Studi dan Implementasi Spark Streaming untuk Mengumpulkan Big Data Stream

Muhammad Ravi–2016730041

1 Data Skripsi

Pembimbing utama/tunggal: **Veronica Sri Moertini**
Pembimbing pendamping: -
Kode Topik : **VSM4702**
Topik ini sudah dikerjakan selama : **1** semester
Pengambilan pertama kali topik ini pada : Semester **47 - Ganjil 19/20**
Pengambilan pertama kali topik ini di kuliah : **Skripsi 1**
Tipe Laporan : **B -** Dokumen untuk reviewer pada presentasi dan **review Skripsi 1**

2 Latar Belakang

Dalam beberapa tahun terakhir,Perkembangan data melonjak secara cepat. Hal ini disebabkan karena semakin banyak orang yang terhubung secara digital. Website yang diakses, media sosial yang dijelajahi, atau sensor-sensor dari barang-barang elektronik yang terhubung ke internet semua meninggalkan jejak digital berupa data. Data yang terakumulasi ini berukuran besar dengan format yang bervariasi dan berkembang dengan sangat cepat.

Jika data yang terakumulasi tersebut diolah dan dianalisis, banyak informasi-informasi bermanfaat yang bisa didapat. Contohnya, data bisa menjadi bahan pertimbangan untuk pengambilan keputusan bisnis. Tetap, Tidak semua data memiliki nilai dan sifat yang sama. Ada data yang memiliki nilai lebih ketika bisa langsung dianalisis ketika didapatkan. Kebutuhan untuk langsung mendapatkan dan menganalisis data secara *real-time*menjadi sangat penting. Selain itu, teknik pengumpulan data yang digunakan untuk pola data yang datang secara terus menerus berbeda dengan teknik yang digunakan untuk mengumpulkan dan mengolah data biasa. *Big Data* yang perlu diakses secara *real-time* adalah page views pada sebuah website, sensor pada IoT (*Internet of Things*.

Selain itu kebutuhan untuk mengolah data dengan cepat semakin penting karena nilai suatu data cenderung menurun secara eksponensial seiiring bertambahnnya waktu. Banyak Perusahaan dan Organisasi yang membutuhkan data untuk diolah secara cepat. Semakin cepat data bisa diambil, dianalisis, dimanipulasi, dan semakin banyak througput yang bisa dihasilkan maka sebuah organisasi akan lebih *agile* dan responsif. Semakin sedikit waktu yang digunakan untuk ETL (*Extract, Load, Transform*) pekerjaan akan semakin fokus untuk melakukan analisis bisnis.

Untuk menjawab masalah di atas, *Spark Streaming* merupakan teknologi yang menjadi salah satu solusi terhadap adanya kebutuhan untuk menganalisis *big data* secara *real time*. Data hasil streaming kemudian dapat dianalisis dengan teknik-teknik analisis data berbasis statistik maupun *machine learning/data mining* dan divisualisasikan agar lebih mudah dimengerti.

3 Rumusan Masalah

Berdasarkan latar belakang yang sudah dijabarkan, berikut adalah rumusan masalah untuk skripsi ini.

1. Bagaimana Karakteristik *data stream* dan contoh-contoh analisisnya?
2. Bagaimana cara kerja *Spark Streaming*?
3. Bagaimana cara mengintegrasikan *Spark Streaming* untuk mengumpulkan data?
4. Bagaimana cara menganalisis data yang telah terkumpul?

4 Tujuan

Berikut ini adalah tujuan yang ingin dicapai pada skripsi ini.

1. Melakukan studi tentang definis, pola-pola, arsitekstur, dan manfaat analisis dari data stream
2. Mempelajari konsep, arsitektur, cara kerja Spark Streaming dan integrasinya dengan teknologi-teknologi lain
3. Mengimplementasikan *Spark Streaming* pada sebuah sistem untuk mengumpulkan data stream dengan kasus-kasus tertentu.
4. Menganalisis dan mempresentasikan data

5 Detail Perkembangan Pengerjaan Skripsi

Detail bagian pekerjaan skripsi sesuai dengan rencan kerja/laporan perkembangan terkahir :

1. **Studi literatur mengenai *Big Data***
   Status : Ada sejak rencana kerja skripsi.
   Hasil : *Big Data* merupakan data yang melebihi kapasitas pemrosesan dari sistem basis data konvensional. Data Tersebut berukuran terlalu besar, tumbuh dengan sangat cepat, memiliki banyak variasi tipe data, dan tidak cukup pada arsitektur basis data konvensional.

   Big Data memberikan dua kegunaan untuk sebuah organisasi, yaitu untuk keperluan analisis dan keperluan bisnis. *Big Data* dapat dianalisis untuk mendapatkan informasi seperti hubungan antar pelanggan. Hal ini dapat dilihat dari hasil analisis transaksi setiap pelanggan, graf sosial, dan graf geografis.

   suatu set data dapat dikatakan sebagai big data jika set data tersebut memenuhi salah satu dari lima karakteristik big data. Kelima karakteristik *Big Data* yang sering disebut dengan 5V adalah *volume,velocity, variety, veracity, dan value.*

   (a) *Volume*
   volume merupakan istilah untuk menggambarkan ukuran dari data. Data dengan ukuran besar mempunyai kebutuhan penyimpanan dan pemrosesan yang berbeda serta tambahan dalam persiapan, pengolahan pemrosesan data. Data berukuran besar tersebut dapat berasal dan transaksi online, eksperimen penelitian ilmiah, sensor, dan media sosial.

   (b) *Velocity*
   Velocity dari data merupakan waktu yang diperlukan untuk melakukan pengolahaan data ketika data tersebut masuk ke penyimpanan. Untuk menangani data yang masuk dengan cepat,

perusahaan atau organisasi memerlukan solusi pemrosesan data yang elastis dan terbuka, dan sesuai.

Kecepatan dari data tidak selalu tinggi dan tergantung pada sumber data. Kecepatan data dapat dipertimbangkan ketika data berukuran besar dan dapat dihasilkan dalam waktu yang singkat. Seperti; 350.000 cuitan, 300 jam video, 171 juta surat elektronik, dan 330 *gigabytes* sensor.

(c) *Variety*
Variety atau variasi data mengacu pada banyaknya format dan tipe data yang perlu didukung oleh solusi *Big Data*. Variasi data ini merupakan tantangan bagi yang ingin melakukan integrasi, transformasi, pemrosesan, dan penyimpanan data.

(d) *Veracity*
Veracity mengacu pada kualitas atau akurasi data. Data yang masuk akan diperiksa untuk menentukan kualitas dan menghindari adanya data yang tidak valid serta menghilangkan *Noise*. Data yang masuk tersebut dapat terdiri dari sinyal dengan informasi tersimpan dan *noise* dari suatu set data. Noise tersebut tidak menyimpan informasi bermakna dan tidak bernilai. Oleh karena itu, data dengan perbandingan sinyal terhadap noise yang besar, memiliki veracity yang lebih besar.

(e) *Value*
Value atau nilai didefinisikan sebagai kegunaan data tersebut bagi perusahaan atau organisasi. Karakteristik value berhubungan *linear* dengan karakteristik *veracity* karena besarnya veracity tersebut menentukan besarnya nilai yang dimiliki suatu data dapat bernilai hanya pada rentang waktu tertentu saja dan tidak bernilai di luar rentang waktu tersebut.

Big Data yang diolah dapat dihasilkan oleh manusia maupun mesin. Data yang dihasilkan manusia berapa hasil interaksi antara manusia dengan sistem. Data yang dihasilkan oleh mesin dapat berupa hasil dari perangkat lunak maupun perangkat keras sebagai respon dari aktivitas dunia nyata. Data yang dihasilkan tersebut dapat dikelompokan menjadi tiga tipe dasar yaitu; data terstruktur, data tidak tersturktur, dan data semi terstruktur.

2. **Studi literatur mengenai *Data Stream* dan *Big Data Stream***
Status : Ada sejak rencana kerja skripsi.
Hasil : *Data Stream* adalah urutan rekaman atau kejadian (*events*) yang tidak pernah berhenti. Serangkaian *data stream* bersifat tidak terbatas dan akan terus dihasilkan *(in-motion)* sedangkan waktu dan tempat yang dialokasikan untuk mengolah data terbatas. Selain itu, sifat khusus dari *data stream* adalah interaksi yang terbatas dengan sumber data, *data stream* hanya bisa menerima data dari sumber dan tidak bisa mengirim informasi kembali dan data yang dihasilkan hanya bisa di akses dan diproses sekali saja sebelum ditumpuk dan dikumpulkan di tempat penyimpanan. Sehingga, hanya satu atau beberapa elemen terbaru saja yang bisa diproses.

Data stream banyak digunakan untuk proses agregasi, korelasi, dan *filtering* secara *real-time*. *Data Stream* memungkinkan pengguna untuk melihat *insights*, menganalisisnya, dan menarik kesimpulan dari *insights* tersebut. Beberapa contoh analisis adalah; memantau informasi dari suatu sosial media seperti *twitter* untuk mendapatkan informasi untuk mengetahui tingkah pengguna, memantau aktivitas web dengan mencatat pembaharuan *web logs* setiap detiknya, mendeteksi anomali dari suatu jaringan atau sensor yang terus menerus mengirimkan data untuk dianalisis.Contoh beberapa kasus nyata adalah:

- institusi keuangan yang memantau perubahan pasar saham. Sehingga bisa mengubah portofolio berdasarkan batasan batasan tertentu(menjual saham ketika saham mencapai titik nilai tertentu)

- Memantau suatu jaringan listrik berdasarkan througput dan akan memberi peringatan jika melebihi batas tertentu.

- Kantor berita yang menggunakan clickstream dari berbagai platform untuk memperkaya data dengan informasi demografik sehingga bisa merekomendasikan artikel ke pembaca yang relevan.

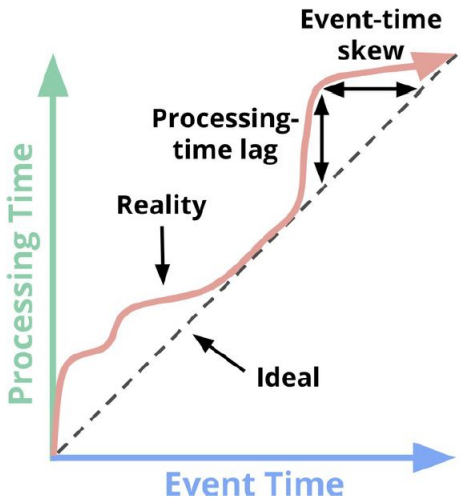
- perusahaan *e-commerce* menggunakan data stream untuk melihat apakah ada anomali pada *web logs* dan akan mengirimkan peringatan jika terjadi anomali.

Big Data Stream adalah gabungan sifat antara *big data* dan *data stream*.Selain memiliki sifat-sifat big data seperti ukurannya yang besar, ukuran pertumbuhannya cepat, dan bervariasi. *Big Data Stream* juga datang secara terus menerus dan tidak terbatas. Selain itu, karena *big data stream* datang secara cepat dan terus menerus nilai (*value*) yang didapatkan akan menurut secara eksponensial seiring bertambahnya waktu. Sehingga, diperlukan pendekatan dan pemrosesan yang bisa langsung mengolah dan memilih informasi dari *Big Data Stream*.

3. **Studi Literatur mengenai teknik *stream processing***
   Status : Ada sejak rencana kerja skripsi.
   Hasil : *Stream Processing* adalah proses yang dirancang untuk mengatasi untaian data yang tidak terbatas atau tidak memiliki awalan dan akhiran yang jelas. *Stream Processing* menawarkan pemrosesan data dengan *latency* yang rendah dan hasil yang spekulatif. Untuk memproses suatu data, *stream processing* mempunyai dua dimensi penting yaitu; *cardinality* dan *constitution*. *cardinality* adalah ukuran dari data tersebut; terbatas (*bounded*) atau tidak terbatas (*unbounded*). *Constitution* adalah bagaimana data ditampilkan; adalah bagaimana data ditampilkan; tabel yang menampilkan data secara menyeluruh seperti SQL atau potongan-potongan file pada HDFS seperti *Map Reduce*.

   Walaupun *Stream Processing* menawarkan *Latency* yang rendah, namun hasil dari data yang diproses kurang akurat. Hal ini disebabkan karena *Stream Processing* menggunakan algoritma pendekatan. Salah satu cara mengatasi masalah ini adalah dengan perancangan arsitektur.

