Implementasi Manajemen *Transfer rate* pada Proses HDFS Berbasis SDN

Narendra Hanif Wicaksana, F.X Arunanto, dan Hudan Studiawan
Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: anto@if.its.ac.id

Abstrak*—Pada *cluster Hadoop* perpindahan data akan sering terjadi, karena data yang disimpan akan tersebar ke dalam *Datanode* terutama pada saat melakukan proses penyimpanan ke dalam HDFS. Lalu lintas jaringan data mempengaruhi performa kinerja *cluster Hadoop* secara keseluruhan. Permasalahan ketersediaan *bandwidth* dan juga *congestion* yang disebabkan lalu lintas data lain dapat mempengaruhi proses penyimpanan data ke dalam HDFS. SDN memiiliki fungsi untuk melakukan pengaturan manajemen *transter rate* sehingga dapat mengkategorikan lalu lintas data dan juga menyediakan nilai *transfer rate* dengan menggunkan mekanisme *queue*. Memanfaatkan arsitektur jaringan SDN, pada *cluster Hadoop* dilakukan manajemen *transfer rate* untuk dapat mengoptimalkan proses perpindahan data pada saat penyimpanan ke HDFS. Manajemen *transfer rate* dilakukan dengan cara memanfaatkan fitur *queue* pada *switch Openflow*. Tiap *queue* digunakan untuk mengkategorikan lalu lintas data pada jaringan *cluster Hadoop*. Nilai *transfer rate* untuk lalu lintas data HDFS dipisahkan dan diberikan nilai *transfer rate* yang lebih tinggi. Berdasarkan hasil uji coba dengan melakukan manajemen *transfer rate* waktu proses penyimpanan data ke HDFS tidak terpengaruh walaupun pada saat proses penyimpanan data terdapat lalu lintas data lain yang mengakibatkan *congestion.

***Kata Kunci—Hadoop*, HDFS, *Openflow*, *Software Defined Networking*, *QoS*.**

I. PENDAHULUAN

SAAT berkembang arsitektur baru dalam mengelola jaringan yang dikenal sebagai *Software Defined Networking* [1]. Dimana kontrol jaringan yang sebelumnya berada pada tiap perangkat, dipindahkan menjadi terpusat. Ide *Software Defined Networking* muncul dari prinsip jaringan yang dapat diprogram (*programmable network*) sehingga konfigurasi perangkat jaringan menjadi lebih fleksibel, dinamis dan dapat diprogram. Hal tersebut dilakukan dengan cara mengambil alih proses penanganan paket (*packet handling*) dari yang sebelumnya dilakukan ditiap perangkat menjadi terpusat. Aristektur *Software Defined Networking* dibagi menjadi tiga komponen, yaitu infrastruktur, kontrol dan juga aplikasi. Konsep pada *Software Defined Networking* dapat mempermudah dan mempercepat inovasi pada jaringan.

Pada *switch* atau *router* saat ini *data path* dan juga *control path* berada pada perangkat yang sama. *Switch Openflow* memisahkan dua fungsi tersebut. *Data path* masih tetap berada pada *switch* sedangkan keputusan pemilihan rute dipindahkan ke *controller* terpisah. Ketika *switch Openflow* menerima paket yang baru diterima yang tidak berada pada alur masuk (*flow entry*), paket akan dikirimkan ke *controller*. *Controller* akan mengambil keputusan bagaimana menangani paket tersebut. Paket dapat di-*drop* atau dapat juga dimasukan ke dalam alur masuk *switch*, sehingga *switch* dapat mengetahui bagaimana jika mendapatkan paket yang serupa. Fitur lain yang dimiliki oleh *Openflow* adalah pengaturan *transfer rate*. *Switch Openflow* dapat melakukan pengaturan *transfer rate* dengan cara pembuatan *queue*.

Hadoop merupakan *platform* pengolahan *big data*. *Hadoop* menyediakan mekanisme penyimpanan data dan juga pengolahan data. Data-data pada *cluster Hadoop* disimpan pada sistem *file* yang dikenal dengan sebutan HDFS (*Hadoop Distributed File System*). Sedangkan pengolah data pada *Hadoop* menggunakan model pemograman *MapReduce*.

