TUGAS AKHIR - KI141502

RANCANG *GATEWAY CLIEN* UNTUK *MULTIPLE TOPIC-BASED PUBLISH-SUBSCRIBE SERVER* DENGAN MEKANISME *CONTENT FILTERING*

NAUFAL FAKHRI MUHAMMAD
NRP 5113 100 024

Dosen Pembimbing I
Waskitho Wibisono, S.Kom., M.Eng., Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - KI141502

RANCANG *GATEWAY CLIEN* UNTUK *MULTIPLE TOPIC-BASED PUBLISH-SUBSCRIBE SERVER* DENGAN MEKANISME *CONTENT FILTERING*

NAUFAL FAKHRI MUHAMMAD
NRP 5113 100 024

Dosen Pembimbing I
Waskitho Wibisono, S.Kom., M.Eng., Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

BUILDING GATEWAY CLIENT FOR MULTIPLE TOPIC-BASED PUBLISH-SUBSCRIBE WITH CONTENT FILTERING MECHANISM

NAUFAL FAKHRI MUHAMMAD
NRP 5113 100 024

Supervisor I
Waskitho Wibisono, S.Kom., M.Eng., Ph.D

Supervisor II
Bagus Jati Santoso, S.Kom., Ph.D

Department of INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

RANCANG *GATEWAY CLIENT* UNTUK *MULTIPLE TOPIC-BASED PUBLISH-SUBSCRIBE SERVER* DENGAN MEKANISME *CONTENT FILTERING*

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :

NAUFAL FAKHRI MUHAMMAD
NRP: 5113 100 024

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Waskitho Wibisono, S.Kom., M.Eng., Ph.D
NIP: 197410222000031001 ..... (Pembimbing 1)

Bagus Jati Santoso, S.Kom., Ph.D
NIP: 051100116 ..... (Pembimbing 2)

SURABAYA
Juli 2017

(Halaman ini sengaja dikosongkan)

RANCANG *GATEWAY CLIEN* UNTUK *MULTIPLE TOPIC-BASED PUBLISH-SUBSCRIBE SERVER* DENGAN MEKANISME *CONTENT FILTERING*

Nama	:	**NAUFAL FAKHRI MUHAMMAD**
NRP	:	**5113 100 024**
Jurusan	:	**Teknik Informatika FTIf**
Pembimbing I	:	**Waskitho Wibisono, S.Kom., M.Eng., Ph.D**
Pembimbing II	:	**Bagus Jati Santoso, S.Kom., Ph.D**

Abstrak

Perkembangan teknologi menyebabkan informasi semakin mudah dan cepat didapatkan. Informasi yang didapatkan tentunya sangat beragam. Selain itu kebutuhan terhadap pengiriman pesan secara real-time juga semakin meningkat. Topic-based publish-subscribe merupakan salah satu pola pengiriman pesan yang mulai berkembang. Kelemahan pada topic-based publish-subcribe adalah ketika pengguna ingin memperoleh suatu pesan berdasarkan isinya. Pada pengerjaan tugas akhir ini nantinya akan dibuat gateway client yang bertujuan untuk menyelesaikan masalah tersebut. Gateway client yang akan dibangun nantinya akan memungkinkan pengguna untuk mendapatkan pesan berdasarkan isinya. Pencocokan antara kata kunci (query pencarian) oleh pengguna dengan isi pesan menggunakan cosine similarity. Sistem gateway client yang dibuat diuji beberapa performanya. Pada uji kecepatan menghasilkan peningkatan kecepatan rata-rata 98%. Pada uji ketahanan, sistem dapat melayani hingga 600 permintaan. Pada uji ketepatan menghasilkan recall sebesar 64%. Pada uji sumber daya terdapat penghematan penyimpanan dengan hanya mengalami peningkatan rata-rata sebesar 6.7%

penyimpanan dari jumlah permintaan sebelumnya. Dari hasil uji coba yang telah dilakukan dapat disimpulkan bahwa sistem yang dibuat memiliki kecepatan pemrosesan yang tinggi.
***Kata-Kunci**: Publish-Subscribe, Real Time, Information Retrieval*

BUILDING GATEWAY CLIENT FOR MULTIPLE TOPIC-BASED PUBLISH-SUBSCRIBE WITH CONTENT FILTERING MECHANISM

Name	:	**NAUFAL FAKHRI MUHAMMAD**
NRP	:	**5113 100 024**
Major	:	**Informatics FTIf**
Supervisor I	:	**Waskitho Wibisono, S.Kom., M.Eng., Ph.D**
Supervisor II	:	**Bagus Jati Santoso, S.Kom., Ph.D**

Abstract

Technological developments makes information easily to get. There are many kind of information. Other than that, needs of real time message increase. Topic based publish-subscribe is one of real time message pattern. Disadvantages of topic based publish-subscribe is when users want to get information based on its content. In this thesis client gateway that aim to solve that problem will be created. The gateway client allow users to get message (information) based on its content. Calculating similarity between keyword from users and message using cosine similarity. The gateway client performance will be test. In speed test, it shows average increment of speed 98%. In durability test, it shows that system can handle up to 600 request. In query test, it shows 64% recall. In resource test, it shows that the average increment of memory used is 6.7% of previous number of request. Based on test that have been done can be conclude that the gateway client system has high speed processing.