   Stream processing memodelkan waktu menjadi dua domain; *Event time* dan *Processing Time*. *Event Time* adalah waktu saat data sedang dihasilkan. Setiap data yang dihasilkan akan diber *time stamp* agar semua data dari sumber yang sama bisa diurutkan secara kronologis. *Event time* digunakan diantaranya untuk menggolongkan perilaku pengguna dari waktu ke waktu.*Processing time* adalah waktu ketika data diobservasi pada *Stream-Processing*. Secara ideal, *event time* dan *processing time* bernilai sama. Artinya, sistem langsung mengobservasi data saat data itu sedang dihasilkan. Namun pada nyatanya, nilai *processing time* dan *event time* tidak selalu sama karena dipengaruhi sumber data, waktu ekseskusi, dan performa mesin *hardware*. Hubungan antara *event time* dan *processing time* bisa dilihat di Gambar 1:

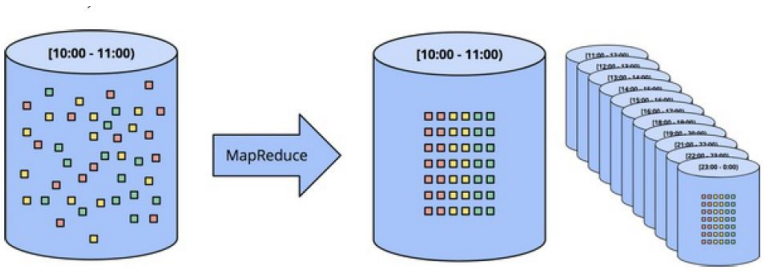


Gambar 1: Pemetaan *Time-Domain*

Sumbu X adalah *event time* atau *completeness* dalam sistem. Sumbu Y adalah *Processing time* waktu jam biasa yang diamatai oleh sistem data processing ketika sedang dieksekusi. *Processing-time lag* adalah jarak vertikal antara garis ideal dengan garis merah yang menunjukan ada berapa banyak waktu delay yang sedang diobservasi di antara kejadian pada waktu tertentu dan kapan delay itu terjadi. *event-time skew* adalah garis horizontal dari garis ideal dengan garis merah yang menunjukan banyaknya distribusi data di *pipeline* pada saat itu dan menunjukan seberapa ketinggalan suatu *pipeline* dari *pipeline* ideal.

4. **Studi literatur mengenai pola teknik pemrosesan *stream processing***
   Status : Ada sejak rencana kerja skripsi.
   Hasil : Pola-pola teknik pemrosesan *stream procesing* dibagi menjadi dua yaitu *batch* dan *streaming*. Pola yang tergolong ke dalam *batch* tidak dirancang untuk data yang tidak terbatas. Tetapi pemrosesan data tidak terbatas bisa dilakukan dengan membagi dataset menjadi beberapa bagian yang berisi potongan-potongan dari dataset tersebut yang lebih kecil dan terbatas. Sehingga bisa diproses dengan *batch processing*. Beberapa jenis *batch processing* adalah *fixed windows* dan *Session*:

   - ***fixed windows*** adalah pemrosesan paling umum. *batch processing* ini bekerja dengan membagi input data ke beberapa *windows* dan memproses *window* tersebut secara terpisah. Proses ini dilakukan secara berulang-ulang. Jika data terkena delay gara-gara partisi *network* perlu dilakukan mitigasi yang memperlambat proses yang memperlambat sampai semua data terkumpul.

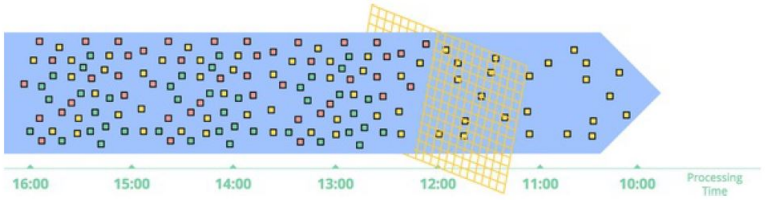


Gambar 2: pemrosesan data tidak terbatas dengan menggunakan *fixed window*

   - *Session* memiliki cara kerja yang hampir sama dengan *fixed windows* bedanya pemotongan data dipisah berdasarkan session sehingga pembagian data jadi tidak seimbang data yang sama mungkin berakhir di *batch* berbeda.

Sedangkan pola-pola yang dikelompokan menjadi *streaming* khusus dibangun untuk memproses data tidak terbatas. Karena pada dunia nyata banyak data yang tidak terstruktur dan tidak sekuensial. Sehingga jika suatu data ingin dianalisis dalam konteks data itu masih baru, harus ada sebuah metode pada *pipeline* untuk mengurutkan data berdasarkan waktu ada 3 pola yang digunakan untuk *stream processing* yaitu; *filtering*, *approximation algorithm* dan *windowing*.

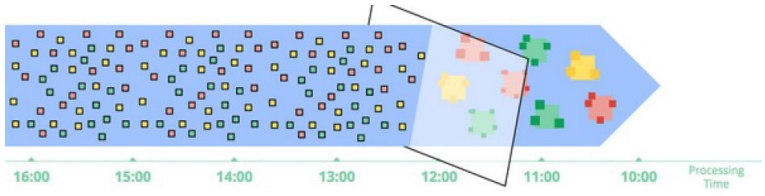
- ***Filtering*** adalah operasi mendasar dan dilakukan secara *Time-Agnostic* yang memilah data yang masuk. Pola *Time-Agonstic* digunakan untuk kasus-kasus dimana waktu tidak relevan. Artinya, semua logika dan informasi pada data yang relevan ada pada data dan yang lebih menentukan relevansi dari suatu data adalah urutan kedatangan data tersebut. Jadi, pola ini hanya membutuhkan streaming engine yang mendukung pengiriman data yang sederhana karena itu semua sistem streaming bisa menggunakan pola Time- AgnosticSistem melihat setiap rekord yang datang dan melihat apakah domain data sama dengan domain tujuan. Bila tidak data akan dibuang karena itu proses ini hanya bergantung pada urutan kedatangan data bukan dari *event-time*.



Gambar 3: *Filtering data*

proses di atas adalah proses *filtering data* dari kumpulan data yang heterogen menjadi homogen dengan tipe yang sama dan diletakan pada klaster yang sama.salah satu cara untuk mengelompokan data yang bersifat sama adalah dengan melakukan *Inner Join.Inner Join* adalah salah satu bagian dari filtering dimana proses menggabungkan dua sumber data yang tidak terbatas. Jika ada suatu data datang sistem akan menyimpan data tersebut ke *persistent state* ketika data berikutnya datang sistem akan menggabungkan data tersebut dengan data yang ada di persistent state.

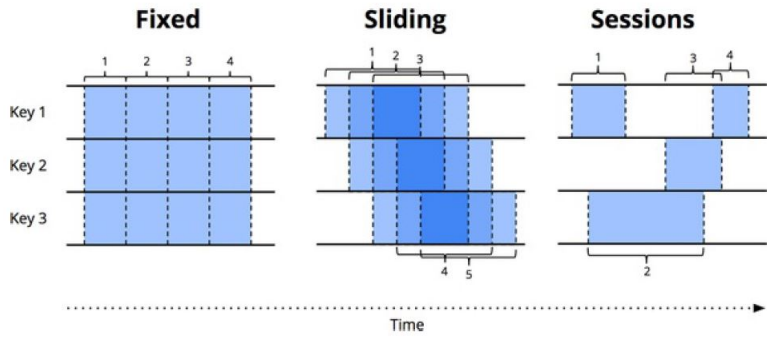
- ***Approximation Algorithm*** adalah algoritma pendekatan yang menerima input data tidak terbatas dan mengelompokan data tersebut menjadi berdekatan jika memiliki sesuatu kesamaan. Konsep cara kerjanya hampir sama dengan *clustering*. Tetapi *approximation algorithm* merupakan algoritma yang rumit sehingga algoritma pendekatan susah untuk dipanggil ketika dibutuhkan dan proses pengelompokan dari algoritma ini cenderung terbatas.



Gambar 4: Algoritma pendekatan pada *unbounded data*

Data dijalankan melewati algoritma yang kompleks dan menghasilkan data output yang terlihat lebih mirip hasil yang diinginkan. Algoritma pendekatan langsung memproses data yang datang karena itu melibatkan *processing-time* digunakan algoritma sebagai pengecek *error* pada data dengan membaca waktu urutan kedatangan dari data.

- ***Windowing*** adalah fungsi yang menerima sumber data sebagai input. Lalu, membagi data tersebut menjadi beberapa bagian dan memberi batasan pada potongan data tersebut bisa dilihat pada gambar 4



Gambar 5: Teknik Windowing

Fixed Windows membagi waktu menjadi segmen-segmen dengan ukuran yang tetap. Seperti pada gambar 4. Segmen untuk fixed window diterapkan secara seragam pada seluruh dataset. Pembagian segmen dengan ukuran yang sama disebut aligned window. Tetapi, terkadang window tidak dibagi dengan ukuran yang sama. Dalam beberapa kasus, pembagian data bergantung pada ukuran dataset dan bervariasi tiap dataset. Pembagian waktu yang tidak merata, unaligned window, membantu meratakan penyebaran waktu penyelesaian.

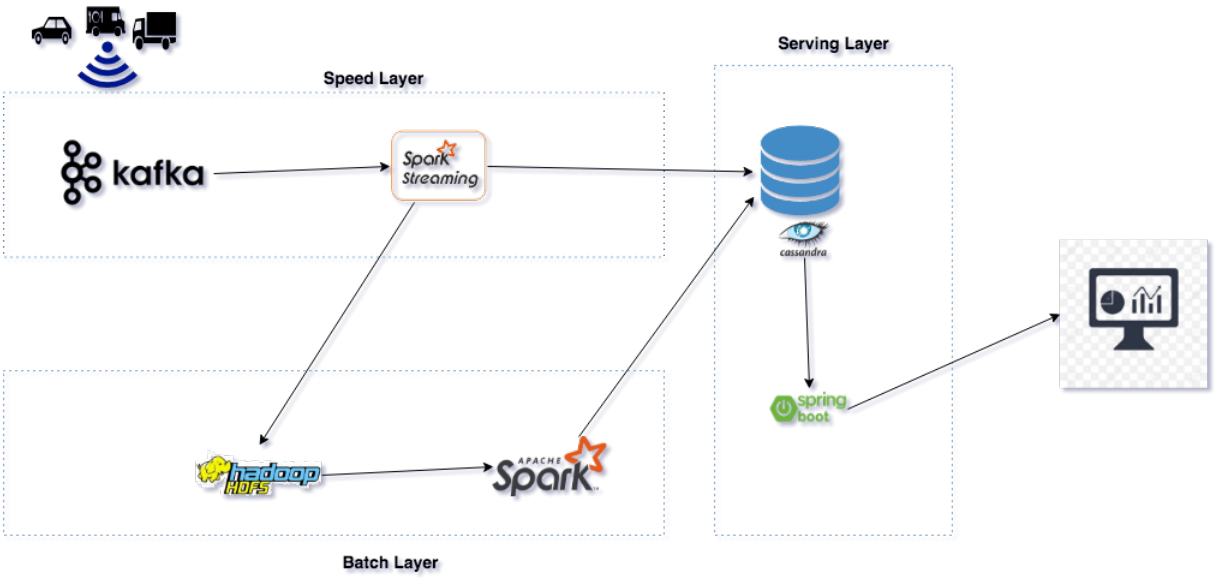
Sliding Windows memiliki panjang dan periode yang tetap. Jika periode lebih kecil dari length maka terjadi overlap pada windows. Jika periode sama dengan waktu maka window akan menjadi fixed window. Jika periode lebih besar dari length maka akan menjadi sampling window yang hanya akan melihat suatu subset data dengan waktu yang lama.

Dynamic Sessions biasanya digunakan untuk menganalisa perilaku pengguna secara berkala dengan megelompokan suatu rantaian peristiwa yang berhubungan. contohnya adalah berapa banyak video yang ditonton oleh pengguna dalam sekali duduk. Panjang dari suatu sesi tidak bisa ditentukan terlebih dahulu. Panjang sesi tergantung dari seberapa banyak data yang terlibat. *Dynamic Session* merupakan salah satu penerapan unaligned window karena untuk setiap session subset pada suatu dataset tidak pernah identik.

Selain itu Windowing bekerja dengan dua cara berbeda. Seperti *Processing-time Windowing* dimana Sistem menyimpan sementara data yang datang pada window untuk beberapa saat sampai processing time telah lewat. Contohnya; misalkan ada fixed windows dengan durasi 5 menit, sistem akan menyimpan sementara data selama lima menit waktu pemrosesan. Lalu, sistem akan mengirim data yang telah diobservasi selama lima menit tersebut ke *downstream* untuk diproses. Proses windowing ini sangat simpel dan implementasinya mudah karena sistem tidak harus mengatur data sesuai waktu. data hanya akan disimpan sementara ketika datang dan langsung dilempar ke downstream ketika processing time selesai. Karena sistem bisa mengetahui semua input karena telah dilihat oleh window. Sehingga sistem bisa dengan baik memprediksi kapan suatu window akan selesai. metode yang kedua adalah *Event-time Windowing* Event-time Windowing digunakan ketika sistem mengobservasi sumber data yang tidak terbatas dalam potongan-potongan data yang terbatas berdasarkan kapan data itu terjadi.

5. **Studi literatur arsitektur *Stream Processing***
   Status : Ada sejak rencana kerja skripsi.
   Hasil : Arsitektur pada *stream processing* dibagi menjadi dua Arsitektur lambda dan arsitektur kappa. **Arsitektur Lambda** adalah teknik pemrosesan data yang bisa menangani data yang besar dengan cara mneggabungkan metode *batch* dan *stream processing*. Teknik ini menyeimbangkan antara *latency, throuhput, dan fault- tolerance* dengan menggunakan batch processing yang menyediakan penyimpanan data yang akurat dan komperhensif. Juga memanfaatkan stream processing agar mendapat data tidak terbatas secara *realtime*.

   Banyak perusahaan yang menggunakan metode stream procesing untuk memprediksi updates dari model dan menyimpan event yang berbeda yang digunakan sebagai bahan untuk memprediksi. Untuk menangani kejadian seperti itu, Arsitektur Lambda memilki tiga layer; *Batch layer*, *speed layer (stream layer)*, dan *Serving layer*.



Gambar 6: Arsitektur *Lambda*

batch layer terlebih dahulu memproses data dengan menggunakan sistem terdistribusi yang bisa menangani data yang besar. tujuan dari batch layer adalah untuk meningkatkan akurasi dengan cara memproses semua data yang ada ketika membangun view. Artinya, batch layer bisa memperbaiki error pada data dengan memproses data kembali berdasarkan dataset yang sudah lengkap.