Pada *cluster Hadoop* perpindahan data akan sering terjadi, karena data yang disimpan akan tersebar ke dalam *Datanode* terutama pada saat melakukan proses penyimpanan ke dalam HDFS. Lalu lintas jaringan data mempengaruhi performa kinerja *cluster Hadoop* secara keseluruhan. Permasalahan ketersediaan *bandwidth* dan juga *congestion* yang disebabkan lalu lintas data lain dapat mempengaruhi proses penyimpanan data ke dalam HDFS.

Pada penelitian ini dengan memanfaatkan arsitektur jaringan SDN, pada *cluster Hadoop* dilakukan manajemen *transfer rate* untuk dapat mengoptimalkan proses perpindahan data pada saat penyimpanan ke HDFS. Manajemen *transfer rate* dilakukan dengan melakukan pengkategorian lalu lintas jaringan dan juga memanfaatkan fitur *queue* pada *switch Openflow*. Tiap *queue* yang dibuat dapat diatur nilai maksimum dan juga nilai minimum *transfer rate*. Dengan menetapkan nilai *transfer rate* pada *cluster Hadoop* maka lalu lintas untuk proses penyimpanan data ke HDFS terbebas dari *congestion* yang disebabkan dari lalu lintas data lain dan juga ketersedaian *bandwidth*.

II. DASAR TEORI

A. Hadoop Distributed File System

HDFS (*Hadoop Distributed File System*) [2] adalah *file* sistem distribusi untuk menyimpan dan juga mengolah data pada satu *cluster Hadoop*. Dalam mekanisme penyimpanan HDFS terdapat satu *node* pusat yang disebut *Namenode* yang

menyimpan metadata dari sistem *file* dan juga node yang lainnya yang disebut sebagai *Datanode*. *File* dalam HDFS dipecah menjadi blok-blok data yang lebih kecil, biasanya bernilai 64 MB tiap blok dan dapat juga diatur sesuai dengan keinginan. Tiap blok data akan disimpan secara terpisah pada *Datanode* dan akan direplikasi sesuai dengan pengaturan. Klien HDFS menghubungi *Namenode* untuk mendapatkan informasi lokasi tiap blok data.

B. SDN dan Openflow

Software Defined Networking (SDN) muncul dari prinsip jaringan yang dapat diprogram (*programmable network*) [3]. Ide SDN muncul untuk mencoba memenuhi kebutuhan konfigurasi perangkat jaringan sehingga menjadi lebih fleksibel dan juga dinamis. Hal tersebut dilakukan dengan cara mengambil alih proses penanganan paket *(packet handling*) dari yang sebelumnya dilakukan di tiap perangkat menjadi terpusat.

Terdapat tiga komponen dasar dalam SDN, dimana diantara tiap komponen terdapat antarmuka. Komponen pertama adalah aplikasi (*application layer/ application plane*). Aplikasi berada pada lapisan paling atas, aplikasi-aplikasi SDN berada pada bagian aplikasi, beberapa aplikasi dapat saling berjalan dan berkolaborasi satu sama lain. Aplikasi-aplikasi ini berkomunikasi dengan *controller* menggunakan API, API ini sering disebut sebagai *Northbound Interface* atau juga sering disebut dengan NBI.

Komponen kedua adalah *controller* (*control layer/ controller plane*). *Controller* SDN akan menerjemahkan kebutuhan aplikasi dengan infrastruktur dengan memberikan instruksi yang sesuai untuk SDN *datapath* serta memberikan informasi yang relevan dan dibutuhkan oleh lapisan aplikasi.

Komponen ketiga adalah infrastruktur (*infastructure layer/ data plane*). Terdiri dari elemen-elemen jaringan yang dapat mengatur SDN *datapath* sesuai dengan instruksi yang diberikan oleh *controller*. *Control plane* mengirim intruksi ke *infrastructure layer* melalui antarmuka yang disebut dengan *Soutbound interface.*

Dalam arsitektur SDN protokol *Openflow* merupakan salah satu elemen terpenting. Protokol *Openflow* [4] digunakan untuk berkomunikasi antara *controller* dengan *switch Openflow* atau yang disebut dengan atau yang disebut *southbound interface*. Protokol ini dapat mengubah *flow table* dari *switch Openflow*. Ketika *switch Openflow* menerima paket, *switch* akan mencoba mencocokan nilai *header* dari paket dengan aturan yang telah ada pada *flow table*. Apabila paket tidak ditemukan pada *flow table* maka *switch Openflow*, akan mengirimkan ke *controller*. *Controller* akan mengambil keputusan bagaimana menangani paket tersebut. Paket dapat di-*drop* atau dapat juga dimasukan kedalam *flow table switch*, sehingga berikutnya *switch* dapat mengetahui bagaimana jika mendapatkan paket yang serupa.

