( ) | Menerapkan sebuah fungsi untuk setiap elemen pada RDD dan mengembalikan sebuah RDD dengan memaparkan konten RDD. Biasanya digunakan untuk mengekstrak kata. |
| 3. | filter( ) | Mengembalikan sebuah RDD yang terdiri dari elemen yang memnuhi kondisi pada function filter() |
| 4. | distinct( ) | Menghapus duplikat. |
| 5. | reduceByKey(func) | Mengkombinasikan nilai dengan key yang sama. |
| 6. | groupByKey() | Mengelompokkan nilai dengan key yang sama. |
| 7. | mapValues(func) | Mengaplikasikan sebuah fungsi ke setiap nilai dari sebuah pair RDD tanpa merubah key. |
| 8. | flatMapValues(func) | Mengaplikasikan sebuah fungsi yang mengembalikan sebuah iterator dari setiap nilai pada pair RDD dan setiap elemen yang dikembalikan, menghasilkan sebuah key/nilai dengan key yang lama. |
| 9. | sortByKey() | Mengembalikkan sebuah RDD yang sudah diurutkan berdasarkan key. |

#### Actions

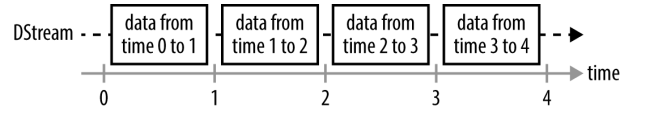
Actions merupakan tipe operasi kedua RDD. Operasi ini mengembalikan sebuah nilai final ke program driver atau menyimpan data ke sistem penyimpanan eksternal. Actions melakukan evaluasi pada transformations RDD, kemudian menghasilkan output (Karau, 2015).

**Tabel 2.2 RDD actions pada sebuah RDD.**

|  |  |  |
| --- | --- | --- |
| **No.** | **Nama Function** | **Penjelasan** |
| 1. | collect() | Mengembalikan semua elemen dari RDD. |
| 2. | count() | Menghitung jumlah elemen pada RDD. |
| 3. | countByValue() | Menghitung jumlah masing-masing elemen pada RDD. |
| 4. | take(num) | Mengembalikan elemen num dari RDD. |
| 5. | top(num) | Mengembalikan nilai top pada elemen num dari RDD. |
| 6. | reduce(func) | Mengombinasikan elemen pada RDD secara bersamaan dan paralel. |
| 7. | foreach(func) | Mengaplikasikan function ke setiap elemen pada RDD. |
| 8. | countByKey() | Menghitung jumlah elemen setiap key. |

### Spark Streaming

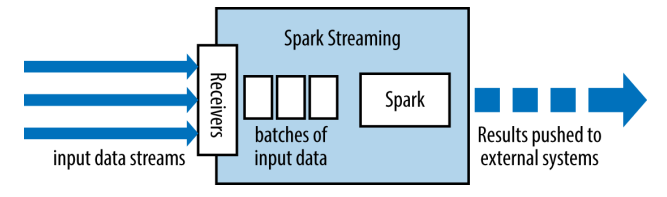
Terdapat banyak aplikasi yang memerlukan aksi bersamaan dengan data tersebut tiba. Spark Streaming merupakan modul Spark yang digunakan untuk aplikasi semacam itu. Seperti Spark yang menggunakan konsep RDD, Spark Streaming menyediakan abstraksi yang disebut DStream atau discretized streams. Sebuah DStream merupakan rangkaian data yang datang dari waktu ke waktu. DStream dapat dibuat dari berbagai macam sumber seperti Flume, Kafka atau HDFS. Setelah terbentuk, ada 2 jenis operasi pada DStream yaitu transformation dan output operations, yang terhubung dengan sistem eksternal. DStream menyediakan banyak operasi yang sama pada RDD, ditambah operasi baru yang berkaitan dengan waktu seperti sliding windows.



Gambar 2.5 DStream merupakan rangkaian dari RDD

Sumber: Karau (2015)

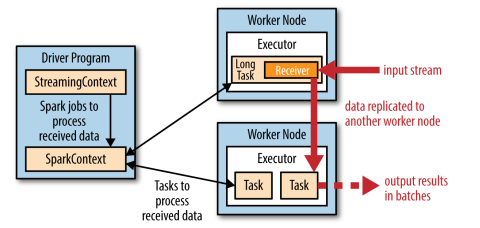
Spark Streaming menggunakan arsitektur micro-batch, dimana komputasi streaming adalah komputasi batch yang kontinyu pada ukuran batch yang kecil. Spark Streaming menerima data dari berbagai macam sumber input dan mengelompokkan ke dalam ukuran batch yang kecil. Batch yng baru dibuat pada setiap interval waktu. Pada saat batch baru dibuat dan data yang datang pada interval waktu itu akan dimasukkan kedalam batch. Ukurang pada waktu interval ditentukan dengan parameter batch interval. Batch interval diantara 500 ms dan beberapa detik, tergantung konfigurasi developer aplikasi. Setiap input batch membentuk sebuah RDD dan diproses menggunakan job Spark untuk membuat RDD lainnya.



Gambar 2.6 High-level architecture pada Spark Streaming

Sumber: Karau (2015)

Abstraksi pada Spark Streaming adalah DStream, yang merupakan rangkaian RDD, dimana setiap RDD mempunyai potongan waktu pada data di stream. DStream dapat dibuat dari berbagai maam sumber seperti Apache Flume, Apache Kafka, HDFS dll. Setelah DStream dibuat, terdapat 2 macam operasi yaitu transformasi dan output. DStream menyediakan banyak operasi yang sama seperti RDD, tetapi ada penambahan operasi yang berelasi dengan waktu.



Gambar 2.7 Eksekusi Spark Streaming di dalam komponen Spark

Sumber: Karau (2015)

Setiap sumber input, Spark Streaming meluncurkan receivers dimana task yang berjalan dalam aplikasi executor yang mengumpulkan data dari sumber input dan menyimpannya sebagai RDD. Inputan data diteria dan kemudian direplikasi ke executor lainnya untuk fault-tolerance. Data in akan disimpan di memori executor. StreamingContext pada program driver secara periosik menjalankan Spark job untuk memproses data ini dan mengkombinasikan dengan RDD sebelumnya. Selama input data tersedia, dapat dikomputasi ulang berdasarkan lineage graph jika terjadi kegagalan pada node.

#### Transformations

Transformation pada DStream dapat dikelompokkan menjadi stateless atau stateful. Pada stateless transformations pengolahan setiap batch tidak tergantung dengan data pada batch sebelumnya. Seperti map(), filter() dan reduceByKey(). Sedangkan stateful transformations menggunakan data atau hasil dari batch sebelumna untuk melakukan komputasi hasil pada batch saat ini. Tarnsformastions dilakukan berdasarkan sliding windows dan pada keadaan track saat itu.

**Tabel 2.3 Stateless DStream Transformations**

|  |  |  |
| --- | --- | --- |
| **No.** | **Nama Function** | **Penjelasan** |
| 1. | map() | Mengaplikasikan sebuah function pada setiap elemen di DStream dan mengembalikan sebuah DStream sebagai hasil. |
| 2. | flatMap() | Mengaplikasikan sebuah fungsi untuk setiap elemen pada DSTream dan mengembalikan sebuah DStream pada konten iterator yang dikembalikan. |
| 3. | filter() | Mengembalikan sebuah DStream yang terdiri dari hanya elemen yang masuk dalam kondisi saat filter. |
| 4. | reduceByKey() | Mengkombinasikan nilai dengan key yang sama setiap batch. |
| 5. | groupByKey() | Mengelompokkan nilai dengan key yang sama setiap batch. |

Pada stateful transformation, beberapa data dari batch sebelumnya digunakan untuk menghasilkan nilai hasil untuk sebuah batch baru. Ada 2 tipe utama yaitu operasi windowed yang digunakan untuk sliding window pada periode waku dan updateStateByKey() yang digunakan untuk mengetahui state setiap key.

#### Output Operations

Operasi output memspesifikasikan yang dibutuhkan untuk menyelesaikan trasformasi akhir pada sebuah stream. Output debugging yang paling umum adalah menggunakan print(). Function ini akan mengambil 10 elemen pertama setiap batch pada DStream dan menampilkan hasilnya. Kemudian foreachRDD() merupakan operasi output yang membiarkan kita menjalankan komputasi arbitrary pada DStream (Karau et al, 2015).

#### Sumber Input

Spark Streaming mendukung sumber data yang berbeda. Sumber data dapat berasal dari Twitter, Apache Flume, Apache Kafka dan berbagai sumber data lainnya. Pada penelitian ini menggunakan sumber data yang berasal dari Apache Kafka. Apache Kafka menjadi sumber data populer karena kecepatan dan ketahanannya. Untuk menggunakannya dibutuhkan Maven artifact kedalam projek yang dikerjakan. Objek KafkaUtils yang bekerja dengan StreamingContext untuk membuat sebuah DStream dari Kafka message (Karau et al, 2015).

## Apache Kafka

*Streaming platform* memungkinkan melakukan *publish* dan *subscribe* untuk stream. Dalam hal ini sama seperti *messaging system*. Apache Kafka dapat digunakan dalam membangun *streaming data pipeline* yang reliabel antara sistem atau aplikasi. Apache Kafka dapat dijalankan sebagai cluster pada satu atau lebih server. Komunikasi antara client dan server menggunakan protokol TCP. Dalam penyimpanan stream, setiap *record* ditampung di sebuah kategori yang disebut dengan topic (Apache Kafka, 2019).

*Client* Kafka merupakan pengguna pada sistem, terdapat 2 tipe dasar, *producer* dan *consumer*. *Producer* mengirim sebuah *message*. Seperti sistem *publish/subscribe*, *producer* dapat dikatakan sebagai *publisher*. Sebuah *message* akan diproduksi untuk sebuah topik yang spesifik. Secara *default*, *producer* tidak memerdulikan partisi yang akan diisi *message* dan *message* akan seimbang di semua partisi.

*Consumer* untuk membaca *message*. *Consumer* sama halnya dengan subscriber yang akan men-*subscribe* satu atau beberapa topik dan kemudian membacanya secara berurutan berdasarkan saat data diproduksi. *Consumer* akan menjaga *track* dari *offset message*. *Offset* merupakan bit metadata, sebuah nilai integer yang terus-menerus meningkat. Setiap message yang diproduksi, Kafka akan menambahkan *offset* (Narkhede et al, 2017).

**Kafka-Cassandra connector**

Berdasarkan platform yang akan dibangun, *raw data* akan disimpan ke Cassandra melalui Kafka. Bahasa pemrograman yang digunakan adalah Python sehingga dibutuhkan *connector* Python untuk melakukan *query* pada Cassandra. Cassandra node sebagai *consumer* pada sistem Kafka. *Connector* yang dipakai berasal dari DataStax (DataStax, 2019).

## Apache Cassandra

Apache Cassandra merupakan database NoSQL terdistribusi yang mengelola jumlah data yang besar dengan menyediakan layanan high available dan *no single point of failure*.

Beberapa poin dari Apache Cassandra sebagai berikut :

*Scalable, fault-tolerant* dan konsisten.

*Column-oriented database*.

Desain distribusi berdasarkan DynamoDB dari Amazon dan data model berdasarkan Bigtable dari Google(Datastax Academy, 2018).

Pada dasarnya, struktur data Cassandra dibagi menjadi *column* dan *column family*. Pada *column* berisi *name/value pair* dan *column family* berisi kontainer dari *row* yang terlihat sama tetapi tidak identik. Kontainer pada data di Cassandra disebut dengan *keyspace* yang mirip dengan nama *database* pada database relational (Hewitt, 2011).

Berdasarkan paltform yang dibangun, hasil pengolahan data streaming disimpan dalam database Cassandra. Untuk menghubungkannya, memerlukan Spark Cassandra *connector* dari DataStax yang dikembangkan oleh Julian Anguenot. Karena konektor tersebut bukan dari bagian Spark, sehingga harus menambahkan *dependencies* untuk membangun aplikasi Spark. Dalam implementasinya, data terus dihasilkan sehingga menggunakan *function* foreach() untuk menyimpan setiap record (Karau, 2015).

## MQTT

MQTT merupakan protokol konektivitas *machine-to-machine* (M2M), *server-to-server* (S2S), *machine-to-server* (M2S) dan mekanisme *routing* (*one-to-many, one-to-one, many-to-many*). MQTT didesain sebagai protokol *publish/subcribe* yang sangat ringan. Secara default, port MQTT bekerja pada port TCP/IP 1883. MQTT merupakan protokol M2M dan IoT yang digunakan untuk *publish/subscribe* *messaging* menyediakan implementasi yang sederhana dan transisi yang fleksibel (Yassein et al, 2017). Sistem MQTT terdiri dari komunikasi client ke sebuah server yang sering disebut broker.