Setiap data yang terus menerus datang ke sistem akan diteruskan ke batch layer dan stream layer secara bersamaan. Data Stream yang baru masuk ke batch layer langsung diproses pada data lake. Data disimpan pada data lake menggunakan in-memory database atau long term persistent database seperti NoSQL. Data akan diproses menggunakan MapReduce atau machine- learning. Apache hadoop digunakan di layer ini karena memiliki throughput yang paling tinggi.

Speed Layer(Stream Layer) memproses data stream secara real-time tanpa memperdulikan *completeness* atau akurasi dari data. layer ini mengurangi *throughput* untuk mengurangi *latency*. Sehingga data yang terbaru bisa langsung dilihat. *Speed layer* digunakan untuk mengisi jarak yang disebabkan oleh batch layer dengan memberikan informasi tentang data terkini. View yang dihasilkan dari layer ini belum tentu akurat namun bisa langsung dilihat dan diakses. Data yang lebih akurat akan disediakan dan diganti nanti oleh hasil data yang telah diolah oleh batch layer ketika sudah tersedia.

Serving Layer Output dari batch dan Speed layer diteruskan dan disimpan di layer ini dan proses *ad-hoc queries* akan dilakukan di layer ini dengan mengembalikan views dari data yang telah diproses.

Arsitektur Lambda dapat dianggap sebagai arsitektur pemrosesan data yang real-time. Seperti disebutkan di atas, dapat menahan kesalahan serta memungkinkan skalabilitas. Arsitektur ini menggunakan fungsi-fungsi batch dan stream lalu menambahkan data baru ke penyimpanan utama sambil memastikan bahwa data yang ada akan tetap utuh. Perusahaan seperti Twitter, Netflix, dan Yahoo menggunakan arsitektur ini untuk memenuhi kualitas standar layanan.

Keuntungan dari Arsitektur Lambda antara lain adalah; *Batch Layer* dari arsitektur ini mengatur histori data dengan penyimpanan terdistribusi yang *fault tolerant* yang mana akan memperkecil terjadinya error walaupun sistem *crash*, seimbang antara kecepatan dan keandalan, Scalable dan fault tolerant untuk data processing. Tetapi, arsitektur ini memiliki kelemahan yaitu; penerapan yang cukup sulit karena melibatkan *batch* dan *stream processing*, memproses setiap batch pada beberapa *cycle* tidak menguntungkan untuk beberapa skenario, data yang dimodelkan dengan arsitektur ini susah untuk dimigrasi dan diorganisir ulang.

Arsitektur Kappa simplifikasi dari arsitektur lambda. Susunan arsitektur ini hampir sama dengan sistem arsitektur lambda namumn tidak memiliki *batch layer*. Untuk mengganti *batch processing* data langsung diteruskan ke sistem streaming. Arsitektur ini digunakan model data berupa; beberapa *event* atau *query* data dicatat dalam suatu antrian untuk disesuaikan dengan penyimpanan atau riwayat sistem file terdistribusi, urutan *event* dan *query* tidak ditentukan sebelumnya, platform *stream processing* dapat berinteraksi dengan basis data kapan saja. model ini sangat *resilient* dan bisa menangani beberapa terabyte data untuk penyimpanan yang diperlukan untuk setiap sistem node yang mendukung replikasi.

Skenario data yang disebutkan di atas ditangani oleh *Apache kafka* yang cepat, toleran terhadap kesalahan dan dapat diskalakan secara horizontal. Hal ini memungkinkan mekanisme yang lebih baik untuk mengatur aliran data. kerena kontrol yang seimbang pada *stream processing* dan database maka hal ini memungkinkan aplikasi untuk bekerja sesuai ekspektasi.Kafka menyimpan data yang diminta untuk jangka waktu yang lebih lama dan meayani *queries* analog dengan menautkannya ke posisi yang sesuai dari log yang disimpan. Manfaat dari arsitektur ini adalah bisa mempertahankan sejumlah besar data untuk menyelesaikan *queries*

keuntungan dari arsitektur ini adalah dapat digunakan untuk mengembangkan sistem data yang merupakan yang tidak membutuhkan *batch layer*, Pemrosesan ulang hanya diperlukan saat kode berubah dan dapat digunakan dengan memori tetap,Lebih sedikit sumber daya yang diperlukan karena pembelajaran mesin dilakukan secara *real-time*. Tetapi, tidak adanya **batch layer** dapat mengakibatkan kesalahan selama pemrosesan data atau saat memperbarui database yang mengharuskan untuk ada pemrosesan ulang atau rekonsilisasi.

6. **Studi literatur mengenai sistem terdistribusi *Hadoop***
   Status : Baru ditambahkan semester ini.
   Hasil :*Apache Hadoop* merupakan sebuah *framework* yang bersifat *open-source* untuk menulis dan menjalankan aplikasi terdistribusi untuk mengolah data dalam ukuran besar. Proyek Hadoop disusun dengan tujuan untuk mengatasi masalah skalabitas pada *nutch*, sebuah *open-source crawler* dan mesin pencarian. Hadoop merupakan bagian dari implementasi hasil riset google mengenai sistem file terdistribusi dan komputasi pararel.

   Hadoop dapat berjalan di atas klaster mesin-mesin dengan kekuatan pemrosesan yang setara dengan komputer komersial maupun berjalan dengan layanan komputasi *cloud* yang dapat diakses dengan mudah oleh klien. Sistem terdistribusi Hadoop tidak perlu menggunakan mesin-mesin berspesifikasi

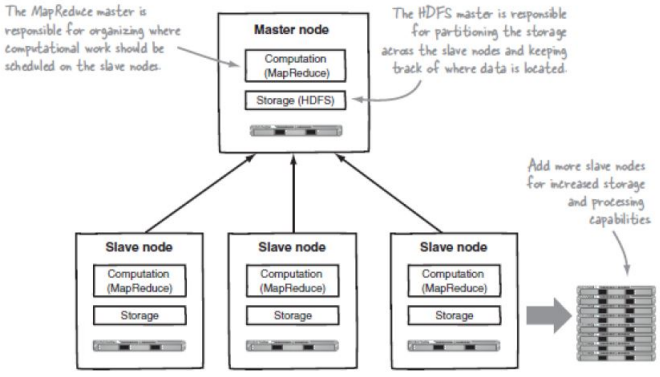
tinggi, karena beban pemrosesan akan didistribusikan ke masing- masing mesin di dalam klaster. Sistem tersebut dapat memudahkan pengguna ketika sistem diperlukan untuk pengolahan data dengan ukuran lebih besar, karena penambahaan ukuran data hanya memerlukan tambahan mesin ke dalam klaster dengan spesifikasi yang sama atau mendekati mesin-mesin lain dalam klaster.

Mesin-mesin dalam klaster dapat menjadi lebih ekonomis dibandingkan dengan menggunakan sebuah mesin dengan sepsifikasi yang lebih baik. Hal ini terkait dengan penambahan ukuran data yang akan diproses serta kerusakan perangkat keras yang mungkin terjadi. Hadoop dapat mengatasi kasus kerusakan tersebut tanpa ada kehilangan data, tetapi penggunaan sebuah mesin saja memiliki resiko kehilangan data saat ada kerusakan. selain itu, penggatian sebuah mesin yang rusak di dalam klaster akan memerlukan biaya yang lebih kecil dibanding menggani mesin dengan spesifikasi tinggi.

Hadoop memerlukan pengolahaan data dengan ukuran yang sangat besar, Sehingga pemindahan data berukuran besar tersebut melalui jaringan akan memperlambat jalannya proses. Oleh karena itu, data tersebut diperkecil dengan membagi data tersebut menjadi blok-blok data dan mendistribusikannya ke masing-masing mesin dalam klaster. Kode aplikasi yang merupakan sebuah pekerjaan di lingkungan Hadoop dan berukuran lebih kecil dibandingkan dengan data akan didistribusikan ke dalam mesin-mesin klaster. Dengan demikian, pekerjaan pemrosesan data akan terjadi di setiap mesin di dalam klaster terhadap blok-blok data yang ada pada masing-masing mesin klaster.

Arsitektur Hadoop

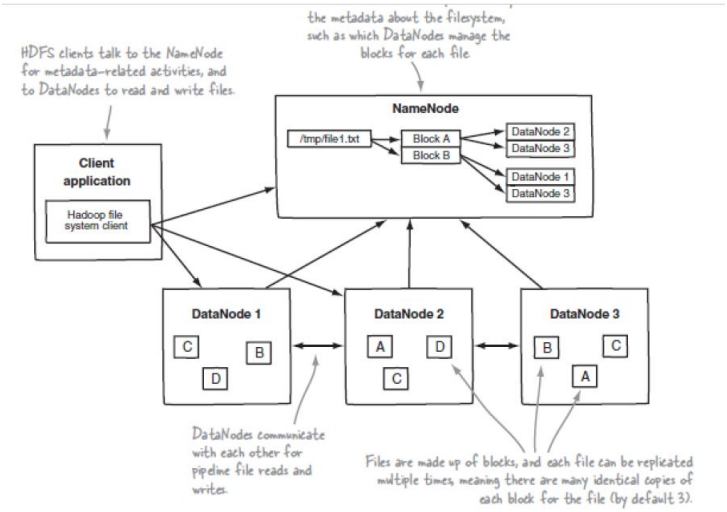
Sebuah klaster hadoop terdiri dari mesin-mesin yang saling terhubung. Klaster hadoop mempunyai arsitektur master/slave dengan sebuah mesin sebagai master node dan mesin-mesin lain sebagai slave node seperti yang ditunjukan pada gambar. Setiap node tersebut akan mempunyai komponen penyimpanan berupa HDFS dan komponen komputasi berupa MapReduce. Seperti yang ditunjukan pada gambar dapat dilihat bahwa HDFS dan MapReduce pada node mempunyai *daemon* dan tugas yang berbeda dengan komponen HDFS dan MapReduce yang ada pada slave node. Perbedaan tugas-tugas tersebut akan dijelaskan dibagaian.



Gambar 7: Arsitektur *Hadoop*

***Hadoop Distributed File System*(HDFS)**

Hadoop distributed file system atau HDFS adalah komponen penyimpanan data dalam Hadoop. HDFS dirancang untuk memiliki *throughput* tinggi dan cocok untuk melakukan operasi baca dan tulis pada file dengan ukuran yang sangat besar. Untuk mendukung hal tersebut, HDFS memanfaatkan ukuran blok data yang besar dan optimasi lokalitas data untuk mengurangi input output jaringan. Selain itu, data yang tersimpan di dalam HDFS tidak akan hilang ketika ada kerusakan pada salah satu mesin. Hal ini disebabkan adanya replikasi untuk setiap blok data yang terdistribusi di dalam klaster. Banyaknya replikasi yang terjadi pada awalnya adalah tiga, tetapi angka ini dapat dikonfigurasi ulang menjadi lebih sedikit maupun lebih banyak.

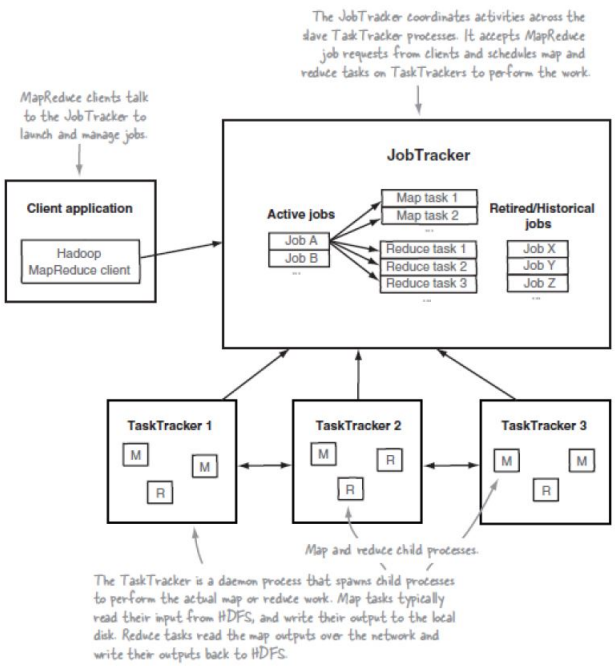


Gambar 8: Arsitektur *HDFS*

Komponen HDFS pada master node menjalankan sebuah daemon yang disebut dengan NameNode. NameNode bertugas untuk mengatur pembagian blok-blok data ke slave node dan mencatat lokasi masing-masing blok data tersebut. NameNode Merupakan komponen yang penting dalam ekseskusi pemrosesan data dalam klaster. Berbeda dengan Master Node, komponen HDFS pada slave node menjalankan daemon yang disebut dengan DataNode. Data Node bertugas untuk melakukan proses baca tulis blok-blok data pada file asli yang terdapat sistem file lokal. Komunikasi pada awal operasi tulis atau baca terjadi diantara klien dan NameNode untuk mendapatkan lokasi blok-blok data yang akan diproses. setelah itu klien dapat berkomunikasi dengan DataNode lain untuk melakukan replikasi blok-blok data yang ada.