Selain fitur tersebut *switch Openflow* memiliki fitur pengaturan QoS. Pengaturan QoS pada *switch Openflow* dilakukan dengan mekanisme *queuing*. Tiap *queue* dapat digunakan untuk mengkategorikan jaringan sesuai dengan karakteristik jaringan yang dibutuhkan. Selain itu juga tiap *queue* yang dibuat dapat diatur nilai maksimum dan juga nilai *minimum transfer rate*.

III. PERANCANGAN

A. Arsitektur Jaringan SDN

Arsitektur jaringan SDN memiliki tiga komponen utama yaitu infrastruktur, *controller*, dan juga aplikasi. Ketiga komponen harus diimplementasikan untuk membagun arsitektur jaringan SDN. Salah satu permasalahan dalam pengimplementasian arsitektur jaringan SDN adalah dibutuhkan *switch* yang dapat mendukung protokol *Openflow*. Untuk menyiasati hal tersebut dibangun *switch Openflow* menggunakan *low cost embedded* Linux *machine* Raspberry Pi dan juga *Open* v*Switch* yang pernah juga dilakukan pada penelitian sebelumnya [5].

Pada bagian infrastruktur digunakan Raspberry Pi yang telah diinstal dengan *Open* v*Switch* sehingga dapat dijadikan sebagai *switch Openflow*. Untuk menambah jumlah antarmuka *Ethernet* pada Raspberry Pi digunakan *adaptor* USB *to Ethernet*.

Bagian *Controller* yang merupakan otak dari jaringan SDN digunakan *Ryu* [6]. Dengan menggunakan *Ryu*, *controller* dan juga *switch* dapat berinteraksi dengan menggunakan protokol *Openflow* dan juga OVSDB.

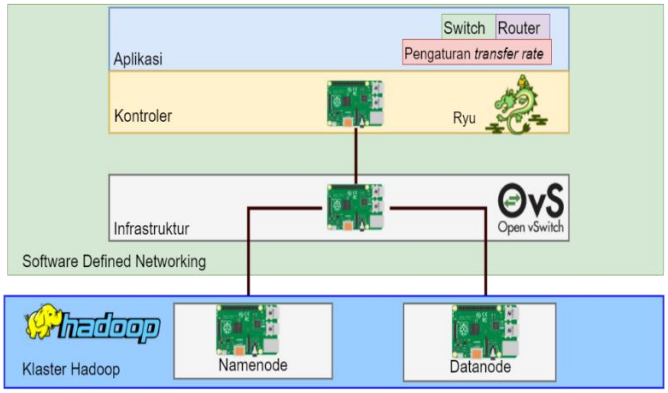
Pada bagian aplikasi, untuk mendukung platform SDN yang dibangun, digunakan aplikasi untuk menghubungkan antar *host*, *switch hub*, *router* dan juga aplikasi pengaturan *transfer rate*.

B. Cluster Hadoop

Cluster Hadoop dibangun terdiri dari dua *node*,yang terdiri dari satu *Namenode* dan satu *Datanode*. *Node* yang digunakan pada *cluster* yang dibangun menggunakan perangkat keras Raspberry Pi dan menggunakan *Hadoop* versi 1.2.

C. Topologi Hadoop dan SDN

Cluster Hadoop memiliki arsitektur *master-slave* dan SDN memiliki tiga komponen dalam arsitekturnya. *Switch Openflow* akan menjadi *switch* penghubung antara *master* dan juga *slave* pada *cluster Hadoop* seperti yang dapat dilihat pada Gambar 1. *Datanode* dan juga *Namenode* akan terhubung dengan *port Openflow* sedangkan *switch Openflow* akan terhubung dengan *controller* SDN melalui bukan *port Openflow*.