### Broker

Broker merupakan tempat dimana topik dan data sementara ditampung. Salah satu broker MQTT adalah Eclipse Mosquitto. Mosquitto cocok digunakan untuk layanan IoT karena ringan (Mosquitto, 2019). Untuk memproteksi data yang dikirim sesuai, dapat membuat autentikasi berisi *username* dan *password*.

### Client

Client merupakan bagian dari MQTT sistem yang membuat/mengirimkan data dan menerima data. Client terdiri dari 2 tipe yaitu *publisher* dan *subscriber*. Salah satu *library* yang digunakan MQTT adalah Paho. Paho merupakan *client* MQTT menggunakan Python. *Class client* mengizinkan aplikasi berkoneksi dengan broker MQTT untuk *publish* data dan *subscribe* ke topik serta menerima data (Pypi, 2018).

# METODOLOGI PENELITIAN

Pada bab 1 dijelaskan tentang latar belakang penelitian, rumusan masalah dan batasan penelitian. Pada bab 2 dijelaskan tentang landasan kepustakaan serta dasar teori yang berkaitan dengan masalah penelitian. Setelah penjabaran pada bab sebelumnya, maka diperlukan dalam merumuskan metodologi penelitian. Pada bab 3 dijabarkan langkah dalam menyelesaikan masalah penelitian secara sistematik dan spesifik pada pengembangan platform untuk pengolahan data sensor IoT berjenis streaming. Penjabaran tersebut digambarkan dengan diagram alur yang dijelaskan pada Diagram 3.1.

Selesai

Mulai

Studi dan analisis literatur

Analisis kebutuhan sistem

Implementasi

Desain alur komunikasi

Desain struktur data streaming

Pengujian dan analisis

Kesimpulan dan saran

Perancangan pengolahan data pada Spark Streaming

Perancangan tiap komponen

**Diagram 3.1 Diagram Alur Metode Penelitian**

## Tipe Penelitian

Penelitian yang dilakukan termasuk kedalam penelitian implementatif pengembangan lanjut, karena penelitian ini mengembangkan platform dalam pengolahan data sensor IoT berjenis streaming dengan komputasi terdistribusi menggunakan Spark Streaming. Pada penelitian ini menghasilkan sebuah *data pipeline* dalam pengolahan data *streaming* IoT.

Dalam menyelesaikan masalah yang terkait pengolahan data sensor IoT berjenis *streaming*, Spark Streaming menjadi inti dari pengembangan platform. Pengolahan data *streaming* tidak hanya diperlukan data processor, tetapi membutuhkan komponen-komponen lain dalam menunjang tujuan dari penelitian ini.

## Studi Literatur

Studi literatur digunakan untuk menemukan solusi dan penyelesaian dalam pemecahan masalah pada penelitian yang dilakukan. Literatur yang digunakan merupakan penelitian tentang penyelesaian pengolahan data streaming pada IoT. Tujuan dari studi literatur agar memperkuat dasar penelitian yang akan dilakukan dan menjadi acuan dalam penyelesaian masalah penelitian.

Kajian penelitian yang pertama adalah menelaah data *streaming* pada IoT dan pengolahannya. Kajian ini juga berfungsi untuk menelaah dasar teori yang mendukung pengembangan platform pengolahan data *streaming* menggunakan Spark Streaming. Dalam pengolahan data *streaming* harus memperhatikan *data pipeline* yang digunakan sehingga dapat mencapai tujuan penelitian. Dalam membangun platform pengolahan data *streaming*, dibutuhkan komponen-komponen merujuk pada pengantar bab 2.

## Batasan Perancangan dan Implementasi

Data yang dikirimkan oleh publisher menggunakan data dummy yang dihasilkan dari Python.

Dalam menguji sistem menggunakan satu aliran data dan mengindahkan *packet loss*.

Spark Streaming dapat bekerja jika data pada topik Kafka sudah tersedia secara real-time.

Sistem penampil data pada website akan bekerja jika Spark Streaming berhasil mengirimkan hasil ke topik pada broker Kafka.

## Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk menentukan semua kebutuhan sebelum melakukan penelitian. Dalam tahap ini dilakukan pemetaan bagian-bagian dasar dari suatu penelitian sebagai pedoman dan memperkuat dasar teori yang telah disampaikan. Sesuai dengan subbab 2.3 tentang Spark Streaming sebagai solusi dalam mengolah data sensor IoT berjenis *streaming* dengan komputasi terdistribusi. Spark Streaming merupakan API *streaming* dari Apache Spark. Dengan optimasi yang dimiliki, Spark Streaming menjadi solusi dalam pengolahan data *streaming*. Berdasarkan pengantar bab 2, dalam mengembangkan platform terdiri dari 4 bagian yaitu *data integration*, *data processor*, *data storage* dan visulisasi.

Fokus penelitian ini adalah pengembangan platform pengolahan data streaming secara terdistribusi menggunakan Spark Streaming. Dalam pengembangan platform, Spark Streaming dibantu dengan beberapa teknologi untuk mencapai tujuan penelitian. Dalam menguji sistem, dibutuhkan data *streaming* untuk mengetahui sistem berjalan sesuai dengan perancangan. Karena penelitian ini tidak melibatkan *hardware layer* (konektivitas sensor), maka data *streaming* yang dipakai berasal dari *data dummy* yang dihasilkan dari *script* Pyhton. Data *streaming* tersebut menggambarkan data yang dihasilkan dari sensor. Data sensor dapat direpresentasikan dengan berbagai format tergantung dengan kebutuhan. Format yang digunakan dalam penelitian ini adalah JSON. Pemilihan JSON sebagai format *data-interchange* dikarenakan ringan dan mudah dibaca.

Data *streaming* dikirim menggunakan protokol MQTT. MQTT merupakan protokol komunikasi yang digunakan untuk mengirimkan data *machine to machine* yang ringan dan mudah serta memainkan peran yang penting untuk IoT. Konsep dasar MQTT adalah *publish*/*subscribe*. Pusat dari komunikasi MQTT adalah broker. Eclipse Mosquitto sebagai broker MQTT berguna untuk menampung sementara data sensor dalam sebuah topik. Client yang mengirimkan data ke broker disebut dengan publisher, sedangkan client yang menerima data dari broker disebut dengan subscriber. Client MQTT yang digunakan dalam penelitian ini adalah Eclipse Paho.

Sebelum data diolah, data sensor diintegrasikan ke sebuah m*essaging system*. *Messaging system* memudahkan pendistribusian dan manajemen data pada sistem yang berjalan. *Messaging system* mampu menampung sementara data sensor sebelum data tersebut masuk ke proses selanjutnya. *Messaging system* yang digunakan adalah Apache Kafka. Dibandingkan *messaging system* lainnya, Kafka mempunyai *throughput* yang lebih baik dan *fault-tolerant* yang membuatnya menjadikan sebuah solusi untuk aplikasi pengolahan data dengan skala yang besar. Meskipun Apache Kafka mempunyai konsep *publish/subscribe*, tetapi hal tersebut tidak bisa digantikan oleh MQTT. Karena Apache Kafka tidak dapat berkerja *machine-to-machine*. Dengan adanya Apache Kafka, data siap diolah sesuai dengan kebutuhan aplikasi.

Data sensor pada Apache Kafka, siap diolah melalui Apache Spark untuk pengolahan streaming secara terdistribusi. Spark Streaming merupakan API Apache Spark dalam menangani pengolahan data streaming. Sebelum dilakukan pengolahan dibutuhkan konektor yang mampu mengambil data dari Apache Kafka. Spark Streaming sudah menyediakan konektornya, sehingga aplikasi dapat dengan mudah dalam pengambilan data. Setelah data didapatkan, Spark Streaming dapat menggali nilai dari data streaming. Sehingga mampu menyediakan kebutuhan aplikasi IoT.

Kebutuhan data historik dalam dunia IoT juga dibutuhkan. Data historik dapat menyediakan informasi secara keseluruhan dan dapat digunakan dalam pengolahan *batch* jika diperlukan, sehingga dibutuhkan penyimpanan data. Penyimpanan data yang digunakan dalam penelitian ini adalah Apache Cassandra. Apache Cassandra dipilih karena mempunyai *avaibility* yang tinggi sehingga jika salah satu node database bermasalah, tidak mempengaruhi node lainnya. Selain itu, Apache Cassandra mempunyai skalabilitas yang baik dan latency yang rendah dengan kinerja yang tinggi. Agar lebih efisien, *raw data* yang berasal dari Apache Kafka disimpan kedalam Apache Cassandra. Dibandingkan disimpan dari Spark Streaming. Penyimpanan dari Apache Kafka ke Apache Cassandra belum tersedia konektor. Dengan cara lain, implementasi dapat menggunakan Datastax cassandra-python driver. Karena implementasi menggunakan bahasa pemrograman Python. Selain itu hasil pengolahan dari Spark Streaming juga harus disimpan untuk keperluan analisis kedepannya. Untuk menyimpan data dari Spark Streaming ke Cassandra dibutuhkan konektor yang dikembangkan oleh Anguenot yang berasal dari konektor DataStax.

Pengguna membutuhkan aplikasi yang dapat melakukan monitoring hasil pengolahan data sensor secara streaming. Peneliti menggunakan aplikasi website sebagai media visualisasi. Peneliti menggunakan Node JS, socket IO dan D3 dalam pengembangan aplikasi website.

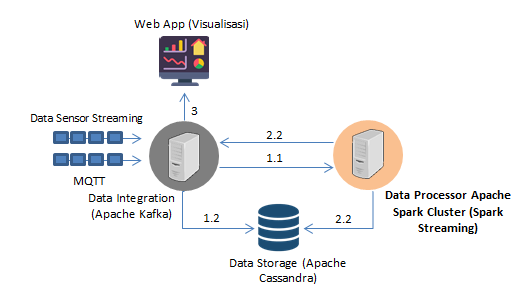
Untuk melakukan monitor resource pada spark worker, dibutuhkan Ganglia Monitoring System. Ganglia digunakan untuk melihat paramater pengujian berupa CPU, memori dan *disk*. Ganglia merupakan sistem monitor yang mempunyai skalabilitas dan terdistribusi untuk kinerja sistem komputasi yang tinggi seperti pada cluster. Selanjutnya kebutuhan penelitian terangkum pada tabel 3.1 berikut.

**Table 3.1 Tabel Kebutuhan Sistem**

|  |  |
| --- | --- |
| Proses/Kebutuhan | Perangkat Lunak |
| Format Data | JSON |
| *Data Integration/Messaging System* | Apache Kafka |
| Broker MQTT | Eclipse Mosquitto |
| Client MQTT | Eclipse Paho |
| Kafka-Cassandra Connector | DataStax Python Cassandra Driver |
| *Data Processor* | Apache Spark (Spark Streaming) |
| Structured Streaming-Kafka Connector | Spark Streaming Kafka Integration |
| Spark-Cassandra Connector | Anguenot Spark-Cassandra Connector |
| *Data Storage* | Apache Cassandra |
| Visualisasi | Node.js, Socket IO, D3 |
| Sistem monitor cluster | Ganglia Monitoring System |

## Perancangan Sistem

Perancangan sistem dilakukan untuk menggambarkan alur kerja sistem secara keseluruhan dan Spark Streaming sebagai inti dalam pengolahan data sensor. Perancangan ini digunakan dalam penelitian berdasarkan analisis kebutuhan yang sudah disampaikan. Dalam pengolahan data sensor IoT berjenis streaming, memerlukan komponen lain untuk menunjangkan penelitian. Secara garis besar, sistem ini terbagi ke dalam 4 komponen yaitu data integration, data processor, data storage dan visualisasi.



**Gambar 3.1 Rancangan Umum Sistem**

### Perancangan Integrasi Data

Integrasi data dibutuhkan untuk mengumpulkan data sensor streaming IoT. Penelitian ini tidak melibatkan *hardware layer*, sehingga menggunakan *data dummy* untuk melakukan pengujian pada sistem. Data sensor IoT dibuat menggunakan script python dengan format JSON. Data tersebut berjenis streaming sehingga dapat mendukung dalam pengujian sistem. Untuk mendekati implementasi yang sudah ada, MQTT digunakan sebagai *gateway* IoT. Data streaming dikirimkan dan diterima oleh MQTT *client* menggunakan Eclipse Paho. Data dikirim ke sebuah topik yang spesifik pada broker. Broker MQTT yang digunakan adalah Eclipse Mosquitto. Pada broker MQTT, ditambahkan autentikasi sehingga tidak semua *node* bisa mengirimkan datanya ke broker. Autentikasinya berupa pembuatan username dan password untuk broker MQTT.

