Analisa Hadoop *High Availability* Menggunakan Quorum Journal Manager dan Zookeeper dengan Studi Kasus Namenode *Failover*

Andhika Willy Satrio [1], Dr. Maman Abdurohman, ST., MT.[2]

[1,2]Fakultas Informatika Universitas Telkom, Bandung
[1]andhikawilly@student.telkomuniversity.ac.id [2]mma@telkomuniversity.ac.id,

Abstrak

Hadoop adalah *software framework opensource* yang memungkinkan untuk pemrosesan secara terdistribusi dari kumpulan set data yang besar pada *Cluster* komputer dengan menggunakan model pemrograman tertentu. Hadoop terdiri dari 2 buah komponen utama dalam menyelesaikan permasalahan *Big Data*, yaitu MapReduce dan HDFS (*Hadoop Distributed File System*).

Dari kedua komponen tersebut maka HDFS yang memegang peranan penting dalam mengatur *File System*. Namun saat *MasterNode* yang di dalamnya terdapat *Namenode* dan *Jobtracker* mengalami gangguan maka *File System* tidak dapat melayani pengguna. Maka dari itu, diperlukan cara khusus untuk dapat mengatasinya dengan membuat *cloning* dari *MasterNode* di server yang lain sehingga saat *MasterNode* mengalami masalah dapat digantikan oleh *cloning*-nya yang disebut dengan *StandbyNode*.

Pada pengerjaan Tugas Akhir ini menerapkan metode Zookeeper dan QJM pada *cluster* Hadoop untuk dapat meningkatkan *High Availability* pada Hadoop dengan *downtime failover* berkisar antara 4 detik sampai 63 detik. Jika dibandingkan dengan metode DRBD dan Heartbeat maka metode Zookeeper dan QJM yang digunakan sekarang lebih baik untuk digunakan dalam jangka panjang. Dan dengan menggunakan Zookeeper dan QJM, *downtime failover* relatif stabil tanpa adanya peningkatan *downtime* terhadap jumlah blok data.

Kata kunci : Hadoop, *High Availability, NameNode, Failover, Downtime*

Abstract

Hadoop is an open source software framework that allows data processing in a distributed from a collection of large data sets on clusters of computer by using a particular programming model. Hadoop consists of 2 core components in solving the problems of Big Data, there are MapReduce and HDFS (Hadoop Distributed File System).

These two core components, the HDFS is an important role in managing File System. When MasterNode which has Namenode and Jobtracker roles got a failure, then the File System can not serve user client. Therefore, there is a solution for it by making a cloning of MasterNode on another server so that when MasterNode experiencing failure then it could be replaced by its cloning called StandbyNode.

This research applying Zookeeper and QJM method on Hadoop cluster to be able to improve on Hadoop High Availability with failover downtime ranges from 4 seconds to 63 seconds. When compared with DRBD and Heartbeat method then Zookeeper and QJM method used today is better to use in a long term. Also by using Zookeeper and QJM, downtime failover relatively stable without increasing the downtime towards the number data blocks.

Key Words : Hadoop, High Availability, NameNode, Failover, Downtime

1. Pendahuluan

Saat ini perkembangan manusia menuju pada era digital, dimana setiap data yang dibuat dan dikirimkan setiap detiknya berbentuk digital. Dalam berita BBC menyebutkan bahwa jumlah data yang tersimpan diseluruh dunia pada tahun 2007 berjumlah 295 exabytes. Ukuran ini sama dengan jumlah 1,2 milyar hard drive pada umumnya. Tetapi kapasitas penyimpanan hard drive yang bertambah setiap tahunnya tidak sebanding dengan kecepatan pertambahan data.

Selain itu, dibutuhkan sistem yang tepat dalam pengolahan data terutama dalam menjaga kelangsungan proses bisnis, keamanan data, dan *recovery* data sehingga dilakukanlah analisis dan perancangan sistem *High Availability*.

Dalam pengolahan data yang besar, terdapat 3 aspek yang menjadi poin terpenting, yaitu data terkumpul sangat cepat, volume data akan bertambah banyak, dan variasi data yang terkumpul. Salah satu solusi yang dapat menjawab permasalahan diatas adalah Hadoop.