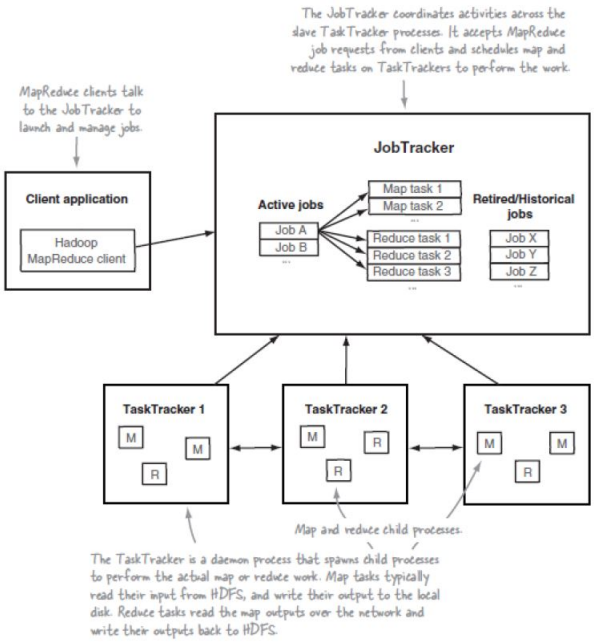
MapReduce

MapReduce merupakan komponen komputasi dalam Hadoop. Model pemrograman yang dimiliki oleh MapReduce memungkinkan seorang programmer mengimplementasikan sebuah aplikasi yang berjalan paralel dengan mudah. Konfigurasi mengenai paralelisasi komputasi distribusi pekerjaan, dan cara mengatasi kegagalan perangkat lunak maupun keras telah ditangani oleh hadoop. Sehingga programmer hanya perlu mengimplementasikan pekerjaan pemrosesan apa saja yang perlu dilakukan.



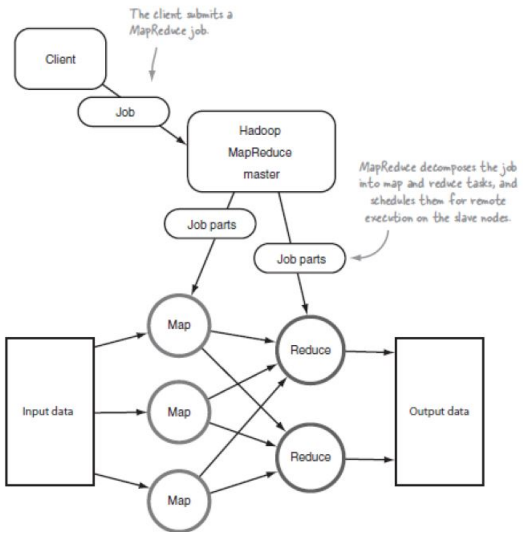
Gambar 9: Arsitektur Map Reduce

komponen MapReduce pada *master node* menjalankan *daemon* JobTracker yang merupakan *daemon* JobTracker. MapReduce merupakan komponen komputasi dalam Hadoop. Model pemrograman yang dimiliki oleh MapReduce memungkinkan seorang programmer mengimplementasikan sebuah aplikasi yang berjalan paralel dengan mudah. Konfigurasi mengenai paralelisasi komputasi, distribusi pekerjaan, dan cara mengatasi kegagalan perangkat lunak maupun keras sudah ditangani oleh Hadoop, sehingga programmer hanya perlu mengimplementasikan pekerjaan pemrosesan apa saja yang perlu dilakukan



Gambar 10: Arsitektur MapReduce

Komponen MapReduce pada master node menjalankan daemon JobTracker yang merupakan daemon yang menghubungkan proses pada Hadoop dengan aplikasi. JobTracker bertugas melakukan pemecahan pekerjaan yang dikirimkan klien menjadi unit-unit pekerjaan map dan reduce. JobTracker akan mendistribusikan unit-unit pekerjaan tersebut ke slave node, melakukan penjadwalan komputasi, dan melakukan pengawasan terhadap pemrosesan yang dilakukan dalam slave node. Komponen MapReduce pada setiap slave node menjalankan daemon TaskTracker yang bertugas untuk melakukan eksekusi pemrosesan di dalam slave node. TaskTracker akan selalu berkomunikasi dengan JobTracker untuk memantau jalannya proses. Jika komunikasi tersebut terputus, dapat diasumsikan proses pada TaskTracker tersebut gagal dan unit pekerjaan yang sesuai akan dikirimkan oleh JobTracker ke slave node lain yang terdapat di dalam klaster. MapReduce terdiri dari komponen-komponen mapper dan reducer. Pekerjaan yang dikirimkan ke dalam klaster akan dipecah menjadi pekerjaan map dan reduce yang berjalan secara paralel. Setiap node dalam map dan reduce dapat berdiri sendiri dan tidak tergantung pada node-node map atau reduce lainnya. Ketergantungan yang ada hanya ketergantungan node-node reduce terhadap node-node map. Pemrosesan ini dilakukan dengan model share-nothing, yaitu data yang diolah tidak dibagikan antarnode untuk mencegah node-node tersebut saling menunggu untuk memakai sumber daya. Implementasi aplikasi dengan MapReduce dapat dilakukan dengan mendefinisikan fungsi map dan reduce. Fungsi map menerima masukan berupa pasangan-pasangan key dan value dan memberikan keluaran berupa list key dan value. Fungsi reduce menerima masukan berupa key dan list value dan memberikan keluaran berupa pasangan key dan value.



Gambar 11: Proses MapReduce

Gambar merupakan gambaran proses yang terjadi ketika klien mengirimkan sebuah pekerjaan MapReduce ke dalam klaster. Pekerjaan yang dikirimkan oleh klien dapat berupa file jar atau xml. Pekerjaan tersebut akan dipecah menjadi unit-unit pekerjaan map dan reduce oleh komponen MapReduce yang terdapat pada master node sebelum didistribusikan ke slave node. Data masukan akan diproses menggunakan komponen mapper pada MapReduce dan format data harus berupa pasangan key dan value. Untuk setiap pasang key dan value tersebut, dilakukan pemrosesan menggunakan fungsi map dan mengembalikan keluaran berupa list pasangan key dan value yang baru. Nilai key pada tahap pemrosesan ini pada umumnya tidak diperhitungkan. Hasil keluaran dari mapper akan diproses terlebih dahulu sebelum dijadikan masukan untuk komponen reducer. Tahap pemrosesan ini disebut sebagai tahap shuffle and sort. Setiap pasangan key dan value akan diurutkan berdasarkan key yang dimiliki dan mengelompokkan semua value yang mempunyai key

yang sama untuk dimasukkan ke node reducer yang sama. Proses ini akan menjadi kompleks ketika semua key yang ada dalam list keluaran mempunyai nilai yang berbeda-beda. Komponen reducer menerima masukan yang berasal dari hasil pada tahap shuffle and sort yang berupa pasangan key dan list value. Untuk setiap nilai key yang ada, fungsi reduce pada reducer akan dipanggil satu kali dan memberikan keluaran berupa pasangan key dan value yang baru. Keluaran dari proses reduce tersebut dapat ditulis ke file yang berada di HDFS maupun ke dalam basis data.

7. **Studi literatur mengenai sistem terdistribusi *Spark***
   Status : Ada sejak rencana kerja skripsi.
   Hasil :*Apache Spark* merupakan *platform* komputasi klaster yang dirancang untuk berjalan dengan cepat ketika mengolah data yang sangat besar dan untuk tujuan penggunaan umum. *Spark* merupakan penerus dari model pemprosesan *MapReduce* pada *Hadoop* dengan jenis komputasi lebih banyak yang dapat dilakukan. Salah satu fitur utama yang ditawarkan *Spark* untuk kecepatan adalah kemampuan untuk menjalankan komputasi di memori. Namun, sistem ini lebih efisien dari *Map Reduce* untuk aplikasi kompleks yang berjalan pada disk karena *Spark* menggunakan DAG(*Directed Acyclic Graph*) *Engine* yang mengoptimasi workflow. DAG bekerja dengan cara menentukan jenis suatu flow yang akan memproses data yang masuk. *Spark* akan mencari cara pengerjaan mana yang paling optimal untuk melakukan pendekatan bagi masalah ini secara keseluruhan. *Spark* juga akan mengoptimasi *workflow* dari pengerjaan tersebut. *Spark* lebih bisa beradaptasi dengan pengerjaan pemrosesan data secara menyeluruh karena itu spark bisa bekerja dengan cepat.

   Spark telah dioptimalkan untuk berjalan pada memori sehingga mempercepat pengolahaan data dibandingkan dengan pendekatan alternatif lain seperti *Map Reduce* pada hadoop yang menulis dan membaca data secara langsung pada *hard drive* komputer pada setiap tahap pemrosesan. Dengan demikian kecepatan pengolahaan data menggunakan spark dapat menjadi lebih cepat dibandingkan dengan Hadoop.

   Spark sering digunakan dalam pemanggilan kueri interaktif pada set data yang besar, pengolahaan data streaming dari sensor maupun sistem finansial, dan tugas-tugas pembelajaran mesin. Selain itu, pengembangan juga dapat menggunakan spark untuk tugas-tuga pemrosesan data lainnya dengan memanfaatkan *library* pengembang dan API serta dukungan untuk bahasa pemrograman java, Python, R, dan Scala. Spark biasa digunakan bersama dengan HDFS(*Hadoop distributed File System*) sebagai pengganti media penyimpanan. *Spark* mempunyai lima buah fitur kunci, yaitu; *easy to use, fast, general purpose, scalable*, dan *falut tolerant*.

   (a) *Easy to Use*: Spark menyediakan lebih dari 80 jenis operator untuk melakukan proses pengolahan data. Sehingga pengolahaan data yang lebih kompleks bisa dilakukan dengan mudah.

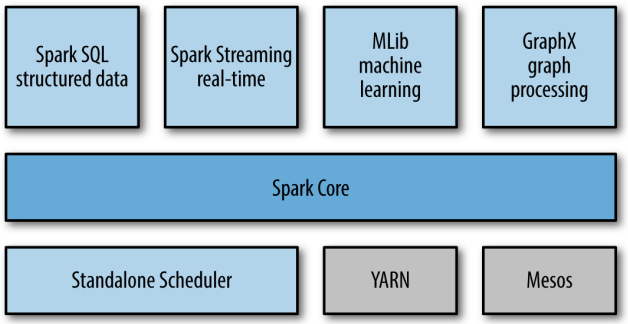
   (b) *Fast*:*Spark* meminimalisir akses data pada disk dengan menyimpan data pada *cache*. Sehingga pengolahaan data menggunakan set data yang sama hanya memerlukan akses pada disk sebanyak satu kali.

   (c) *General Purpose*: *Spark* sudah mempunyai *library* sendiri untuk melakukan *batch processing*, analisis interaktif pada pengolahan *data stream*, pembelajaran mesin dan komputasi graf. Setiap mesin tersebut tidak memerlukan mesin klaster tersendiri untuk melakukan jenis pengolahan tertentu sehingga mengurangi kompleksitas operasional dan menghindari duplikasi kode maupun data.

   (d) *Scalability*: Sama seperti pada Hadoop, peningkatan kapasitas pengolahan data dapat dilakukan dengan menambahkan mesin ke dalam klaster. Selain itu, penambahan mesin pada Spark memengaruhi kode aplikasi yang sudah ada.

(e) *Fault Tolerant*: Kerusakan pada salah satu mesin pada klaster sudah ditangani oleh spark. Tetapi, perlu ada kode untuk menangani hal tersebut walaupun kerusakan tersebut tidak memengaruhi kinerja aplikasi.

Sebuah proyek Spark mempunyai beberapa komponen yang terintegrasi dalam Spark Pada intinya, Spark adalah sebuah mesin komputasi yang bertugas untuk mendistribusikan dan memantau aplikasi yang terdiri dari banyak tugas komputasi yang tersebar ke mesin pekerja atau klaster komputasi. Mesin inti Spark yang dapat berjalan cepat dengan tujuan penggunaan umum memberikan kekuatan tambahan untuk komponen dengan tingkatan yang tingkatan pada susunan yang dikhususkan untuk beban kerja yang beragam. Komponen-komponen ini dirancang untuk beroperasi dengan erat dan dapat digunakan dengan memanggil komponen ini sebagai library di dalam sebuah proyek perangkat lunak.



Gambar 12: *Susunan Spark*

- *Spark Core Spark Core* merupakan fungsi dasar dari *Spark* dan mempunyai komponen-komponen untuk penjadwalan tugas, pengelolaan memori, pemulihan kegagalan, berinteraksi dengan sistem penyimpanan, dan lainnya. *Spark Core* juga mempunyai API untuk mendefinisikan *Resilient Distributed Dataset*(RDD) serta *Spark Context*.
- *Spark SQL* Modul yang bekerja dengan data terstruktur menggunakan Hive yang memungkinkan programmer untuk menggabungkan SQL dengan bahasa pemrogramman spark seperti *python, scala, dan java*.
- *Spark Streaming* API yang menyediakan pemrosesan data secara *real- time*.komponen-komponen dari *Spark Streming* hampir sama dengan *Spark Core*. Seperti pengeloaan memori, pemulihan kegagalan, dan skalabilitas. *Spark Streaming* mempunyai abstraksi dan API berupa *Dstream* dan *Streaming Context*. Spark Streaming akan dibahas lanjut pada 2.4.3
- *Mlib* Sebuah *library* untuk *machine learning*yang menyediakan beberapa tipe algoritma pembelajaran mesin yang dirancang untuk bekerja lintas klaster.
- API untuk pemrosesan *graph* dan komputasi *graph-parallel*.
- *Cluster Manager* Spark Dirancang untuk tetap efisien dalam peningkatan mesin dari satu hingga ribuan mesin. Untuk mencapai efisiensi tersebut sekaligus memaksimalkan fleksibilitas. Spark dapat menjalankan *cluster manager* termasuk Hadoop dan YARN, Apache Mesos, dan *Cluster Manager* yang sudah termasuk dalam spark yaitu Standalone Scheduler.