Sebelum data diolah secara terdistribusi, data diintegrasikan kedalam sebuah *messaging system*. Messaging system yang dipakai adalah Apache Kafka. Kafka berfokus pada pengumpulan data sebelum diolah oleh data processor dan mempermudah pengambilan data untuk visualisasi. Producer Kafka menjadi subscriber dari MQTT broker. Data diambil dari sebuah topik pada broker, kemudian dikirimkan ke sebuah topik Kafka. Data tersebut dapat digunakan untuk keperluan analisis di data processor dan disimpan ke dalam database. Untuk menyimpan raw data kedalam data storage, diperlukan driver Cassandra-Python *driver*. Consumer Kafka menerima data dari sebuah topik kemudian disimpan ke dalam Cassandra.

### Perancangan Pengolahan Data

Sentral dari penelitian ini adalah Apache Spark. Apache Spark sebagai *data processor* bertugas untuk mengolah data *streaming* dengan komputasi terdistribusi menggunakan Spark Streaming. Spark Streaming mengambil data yang siap diolah di topik Kafka menggunakan Spark Streaming-Kafka *connector*. Setelah mendapatkan data streaming, dilakukan analisis secara *near real time*. Dikatakan *near real time* karena pengolahan dilakukan dengan besaran detik. Karena fokus penelitian ini pengembangan platform pengolahan data streaming bukan algoritma pengolahan data, sehingga analisis datanya berupa kalkulasi aritmatika nilai rata-rata, maksimum dan minimum.

Untuk mencapai tujuan skalabilitas tinggi dan fault-tolerant, Spark dibangun diatas cluster. Pada model distribusi, Spark menggunakan arsitektur master/slave dengan satu koordinator pusat dan banyak distribusi worker. Koordinator pusat disebut dengan *driver*. *Driver* berkomunikasi dengan sejumlah worker yang berkerja secara terdistribusi yang disebut dengan *executor*. Sebuah driver dan executor berkerja sama membentuk sebuah aplikasi Spark. Sebuah aplikasi Spark dijalankan pada sebuah kumpulan mesin menggunakan sebuah layanan eksternal yang disebut *cluster manager*. Pada penelitian ini menggunakan Standalone *cluster manager*.

### Perancangan Penyimpanan Data

Penyimpanan dilakukan untuk keperluan data histori yang bisa digunakan untuk analisis selanjutnya. Data storage yang digunakan adalah Cassandra. Cassandra dibangun secara cluster terdiri dari 2 *node*. Diperlukan perancangan data model untuk mengambarkan skema basis data dalam menyimpan raw data dan hasil kalkulasi dari Spark Streaming. Perancangan disesuaikan dengan sistem basis data Cassandra yang berorientasi kolom. Peneliti membutuhkan untuk mendefinisikan sebuah *keyspace*, yang terdiri dari *column family*. Tabel data tidak bersifat mengikat, sehingga dapat menambahkan data tanpa membuat kolom baru. Untuk memudahkan kebutuhan data historik, dibuat 4 tabel yang akan menampung raw data, nilai rata-rata, nilai maksimum dan nilai minimum. Untuk menyimpan hasil kalkulasi dari Spark Streaming ke Apache Cassandra dibutuhkan konektor pyspark-cassandra yang dikembangkan oleh Julian Anguenot.

### Perancangan Visualiasi

Perancangan visualiasasi menjelaskan rancangan aplikasi website untuk melihat hasil pengolahan secara real time. Aplikasi website dibangun menggunakan Node JS dengan dibantu Socket IO untuk membantu pengambilan data dari topik Kafka secara real time. Antarmuka pengguna akan menampilkan grafik hasil pengolahan secara real time menggunakan D3.

### Perancangan Pengujian

Setelah perancangan sistem sudah selesai, kemudian menentukan perancangan untuk pengujian. Pengujian difokuskan pada performa Spark Streaming dalam mengolah data streaming dengan komputasi terdistribusi. Artinya pengolahan data pada Spark Streaming dilakukan dalam sebuah cluster. Parameter yang menjadi acuan adalah *latency, throughput, fault tolerance* dan penggunaan CPU dan memori. *Latency* adalah waktu yang dibutuhkan Spark Streaming untuk mengolah data dan menghasilkan nilai. *Throughput* adalah jumlah data setiap detik yang dapat diolah oleh Spark Streaming. Fault tolerance didefinisikan dengan *throughput penalty factor* dan *latency penalty factor*. Perhitungan *penalty factor* dilihat dari *throughput* dan *latency* pada saat jumlah *node worker* sebelum gagal, saat satu *node worker* gagal dan saat jumlah node worker setelah gagal. *CPU, memory, disk utilization* adalah jumlah penggunaan CPU, memory dan disk pada *node worker* dalam mengolah *data streaming*. Skenario yang dilakukan untuk menguji masing-masing parameter adalah jumlah node worker dan jumlah data setiap detik yang akan diolah. Jumlah *node worker* yang akan dibangun adalah 1 node worker di mesin yang sama dengan node master, 1 node worker terpisah dengan node master, 4 node, dan 8 node. Skenario selanjutnya menggunakan jumlah data setiap detik yang akan diolah dengan jumlah 1000 data, 1500 data, 2000 data dan 2500 data. Skenario selanjutnya menggunakan batch interval 3 detik, 9 detik, 18 detik dan 27 detik. Setiap skenario akan menguji Spark Streaming dalam mengolah data dan dianalisis sesuai parameter pengujian.

Karena keterbatasan *resource* laptop peneliti, lingkungan pengujian dilakukan menggunakan cloud. Cloud yang digunakan adalah Amazon Web Service (AWS). Pada pengujian ini, layanan AWS yang dipakai adalah Elastic Compute Cloud (EC2). EC2 menyediakan berbagai macam jenis instance terdiri dari kombinasi kapasitas CPU, memory, penyimpanan dan jaringan yang memberikan pilihan untuk aplikasi yang akan dibangun. Penelitian ini menggunakan jenis instance t2.micro dengan spesifikasi vCPU 1 dan memory 1 GiB. Layanan EC2 ditempatkan di berbagai macam lokasi didunia. Region yang dipakai dalam pengujian adalah N. Virginia dengan kode us-east-1. Semua komponen sistem diinstal kedalam EC2 yang terdiri dari beberapa Availibility Zone yang tersedia pada Region N. Virginia. Setelah sistem selesai dibangun, dilakukan pengujian berdasarkan parameter dan skenario.

Pengambilan data pengujian melalui Spark Streaming UI. Pada Spark Streaming UI terdapat statistika streaming untuk menjalankan aplikasi Spark. Spark Streaming UI menyajikan beberapa grafik dan tabel yang menggambarkan pengolahan data setiap task dan job pada aplikasi Spark. Untuk mengambil data pengujian *throughput* menggunakan grafik *input rate*. Grafik tersebut menampilkan kecepatan data saat aplikasi Spark menerima data streaming dari sumber. Untuk mengambil data pengujian *latency* menggunakan grafik *processing time* dan *scheduling delay*. Hasil kedua grafik dijumlahan untuk memperoleh latency. Sedangkan untuk mengambil data pengujian penggunaan CPU, memori dan disk menggunakan Ganglia Monitoring System.

## Implementasi

Implementasi pengembangan platform pengolahan data sensor IoT berjenis *streaming* berdasarkan tahapan perancangan. Implementasi dilakukan pada virtual machine (VM). Pada komponen data integration, dilakukan instalasi MQTT node yang terdiri dari publisher dan broker. Masing-masing MQTT node diinstall dalam VM yang berbeda. Node publisher diinstall dengan Eclipse Paho, kemudian dilanjutkan instalasi Eclipse Mosquitto pada node broker. Pada MQTT broker dilakukan konfigurasi autentikasi MQTT client menggunakan username dan password sebelum sebuah koneksi diizinkan. Penggunaan autentikasi username dan password menyediakan cara yang mudah dalam membatasi akses ke broker. Metode yang digunakan untuk membuat autentikasi ini dengan menggunakan file text. Membuat file text berisi username dan password kemudian dilakukan enkripsi pada file password tersebut. Setalah dilakukan konfigurasi autentikasi pada broker, membuat sumber kode data dummy pada MQTT publisher menggunakan bahasa pemrograman Python.

Pada integrasi data, diperlukan messaging system sebagai perantara dan mengatur data dari pengirim ke penerima. Sehingga dilakukan instalasi Apache Kafka. Kafka berjalan di atas JVM, sehingga membutuhkan instalasi Java. Setelah Java berhasil di instal kemudian melakukan instalasi Kafka. Pada penelitian ini, Kafka dibangun di satu node. Node Kafka mempunyai peran sebagai subscriber MQTT. Sehingga diperlukan instalasi Eclipse Paho. Data yang diterima dari broker MQTT kemudian dikirimkan ke sebuah topik di broker Kafka. Producer Kafka berperan dalam pengiriman data tersebut. Untuk menggunakan producer Kafka dilakukan instalasi kafka-python *package*. Setelah semua instalasi selesai, membuat sumber kode yang dapat mengambil data dari topik di broker MQTT dan mengirimkan ke data tersebut ke sebuah topik di broker Kafka. Data yang belum diolah, disimpan ke dalam Cassandra untuk keperluan data histori. Penyimpanan data ke Cassandra membutuhkan driver kafka-python, karena implementasi *consumer* Kafka menggunakan bahasa pemrograman Python.

Setelah data siap diolah, dilakukan instalasi Apache Spark. Apache Spark sebagai sentral dari penelitian ini. Apache Spark berjalan di atas Java Virtual Machine (JVM) sehingga membutuhkan instalasi Java. Setelah instalasi Java selesai, melakukan instalasi Apache Spark. Apache Spark dibangun diatas cluster yang terdiri dari satu *node* *master* dan beberapa *node* *worker*. Hal tersebut berguna untuk melakukan komputasi data secara terdistribusi. Pengaturan resource dalam menjalankan aplikasi Spark menggunakan cluster manager Standalone. Instalasi Apache Spark memerlukan konfigurasi antar master dan worker, sehingga dalam satu cluster dapat mengolah data secara terdistribusi. Setelah instalasi Apache Spark selesai, dilakukan penulisan sumber kode untuk menerima data dari sebuah topik Kafka kemudian mengolahnya. Pengolahan data secara streaming menggunakan salah satu API Spark yaitu Spark Streaming. Pengambilan data dari topik Kafka memerlukan sebuah package. Spark Streaming menerima data dari Kafka menggunakan sebuah package Spark Streaming Kafka Integration yang sudah disediakan oleh Apache Spark. Dalam implementasinya, harus melihat spesifikasi yang diperlukan seperti versi Kafka dan bahasa pemrograman yang dipakai dalam menjalankan aplikasi Apache Spark. Setelah Spark Streaming menerima data dari Kafka, dilakukan kalkulasi nilai rata-rata, nilai minimum dan nilai maksimum setiap detik. Setelah mendapatkan hasil, dilakukan operasi output. Terdapat 2 operasi output yang dilakukan aplikasi Spark yaitu mengirimkan hasil ke masing-masing topik pada broker Kafka dan menyimpan hasil ke masing-masing tabel pada Cassandra. Pengiriman data ke topik broker Kafka memerlukan instalasi sebuah driver Kafka-Python. Sedangkan untuk menyimpan ke Cassandra dibutuhkan konektor pyspark-cassandra.

Pada penyimpanan data membutuhkan instalasi Cassandra. Cassandra dibangun dalam sebuah cluster. Cluster terdiri 2 node. Cassandra berjalan di JVM, sehingga membutuhkan instalasi Java. Setelah instalasi Java di masing-masing node selesai, kemudian dilakukan instalasi Cassandra. Konfigurasi di semua node harus sama dan sesuai. Sehingga antar node dapat terhubung. Jika instalasi Cassandra sudah benar, dilakukan pembuatan keyspace dan tabel. Tabel pada Cassandra bersifat fleksibel karena Cassandra mempunyai konsep schema-free.

Pada visualisasi data, pengguna memerlukan monitoring hasil pengolahan data sensor secara streaming. Pada sisi server menggunakan Node.JS. Node.JS mengambil data dari topik Kafka dengan bantuan Socket.IO. Socket.IO digunakan untuk mengambil data secara real time. Visualiasi dilakukan dalam bentuk grafik sehingga pengguna dapat melihat pola dari hasil pengolahan data. D3.js membantu visualisasi dengan menampilkan grafik hasil pengolahan secara real time.