Hadoop terdiri dari 2 buah komponen utama dalam menyelesaikan permasalahan *Big Data*, yaitu MapReduce dan HDFS (*Hadoop Distributed File System*). Dari kedua komponen tersebut, HDFS yang

memegang peranan penting dalam mengatur *File System*.[1] Namun saat *Master Node* Hadoop mengalami gangguan maka *File System* tidak dapat melayani pengguna. Maka dari itu, diperlukan cara khusus untuk dapat mengatasi masalah *Single Point Of Failure* (SPOF) ini dengan membuat *mirror* dari *Master Node* di server yang lain sehingga saat *Master Node* (*Active NameNode*) mengalami masalah dapat digantikan oleh *mirror*-nya (*Standby NameNode*).

Selain itu, diperlukan metode *failover* secara otomatis sehingga ketika *Master Node* digantikan oleh *mirror* maka semua layanan Hadoop langsung berpindah ke *mirror* tanpa adanya konfigurasi manual oleh administrator. Hal lainnya yang perlu diperhatikan adalah meminimalisir *delay* saat terjadi gangguan sampai *mirror* menyala agar tingkat *availability* sistem menjadi lebih tinggi.

Dengan berbagai macam kelebihan Hadoop ini, maka diharapkan dapat menyelesaikan jumlah data dalam skala besar. Selain itu, hasil dari penelitian ini dapat meningkatkan *availability* pada *server* Hadoop saat terjadi gangguan.

Berdarsarkan latar belakang yang sudah diuraikan, maka terdapat beberapa permasalahan yang akan diselesaikan, yaitu :

a. Bagaimana implementasi dan performansi *Active* NameNode pada Hadoop sebelum dan setelah *failover*.
b. Bagaimana proses *failover* terjadi ketika *Active* NameNode mengalami *down*
c. Bagaimana pengaruh jumlah blok data yang disimpan di dalam Hadoop terhadap waktu *failover*.
d. Apa yang mempengaruhi durasi *failover* ketika *Active* NameNode mengalami *down*

Berikut ini merupakan tujuan dari pembuatan Tugas Akhir ini :

a. Dapat mengevaluasi hasil dari performansi *Active* NameNode pada Hadoop sebelum dan setelah *failover*.
b. Dapat memahami proses *failover* ketika *Active* NameNode mengalami *down*
c. Menganalisa pengaruh jumlah blok data yang disimpan di dalam Hadoop terhadap *downtime failover*.
d. Menganalisa *downtime failover* ketika *Active* NameNode mengalami *down*
e. Dapat mengimplementasi Quorum Journal Manager & Zookeeper

2. Landasan Teori

2.1 Hadoop

Hadoop adalah *software framework opensource* yang memungkinkan untuk pemrosesan secara terdistribusi dari kumpulan set data yang besar pada *Cluster* komputer dengan menggunakan model pemrograman tertentu. Hal ini dirancang untuk meningkatkan kinerja *server* dengan menawarkan komputasi dan penyimpanan lokal pada banyak komputer. Dengan adanya Hadoop dimungkinkan untuk mengolah data dalam jumlah yang sangat besar hingga ukuran petabyte dan dijalankan di atas ribuan komputer.

2.2 Prinsip Kerja Hadoop

MapReduce merupakan model pemrosesan dari data-data yang tersimpan secara terdistribusi dengan menguraikan proses data menjadi fase Map dan fase Reduce. Terdapat 2 komponen di dalam MapReduce, yaitu TaskTracker dan JobTracker. TaskTracker ini yang bertugas untuk menguraikan proses data. JobTracker bertugas untuk membagi proses pengolahan data menjadi proses kecil-kecil yang nantinya akan ditangani oleh TaskTracker. Lalu HDFS adalah sebuah file system terdistribusi yang digunakan dalam menyimpan data-data dalam jumlah besar dalam sebuah *Cluster* komputer. HDFS juga memiliki 2 komponen didalamnya, yaitu DataNode dan NameNode. DataNode merupakan tempat menyimpan blok-blok data yang sudah terdistribusi. Sedangkan NameNode bertugas untuk menyimpan metadata dan membagi data menjadi blok-blok data kecil yang nantinya akan disimpan kedalam DataNode.[1]

Ketika dilakukan proses *write* (*upload*) dan *read* (*download*) di HDFS, semua proses data dilakukan oleh NameNode dan JobTracker yang terdapat di *MasterNode*. Lalu *MasterNode* akan menyebarkan data kepada para *SlaveNode* yang berisi DataNode dan TaskTracker.