Application Programming Interface (API) Spark adalah Kemampuan komputasi pada aplikasi spark ada dalam bentuk *library*. *library* tersebut ditulis dalam bahasa scala. Tetapi, menyediakan *Applicataion Programming Interface* atau API dalam berbagai bahasa. Spark API mempunyai dua abstraksi penting, yaitu *Spark Context* dan *Resilient Distributed Datasets*(RDD). Kedua Abstraksi

ini memungkinkan sebuah aplikasi untuk berinteraksi dengan Spark, terhubung dengan klaster, dan menggunakan sumber daya dalam *Cluster*.

Spark Context

Spark Context Merupakan sebuah kelas yang didefinisikan dalam *library Spark*. Spark Context merepresentasikan sebuah koneksi ke cluster spark dan diperlukan untuk membuat objek-objek lain yang disediakan oleh *Spark* API. Sebuah aplikasi harus mempunyai objek *Spark Context* yang aktif. Objek *Spark Context* tersebut harus mempunyai konfigurasi untuk alamat spark master dan nama aplikasi. Spark Master merupakan cara SparkContext terkoneksi dengan klaster. Penggunaan kata kunci lokal menjalankan Spark dengan menggunakan sebuah thread pada satu mesin saja. Penggunaan thread tersebut dapat dikonfigurasi menjadi local[n] untuk n buah thread atau local[$*$] untuk menggunakan thread sejumlah core.

Resilient Distributed Dataset(RDD)

RDD adalah abstraksi dasar untuk merepresentasikan kumpulan objek yang dapat didistribusikan pada mesin-mesin dalam sebuah klaster. RDD dapat dibuat dengan menggunakan data yang bersumber dari luar seperti file dalam HDFS, tabel basis data, atau kumpulan objek local dan hasil transformasi yang dilakukan pada RDD yang sudah ada. Pembuatan RDD dengan data dari sumber luar memerlukan objek *spark context*. karakteristik RDD.

(a) *Immutable*
RDD merupakan sebuah struktur data yang permanen. RDD yang dibuat tidak dapat dimodifikasi lebih lanjut. Sehingga operasi yang mengubah RDD akan mengembalikan RDD yang baru.

(b) *Partitioned*
Data yang direpresentasikan oleh RDD terbagi menjadi partisi-partisi yang didistribusikan pada klaster mesin-mesin. Akan tetapi, partisi-partisi tersebut akan berada pada sebuah mesin yang sama jika *Spark* hanya berjalan pada satu mesin saja. Di antara partisi RDD dengan partisi fisik set data terdapat pemetaan. Jenis Pemetaan tergantung pada sumber data seperti, blok-blok data HDFS mempunyai pemetaan satu ke satu dengan partisi-partisi RDD dan berapa partisi-partisi RDD dan beberapa partisi *cassandra* dipetakan ke satu buah partisi RDD.

(c) *Fault Tolerant*
RDD mengatasi kegagalan dari mesin klaster secara otomatis. Partisi RDD yang hilang pada mesin yang rusak tersebut akan dibuat ulang pada mesin lain. Hal ini dapat dilakukan karena spark menyimpan informasi keterhubungan antara RDD dan informasi tersebut digunakan untuk memulihkan bagian atau keseluruhan RDD yang hilang.

(d) *Interface*
RDD merupakan sebuah antar muka untuk pemrosesan data yang didefinisikan sebagai kelas abstrak dalam *library Spark*. RDD menyediakan antarmuka yang seragam untuk pemrosesan data yang berasal dari berbagai sumber. Selain itu RDD juga menyediakan kelas- kelas implementasi konkret untuk sumber data yang berbeda.

(e) *Strongly typed*
Definisi kelas RDD mempunyai parameter tipe yang memungkinkan RDD untuk merepresentasikan data dengan tipe berbeda. RDD merupakan kumpulan elemen homogen yang terdistribusi dan elemen-elemen tersebut dapat bertipe *Integer, Long, Float, String*, atau tipe yang didefinisikan oleh pengembang aplikasi.

(f) *In Memory*
Kelas RDD menyediakan API untuk komputasi klaster dalam memori. *Spark* memungkinkan RDD untuk di-cache atau dipertahankan dalam memori. Operasi pada RDD yang ada dalam

cache berjalan lebih cepat dibandingkan dengan operasi pada RDD yang tidak berada dalam cache.

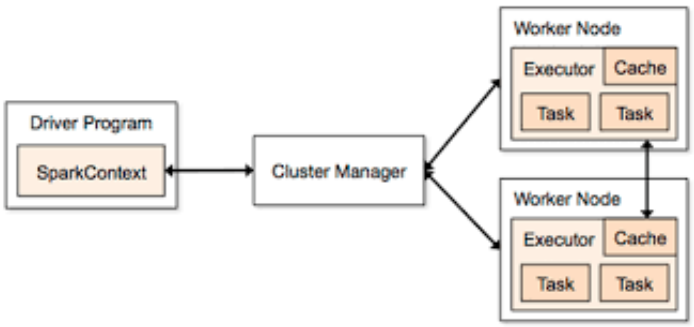
Data yang sudah direpresentasikan dalam RDD dapat dioperasikan dengan menggunakan dua jenis operasi dasar, yaitu transformasi dan aksi.

(a) *transformasi*
RDD yang menjadi masukan dari sebuah operasi transformasi akan mengalami perubahan struktur atau nilai yang ada pada RDD. karena RDD bersifat *immutable*, maka perubahan dari operasi ini secara konseptual akan dikembalikan dalam bentuk RDD baru.

(b) *Aksi*
Operasi menjadi titik awal dari komputasi-komputasi yang telah terjadi pada RDD masukan. Pemanggilan operasi aksi akan memulai pembentukan RDD yang diperlakukan untuk komputasi. Operasi ini menerima masukan berupa RDD dan meberikan hasil berupa sebuah nilai.

Operasi yang dilakukan pada RDD bersifat *lazy* yang berarti operasi tersebut tidak akan dieksekusi oleh spark sampai ada operasi aksi. Evaluasi *lazy* berarti ketika ada pemanggilan operasi transformasi pada RDD. Operasi tersebut tidak langsung dilakukan spark mencatat metadata untuk menandakan bahwa operasi tersebut sudah pernah diminta. Dengan demikian, RDD dapat disebut sebagai kumpulan instruksi untuk melakukan komputasi data yang dibuat dari kumpulan operasi transformasi.

8. **Studi Literatur Spark: Arsitektur Spark**
Status : Ada sejak rencana kerja skripsi
Hasil :



Gambar 13: Arsitektur *Spark*

Berdasarkan pada gambar sebuah aplikasi melibatkan lima entitas penting yaitu; driver program, cluster manager, worker, executor, dan task.

(a) *Driver Program*
Driver Program merupakan bagian dari aplikasi yang memulai menjalankan proses pengolahan data yang akan dilakukan. Driver program terhubung dengan komponen-komponen lain melalui objek *Spark Context* yang terdapat dalam komponen ini. Objek *Spark Context* akan melakukan koneksi dengan *cluster manager*. Setiap klaster hanya akan mempunyai satu buah driver program atau satu buah objek spark context.

(b) *Cluster Manager*
Cluster Manager berfungsi untuk mengelola sumber daya yang digunakan untuk proses pengolahan. Spark Context pada komponen driver program dapat terhubung dengan salah satu dari jenis-jenis cluster manager yang didukung oleh Spark, yaitu *cluster manager* seperti *Apache mesos* dan *Yarn*. Setiap Cluster hanya memiliki satu buah *cluster manager*.

(c) *Worker*
Worker merupakan komponen yang menyediakan unit pemrosesan dan alokasi memori untuk menyimpan sumber daya yang digunakan dalam proses yang berjalan. Setiap klaster dapat memiliki lebih dari satu buah worker dan setiap worker tersebut akan menjalankan aplikasi sebagai proses yang terdistribusi pada sebuah klaster.

(d) *Executor*
Executor merupakan proses yang dibuat oleh *spark* pada setiap node worker untuk menjalankan aplikasi. *Executor* yang terdapat pada setiap worker hanya dapat menangani proses untuk sebuah aplikasi saja, sehingga aplikasi yang berbeda akan mempunyai eksekutor yang berbeda. Setiap eksekutor mempunyai lama hidup yang sama dengan aplikasi sehingga akan berhenti ketiak aplikasi berhenti berjalan. Setiap Klaster dapat mempunyai lebih dari satu eksekutor dan jumlah tergantung pada banyak aplikasi.

(e) *Task*
Task merupakan unit pekerjaan terkecil yang dikirimkan ke executor. Unit pekerjaan tersebut akan dijalankan pada sejumlah *thread* yang terdapat pada executor. Banyak *Thread* yang digunakan berbanding lurus dengan banyak partisi data yang diolah.

9. **Studi literatur mengenai *Spark Streaming***
Status : Ada sejak rencana kerja skripsi.
Hasil :*Spark Streaming* adalah ekstensi dari API *Spark Core* yang menyediakan pemprosesan *Data Stream* yang bisa ditingkatkan performanya dengan menambah *hardware baru*, bisa memproses data dengan banyak dan cepat, dan sistem masih bisa beroperasi ketika terjadi kegagalan. Data bisa dikumpulkan dari berbagai sumber seperti *Kafka, Flume, Kinesis, atau TCP Socket.* Data yang terkumpul akan langsung diproses dengan algoritma yang kompleks seperti *Map, Reduce, Join* dan *Windowing.* Terakhir data yang telah diproses langsung dikirim ke *File Systems, database* dan *live dashboard.* Penjelasan lebih jelas ada pada gambar 2.12 berikut:

Gambar 14: Arsitektur *Spark Streaming*

Arsitektur *Spark Streaming* *Spark Streaming* bekerja dengan cara menerima input *data streams*secara langsung dan membagai data tersebut menjadi beberapa potongan-potongan(*batches*), yang nanti akan diproses oleh mesin *Spark* untuk menghasilkan *stream* akhir pada *batches. Spark Streaming* tidak memproses data secara periodik, hanya memproses yang duluan datang dan memutakhirkan hasil dari *Spark Streaming* dari waktu ke waktu. Data langsung bisa dianalisis ketika datang dengan mengelompokannya ke beberapa bagian kecil dan langsung melakukan agregasi pada potongan data tersebut.

Gambar 15: Arsitektur *Spark Streaming*

Berdasarkan gambar 2.13 *Data Streams* yang masuk akan diterima oleh *reciever* lalu potongan-potongan data yang masuk pada selang waktu tertentu akan dihasilkan secara terus menerus dengan kata lain proses transformasi akan terus berlangsung ketika program dihentikan.Lalu, data yang telah dihasilkan dapat dikirimkan langsung ke *external database* dengan data yang telah diagregasi sebelumnya.

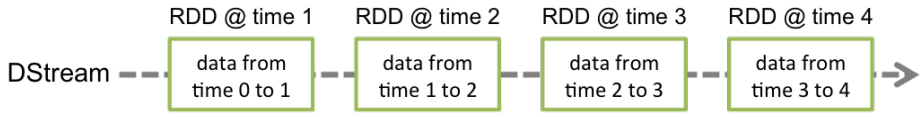
transformasi dan aksi pada RDD bisa terjadi secara pararel pada node *worker*. Artinya, proses RDD dibagi menjadi potongan kecil dan didistribusikan, potongan yang berbeda akan dikirim ke node yang berbeda. Potongan RDD tersebut akan didistribusikan ke seluruh klaster.

Aplikasi *Spark Streaming* membutuhkan pengaturan tambahan untuk beroperasi tanpa henti. Aturan yang dimaksud adalah *checkpointing* yang merupakan mekanisme utama pada *Spark Streaming*. *Checkpointing* memungkinkan penyimpanan data pada *file system* seperti HDFS dan yang membuat *Spark Streaming* menjadi *fault tolerant*.

Abstraksi *Spark Streaming*

Abstraksi dasar yang disediakan oleh *Spark Streaming* disebut *Discretized Streams* (*DStreams*). *Dstream* merupakan seluruh alur data yang datang dari *recievers* Setiap Dstream dibuat pada potongan-potongan RDD yang merepresentasikan aliran data yang kontinu. Dstream memiliki dua buah operasi; transformasi dan *output operation*. Transformasi bertugas untuk menghasilkan Dstream baru dan *Output operation* bertugas untuk menuliskan data ke sistem eksternal. *Dstream* menyediakan operasi yang hampir sama dengan operasi RDD dan mempunyai operasi sendiri yang digunakan untuk mengatur waktu seperti *sliding window*.