## Pengujian dan Analisis

Pengujian dilakukan untuk mengetahui bahwa platform pengolahan data sensor berjenis streaming sudah dibangun sesuai dengan kebutuhan yang telah dijabarkan pada tahap analisis dan perancangan. Spark Streaming juga harus diuji untuk mengetahui kinerja Spark Streaming dalam mengolah data streaming dengan menggunakan komputasi terdistribusi. Ada 2 jenis penelitian yang akan dilakukan, antara lain :

### Pengujian Fungsional

Pengujian fungsional dilakukan untuk menguji keseluruhan komponen pada platform dapat berjalan. Pertama menguji apakah subscriber broker MQTT atau producer Kafka dapat menerima data streaming dari broker MQTT kemudian mengirimkan ke sebuah topik Kafka. Terdapat 2 consumer pada Kafka yaitu database Cassandra dan Spark Streaming. Setelah data berhasil dikirimkan ke sebuah topik Kafka, kemudian menguji apakah data tersebut dapat disimpan ke dalam Cassandra. Selain itu, menguji apakah Spark Streaming dapat menerima data dari sebuah topik Kafka, kemudian dilakukan kalkulasi untuk mendapatkan nilai rata-rata, nilai maksimum dan nilai minimum. Setelah itu, hasil kalkulasi dikirimkan ke 2 output yang berbeda yaitu Cassandra dan dikirimkan ke topik yang berbeda di Kafka. Pengujian pada Cassandra dilakukan dengan melihat apakah hasil pengolahan berhasil disimpan kedalam basis data. Selain itu, menguji apakah hasil kalkulasi sudah berhasil masuk ke topik yang berbeda di Kafka. Pengujian tersebut dapat dilihat dengan menggunakan perintah consumer Kafka dengan menampilkan isi pada masing-masing topik Kafka. Setelah itu, dilakukan pengujian bagian visualisasi. Pengujian dilakukan dengan melihat apakah aplikasi website dapat menampilkan hasil pengolahan secara real time berupa grafik.

### Pengujian Non-fungsional

Pengujian dilakukan berdasarkan parameter dan skenario yang sudah dijelaskan pada perancangan pengujian. Pengujian non-fungsional berguna untuk menguji kinerja Spark Streaming dalam pengolahan data secara terdistribusi. Pengujian dilakukan menggunakan beberapa parameter yaitu *latency*, *throughput*, *fault tolerance* dan penggunaan CPU dan memori. Masing-masing parameter diuji dengan beberapa skenario yaitu berdasarkan batch interval, jumlah worker dan jumlah inputan data. Pengujian dilakukan di lingkungan cloud dan menghindari instalasi komponen lain di node yang sama dengan Spark. Sehingga dapat menampilkan hasil yang optimal untuk mengetahui kinerja Spark. Pengujian dilakukan beberapa kali untuk menghasilkan nilai rata-rata setiap parameter. Parameter throughput dan latency dilihat pada Spark Streaming UI. Sedangkan parameter penggunaan CPU, memori dan disk dilihat pada sistem monitoring Ganglia. Hasil pengujian ditampilkan dalam bentuk grafik.

### Analisis Hasil Pengujian

Setelah dilakukan pengujian maka tahap selanjutnya akan dilakukan analisis sesuai dengan pengujian yang dilakukan. Analisis dilakukan untuk mengetahui fungsional dan kinerja Spark Streaming apakah sudah sesuai dengan tahap kebutuhan dan perancangan sebelumny.

## Kesimpulan

Berdasarkan hasil pengujian platform menggunakan Spark Streaming kemudian dilakukan pembahasan hasil. Pengambilan kesimpulan dilakukan setelah melalui tahapan perancangan, implementasi, pengujian dan analisis. Kesimpulan penelitian menjawab rumusan masalah pada bab 1, yang juga menjadi dasar dilakukan penelitian serta hasil pengujian dan analisis sistem yang dibangun. Tahap akhir dari penulisan adalah saran yang digunakan untuk memperbaiki kesalahan-kesalahan yang ada dan menyempurnakan penulisan serta pengembangan untuk penelitian selanjutnya.

# PERANCANGAN SISTEM

Perancangan sistem dilakukan untuk menggambarkan alur kerja sistem secara keseluruhan. Perancangan meliputi perancangan komponen integrasi data, perancangan komponen pengolahan data, perancangan komponen penyimpanan data, perancangan visualisasi dan perancangan pengujian.

### Perancangan Komponen Integrasi Data

Data integration dibutuhkan untuk mengumpulkan data sensor streaming IoT. Dalam praktiknya, dibutuhkan protokol komunikasi untuk menghubungkan mesin ke mesin atau IoT gateway. Protokol komunikasi yang dipakai adalah MQTT. Data streaming dikirimkan oleh publisher MQTT menggunakan Eclipse Paho. Data streaming mempunyai format JSON. Karena penelitian tidak melibatkan hardware layer, memakai data dummy yang dihasilkan dari script Python. Kisaran nilai suhu, kelembapan dan pH berdasarkan karakteristik lahan pada tanaman jagung (Kementrian Pertanian, 2011).

Struktur JSON yang digunakan sebagai berikut:

datasensor = {

"id": id,

"nama\_sensor": nama\_sensor,

"event\_time": event\_time,

"nilai\_suhu": nilai\_suhu,

"nilai\_kelembapan": nilai\_kelembapan,

"nilai\_ph": nilai\_ph

}

**Gambar 4.1 Skema JSON**

Penjelasan dari Gambar 4.1 adalah:

‘id’ merupakan identitas objek yang bersifat unik. Key ‘id’ menggunakan universally unique identifier (UUID) versi 4.

‘nama\_sensor’ merupakan label nama dari setiap sensor yang menghasilkan data suhu, kelembapan dan pH. Penamaan sensor menggunakan angka. Contoh: Sensor1, Sensor2.

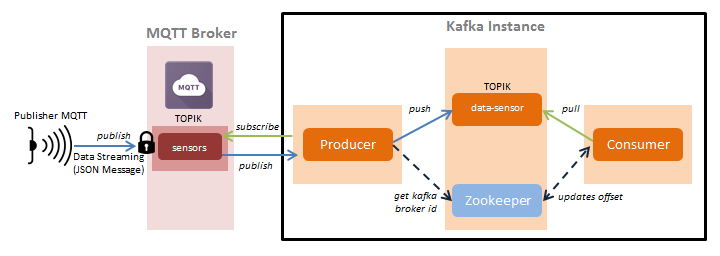
‘event\_time’ merupakan waktu dimana setiap data dihasilkan.

‘nilai\_suhu’ merupakan nilai hasil sensing suhu pada lingkungan. Tipe data nilai suhu menggunakan integer dengan jangkauan nilai 20 sampai 30 dalam derajat celcius (°C).

‘nilai\_kelembapan’ merupakan nilai hasil sensing kelembapan pada lingkungan. Tipe data nilai kelembapan menggunakan integer dengan jangkauan nilai 36 sampai 50 dalam persen (%).

‘nilai\_pH’ merupakan nilai hasil sensing pH. Tipe data nilai pH menggunakan integer dengan jangkauan nilai 5.8 sampai 7.8.

Publisher MQTT (node sensor) mengirimkan datanya ke suatu topik pada broker MQTT. Broker MQTT berada di mesin yang berbeda dengan publisher. Implementasi broker MQTT menggunakan Eclipse Mosquitto.



**Gambar 4.2 Perancangan Integrasi Data**

Pada MQTT broker dilakukan konfigurasi untuk mengharuskan autentikasi client menggunakan sebuah username dan password yang valid sebelum koneksi diizinkan. Penggunaan autentikasi username dan password menyediakan cara yang mudah dalam membatasi akses ke broker. Konfigurasi MQTT broker membutuhkan *password file* dan menyunting *file* konfigurasi pada Mosquitto. File password berisi username dan password. Kemudian melakukan konversi password file untuk mengenkripsi password menggunakan sebuah command. Setelah password file dapat digunakan, melakukan konfigurasi pada file konfigurasi Mosquitto.

MQTT didesain untuk perangkat berdaya rendah, tidak dapat menangani aliran data yang rendah. Kafka dapat menangani hal tersebut. Oleh karena itu, MQTT dan Kafka menjadi sebuah kombinasi yang sangat baik untuk proses integrasi *end-to-end* IoT dari *edge* ke *data center*. Kafka tidak dapat menggantikan MQTT, karena memiliki tujuan yang berbeda.

Kafka menjadi messaging system yang dirancang untuk pengiriman data stream. Kafka sering digunakan untuk menampung data stream dalam jangka waktu menengah sampai nantinya diarahkan ke dalam cluster untuk diproses lebih lanjut. Terdapat 2 client Kafka yaitu producer dan consumer. Producer bertugas untuk menerima data dari server Kafka. Sedangkan consumer bertugas untu menerima data dari server Kafka. API producer menerima data dari broker MQTT. Data tersebut berasal dari topik ‘sensors’. Dalam hal ini, producer Kafka menjadi subscriber MQTT. Sehingga dilakukan autentikasi ke broker MQTT menggunakan username dan password. Jika koneksi diizinkan, maka dapat menerima data dari broker MQTT dan mengirimkan ke server Kafka. Data streaming dikirimkan ke sebuah topik pada server Kafka dengan nama “iot-data-sensor”.

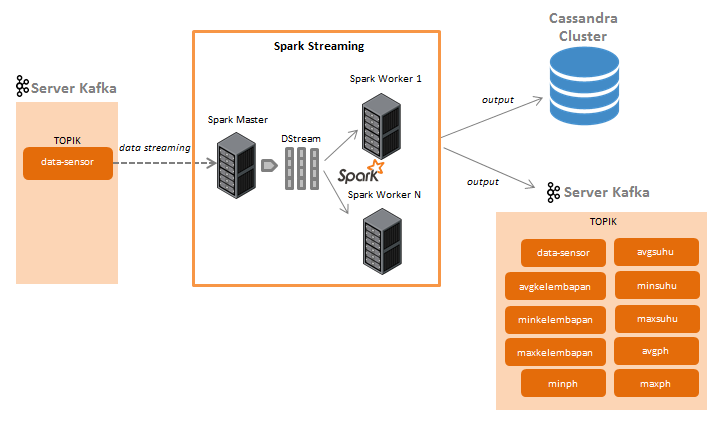
Setelah data berada di server Kafka, data tersebut akan diproses lebih lanjut. Consumer dapat berupa aplikasi, basis data atau sistem lainnya. Dalam hal ini, aplikasi akan membuat onjek consumer, kemudian melakukan subscribe ke topik tertentu dan mulai menerima data, validasi dan menulis hasilnya. Terdapat 2 consumer pada penelitian ini yaitu aplikasi untuk menyimpan data ke basis data dan aplikasi Spark Streaming.

Pada implementasi Kafka, semua komponen Kafka ditempatkan dalam satu node. Selain komponen server dan client, komponen Kafka lainnya yaitu Zookeeper. Zookeeper diperlukan untuk menjaga track Kafka node dengan memperhatikan topik dan partisi. Karena server Kafka dan Zookeeper dijalankan pada node yang sama, sehingga harus memperhatikan kapasitas memori untuk menjalankan JVM.

### Perancangan Komponen Pengolahan Data

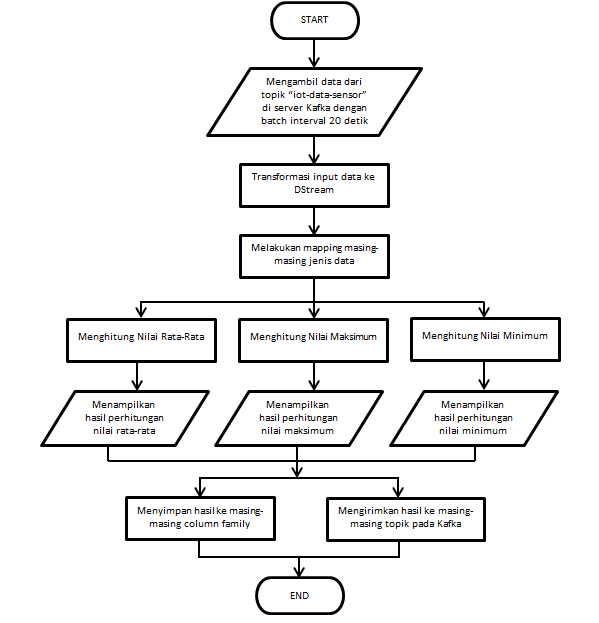
Spark Streaming merupakan data processor yang bertugas untuk mengolah data streaming menggunakan komputasi terdistribusi. Spark Streaming merupakan salah satu API Apache Spark. Apache Spark berjalan pada arsitektur Master-Slave dan koordinator pusat yang disebut driver program. Driver program merupakan proses utama pada aplikasi Spark dan menjalankan kode untuk membuat objek SparkContext. Proses pada driver mengubah sebuah aplikasi user ke dalam eksekusi yang lebih kecil yang disebut dengan task. Task dieksekusi oleh exceutor yang merupakan proses dari node worker.