2.3 Quorum Journal Manager (QJM)

Pada *Cluster* Hadoop, terdapat satu buah *MasterNode* dan satu buah *StandbyNode*. *Master Node* bertindak sebagai *Active NameNode* yang bertanggung jawab untuk semua layanan klien pada *Cluster* Hadoop. Sedangkan *StandbyNode* bertindak sebagai *Standby NameNode* yang bertanggung jawab untuk menggantikan *Master Node* saat terjadi *failover*.

Namun untuk dapat menjaga sinkronisasi antara *Active NameNode* dengan *Standby Namenode* diperlukan daemon QJM yang disebut JournalNode. Fungsi dari JournalNode adalah merekam log secara berkala sehingga saat ada perubahan NameSpace pada *Active NameNode* maka *Standby NameNode* akan mengetahuinya lalu mengubah NameSpace-nya. *Standby NameNode* akan memastikan bahwa NameSpace-nya sudah disinkronkan sebelum terjadi *failover*.[6]

Untuk memberikan *failover* yang cepat dibutuhkan informasi yang *up to date* tentang lokasi dari blok-blok data yang ada di *Cluster*. Maka dari itu, DataNode sudah dikonfigurasi dengan lokasi dari kedua NameNode (*Active NameNode* dan *Standby NameNode*). Lalu DataNode akan mengirimkan informasi lokasi dari blok-blok data dan *Heartbeat* kepada kedua NameNode tersebut.

2.4 Zookeeper

Zookeeper merupakan layanan untuk mengatur koordinasi data, memberitahu klien jika ada perubahan data, dan *monitoring server* jika ada *failure*. Berikut ini merupakan manfaat dari Zookeeper :

- Pendeteksi kegagalan

Setiap NameNode di *Cluster* akan mengatur sesi pada Zookeeper. Jika NameNode mengalami gangguan maka sesi Zookeeper akan berakhir sehingga memberitahu NameNode lainnya bahwa *failover* telah terjadi.

- Pemilihan *Active NameNode*

Zookeeper memiliki mekanisme sederhana untuk secara eksklusif memilih NameNode yang masih aktif. Jika terjadi *failover* maka Zookeeper akan segera menunjuk NameNode lainnya untuk menggantikannya.

Selain itu, terdapat komponen lainnya yang merupakan klien dari Zookeeper yang juga memonitor dan mengelola keadaan NameNode. Komponen itu disebut ZKFailoverController (ZKFC). Baik Zookeeper maupun ZKFC diperlukan untuk konfigurasi *failover* sistem secara otomatis. Berikut ini merupakan manfaat dari ZKFC.

- Pemantuan *Health*

ZKFC melakukan ping ke lokal NameNode secara periodik dengan *command* tertentu. Selama NameNode merespon dengan status yang *healthy*, maka ZKFC akan menganggap bahwa node tersebut masih dalam kondisi baik. Jika node mengalami *crash*, *frozen*, atau dalam keadaan yang tidak baik maka *health monitor* akan menandainya sebagai *unhealthy*.

- Manajemen sesi Zookeeper

Ketika lokal NameNode dalam kondisi baik, ZKFC memegang sesi *open* Zookeeper. Jika NameNode aktif, ZKFC juga memegang spesial *lock* znode. Dan ketika sesi berakhir maka *lock* akan terhapus secara otomatis.

- Zookeeper *based election*

Jika lokal NameNode dalam kondisi baik dan ZKFC melihat bahwa tidak ada node yang memegang *lock* znode maka akan dengan sendirinya mencoba untuk mendapatkan *lock*. Jika sukses maka ZKFC telah "memenangkan pemilihan (*election*)" dan bertanggung jawab untuk melakukan *failover* dengan membuat lokal NameNode menjadi aktif. Proses *failover* mirip dengan manual *failover* pada umumnya.[5]

2.5 *High Availability*

High Availability bertujuan untuk meningkatkan ketersediaan layanan *Cluster* komputer kepada klien. Ketesediaan yang dimaksud adalah kemampuan *server* agar dapat terus melayani klien dengan mengurangi dampak yang terjadi saat adanya gangguan atau *maintenance* pada *server*.