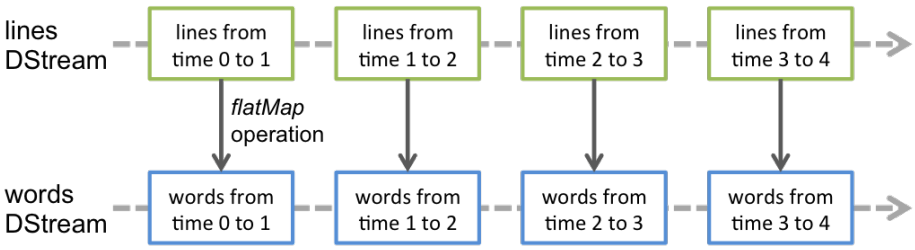
setiap RDD yang ada pada Dstream mempunyai data dari interval tertentu yang akan ditunjukan pada gambar 2.14



Gambar 16: Alur *Dstream*

Pada setiap awal interval, *batch* baru selalu dibuat dan setiap data yang muncul pada interval tersebut akan dimasukan ke *batch* tersebut. Saat interval berakhir *batch* telah selesai berkembang. Ukuran dari suatu interval ditentukan oleh sebuah parameter yang disebut *batch interval*. Biasanya, ukurandari interval berkisar antara 500 milidetik sampai beberapa detik. Setiap input yang ada di dalam batch membentuk RDD dan diproses menggunakan *Spark Jobs* untuk membuat RDD lain. RDD akan terus dibuat dan ditransformasi terus menerus sampai ada aksi yang memberhentikan *Dstream*. Hasil dari transformasi *Dstream* akan langsung dikirimkan ke sistem eksternal dalam bentuk *batch*.

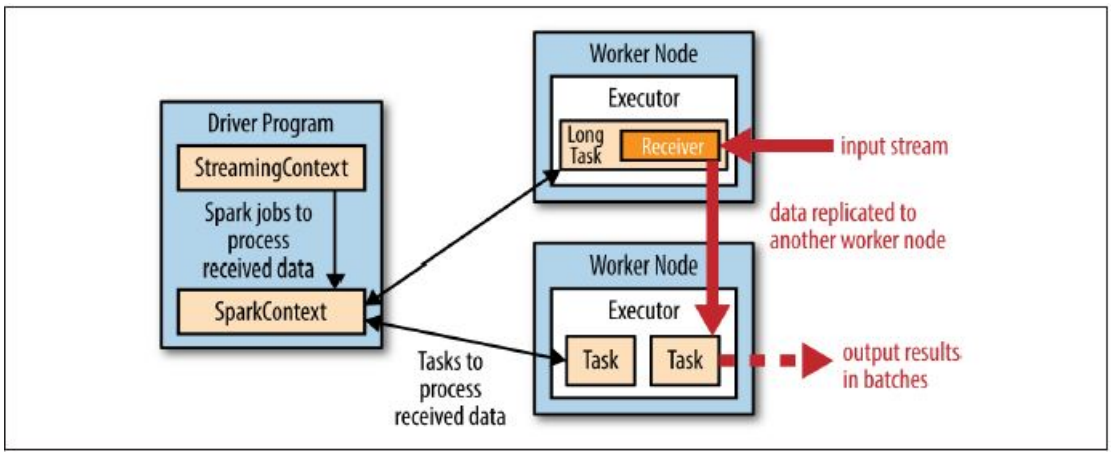
Dstream merupakan salah satu abstraksi yang paling memudahkan pada spark streaming karena transformasi langsung diterapkan ke *Dstream* bukan masing-masing RDD. Jadi, jika melakukan transformasi pada Dstream seluruh potongan RDD akan ikut bertransformasi. Namun, masing-masing RDD masih bisa diakses melalui *Dstream*.



Gambar 17: mengubah*Data Stream* dari lines ke words

DStream dapat dibuat dari sumber eksternal ataupun menggunakan hasil transformasi dari *Dstream* lain. *DStream* juga memiliki *Stateful transformations* yang bisa mengagregasi data pada seluruh interval yang ada. pembahasan tentang *stateful transformation* akan dibahas lebih jelas di bab berikutnya.

Untuk setiap sumber input, *spark streaming* meluncurkan *recievers* yang mana adalah *task* yang berjalan pada eksekutor yang mengumpulkan data dari sumber input dan menyimpannya sebagai RDD. Selain menyimpan data, *recievers* juga mereplikasi data ke eksekutor lain untuk mencapai *fault-tolerance*. Data akan disimpan di memori eksekutor sama seperi *cache* pada RDD. *Receivers* juga bisa mereplikasi data ke HDFS. *Streaming Context* pada *driver* than secara periodik menjalankan *Spark Jobs* untuk memproses data dan menggabungkannya dengan RDD sebelumnya.



Gambar 18: eksekusi*Spark Streaming* pada komponen *Spark*

Spark Streaming memiliki sifat *fault-tolerant* yang sama dengan *Spark* untuk RDD selama replika input data masih tersedia. *Spark Streaming* bisa mengkomputasi ulang setiap *state* yang diturunkan dari *lineage*suatu RDD dengan cara menjalankan kembali operasi yang memproses RDD tersebut. Biasanya, data yang diterima direplika dalam dua node sehingga *spark streaming* bisa mentoleransi satu worker yang gagal. Namun, jika menggunakan *lineage* penghitungan ulang bisa memerlukan waktu yang lama karena datanya telah dibuat duluan. Karena itu, *Spark Streaming* menyediakan mekanisme yang disebut *checkpointing* yang akan menyimpan state secara berkala ke suatu file system seperti HDFS. *checkpointing* akan dijalankan setiap lima atau sepuluh *batch*. Ketika terjadi ingin memperbaiki data

yang gagal *Spark Streaming* hanya perlu kembali ke *checkpoint* paling baru.

Transformasi

Transformasi pada *Spark Streaming* dikelompokan menjadi dua yaitu; *stateless* atau *Stateful*:

- Pada transformasi *stateless*, pemrosesan setiap batch tidak bergantung pada data di batch sebelumnya. Transformasi ini memiliki transformasi RDD seperti `map()`,`reduce()`, dan `reduceByKey()`
- Transformasi *stateful* menggunakan data yang dihasilkan oleh *batch* sebelumnya untuk menghitung hasil dari *batch* saat ini. Transformasi ini memiliki *sliding windows* dan bisa mengecek waktu pada seluruh interval.

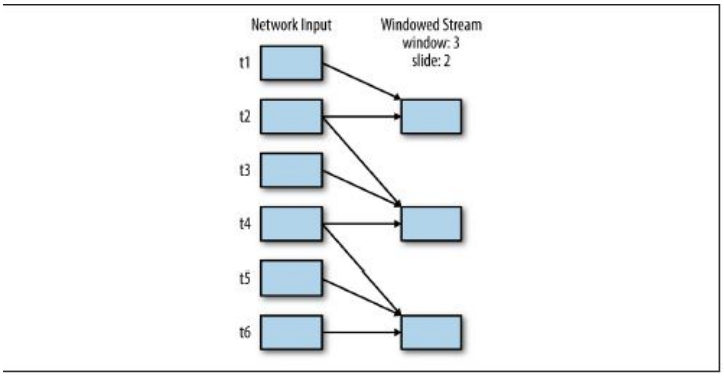
Walaupun setiap fungsi terlihat diterapkan ke pada seluruh aliran data. Namun, secara internal setiap *DStream* tersusun dari beberapa RDD (*batches*) dan setiap transformasi *stateless* diterapkan secara terpisah untuk setiap RDD. Contohnya, `reduceByKey()` akan melakukan *reduce* pada data pada setiap *batch interval*. Untuk menggabungkan data pada seluruh interval diperlukan *Stateful Transformation*

Stateful Transformation

Stateful Transformation adalah sebuah operasi pada *Dstream* yang bisa menelusuri waktu pada semua interval sehingga data pada *batch* sebelumnya bisa digunakan untuk *batch* saat ini.
Spark Streaming memerlukan *checkpointing* untuk bisa diaktifkan di *streaming context* sebagai upaya menghindari kesalahan *fault*.

Windowed Transformation

Transformasi ini menghitung hasil pada interval yang lebih lama dari *streaming context* dengan menggabungkan beberapa *batch* pada interval tertentu. pada transformasi *windowing* ada tiga interval yang digunakan: *batch, slide*, dan *window interval*. *Batch Interval* adalah seberapa sering suatu data diambil ke dalam *Dstream*. Durasi dari batch interval sangat sebentar setengah sampai satu detik. *Batch time* tidak berkorelasi dengan apa yang akan dianalisis nanti. *Batch Interval* hanya mengambil data sebanyak dan secepat mungkin dari suatu sumber data.



Gambar 19: Gambar cara kerja *Windowed Transformation*

Slide Interval adalah titik acuan seberapa sering suatu informasi ingin dikomputasi, dan *Window Interval* adalah bagaimana *Spark Streaming* melihat ke belekang setiap kali bertemu dengan *slide Interval*.

10. **Studi literatur mengenai Twitter API**
    Status : Baru ditambahkan pada semester ini
    Hasil : Twitter API adalah sekumpulan URL yang digunakan untuk mengakses data pada twitter tanpa melewati antar muka web. URL akan digunakan sebagai parameter kode program nanti. data yang diambil pada twitter berupa objek. seperti contoh gambar:

Gambar 20: *Twitter Object*

objek yang diakses akan disimpan dengan format JSON. Objek terdiri dari informasi-informasi yang membangun tweet seperti; tanggal berapa suatu tweet dibuat, id twitter yang menunggah tweet tersebut, isi pesan yang diunggah oleh pengguna(status), informasi pengguna itu sendiri, dan entitas luar yang ikut di dalam suatu tweet seperti link url atau mention. Berikut adalah contoh format JSON yang membangun suatu Tweet:


```
{
 "created_at": "Wed Oct 10 20:19:24 +0000 2018",
 "id": 1050118621198921728,
 "id_str": "1050118621198921728",
 "text": "To make room for more expression, we will
tps://t.co/MkGjXf9aXm",
 "user": {},
 "entities": {}
}
```


Gambar 21: *Twitter Object JSON*

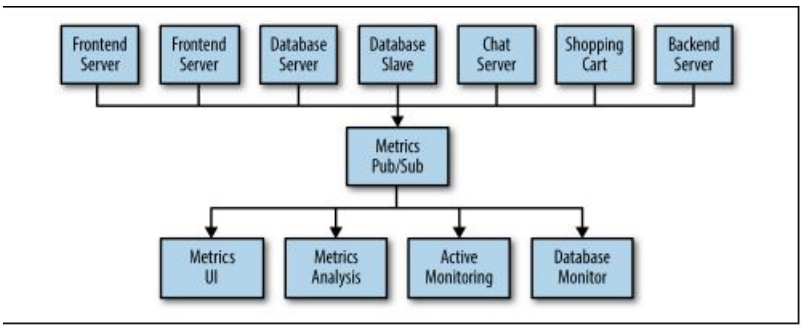
Informasi tentang pengguna terdiri dari beberapa objek lagi.Objek yang dimuat berupa id user, nama, lokasi, url, deskripsi, status verifikasi, jumlah follower, jumlah following, dan lokasi.

```
{ "user": {
    "id": 6253282,
    "id_str": "6253282",
    "name": "Twitter API",
    "screen_name": "TwitterAPI",
    "location": "San Francisco, CA",
    "url": "https://developer.twitter.com",
    "description": "The Real Twitter API. Tweets about API cl
e issues and our Developer Platform. Don't get an answer? It
te.",
    "verified": true,
    "followers_count": 6129794,
    "friends_count": 12,
    "listed_count": 12899,
    "favourites_count": 31,
    "statuses_count": 3658,
    "created_at": "Wed May 23 06:01:13 +0000 2007",
```


Gambar 22: *User Object JSON*

Namun dari beberapa banyak informasi yang terdapat pada twitter seseorang tidak semuanya bisa diakses hal ini bergantung ke pada kebijakan dari twitter dan persetujuan dari user. salah satu contohnya adalah lokasi seseorang. Lokasi seseorang hanya bisa diakses ketika user bersedia untuk menampilkan lokasi tersebut.

11. **Studi literatur mengenai Kafka**
    Status : Baru ditambahkan pada semester ini
    Hasil : Sebelum membahas Kafka, penting untuk mengerti tentang sistem pengiriman *publish/messaging* dan mengapa konsep ini sangat penting. *publish/subscribe messaging* adalah *pattern* yang dicirikan oleh pengirim *publisiher* tidak langsung mengirimkannya ke penerima. Tetapi, pengirim mempublikasikan data yang dimiliki ke sebuah sistem lain yang disebut *broker*, sebuah titik sentral di mana suatu pesan selalu diunggah. Sehingga penerima pesan bisa langsung mengikuti perkembangan data dan informasi yang dimiliki *publisher* di broker. Penerima disebu *Subscriber*.



Gambar 23: *Publisher/Subscriber*

seperti gambar di atas semua server mengirimkan data ke broker *metrics* dan sistem- sistem yang ingin memilki data yang ingin diakses harus diintegrasikan dengan broker. Sistem-sistem tersebut mengikuti perkembangan dan perubahan data yang terjadi melalui broke (subscribe).

Pengertian Kafka

kafka adalah sebuah sistem pengriman data *Publish/subscribe* yang didesain untuk menyelesaikan masalah yang sering disebut dengan *distributed commit log* yang mana suatu *filesystem* atau database

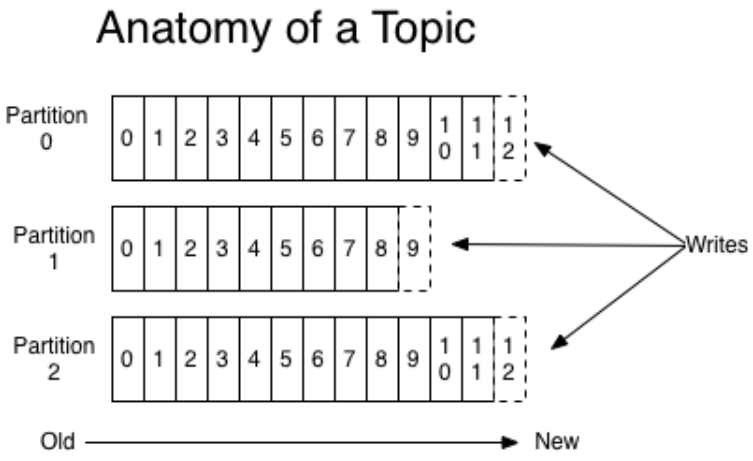
didesain untuk menyediakan data rekord-rekord yang disimpan dengan lama dan bisa diakses kembali secara berkala pada sistem yang stabil. Data pada kafka disimpan dengan lama dan terurut. Berikut dalah komponen-komponen dari kafka:

Messages and Batches
Satuan data di kafka disebut dengan *message*. *Message* hampir sama seperti *row* atau *rekord*. Sebuah *message* adalah array dari sekumpulan bytes karena itu *message* pada kafka tidak memiliki format spesifik atau arti bagi kafka. Suatu *message* bisa memiliki metadata yang disebut dengan *key*. Untuk lebih efisien semua *messages* ditulis pada *batch*. *batch* adalah sekumpulan pesan yang diproduksi oleh topik dan partisi yang sama.