Gambar 1.Topologi *Hadoop* dan SDN

D. Manajemen Transfer rate

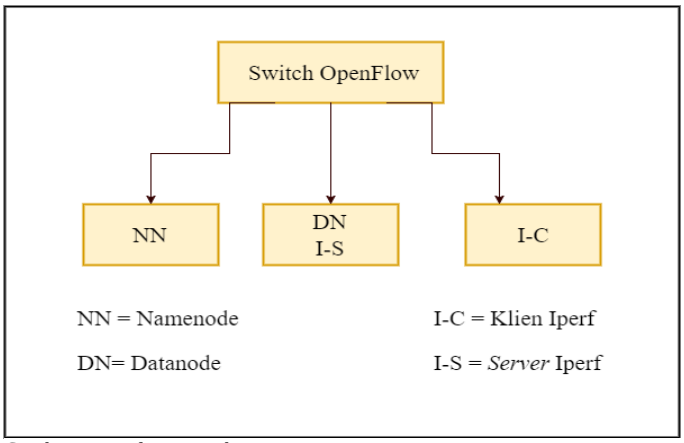
Pada kaster *Hadoop* yang dibangun akan dilakukan manajemen *transfer rate* sehingga proses HDFS *Hadoop*

dapat dioptimalkan. Manajemen *transfer rate* dilakukan dengan cara memanfaatkan fitur yang dimiliki *switch Openflow*. *Switch Openflow* memiliki fitur pengaturan *transfer rate* dengan menggunakan mekanisme *queue*. Dilakukan pengkategorian lalu lintas data dengan menggunakan *queue*. *Queue* untuk proses penyimpanan data ke HDFS dipisahkan dengan lalu lintas data yang lain pada jaringan yang dibangun dan juga diberikan nilai *transfer rate* yang lebih besar dibandingkan dengan lalu lintas data yang lain. Sehingga ketika terjadi *congestion* pada jaringan proses perpindahan data HDFS tidak terganggu dan *bandwidth* tersedia. Pengaturan *transfer rate* pada klster *Hadoop* juga pernah dilakukan pada [7] untuk proses *MapReduce*.

IV. UJI COBA DAN EVALUASI

Pada pengujian dilakukan penyimpanan data ke HDFS. Uji coba dilakukan pada *cluster Hadoop* yang yang terdiri dari satu *Namenode* dan satu *Datanode* yang saling terhubung dengan *switch Openflow* seperti yang dapat dilihat pada Gambar 2. Untuk dapat melihat pengaruh manajemen *transfer rate*, pada *cluster Hadoop* diberikan *traffic* tambahan untuk bertujuan membuat *congestion* sehingga proses perpindahan data menjadi terganggu. *Traffic* tambahan dilakukan dengan menjalankan program *Iperf*. *Datanode* dijadikan sebagai *Iperf server* yang akan menerima kiriman paket data dari klien *Iperf* seperti yang dapat pada Gambar 2. Pada pengujian dilakukan tiga skenario uji coba untuk melihat pengaruh dari manajemen *transfer rate* apabila terdapat *traffic* data lain yang mengakibatkan *congestion* pada *cluster Hadoop*. Tiap skenario dibandingkan waktu proses penyimpanan ke HDFS. *Dataset* yang digunakan terdapat tiga ukuran yaitu 1 MB, 10 MB dan 20 MB.

A. Skenario Uji Coba 1



Gambar 2 Topologi uji coba

Pada skenario uji coba 1 proses penyimpanan data ke HDFS dilakukan tanpa diberikan *traffic* tambahan. Semua data masuk kedalam q_0. Tujuan dari skenario uji coba 1 untuk mendapat nilai tolak ukur proses penyimpanan data ke HDFS. Data dimasukan dari *Namenode* kemudian data dikirimkan ke *Datanode*.

Hasil dari skenario 1 dapat dilihat pada Tabel 1. Berdasarkan hasil uji coba rata-rata waktu yang dibutuhkan untuk memasukan data sebesar 1MB kedalam HDFS 32 detik. Untuk data sebesar 10 MB dan juga 20 MB dibutuhkan waktu 5 menit 23 detik dan 11 menit 5 detik.