Apache Spark dibangun di sebuah cluster yang terdiri dari satu node master dan beberapa node worker. Node master dan node worker berada di node berbeda. Driver program menjalankan proses pengolahan ke executor menggunakan cluster manager. Cluster manager akan melakukan penjadwalan dan alokasi resource yang diperlukan antar node dalam cluster. Cluster manager yang digunakan adalah Spark Standalone. Cluster manager ini tersedia sebagai bagian dari distribusi Spark. Pada proses penjadwalan, Spark Standalone menggunakan sebuah FIFO scheduler untuk aplikasi. Penggunaan resource seperti memori, cpu dan komponen lainnya diatur oleh objek SparkConf. Dalam penelitian ini, melibatkan semua node pada cluster dengan memperhatikan memori yang tersedia.



**Gambar 4.3 Perancangan Pengolahan Data**

Setelah cluster Spark selesai diimplementasikan, kemudian dilakukan pengolahan data streaming menggunakan Spark Streaming. Spark menjalankan aplikasi dalam sebuah cluster menggunakan spark-submit. Projek dijalankan menggunakan bahasa pemrograman Python. Pada saat akan menjalankan aplikasi, memperhatikan beberapa dependencies seperti alamat ip node master, package yang digunakan, script python dan konfigurasi lainnya.

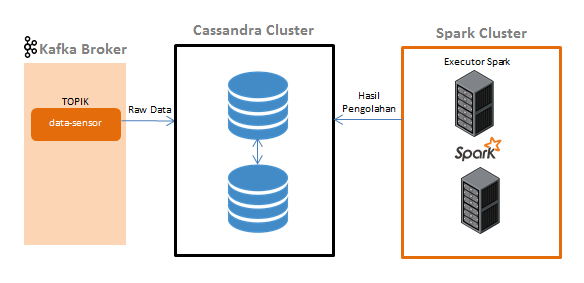


**Gambar 4.4 Flowchart alur pengolahan data pada Spark Streaming**

Aplikasi Spark menerima data dari topik “iot-data-sensor”. Apache Spark menerima data dari Kafka server melalui konektor sparkstreaming-kafka. Pada saat pengambilan data streaming, harus menyertakan batch interval. Batch interval merupakan waktu Spark Streaming dalam mengumpulkan input data menjadi sebuah micro-batch sebelum dilakukan perhitungan. Setelah data diterima, Spark Streaming akan melakukan transformasi data menjadi DStream. Dstream merupakan abstraksi data pada Spark Streaming. Data yang masuk dipecah menjadi sebuah sequence RDD yang menggambarkan data stream. Setelah berhasil ditransformasi menjadi Dstream, aplikasi menggunakan beberapa operasi dasar yang tersedia untuk melakukan perhitungan rata-rata, nilai maksimum dan nilai minimum. Perhitungan nilai tersebut dilakukan secara terdistribusi oleh executor yang berada di node worker. Hasilnya disimpan kedalam basis data Cassandra dan mengirimkan ke topik yang berbeda pada Kafka.

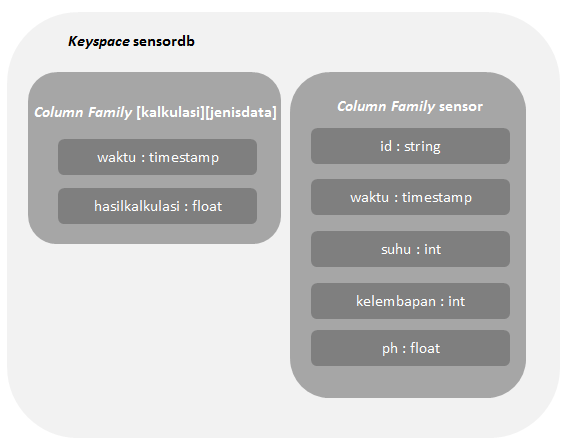
### Perancangan Komponen Penyimpanan Data

Jumlah data yang dihasilkan dari sensor IoT sangatlah besar, hal tersebut mengubah pandangan komputasi dan basis data. Hal ini tidak mengatakan bahwa basis data relasional tidak digunakan, tetapi terdapat beberapa permasalahan yang membutuhkan pendekatan berbeda dalam hal mengolah, menyimpan dan manajemen data. Basis data NoSQL menawarkan kemampuan yang dirancang untuk memenuhi kebutuhan data. Cassandra merupakan salah satu basis data NoSQL yang dapat digunakan sebagai penyimpanan data IoT. Pada sistem ini, Cassandra cluster terdiri dari 2 node. Semua node Cassandra mempunyai kedudukan yang sama. Masing-masing node bekerja sama dalam hal penyimpanan data yang berasal dari Kafka Broker dan Spark Cluster. Data yang disimpan berupa raw data yaitu data yang belum diolah dan hasil pengolahan dari Spark Streaming. Data disimpan dari Kafka Broker menggunakan DataStax Python Cassandra Driver. Sedangkan data yang disimpan dari Spark Cluster menggunakan connector Spark Streaming Cassandra yang dikembangkan oleh Julian Anguenot.



**Gambar 4.5 Perancangan Penyimpanan Data**

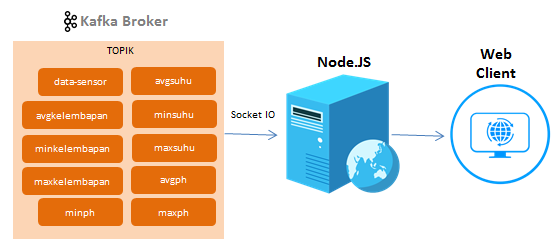
Cassandra menggunakan konsep key-value dimana terdiri dari kumpulan key dan value yang berasosiasi dengan key tersebut. Schema pada Cassandra tidak harus didefinisikan pada saat waktu mendesain, tetapi beberapa properti penyimpanan seperti column family dan beberapa atribut dapat didefinisikan terlebih dahulu. Nama yang mendefinisikan replikasi data antar node disebut dengan *keyspace* dan terdiri dari beberapa *row key* dan *column family*. Pada Cassandra dapat mendefinisikan struktur untuk data dan menambahkan tipe data. Untuk memudahkan penyimpanan, skema didefinisikan terlebih dahulu. Terdapat 1 keyspace dan 10 column family. Column family terdiri dari 1 column family untuk penyimpanan raw data dan 9 column family untuk hasil pengolahan. Schema yang digunakan dijelaskan pada gambar 4.7 berikut.



**Gambar 4.6 Schema Cassandra**

### Perancangan Visualisasi

Visualisasi digunakan untuk melihat hasil pengolahan data secara real time. Aplikasi website digunakan sebagai salah satu alternatif dalam visualisasi pengolahan data. Node JS sebagai server mengambil data dari Kafka Broker. Data diambil dari masing-masing topik pada Kafka Broker. Pengambilan data dari masing-masing topik, mempermudah dalam visualisasi. Karena visualisasi harus real time, pengambilan data menggunakan Socket IO. Untuk ditampilkan ke client, menggunakan D3.



**Gambar 4.7 Perancangan Visualisasi**

Pada sisi client, pengguna akan melihat hasil pengolahan data dalam bentuk grafik. Grafik tersebut akan menggambarkan hasil pengolahan data secara real time. Grafik terdiri dari nilai pengolahan data dan waktu. Pada sumbu X digunakan untuk menggambarkan waktu dari hasil pengolahan data. Sedangkan sumbu Y menggambarkan nilai pengolahan data. Pada halaman website, terdapat 10 grafik. Masing-masing grafik menggambarkan raw data dan hasil pengolahan untuk data suhu, kelembapan serta pH.

### Perancangan Pengujian

Pengujian dilakukan untuk mengetahui bahwa platform pengolahan data sensor berjenis streaming sudah dibangun sesuai dengan kebutuhan yang telah dijabarkan pada tahap analisis dan perancangan.

#### Pengujian Fungsional

Pengujian fungsional dilakukan untuk menguji keseluruhan komponen pada platform dapat berjalan. Pertama menguji apakah subscriber broker MQTT atau producer Kafka dapat menerima data streaming dari broker MQTT kemudian mengirimkan ke sebuah topik Kafka. Terdapat 2 consumer pada Kafka yaitu database Cassandra dan Spark Streaming. Setelah data berhasil dikirimkan ke sebuah topik Kafka, kemudian menguji apakah data tersebut dapat disimpan ke dalam Cassandra. Selain itu, menguji apakah Spark Streaming dapat menerima data dari sebuah topik Kafka, kemudian dilakukan kalkulasi untuk mendapatkan nilai rata-rata, nilai maksimum dan nilai minimum. Setelah itu, hasil kalkulasi dikirimkan ke 2 output yang berbeda yaitu Cassandra dan dikirimkan ke topik yang berbeda di Kafka. Pengujian pada Cassandra dilakukan dengan melihat apakah hasil pengolahan berhasil disimpan kedalam basis data. Selain itu, menguji apakah hasil kalkulasi sudah berhasil masuk ke topik yang berbeda di Kafka. Pengujian tersebut dapat dilihat dengan menggunakan perintah consumer Kafka dengan menampilkan isi pada masing-masing topik Kafka. Setelah itu, dilakukan pengujian bagian visualisasi. Pengujian dilakukan dengan melihat apakah aplikasi website dapat menampilkan hasil pengolahan secara real time berupa grafik.

#### Pengujian Non Fungsional

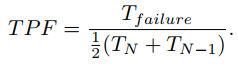
**Metrik**

Kapabilitas platform

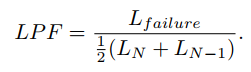
Kemampuan platform dalam melakukan pengolahan data verdasarkan pengujian *throughput* dan *latency*. *Throughput* adalah jumlah data/event yang diproses setiap detik. Sedangkan *latency* adalah rata-rata rentang waktu pengolahan data (*processing time*).

Fault-tolerant

Pengujian dilakukan untuk mengetahui dampak yang ditimbulkan jika ada node yang gagal. Pengujian dilakukan dengan mempertimbangkan kegagalan satu node. Hal tersebut memerlukan kegagalan node secara sengaja di pertengahan waktu eksekusi untuk melihat kemampuan komputasi platform. Jika sejumlah N node yang pertama dijalankan, kemudian salah satu node dimatikan, tersisa N-1 node yang berfungsi. Throughput untuk N node tanpa kegagalan dinyatakan sebagai TN, Throughput untuk N-1 node tanpa kegagalan dinyatakan sebagai TN-1, dan throughput pada saat gagal dinyatakan sebagai Tfailure. Untuk mendefinisikan *penalty factor* TPF sebagai berikut :



Pada umumnya, nilai TPF dibawah 1 dan jika mendekati 1 nilainya makin baik sehingga tidak terlalu berpengaruh dengan komputasi. Hampir sama dengan TPF, latency penalty factor LTF sebagai berikut:



Dimana L menyatakan latency. Pada umumnya, nilai LPF lebih besar dari 1 dan jika mendekati 1 nilainya makin baik (Qian et al, 2016).

Penggunaan Memori dan CPU

Metrik ini digunakan untuk melihat penggunaan memori dan CPU pada Spark Cluster. Monitoring dilakukan di setiap komponen Spark Cluster yaitu Spark Master dan Spark Worker.

**Skenario**

Skenario yang dilakukan untuk menguji masing-masing parameter sebagai berikut:

Jumlah Data

Jumlah data yang dikirimkan dari node sensor sejumlah 100, 400 dan 800 data per detik secara bersamaan.

Jumlah Node

Jumlah node worker yang digunakan untuk memproses data di Spark Streaming. Jumlah node yang digunakan sejumlah 1 node worker yang terletak di node yang sama dengan node master, 1 node, 4 node, 8 node.

Batch Interval

Batch interval adalah jumlah waktu yang dibutuhkan oleh Spark Streaming untuk mengumpulkan data dalam batch sebelum di proses. Pengujian dilakukan dengan memberikan batch interval sejumlah 3 detik, 9 detik, 18 detik dan 27 detik.

**Lingkungan Pengujian**

Lingkungan pengujian dilakukan menggunakan cloud. Cloud yang digunakan adalah Amazon Web Service (AWS). Pada pengujian ini, layanan AWS yang dipakai adalah Elastic Compute Cloud (EC2). EC2 menyediakan berbagai macam jenis instance terdiri dari kombinasi kapasitas CPU, memory, penyimpanan dan jaringan yang memberikan pilihan untuk aplikasi yang akan dibangun. Penelitian ini menggunakan jenis instance t2.micro dengan spesifikasi vCPU 1 dan memori 1 GiB. Layanan EC2 ditempatkan di berbagai macam lokasi didunia. Region yang dipakai dalam pengujian adalah N. Virginia dengan kode us-east-1. Semua komponen sistem diinstal kedalam EC2 yang terdiri dari beberapa Availibility Zone yang tersedia pada Region N. Virginia. Setelah sistem selesai dibangun, dilakukan pengujian berdasarkan parameter dan skenario.

|  |  |
| --- | --- |
| **Keterangan** | **Konfigurasi** |
| *Machine* | AWS Elastic Compute Cloud (EC2) |
| Region | N. Virginia (us-east-1) |
| Jenis EC2 | M5.large |
| vCPU | 2 |
| Memori | 8 GiB |
| *Storage* | 8 GiB General Purpose SSD |

# IMPLEMENTASI

(Di bagian utama kasih pseudocode, keseluruhan kodingan dan konfigurasi di taruh di lampiran)

Pada bab 5 berisi tentang implementasi pengembangan platform pengolahan data streaming. Implementasi dilakukan berdasarkan perancangan yang telah dibuat pada subbab 3.4. Implementasi terdiri dari 4 subbab yaitu implementasi integrasi data, implementasi pengolahan data, implementasi penyimpanan data dan implementasi visualisasi.