Schemas
Kakfa mengetahui *message* adalah sebuah *array of bytes*. Sehingga kafka membutuhkan skema atau struktur yang diterapkan pada pesan sehinggga bisa mudah dimengerti. Ada banyak cara untuk menerapkan skema tergantung kebutuhan aplikasi. Contoh dari skema bisa berbentuk JSON atau XML sehingga manusia bisa membaca pesan yang ada pada kafka.

Topics
Message pada kafka disebut sebagai *topics*. *Topics* adalah sebuah kategori atau sebuah nama *feed* dimana suatu rekord dipublikasi. Analoginya, topik adalah tabel basis data atau suatu folder di filesystem. Suatu rekord bisa disimpan di folder atau basis data dengan identifikasi pengenal.

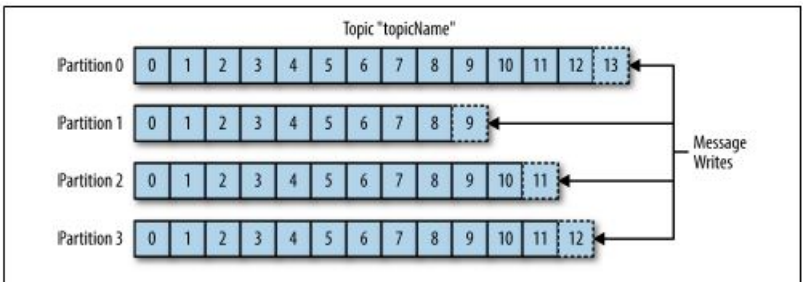
Message pada kafka disebut sebagai *topics*. *Topics* adalah sebuah kategori atau sebuah nama *feed* dimana suatu rekord dipublikasi. Analoginya, topik adalah tabel basis data atau suatu folder di filesystem. Suatu rekord bisa disimpan di folder atau basis data dengan identifikasi pengenal.



Gambar 24: *Topic pada kafka*

Setiap partisi adalah sebuah sekuensi yang terurut, tidak bisa diubah-ubah *immutable* yang terus ada pada *commit log*. Setiap rekord pada p *partition* diberi tanda dengan nomer sekuensial yang disebut offset yang menandai rekord secara unik. Partition adalah tempat dimana topic disimpan.

Klaster pada kafka akan terus menyimpan topic terlepas topik itu digunakan atau tidak selama waktu yang telah ditentukan *(retention period)*. Contoh jika suatu *retention period* pada topic diatur menjadi 2 hari maka topic tersebut akan ada selama dua hari dan tidak bisa dihapus pada interval waktu itu. Sehingga setiap *subscriber* masih bisa mengakses data tersebut. Setelah itu baru dihapus.



Gambar 25: *stream topic*

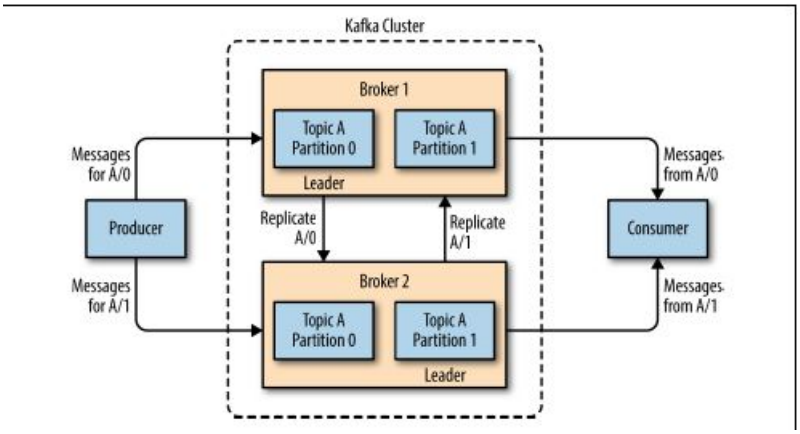
Begitu juga dengan *data stream*. Pada kafka, *Data Stream* dianggap sebagai satu topic terlepas dari banyak partisi. Sebagai contoh keempat partisi pada gambar di atas masih disebut sebagai satu stream.

Producers and Consumers

Pengguna yang menggunakan sistem kafka disebut dengan klien. Ada dua tipe kline *producer* dan *consumer*. Bisa juga kakfa diintegrasikan dengan sistem API lain. Tugas dari *producer* adalah membuat *membuat message* dan mengirimnya ke *topic* tertentu pada broker. *Consumer* adalah yang membaca pesan degan cara mengakses topic-topic tertentu pada broker.

Broker

Sebuah kafka server disebut dengan broker. Sebuah broker menerima pesan dari *producer* memberi *offset* pada pesan tersebut dan menyimpannya pada sebuah disk. Broker juga berinteraksi dengan *consumer* ketika konsumer meminta akses pada suatu partisi dan membalas *consumer* dengan pesan yang diinginkan dan ada di disk.



Gambar 26: Kafka Broker

Broker dirancang untuk berjalan pada klaster. Dari rangkaian *broker* pada klaster satu *broker* akan bertindak sebagai *controller* yang dipilih secara otomatis dan berfungsi untuk mengatur operasi administratif seperti; menentukan partisi mana topic akan disimpan atau mengawasi jika terjadi *failure* pada broker lain. Sebuah partisi bisa disimpan pada dua broker secara bersamaan.

12. **Konfigurasi TCP Socket**

    Status : Sudah ada sejak perencanaan skripsi

    Hasil : Sumber data TCP Socket tidak perlu diinstalasi terlebih dahulu karena tidak ada interverensi dari pihak ketiga yaitu sang penyedia data. TCP socket langsung bisa menerimas data dengan mengakses port lokal dan akan langsung terintegrasi dengan IP address dan port yang kita miliki. Mengkonfigurasikan TCP Socket hanya perlu menyediakan sebuah port kosong yang nantinya akan digunakan data untuk masuk. Tetapi, konfigurasi pada kode program masih diperlukan.

13. **Mengintegrasikan *spark streaming* dengan twitter API**
Status : Ada sejak rencana kerja skripsi.
Hasil : Sebelum mendapatkan hak untuk mengakses Twitter API, user perlu melakukan pendaftaran ke pihak twitter untuk menjadi *developer account* terlebih dahulu. Akan muncul survei yang menanyakan tentang privasi data pengguna twitter.Pertanyaan paling umum adalah data yang didapatkan akan digunakan untuk apa. Untuk kasus skripsi ini, data akan digunakan untuk mempelajari Spark Streaming. Berikut adalah cara membuat API Setelah dikonfirmasi menjadi *developer account*:

    (a) Membuat Aplikasi yang akan digunakan sebagai API.

    (b) Mengisi keterangan dan informasi tentang aplikasi tersebut

    (c) Menggunakan keys dan tokens yang didapatkan yang akan digunakan untuk integrasi dengan Spark Streaming

    keys dan tokens yang didapatkan adalah beberapa nomer seri yang disbut *Consumer API* dan *Access token*. Nomor seri ini yang nantinya akan jadi parameter bagi kode program untuk mengakses Aplikasi yang telah kita buat. Nomor seri ini bersifat rahasia jadi tidak boleh tersebar ke pihak lain karena seseorang bisa mengakses Aplikasi pengumpul data yang telah kita buat. Berikut contoh gambar dari key dan tokens:

Gambar 27: Gambar Token yang digunakan sebagai parameter

Parameter keys dan token nantinya akan dimuat dalam `twitterAuth.txt`. File tersebut akan dijadikan sebagai otentikasi untuk mengakses data yang ada di Twitter.

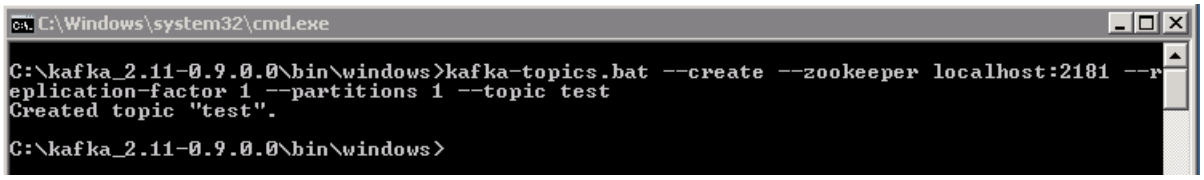
14. **Integrasi dengan Kafka**
Status : Sudah ada sejak perencanaan skripsi
Hasil :kafka adalah sebuah sistem pengriman data *Publish/subscribe* yang didesain untuk menyelesaikan masalah yang sering disebut dengan *distributed commit log* yang mana suatu *filesystem* atau database didesain untuk menyediakan data rekord-rekord yang disimpan dengan lama dan bisa diakses kembali secara berkala pada sistem yang stabil. Kafka akan menjadi Input Sources bagi spark streaming.

    Untuk sistem *standalone* port zookeper bisa diatur. tetapi port default adalah 2181. Untuk menjalankan Zookeper lakukan perintah `zkserver`. Sebelum menginstal broker pastikan zookeeper berjalan terlebih dahulu. Setelah zookeeper berhasil terinstal. perlu menjalankan broker karena *consumer* dan

producer membutukan *broker* untuk bisa diinstal dan saling berkomunikasi. perintah untuk menjalankan broker: `.\bin \windows\kafka-server-start.bat .\config\server.properties`. Setelah menjalankan broker dan zookeeper, membuat *topics* untuk kafka.

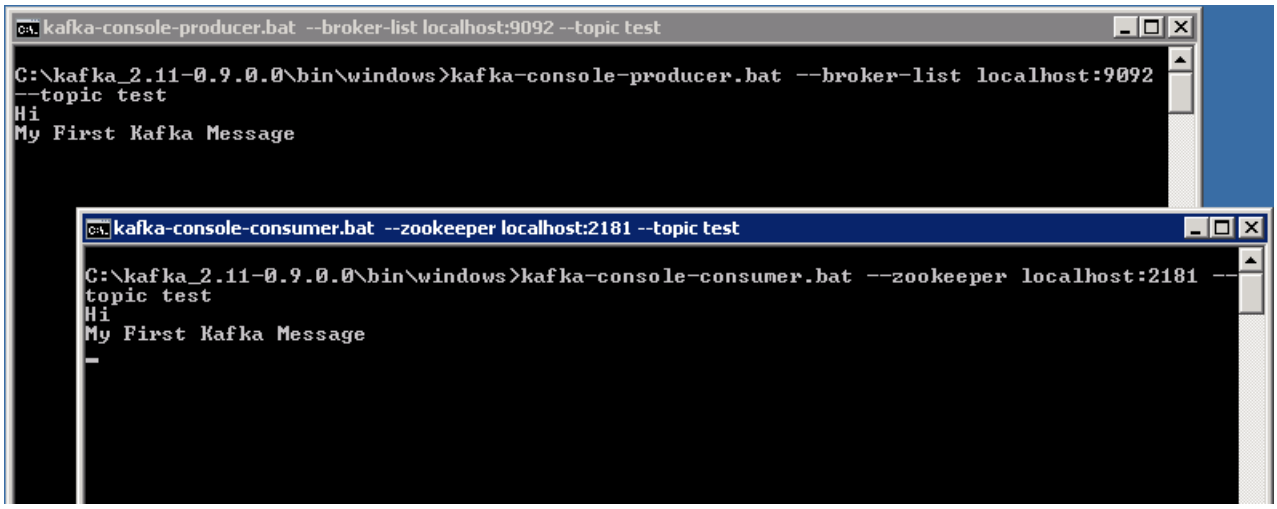
Gambar 28: menjalankan server

Topics yang dibuat dengan nama T̈estd̈an akan direplika sebanyak satu kali jika *standalone*. Jika memiliki klaster lebih dari satu maka bisa mereplikasi lebih dari satu dan replikasi tersebut digunakan sebagai *backup* jika terjadi kegagalan. Topik dapat dibuat dengan perintah: `kafka-topics.bat --create --zookeeper localhost:2181 -- replication-factor 1 --partitions 1 --topic test`



Gambar 29: menjalankan topic

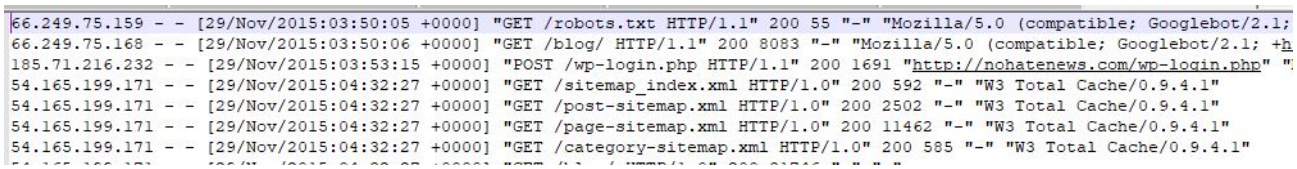
Untuk mengetes apakah suatu server sudah jalan perlu dilakukan pengecekan dengan membuat broker *producer* dan *consumer*. Untuk menjalankan *producer*, lakukan perintah `kafka-console-producer.bat --broker-list localhost:9092 --topic test` dan untuk menjalankan *consumer* lakukan perintah `kafka-console-consumer.bat -- zookeeper localhost:2181 --topic test`. Jika berhasil *consumer* dan *producer* akan saling terhubung.