Tabel 1.
Hasil uji coba uji coba skenario 1

Ukuran	Waktu (Menit:Detik)				
1 MB	00:34	00:33	00:33	00:30	00:30
10 MB	05:43	05:22	05:23	05:09	05:21
20 MB	10:55	11:29	11:00	11:02	11:03

B. Skenario Uji Coba 2

Skenario kedua pada saat menjalankan proses HDFS *Hadoop* diberikan lalu lintas tambahan yang berjalan pada *queue* yang sama. Pemberian lalu lintas tambahan dengan cara klien *Iperf* mengirimkan data berupa TCP ke *Datanode* sehingga lalu lintas data masuk ke *Datanode* terdapat *congestion*. Tujuan dari skenario ini untuk mendapatkan waktu apabila pada proses HDFS terdapat lalu lintas tambahan yang mengakibatkan *congestion* apabila lalu lintas data digabung tidak dilakukan manajemen *transfer rate*.

Hasil dari skenario 2 dapat dilihat pada Tabel 2. Berdasarkan hasil uji coba rata-rata waktu yang dibutuhkan untuk memasukan data sebesar 1MB kedalam HDFS 53 detik. Untuk data sebesar 10 MB dan juga 20 MB dibutuhkan waktu 7 menit 18 detik dan 13 menit 21 detik.

Tabel 2
Hasil uji coba skenario 2

Ukuran	Waktu (Menit:Detik)				
1 MB	00:52	00:55	00:51	00:52	00:55
10 MB	07:04	07:26	07:17	07:17	07:25
20 MB	13:11	13:30	13:25	13:27	13:12

C. Skenario Uji Coba 3

Skenario uji coba 3 sama seperti skenario uji coba 2 dimana pada saat melakukan proses penyimpanan data ke HDFS dari *Namenode* menuju *Datanode* diberikan lalu lintas data tambahan yang dilakukan dengan menjalankan program *Iperf server* pada *Datanode Iperf* klien akan mengirimkan data juga menju *Datanode* sehingga lalu lintas data menuju *Datanode* terdapat *congestion*. Namun pada uji coba 3 dilakukan manajemen *transfer rate*. Lalu lintas data program *Iperf* dipisahkan dengan lalu lintas penyimpanan data ke HDFS. Lalu lintas program *Iperf* dimasukan ke dalam q_1 dan lalu lintas data HDFS dimasukan ke q_0. Q_0 diberikan nilai *transfer rate* yang lebih besar dari nilai *transfer rate* q_1. Pengaturan *queue* dapat dilihat pda Tabel 3. Q_0 diberikan nilai minimum *transfer rate* sebesar 800 Kbps dan nilai maksimum sebesar 1 Mbps, dengan mengimplementasikan aplikasi manajemen *transfer rate* pada arsitektur yang dibangun dipastikan nilai transfer data untuk proses HDFS sesuai dengan nilai maksimum dan juga nilai minimum yang ditentukan.

Tabel 3
Pengaturan *queue*

Queue	Nilai maksimum	Nilai minimum
q_0	1 Mbps	800 Kbps
q_1	300 Kbps	-

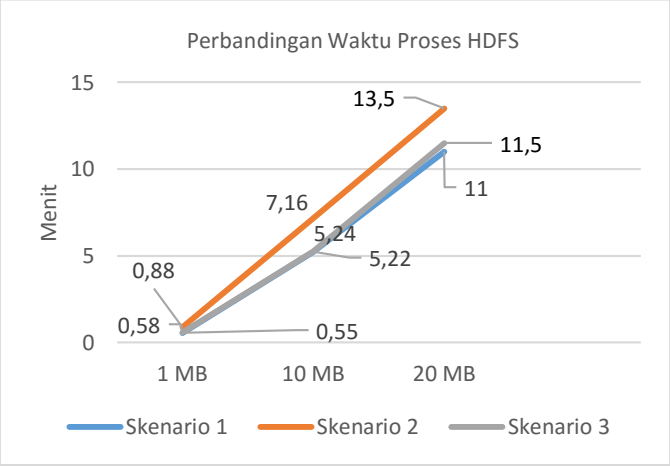
Hasil dari skenario 3 dapat dilihat pada Tabel 4. Berdasarkan hasil uji coba rata-rata waktu yang dibutuhkan untuk memasukan data sebesar 1MB kedalam HDFS 35 detik. Untuk data sebesar 10 MB dan juga 20 MB dibutuhkan waktu 5 menit 31 detik dan 11 menit 25 detik.