High Availability server ditempatkan pada area yang terpisah secara geografis sehingga *backup server* dapat terhindar dari gangguan atau bencana yang sama seperti s*erver* pusatnya. Hal ini dinamakan dengan *Disaster Recovery*.[2]

Pada Hadoop, *High Availability* sudah diimplementasikan dengan *failover*. Awalnya Hadoop hanya menggunakan *single master model* yang akhirnya menghasilkan *Single Point of Failure* (SPOF). Namun seiring dengan meningkatnya kebutuhan terhadap pengelolaan *Big Data* maka peningkatan ketersediaan layanan *server* menjadi sebuah kebutuhan sehingga tingkat *availability* Hadoop harus terus dikembangkan.

3. Implementasi

Pada Tugas Akhir ini, pembahasan akan difokuskan kepada HDFS karena proses *High Availability* yang menggunakan Zookeeper & Quorum Journal Manager hanya di lakukan pada HDFS.

Tabel 1 Cluster sebelum terjadi *failover*

Nama Komputer	Fungsi	Layanan	Status
master-pc	*Master Node*	NameNode, *Secondary* NameNode, JobTracker	Aktif
mirror-pc	*Standby Node*	NameNode	*Standby*
slave1-pc	*SlaveNode*	DataNode, TaskTracker	Aktif
slave2-pc	*SlaveNode*	DataNode, TaskTracker	Aktif
slave3-pc	*SlaveNode*	DataNode, TaskTracker	Aktif

Tabel 2 Cluster setelah terjadi failover

Nama Komputer	Fungsi	Layanan	Status
master-pc	*Master Node*	NameNode	*Standby*
mirror-pc	*Standby Node*	NameNode, *Secondary* NameNode	Aktif
slave1-pc	*SlaveNode*	DataNode, TaskTracker	Aktif
slave2-pc	*SlaveNode*	DataNode, Task Tracker	Aktif
slave3-pc	*SlaveNode*	DataNode, Task Tracker	Aktif

3.1 Proses *Failover Clustering*

Failover merupakan proses peralihan ke *Cluster Server* cadangan baik itu berupa komponen, elemen, atau operasi yang sedang dijalankan sehingga gangggguan yang sedang terjadi dapat diatasi dengan baik.[3] Dan dalam riset sekarang ini, penulis melakukan analisa terhadap waktu *downtime failover*. Ketika *Master Node* mengalami gangguan (*disconnected, crash*, kebakaran, dan sebagainya) maka NameNode tidak dapat melayani servis HDFS sehingga diperlukan proses *failover* dimana *Mirror Node* yang berada di tempat lain akan menggantikan NameNode untuk melayani servis HDFS. Namun akan terjadi *delay* saat proses *failover* tersebut dilakukan. Untuk itu diperlukan metode yang tepat agar mengurangi *downtime failover* sehingga servis HDFS pada NameNode dapat berjalan secara simultan. Dengan proses *failover* yang cepat maka *High Availability* pada *Cluster* Hadoop pun juga akan meningkat.

Untuk meningkatkan *High Availability* pada *Cluster* Hadoop maka diperlukan instalasi Zookeeper dalam mengatur koordinasi layanan pada aplikasi yang terdistribusi dan *monitoring server* jika ada NameNode *failure*. Selain itu, perlu dilakukan konfigurasi *High Availability* pada HDFS dengan menggunakan metode Quorum Journal Manager (QJM). Baik Zookeeper maupun QJM harus dilakukan instalasi dan konfigurasi pada *MasterNode* dan *StandbyNode*.[4]

3.2 Hasil Pengujian

Dari hasil pengujian didapatkan beberapa data tentang delay waktu *failover* yang dilakukan sebanyak 10 kali percobaan pada setiap jumlah blok data. Hasil pengujian ini disajikan menggunakan diagram batang untuk mendapatkan gambaran secara umum tentang peningkatan dan pengurangan *downtime failover*. Berikut ini merupakan data hasil pengujian yang didapatkan :