## Implementasi Integrasi Data

Integrasi data dibutuhkan untuk memudahkan pendistribusian dan manajemen data pada sistem yang berjalan. Untuk menguji sistem, dibutuhkan data streaming. Pada penelitian ini tidak melibatkan hardware layer, sehingga menggunakan data dummy. Data sensor IoT dibuat menggunakan script Python dengan format JSON. Data dihasilkan secara tersu menerus agar mendekati data streaming sesungguhnya. Data yang digunakan terdiri dari 5 macam sensor yang masing-masing mengirimkan data suhu, kelembapan dan pH.

Protokol yang digunakan dalam komunikasi data adalah MQTT. MQTT terdiri dari client dan broker. Data streaming dikirimkan dan diterima oleh MQTT client menggunakan Eclipse Paho. Data tersebut dikirim dan diambil ke sebuah topik yang spesifik pada broker. Implementasi broker MQTT menggunakan Eclipse Mosquitto. Pada broker MQTT ditambahkan autentikasi sehingga tidak semua data bisa dikirim ke broker. Autentikasi yang dilakukan berupa pembuatan username dan password. Konfigurasi autentikasi pada broker dapat dilihat di Lampiran A pada bagian A.1.

**Kode Program 5.1 Deklarasi Broker MQTT dan Jumlah Sensor**

|  |  |
| --- | --- |
| config.json | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | {  "mqtt": {  "username": "fidia",  "password": "12345678",  "host": "172.31.81.182",  "port": 1883,  "topic": "sensors"  },  "sensors": {  "Sensor 1": {  "suhu": [20, 30],  "kelembapan": [36, 50],  "ph": [5.5, 7.8]  },  "Sensor 2": {  "suhu": [20, 30],  "kelembapan": [36, 50],  "ph": [5.5, 7.8]  },  "Sensor 3": {  "suhu": [20, 30],  "kelembapan": [36, 50],  "ph": [5.5, 7.8]  },  "Sensor 4": {  "suhu": [20, 30],  "kelembapan": [36, 50],  "ph": [5.5, 7.8]  },  "Sensor 5": {  "suhu": [20, 30],  "kelembapan": [36, 50],  "ph": [5.5, 7.8]  }  } } |

Pada kode program 5.1, **config.json** berisi tujuan data dikirimkan ke broker MQTT dan jumlah sensor yang digunakan. Pada baris 2-8 menjelaskan tujuan broker MQTT. Data streaming dikirim ke broker dengan alamat IP, *username*, *password*, *port* dan topik. Pada baris 9-35 mendeklarasikan jumlah sensor yang digunakan dan data yang dikirim. Data yang dikirimkan berupa data suhu, kelembapan dan ph.

**Kode Program 5.2 Implementasi Pengiriman Data Streaming ke Broker MQTT**

|  |  |
| --- | --- |
| datastreaming.py | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96 | *#!/usr/bin/env python3* import sys import json import random import time from datetime import datetime import paho.mqtt.client as mqtt import uuid  def generate(host, port, username, password, topic, sensors, interval\_ms):   mqttc = mqtt.Client()   if username:  mqttc.username\_pw\_set(username, password)   mqttc.connect(host, port)   keys = list(sensors.keys())  interval\_secs = 1 *#generate data terus menerus, dengan 100 data per detik* while True:  for counter in range(100):  id = str(uuid.uuid4())  sensor\_id = random.choice(keys)  sensor = sensors[sensor\_id]   *# Suhu* min\_valtemp, max\_valtemp = sensor.get("suhu", [20, 30])  valtemp = random.randint(min\_valtemp, max\_valtemp)   *# Kelembapan* min\_valhum, max\_valhum = sensor.get("kelembapan", [36, 50])  valhum = random.randint(min\_valhum, max\_valhum)   *# pH* min\_valph, max\_valph = sensor.get("ph", [5.8, 7.8])  valph = round(random.uniform(min\_valph, max\_valph), 1)   waktu = datetime.now()  data = {  "id": id,  "sensor": sensor\_id,  "suhu": valtemp, "kelembapan": valhum, "ph": valph, "waktu": waktu.strftime("%Y-%m-%d %H:%M:%S")  }  payload = json.dumps(data)   print("%s%s: %s" % (topic, counter, payload))  mqttc.publish(topic, payload)   time.sleep(interval\_secs)  def main(config\_path):  *"""main, load isi dari config.json dan memanggil function generate"""* try:  with open(config\_path) as handle:  config = json.load(handle)  mqtt\_config = config.get("mqtt", {})  sensors = config.get("sensors")   if not sensors:  print("Tidak ada sensor yang mengirimkan data")  return   host = mqtt\_config.get("host", "172.31.81.182")  port = mqtt\_config.get("port", 1883)  username = mqtt\_config.get("username")  password = mqtt\_config.get("password")  topic = mqtt\_config.get("topic", "sensors")   dateTimeObj = datetime.now()  timeObj = dateTimeObj.time()  time.sleep(1 - (timeObj.microsecond / 1000000))   generate(host, port, username, password, topic, sensors, interval\_ms, verbose)  except IOError as error:  print("Error membuka file config '%s'" % config\_path, error)  if \_\_name\_\_ == '\_\_main\_\_':  if len(sys.argv) == 2:  main(sys.argv[1])  else:  print("Gunakan %s config.json" % sys.argv[0]) |

Kode program 5.2 berisi implementasi pengiriman data streaming ke broker MQTT.

## Implementasi Pengolahan Data

## Implementasi Penyimpanan Data

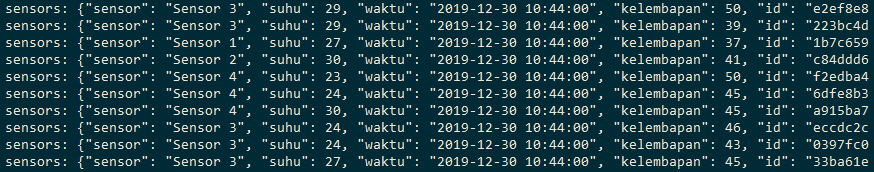
## Implementasi Visualisasi

# PENGUJIAN DAN ANALISIS hasil pengujian

## Pengujian Fungsional

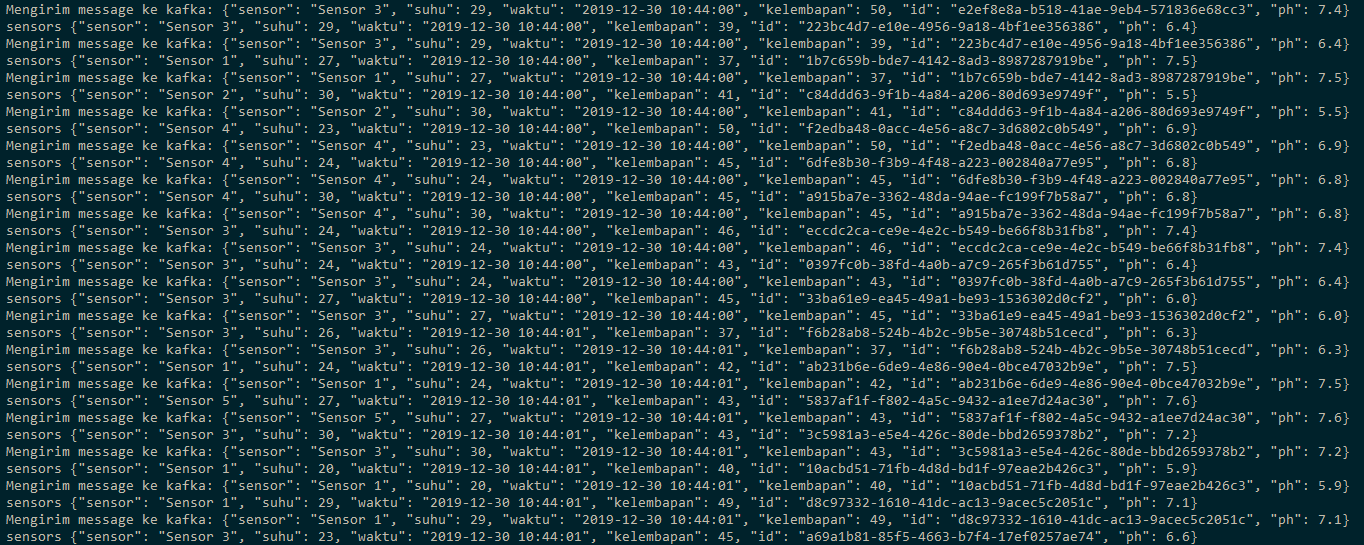
### Publisher mengirimkan data streaming ke broker MQTT



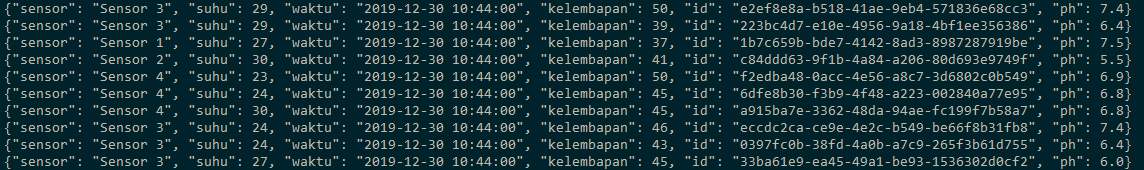


### Subscriber menerima data dari broker MQTT dan mengirimkan ke server Kafka





### Data streaming berada di topik iot-data-sensor (Kafka)



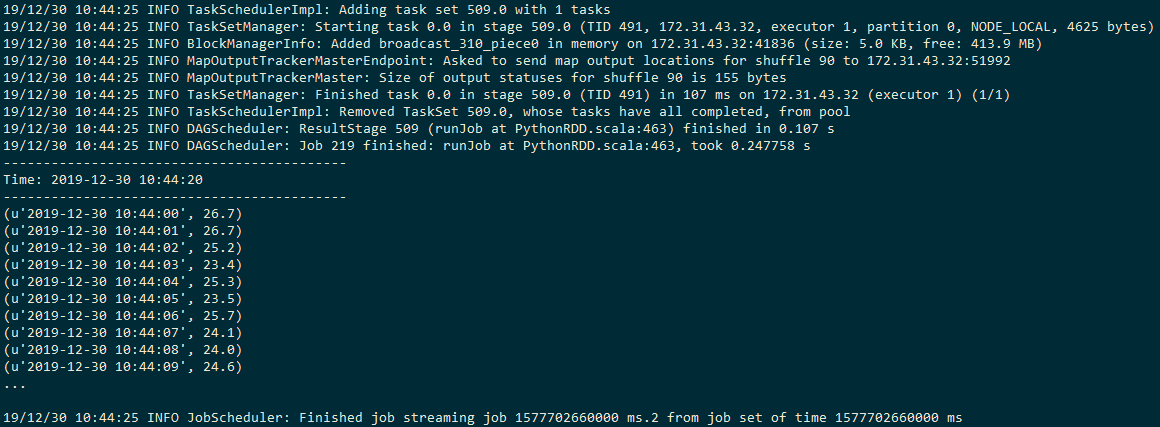
### Aplikasi Spark menerima data dari server Kafka