Gambar 30: *consumer-producer*

15. **Mengeksplorasi dan Menganalisis data yang didapat dari TCP Socket**
Status : Ada sejak rencana kerja skripsi.
Hasil : Studi Eksplorasi dilakukan dengan mencoba mengumpulkan data dari salah satu penyedia data yaitu TCP Socket. karena ini berupa simulasi, maka file input data web logs akan diunduh terlebih dahulu disimpan pada sebuah file `accesslog.txt`. Lalu menggunakan perintah `ncat -lk 9999 <`

`accesslog.txt` yang artinya `ncat` akan memasukan data ke TCP port 9999 satu demi satu seolah-olah data yang masuk berupa aliran data.



Gambar 31: File input

Data yang akan dianalisis adalah simulasi web logs data stream yang datang dari aktivitas suatu website. Eksplorasi ini akan menghitung seberapa sering suatu file dibuka oleh pengguna. Berikut cara membuat spark streaming. Contoh Gambar

```
//1. membuat streaming context dengan ukuran batch 1
val ssc = new StreamingContext("local[*]", "LogParser", Seconds(1))

setupLogging()

// 2.mengambil pattern dari satu weblog IP, Client, urls
val pattern = apacheLogPattern()

// 3.membuat socket stream untuk membaca dari netcat secara local
val lines = ssc.socketTextStream("127.0.0.1", 9999, StorageLevel.MEMORY_AND_DISK_SER)
```

Gambar 32: pengaturan spark streaming

```
//4.cek apakah suatu line sama dengan pola
val requests = lines.map(x => {val matcher:Matcher = pattern.matcher(x); if (matcher.matches()) matcher.group(5)})

// 5.mengambil url dari suatu pola
val urls = requests.map(x => {val arr = x.toString().split("␣"); if (arr.size == 3) arr(1) else "[error]"})

// 6.Reduce Url dengan sliding windows 5 menit
val urlCounts = urls.map(x => (x, 1)).reduceByKeyAndWindow(_ + _, _ - _, Seconds(300), Seconds(1))

// Urutkan dari yang terbesar
val sortedResults = urlCounts.transform(rdd => rdd.sortBy(x => x._2, false))
sortedResults.print()
```

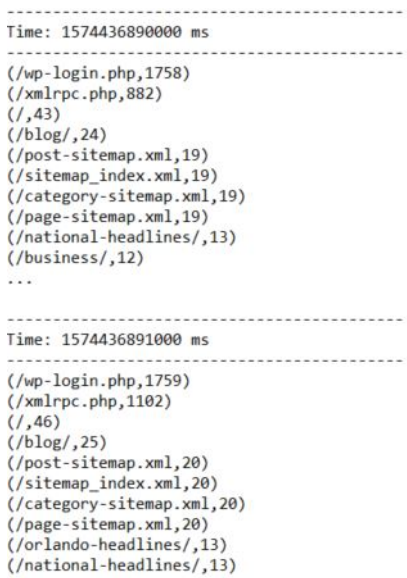
Gambar 33: perhitungan url

Spark streaming dibuat dengan menentukan ukuran batch interval (Streaming Context) selama 1 detik. Pada interval 1 detik Spark Streaming akan mengambil data yang dihasilkan pada interval teresbut. lalu, menggunakan metode dari library ambil web log yang hanya mengikuti pola saja. Jadi, jika ada data lain yang masuk tapi tidak berbentuk web logs akan diabaikan. Menghubungkan batch inteval dengan socket stream untuk mendapatkan data. Mengambil url file dari potongan web logs tersebut contoh `apachepb.gif`.Menghitung dengan ReduceByKeyAndWindow artinya hitung berapa banyak jumlah url pada key dan windows yang sama. Terakhir urutkan url dari yang memiliki hit paling banyak dan tampilkan 10 hasil terbaik. Contoh Gambar

```
// Jalankan Streaming
ssc.checkpoint("C:/checkpoint/")
ssc.start()
ssc.awaitTermination()
```

Gambar 34: Menjalankan Spark Streaming

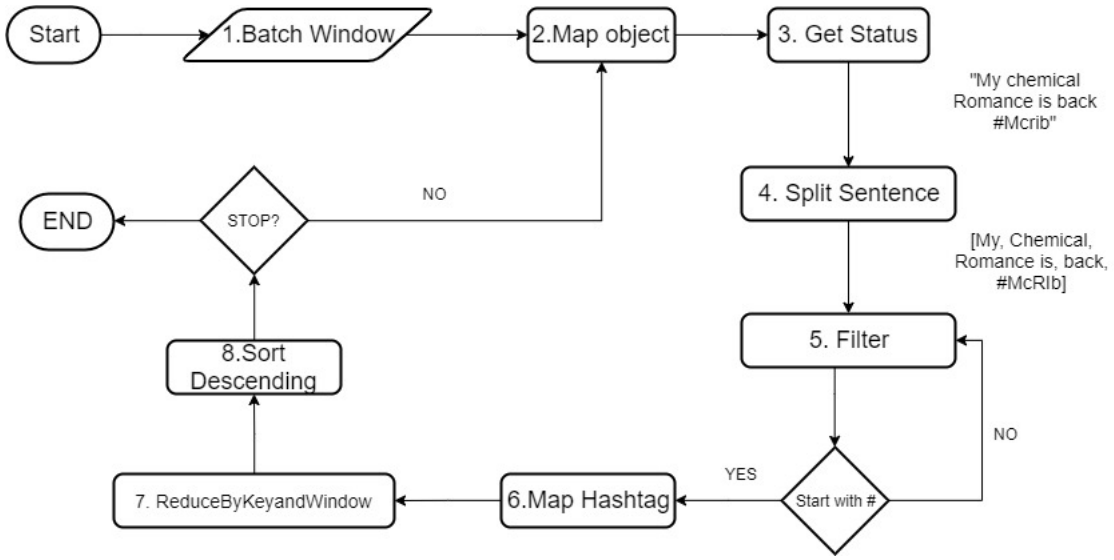
Terakhir jalankan spark streaming yang telah dibuat. `ssc.awaitTermination()` menyatakan bahwa proses pengambilan data tidak akan berhenti sampai ada perintah dari user. Contoh Gambar



Gambar 35: Output Web log

hasil dari eksekusi program tersebut adalah batch dengan interval yang telah kita atur dan pada tiap batch interval tersebut berisi hasil komputasi 10 file yang paling sering diakses. Dari hasil dapat disimpulkan pada batch pertama file yang paling sering diakses dalah `wp-login.php`

16. **Mengeksplorasi dan Menganalisis data yang didapat dari Twitter API**
Status : Ada sejak rencana kerja skripsi.
Hasil : Perangkat lunak yang akan dikembangakan adalah perangkat lunak spark. Perangkat lunak digunakan untuk mengambil data dengan sistem *spark streaming* dan melakukan analisis sederhana secara *real-time*. Program saat ini akan menghitung jumlah hashtag terbanyak pada lima menit terakhir. set data yang akan diambil oleh *spark streaming* bersifat *real-time*. Hal ini dilakukan dengan mengintegrasikan *spark streaming* dengan sistem eksternal yang meyediakan data melalui API. Untuk kasus ini akan digunakan data yang berasal dari twitter.
Data yang didapatkan dari twitter adalah berupa *twitter object* dalam format JSON. Data twitter akan diambil selama 5 menit dan keluaran akan berupa pasangan key value yang bernilai (hashtag,value).Untuk membuat sistem penghitung hashtag bekerja, harus membuat batch window yang nanti akan diintegrasikan dengan Twitter. Lalu Setiap Objek akan dipetakan dan hanya diambil statusnya saja karena informasi hashtag terletak pada status seseorang. Status yang telah didapat akan dipetakan lagi dan dipisahkan perkata. Lalu fungsi filter akan dipanggil dan kita hanya mengambil simbol # yang merepresentasikan *Hashtag*. Hashtag akan dipetakan lagi dengan map hingga berisi pasangan key value. dengan nilai key adalah hashtag dan value adalah jumlahnya. Lalu akan dilakukan *reduce* berdasarkan nilai key dan batch window yang sama. Atur ukuran window menjadi 5 menit.



Gambar 36: Flowchart diagram analisis hashtag

1. Membuat batch window berukuran satu detik. setiap satu detik akan menangkap objek-objek dari twitter.
2. Memetakan setiap objek menjadi statusnya saja artinya dari seluruh objek tweet seperti nama atau location yang kita ambil hanya perkataan yang diposting oleh user saja.
3. Mengambil statusnya satu per satu
4. Membagi kalimat berdasarkan spasi lalu menyimpannya di array
5. Filter setiap elemen pada array jika elemen pada array tidak sama dengan # maka akan terus melakukan filter sampai status pada batch ini habis
6. memetakan hashtag menjadi $[key, value] => [hashtag, 1]$
7. melakukan reduce penjumlahan berdasarkan key dan windows yang sama.
8. Mengurutkan hashtag berdasarkan value dari nilai yang paling tinggi.
9. Jika program sudah diberhentikan maka tidak akan mengumpulkan objek lagi. Tetapi, jika tidak akan terus berlanjut.

berikut kode pengerjaan dari perhitungan hashtag selama lima menit terakhir:

```
// Setting twitter Credentials
setupTwitter()

//setup streaming context ukuran 1 detik
val ssc = new StreamingContext("local[*]", "PopularHashtags", Seconds(1))

// hapus spam selain error
setupLogging()

// Membuat Dstream dengan Streaming context
val tweets = TwitterUtils.createStream(ssc, None)

// ambil status
```

```
val statuses = tweets.map(status => status.getText())

// ambil setiap kata
val tweetwords = statuses.flatMap(tweetText => tweetText.split(" "))

// filter yang bukan hashtag
val hashtags = tweetwords.filter(word => word.startsWith("#"))

// Map setiap hastag menjadi key value (hashtag,1)
val hashtagKeyValues = hashtags.map(hashtag => (hashtag, 1))

//Hitung hashtag perdetik dengan sliding window selama 5 menit
val hashtagCounts = hashtagKeyValues.reduceByKeyAndWindow( (x,y) => x + y,
                    (x,y) => x - y,Seconds(300), Seconds(5))

// Sort berdasarkan banyak hashtag
val sortedResults = hashtagCounts.transform(rdd => rdd.sortBy(x => x._2, false))

// Print the top 10 dan save di HDFS
sortedResults.saveAsTextFiles("hdfs:
                //localhost:50071/Twitter/Hashtag/Output1", "txt")
sortedResults.print

ssc.checkpoint("C:/checkpoint/")
ssc.start()
ssc.awaitTermination()
```

hasil keluaran :

```
-------------------------------------------
Time: 1574386293000 ms
-------------------------------------------
(#Cacerolazo,4)
(#,3)
(#質問箱,2)
(#forex,2)
(#ADonaDoPedaço,2)
(#SLNTeaserDay,2)
(#RT,2)
(#SarileruNeekevvaru,1)
(#진영,1)
(#ExtraordinaryYou,1)
...

-------------------------------------------
Time: 1574386298000 ms
-------------------------------------------
```

Gambar 37: Hasil perhitungan Hashtag

17. **Menulis Dokumentasi Skripsi**
    Status : baru ditambahkan pada semester ini
    Hasil : Penulisan skripsi baru selesai Bab I, Bab II, Bab III dan sebagian Bab IV

6 Pencapaian Rencana Kerja

Langkah-langkah kerja yang berhasil diselesaikan dalam Skripsi 1 ini adalah sebagai berikut:

1. Studi Literatur mengenai *Big Data* dan *Big Data Stream*
2. Studi Literatur mengenai teknik stream processing
3. Studi literatur mengenai pola teknik pemrosesan stream processing
4. Studi literatur arsitektur Stream Processing
5. Studi literatur mengenai sistem terdistribusi Spark
6. Studi Literatur Spark: Arsitektur Spark
7. Studi literatur mengenai Spark Streaming
8. Mengintegrasikan spark streaming dengan twitter API
9. Mengintegrasikan spark streaming dengan TCP socket
10. Mengintegrasikan spark streaming dengan Kafka
11. Mengeksplorasi dan Menganalisis data yang disumulasikan di TCP socket
12. Mengeksplorasi dan Menganalisis data yang didapat dari Twitter API
13. Menulis Dokumentasi skripsi Bab I dan Sebagian Bab II, Bab III, dan Bab IV

7 Kendala yang Dihadapi

Kendala - kendala yang dihadapi selama mengerjakan skripsi :

- Kesulitan meng-instal environment di PC.
- Banyak tugas dari mata kuliah lain
- Sering ada versi yang tidak saling cocok.
- terlalu banyak environment yang di-instal

Bandung, 21/11/2019

Muhammad Ravi

Menyetujui,

Nama: Veronica Sri Moertini
Pembimbing Tunggal