Tabel 4
Hasil uji coba skenario 3

Ukuran	Waktu (Menit:Detik)				
1 MB	00:35	00:35	00:35	00:36	00:34
10 MB	05:44	05:30	05:25	05:25	05:33
20 MB	11:23	11:23	11:52	10:54	11:34

D. *Evaluasi Uji Coba*

Berdasarkan hasil uji coba ketiga skenario yang dapat dilihat pada Gambar 3, skenario uji coba 2 membutuhkan waktu terbanyak untuk proses penyimpanan data HDFS. Sedangkan skenario uji coba 1 dan skenario uji coba 3 memiliki waktu proses yang hamper sama. Proses HDFS mengalami peningkatan waktu apabila terdapat lalu lintas tambahan pada *cluster Hadoop*, hal tersebut dapat dilihat dari perbandingan skenario satu dan dua dengan munculnya lalu lintas tambahan maka terjadi *congestion* yang dapat mempengaruhi waktu proses penyimpanan ke dalam HDFS. Melakukan manajemen *transfer rate* pada *cluster Hadoop* dapat menurukan waktu proses HDFS hal tersebut dapat dilihat dari perbandingan skenario dua dan tiga. Dengan melakukan manajemen *transfer rate* waktu proses HDFS *Hadoop* menjadi tidak terlalu banyak peningkatan walaupun dalam *cluster Hadoop* terdapat lalu lintas tambahan yang dapat mengakibatkan *congestion*.



Gambar 3 Grafik perbandingan waktu HDFS

V. KESIMPULAN DAN SARAN

Berdasarkan hasil uji coba dan analisis, terdapat kesimpulan dan saran.

A. *Kesimpulan*

Dari hasil selama proses perancangan, implementasi, serta pengujian dapat diambil kesimpulan sebagai berikut:

1. Melakukan manajemen *transfer rate* dapat mengoptimalkan proses HDFS *Hadoop* pada *cluster Hadoop*.
2. Dengan menetapkan nilai *transfer rate* pada *cluster Hadoop* maka lalu lintas untuk proses penyimpanan data ke HDFS terbebas dari *congestion* yang disebabkan dari lalu lintas data lain.
3. Manajemen *transfer rate* dapat dilakukan dengan memanfaatkan fitur *queue* yang dimiliki *switch Openflow*.
4. Arsitektur SDN dapat dibangun dengan memanfaatkan Raspberry Pi sebagai *switch Openflow* untuk keperluan uji coba.

B. *Saran*

Saran yang dapat diberikan dalam pengujian sistem ini adalah sebagai berikut:

1. Melakukan pengaturan *transfer rate* dapat dilakukan secara dinamis.
2. Mekanisme konfigurasi perangkat memakan waktu yang lama dan berulang dapat dilakukan dengan menggunakan *script*.

DAFTAR PUSTAKA

[1] Open networking foundation, “Software-Defined Networking: The New Norm for Networks,” 2014.

[2] T. White, *Hadoop* The Definitive Guide, California: O'REILLY, 2009.

[3] Open networking foundation, “SDN Architecture,” 2014.

[4] McKeown, Nick; Anderson, Tom; Balakrishnan, Hari; Parulkar, Guru; Peterson, Larry; Rexford, Jennifer; Shenker, Scott; Turner, Jonathan, “*Openflow*; Enabling Innovation in Campus Networks,” 14 March 2008.

[5] H. Kim, J. Kim and Y.-B. Ko, “Developing Cost-effective *Openflow* Test Bed for Small Scale Software Defined networking,” 2014.

[6] “Ryu,” [Online]. Available: http://github.com/osrg/ryu. [Accessed 10 Juni 2016].

[7] S. Narayan, S. Bailey and A. Daga, “*Hadoop* Acceleration in an *Openflow* based cluster,” November 2012.