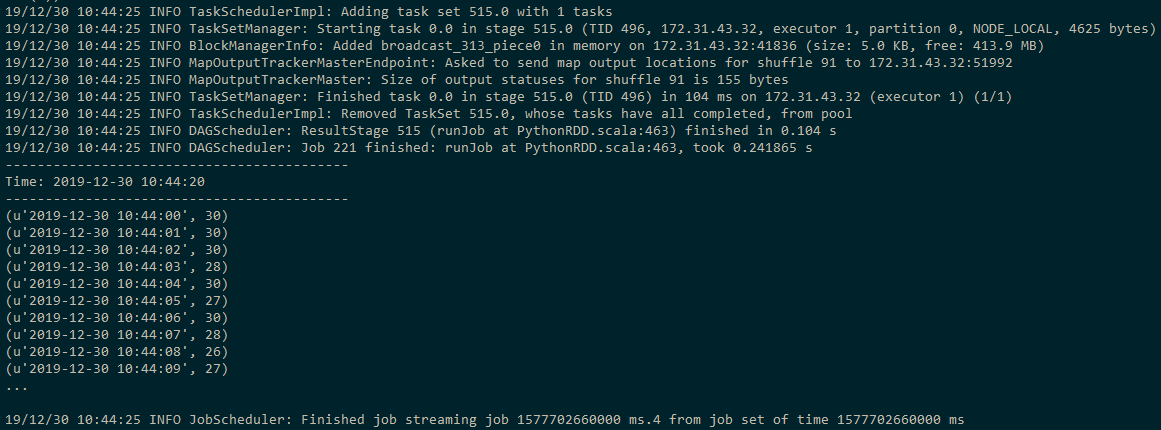


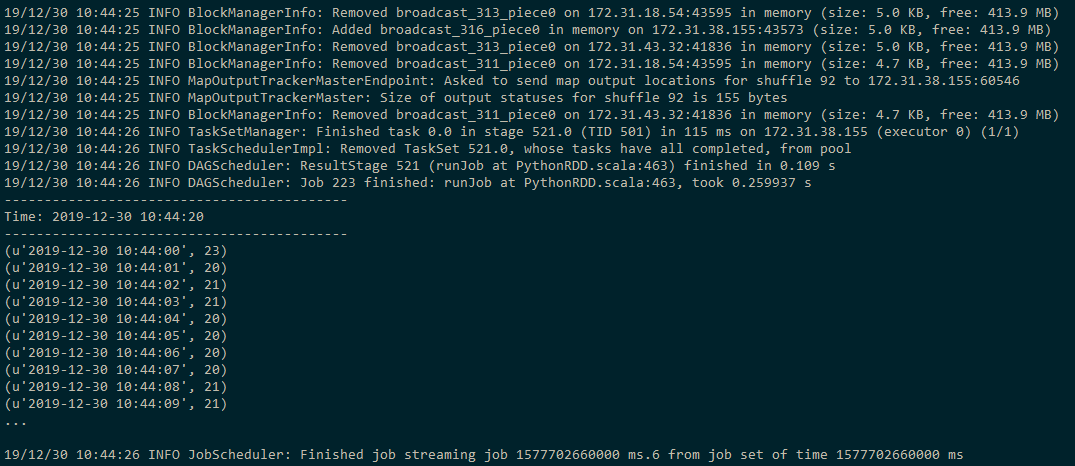
### Aplikasi Spark menampilkan hasil perhitungan

#### Perhitungan nilai rata-rata

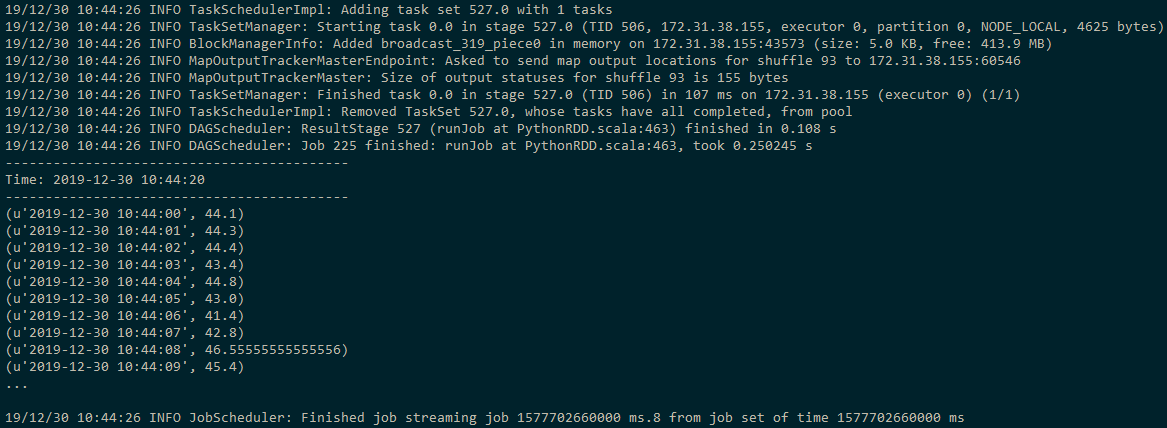
Suhu :

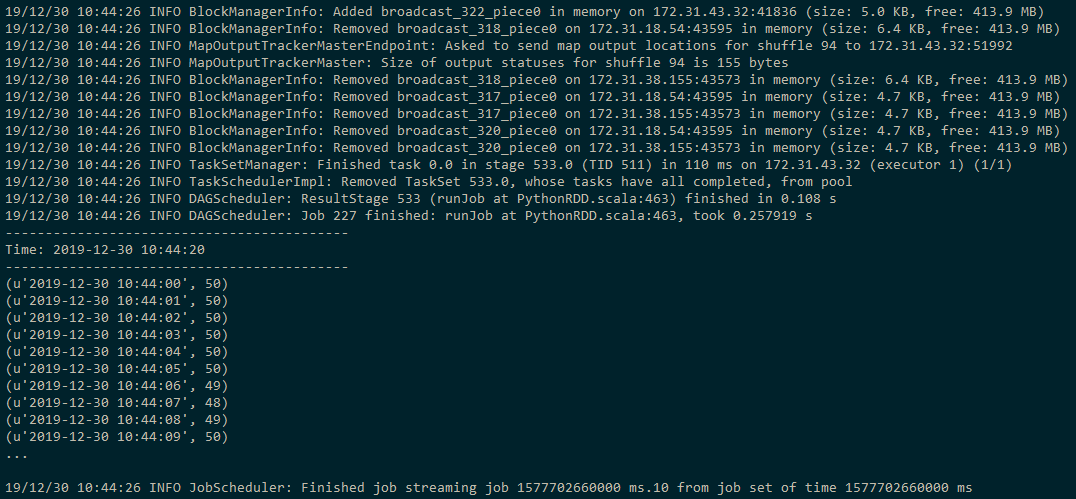


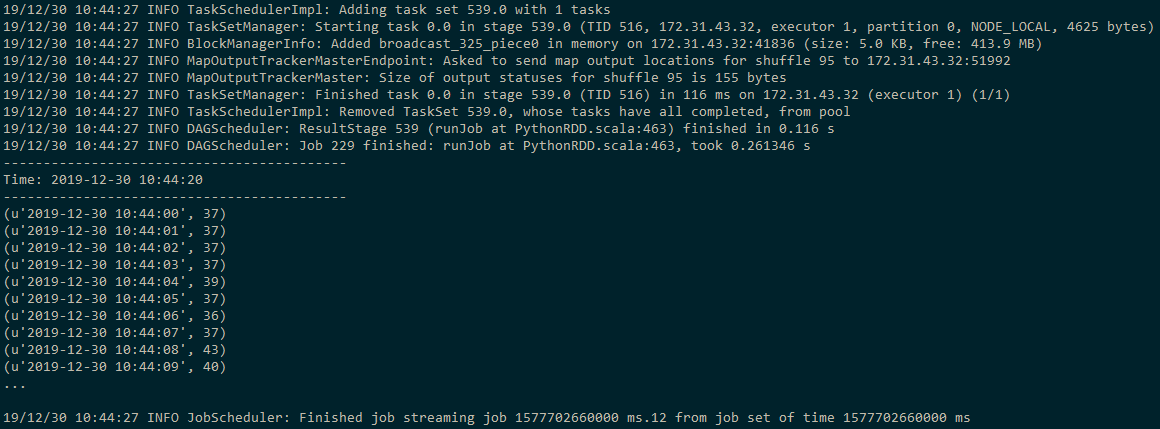




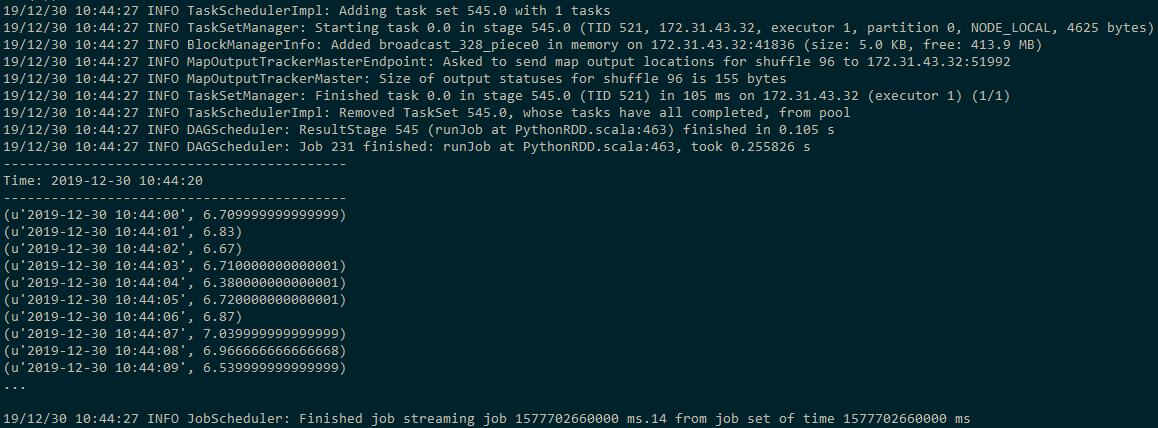
Kelembapan :

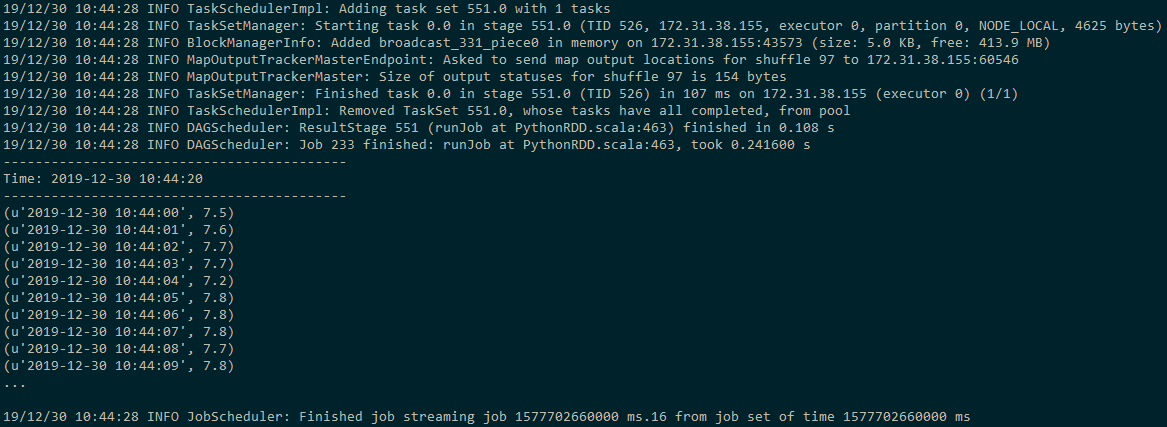


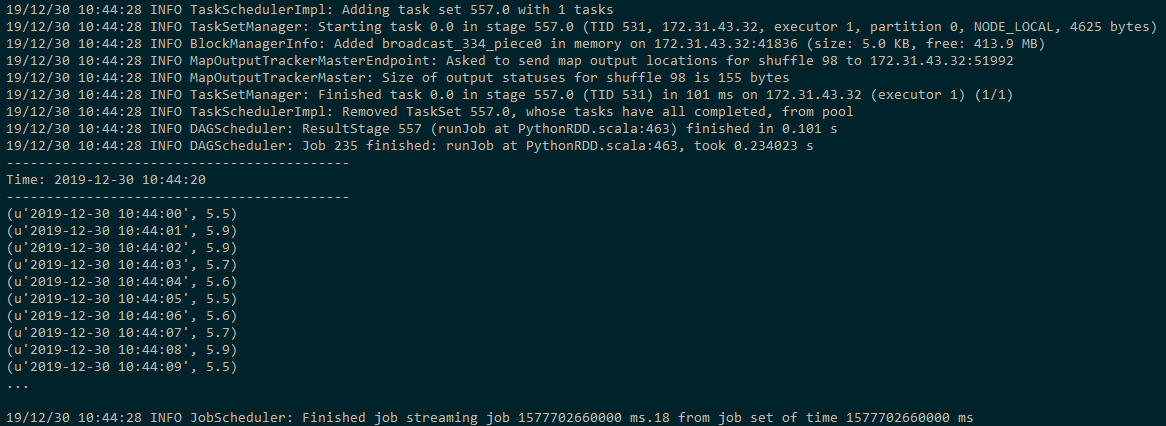




pH :







#### Perhitungan nilai maksimum

Suhu :

Kelembapan :

pH :