Tabel 3 Percobaan 5000 blok data

Percobaan ke-	Waktu *failover* (milisecond)
1	20625
2	41838
3	32629
4	47183
5	38158
6	53968
7	44977
8	25493
9	29223
10	27964

Tabel 4 Percobaan 10.000 blok data

Percobaan ke-	Waktu *failover* (milisecond)
1	28846
2	50310
3	59288
4	47703
5	4773
6	9432
7	24472
8	46849
9	58389
10	31782

Tabel 5 Percobaan 30.000 blok data

Percobaan ke-	Waktu *failover* (milisecond)
1	38574
2	59205
3	47490
4	47950
5	58141
6	8054
7	42127
8	15204
9	37252
10	34834

Tabel 6 Percobaan 50.000 blok data

Percobaan ke-	Waktu *failover* (milisecond)
1	33743
2	41518
3	41730
4	39605
5	25836
6	41525
7	63819
8	41103
9	59501
10	46542

Tabel 7 Percobaan 85.000 blok data

Percobaan ke-	Waktu *failover* (milisecond)
1	47776
2	41751
3	25280
4	32001
5	37951
6	41538
7	50003
8	11041
9	45909
10	30731

Tabel 8 Percobaan 100.000 blok data

Percobaan ke-	Waktu *failover* (milisecond)
1	49946
2	58485
3	48185
4	26499
5	63366
6	41552
7	26810
8	28326
9	43491
10	58073

3.3 Analisis *Failover* Menggunakan QJM dan Zookeeper

Berdasarkan pengujian yang sudah dilakukan, maka *failover* NameNode sudah berhasil di implementasikan. Lalu sinkronisasi blok-blok data dari *Active* NameNode ke *Standby* NameNode sudah berjalan dengan baik. Bahkan ketika sedang ada proses *write* di *Active* NameNode lalu NameNode *down*, maka proses *failover* tetap berjalan baik sedangkan proses *write* yang sedang berlangsung akan dilanjutkan kembali.

Dalam proses *failover* terdapat beberapa *handler* pada Zookeeper dan ZKFC yang digunakan untuk memastikan bahwa *Active* NameNode harus siap terhadap layanannya. *Handler* tersebut adalah ActiveStandbyElector dan ActiveBreadCrumb. ActiveStandbyElector digunakan untuk menentukan node mana yang harus menjadi aktif dan node mana pun yang berhasil membuat *Handler* ini akan menjadi *leader*. Jika sesi sudah *expired* atau koneksinya hilang maka node lainnya akan mendapatkan kesempatan untuk menjadi *leader*. Namun beberapa saat kemudian, *leader* yang lama akan kembali lagi sehingga mengakibatkan adanya 2 *leader* pada saat yang bersamaan. Akibatnya ActiveStandbyElector harus membuat ActiveBreadCrumb yang dapat menyesuaikan datanya sendiri sehingga *leader* yang baru dapat melakukan *fence* sebelum *leader* tersebut menjadi aktif. *Fence* merupakan *method* (org.apache.hadoop.ha.ShellCommandFencer) yang digunakan untuk menentukan sebuah node agar mencegah node lainnya dalam melakukan progress secara kontinyu. Pada umumya, ketika *Standby* NameNode menjadi *Active* NameNode maka diperlukan *fence* kepada *Active* NameNode sebelumnya untuk memastikan bahwa NameNode itu sudah tidak dapat melakukan edit log lagi.