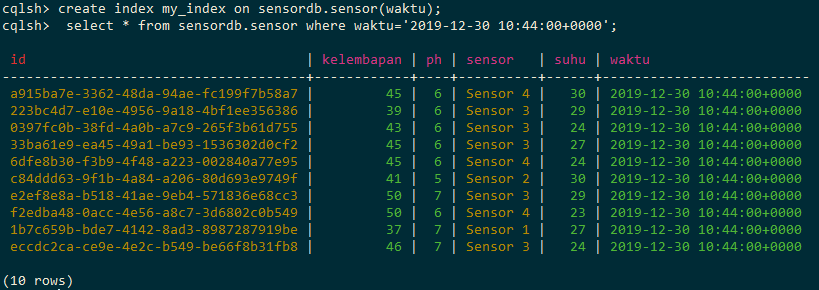
#### Perhitungan nilai minimum

Suhu :

Kelembapan :

pH :

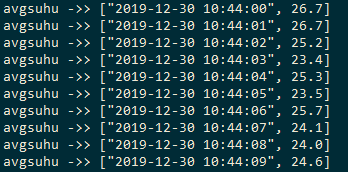
### Aplikasi Spark menyimpan data ke basis data Cassandra

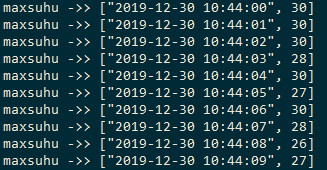


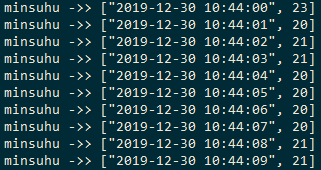
### Aplikasi Spark mengirim hasil ke topik yang berbeda di server Kafka

### Mengambil data dari Topik Kafka dan Menampilkan hasil pengolahan melalui website

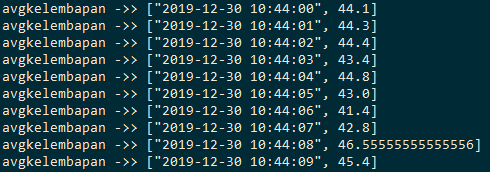
#### Suhu

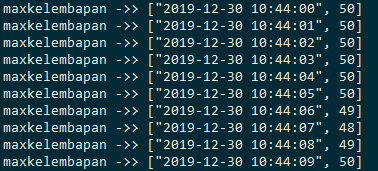


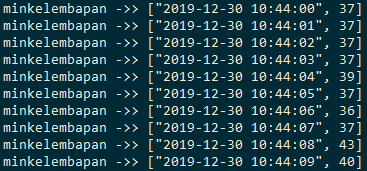




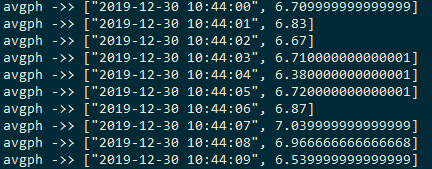
#### Kelembapan

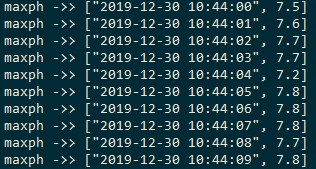


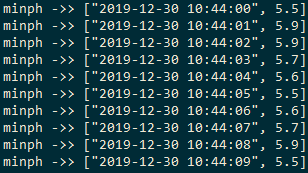


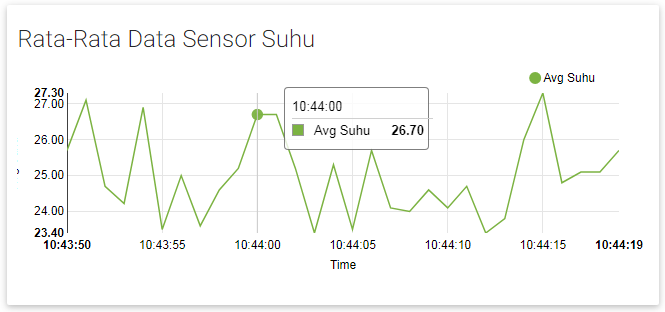


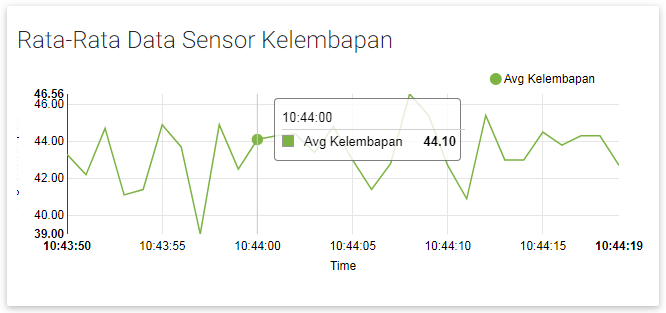
#### pH

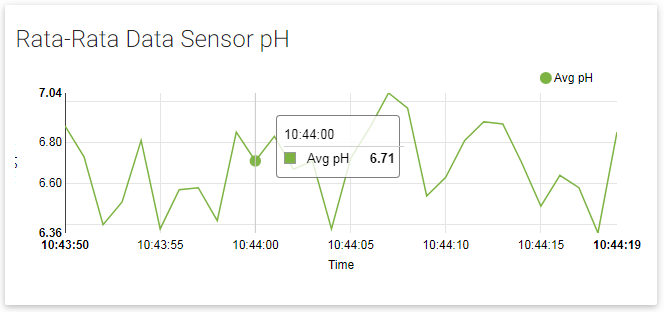


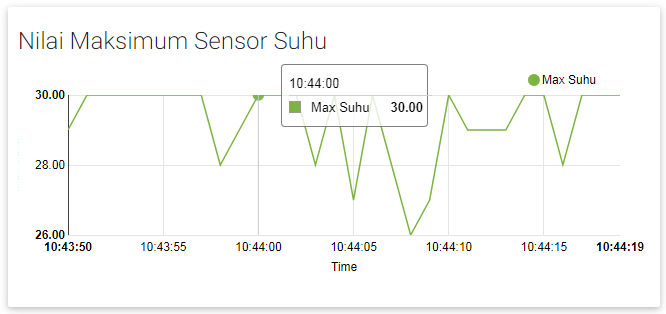


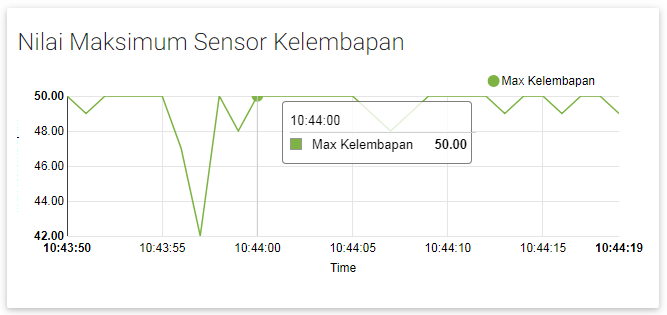


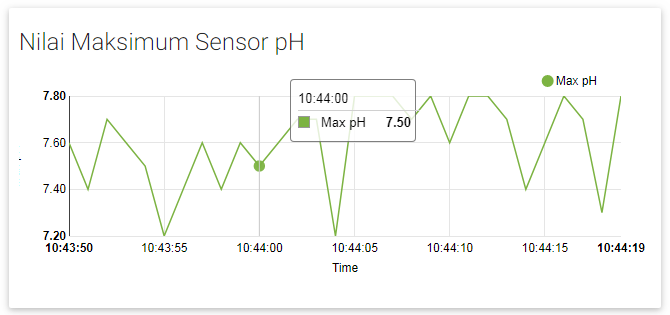


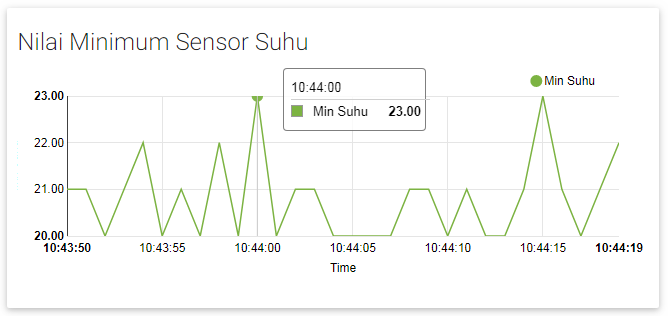


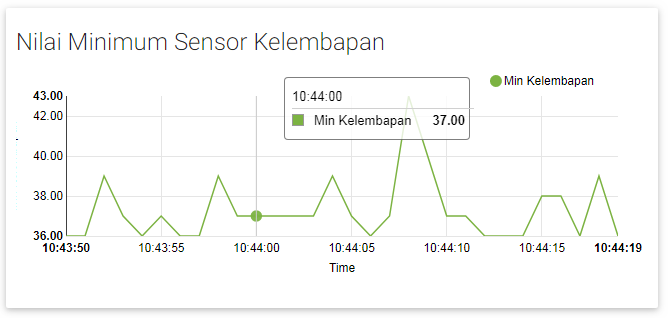


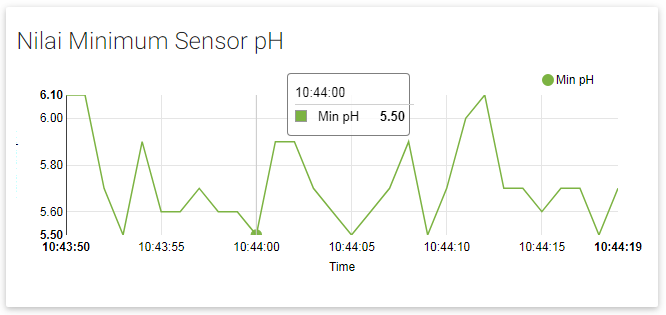












## Pengujian Non Fungsional

### Kapabilitas Platform

***Throughput(records/detik)***

Keterangan : N = Jumlah Node Worker, batch interval = 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Jumlah Node** | **Jumlah Data** | | | |
| **1000** | **1500** | **2000** | **2500** |
| 2 | 450,5 | 556,4 | 742,9 | 809,3 |
| 4 | 573 | 635 | 777,3 | 948,9 |
| 8 | 419,2 | 705,2 | 878,6 | 723,9 |

***Latency(detik)***

Keterangan N = Jumlah Node Worker

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Jumlah Node** | **Jumlah Data** | | | |
| **1000** | **1500** | **2000** | **2500** |
| 2 | 6,6 | 6,9 | 8,7 | 7,1 |
| 4 | 5,6 | 5,8 | 7,1 | 9,9 |
| 8 | 16,4 | 15,8 | 12,5 | 13,6 |

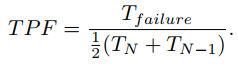
### Penggunaan CPU, memori dan disk

|  |  |  |
| --- | --- | --- |
| Jumlah Node | CPU | Memori |
| 2 |  |  |
| 4 |  |  |
| 8 |  |  |

### Fault-Tolerant

Dilakukan dengan period detik yang sama. Misal setelah 30 detik running aplikasi. Batch Interval = 1 detik, jumlah message = 2500/detik.

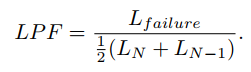
Untuk mendefinisikan *penalty factor* TPF sebagai berikut :



Pada umumnya, nilai TPF dibawah 1 dan jika mendekati 1 nilainya makin baik sehingga tidak terlalu berpengaruh dengan komputasi

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Jumlah Node** | **TN** | **TFAILURE** | **TN-1** | **TPF** |
| 2 | 615,3 | 14,8 | 27,6 | 0,04 |
| 4 | 971,5 | 56 | 177,2 | 0,09 |
| 8 | 1163,4 | 70,6 | 170,8 | 0,1 |

Hampir sama dengan TPF, latency penalty factor LTF sebagai berikut:



Dimana L menyatakan latency. Pada umumnya, nilai LPF lebih besar dari 1 dan jika mendekati 1 nilainya makin baik (Qian et al, 2016).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Jumlah Node** | **LN** | **LFAILURE** | **LN-1** | **LPF** |
| 2 | 6 | 276 | 108 | 2,4 |
| 4 | 11 | 168 | 60 | 2,3 |
| 8 | 10 | 134 | 56 | 2,03 |

# BAB 7 PENUTUP

## 7.1 Kesimpulan

## Saran

# BAB 8 DAFTAR REFERENSI

LAMPIRAN A **KONFIGURASI INTEGRASI DATA**

## **A.1** Konfigurasi Autentikasi Broker MQTT

Pada broker MQTT dilakukan autentikasi, sehingga tidak semua data bisa dikirimkan ke broker MQTT. Autentikasi berupa pembuatan username dan password. Langkah-langkah dalam pembuatan username dan password berikut ini:

|  |
| --- |
| sudo nano autentikasi.txt  mosquitto\_passwd –U autentikasi.txt |

Pada file **autentikasi.txt** berisi username dan password. Dibawah ini merupakan isi dari file **autentikasi.txt.** “fidia” merupakan username yang digunakan dan “12345678” adalah password.

|  |
| --- |
| autentikasi.txt |
| fidia:12345678 |