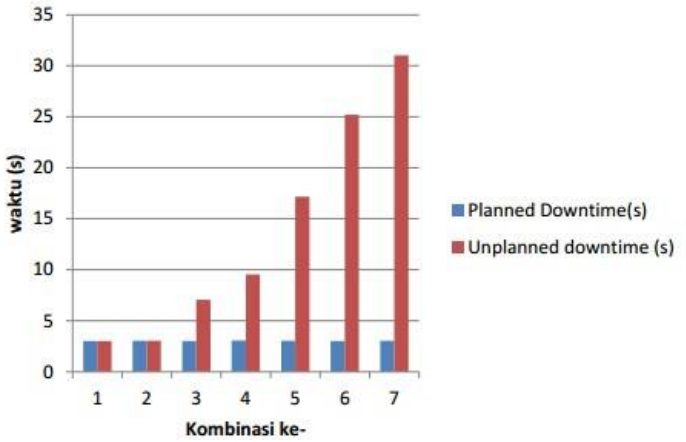
3.4 Analisis Terhadap Jumlah Blok Data

Berdasarkan skenario yang sesuai dengan paper IBM [3] maka dilakukan percobaan berdasarkan jumlah blok data. Namun hasil yang didapatkan tidak sama dengan paper tersebut. Dengan data yang sudah di dapatkan, maka dapat di analisa bahwa blok data dalam jumlah besar tidak mempengaruhi waktu *failover*. Dari data percobaan yang didapatkan maka dapat terlihat bahwa tidak terjadi peningkatan yang signifikan saat *downtime failover* pada percobaan yang dilakukan. Hal ini dapat terlihat bahwa diagram batang dari hasil

percobaan menunjukkan adanya peningkatan bukan karena bertambahnya jumlah blok data melainkan karena ada proses lain yang sedang dilakukan oleh NameNode sehingga *downtime failover* menjadi lebih lama. Maka dari itu, dapat dilihat secara keseluruhan bahwa peningkatan jumlah blok data tidak mempengaruhi waktu *failover* tetapi waktu *failover* berpengaruh pada proses yang sedang terjadi pada *Standby Node*.

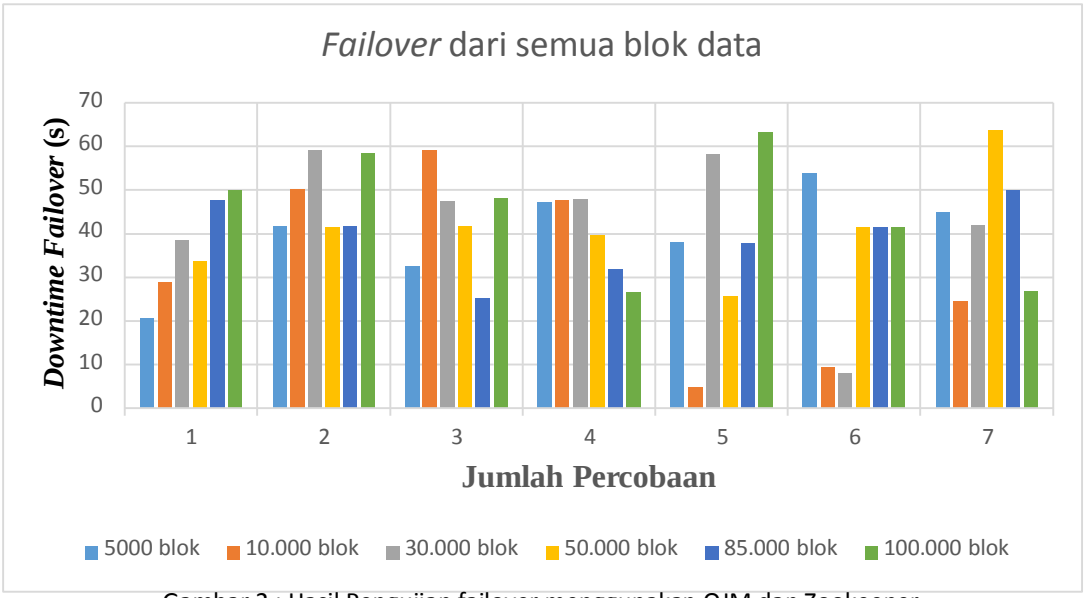
Sedangkan berdasarkan pada paper IBM, dijelaskan bahwa proses *failover* yang didapatkan dari *metadata replication* akan dipengaruhi oleh jumlah metadata yang disimpan dalam NameNode itu sendiri.[3] Tentunya ini berbeda dengan Zookeeper yang dipengaruhi oleh proses yang sedang berjalan di *Standby Node*. Selain itu, Zookeeper menyimpan dan mensinkronisasi log-log pada semua NameNode yang ada.

Pada metode *metadata replication* memindahkan metadata ketika *server* master sudah dipastikan mati/*fail* sehingga saat pemindahan metadata sudah selesai baru dapat dilakukan proses blok mapping. Tetapi Zookeeper menggunakan metode *wait-free coordination* yang menggunakan log4j untuk melihat keadaan dari setiap NameNode lalu melakukan koordinasi antara *Active* dan *Standby* NameNode. Selain itu, proses pemindahan metadata dilakukan secara langsung ketika *user* menggunakan QJM.



Gambar 1 : Hasil Pengujian Penelitian failover menggunakan DRBD dan Heartbeat

Selain itu, pada penelitian DRBD dan Heartbeat dijelaskan bahwa *downtime failover* akan menjadi lebih lama ketika skenario yang digunakan adalah *Unplanned downtime* (terjadi peristiwa diluar dugaan).[6] Sehingga pada akhirnya *downtime failover* yang terjadi akan mengalami peningkatan yang signifikan terutama untuk rencana jangka panjang.



Gambar 2 : Hasil Pengujian failover menggunakan QJM dan Zookeeper

Tentunya hal ini berbeda dengan percobaan menggunakan Zookeeper dan QJM yang menghasilkan *downtime failover* yang relatif sama tergantung dari proses yang sedang terjadi pada NameNode. Sehingga untuk penggunaan jangka panjang, Zookeeper dan QJM memberikan *downtime failover* yang lebih baik.

4. Kesimpulan

Berikut ini beberapa kesimpulan yang dapat diambil dari pembuatan Tugas Akhir ini, diantaranya :

- Jumlah blok data tidak mempengaruhi *downtime failover* pada *Cluster* Hadoop
- Jika dibandingkan dengan metode DRBD dan Heartbeat maka metode Zookeeper dan QJM yang digunakan sekarang lebih baik untuk digunakan dalam jangka panjang. Hal ini disebabkan *downtime failover* yang relatif stabil tanpa adanya peningkatan *downtime* terhadap jumlah blok data.
- Keunggulan Zookeeper lainnya adalah API yang sederhana sehingga mudah untuk diperlajari, dipahami & di implementasikan.
- Berdasarkan percobaan yang dilakukan maka kelemahan Zookeeper adalah pada saat *downtime failover* perlu dilakukan *fencing* pada *Active* NameNode sehingga *downtime*-nya menjadi lebih lama.

5. Saran

Adapun beberapa saran yang dapat digunakan untuk mengembangkan penelitian ini lebih jauh, diantaranya :

1. Dapat dikembangkan metode lainnya yang dapat memberikan hasil *downtime failover* yang lebih cepat.
2. Diperlukan penelitian tentang keamanan dari *Cluster* Hadoop maupun Zookeeper itu sendiri.
3. Dapat diteliti lebih jauh tentang proses *upload* dan *download* dari klien ke HDFS sehingga dapat memberikan *availability* yang maksimal.
4. Terdapat *High Availability* pada bagian MapReduce yang masih dapat diteliti lebih jauh terhadap kinerjanya pada *Cluster* Hadoop.

Daftar Pustaka

[1] Holmes, Alex. (2012). *Hadoop in Practice*. New York : Manning Publications

[2] Junqueira, Flavio. P, et al. 2010. *Zookeeper : Wait-free coordination for Internet-scale systems*. USA : Yahoo Research

[3] Purnomo, Nanang. (2012). *Pemanfaatan Failover Cluster Server Guna Recovery Sistem Pada PT.Lintas Data Prima*. Skripsi Sarjana pada Sekolah Tinggi Manajemen Informatika dan Komputer : tidak diterbitkan.

[4] Cloudera : Introduction to HDFS High Availability http://www.cloudera.com/content/cloudera/en/documentation/cdh4/latest/CDH4-High-Availability-Guide/cdh4hag_topic_2_1.html akses terakhir 22 Februari 2015

[5] Zookeeper : Overview https://zookeeper.apache.org/doc/trunk/zookeeperOver.html akses terakhir 4 Mei 2015

[6] Hadoop : HDFS High Availability Using The Quorum Journal Manager http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithQJM.html akses terakhir 22 Mei 2015