***Keyword**: Publish-Subscribe, Real Time, Information Retrieval*

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Rancang *Gateway Clien* Untuk *Multiple Topic-Based Publish-Subscribe Server* Dengan Mekanisme *Content Filtering***. Pengerjaan Tugas Akhir ini juga menjadi sarana bagi penulis untuk menambah ilmu tentang pembuatan *server* yang bersifat *asynchronous* dan *real-time*. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT atas anugerahnya yang tidak terkira kepada penulis.
2. Keluarga atas dukungan dan doanya.
3. Dosen pembimbing yang memberikan arahan.
4. Laboratorium NCC
5. Laboratorium MI
6. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2017

Naufal Fakhri M.

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB 1

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Perkembangan teknologi menyebabkan informasi semakin mudah dan cepat didapatkan. Informasi yang didapatkan sangat beragam, mulai dari hal yang bersifat tidak terlalu penting sampai yang sangat penting. Semakin berkembangnya teknologi menuntut suatu informasi dapat disampaikan secara cepat bahkan secara *real-time* kepada pihak yang membutuhkan. Informasi yang disampaikan dapat berupa berita mengenai kejadian-kejadian yang sedang disoroti oleh khalayak umum maupun hasil dari suatu pertandingan sepakbola.

Salah satu pola pesan pada sumber informasi adalah *publish-subscribe*. *Publish-subscribe* merupakan pola pesan dimana pengirim pesan (*Publisher*) tidak mengirimkan pesan secara langsung kepada penerima (*Subscriber*) melainkan kepada penampung pesan (*Message broker*). Pengirim pesan dan penerima pesan tidak tahu menahu satu sama lain.

Pada umumnya terdapat dua jenis *publish-subscribe*, yaitu *topic-based publish-subscribe* dan *content-based publish-subscribe*. *Topic-based publish-subscribe* merupakan *publish-subscribe* dimana *subscriber* mendapatkan pesan berdasarkan topik yang diminatinya. Sedangkan *content-based publish-subscribe* merupakan *publish-subscribe* dimana *subscriber* mendapatkan pesan berdasarkan isi dari pesan yang diminati.

Kebanyakan *publish-subscribe* merupakan *topic-based publish-subscribe*. *Topic-based publish-subscribe* memiliki

beberapa kelemahan. Pertama tidak leluasa karena *subscriber* hanya dapat memilih sesuai dengan topik disediakan *publisher*. Kelemahan kedua yaitu kesulitan memilih topik karena *subscriber* hanya menginginkan suatu pesan berdasarkan isinya. Pada *publish-subscribe* dikenal adanya *content-filtering*. *Content-filtering publish-subscribe* merupakan suatu *publish-subscribe* yang menyampaikan informasi sesuai dengan isi dari pesan yang ada. Maka dari itu perlu adanya sitem yang menjembatani antara *topic-based publish-subscribe* dan pengguna agar dapat memberikan pesan sesuai dengan isi yang diinginkan.

Hasil yang diharapkan dari pengerjaan tugas akhir ini adalah berupa sistem yang dapat menjembatani antara *topic-based publish-subscribe* dan pengguna supaya pengguna mendapatkan pesan sesuai dengan isi yang diinginkannya.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut :

1. Bagaimana membuat *gateway client* yang menghubungkan beberapa *topic-based publish-subscribe*?
2. Bagaimana melakukan *content-filtering* pada *topic-based publish-subscribe*?
3. Bagaimana mengurutkan peringkat pada hasil *content filtering*?

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Data yang diolah hanya teks.
2. *Content filtering* dilakukan pada teks.

3. Data *publish-subscribe* dikumpulkan menggunakan *web crawling*.
4. Pesan pada *publish-subscribe* dan *query* menggunakan bahasa Indonesia.
5. Sistem yang dibuat berbasis web

1.4 Tujuan

Tujuan dari pengerjaan Tugas Akhir ini adalah :

1. Membuat sistem yang dapat menjembatani antara *topic-based publish-subscribe* dan pengguna dengan mekanisme *content-filtering*.

1.5 Manfaat

Hasil dari pengerjaan Tugas Akhir ini memiliki manfaat untuk menghasilkan sebuah sistem yang dapat menjembatani *topic-based publish-subscribe* dengan pengguna dengan melakukan *content-filtering*.

1.6 Metodologi

1. Studi literatur
   Studi literatur yang dilakukan dalam pengerjaan Tugas Akhir ini adalah mengenai pola pesan *topic-based publish-subscribe* yang akan dijadikan landasan sistem yang akan dibuat. Selain itu juga dilakukan studi literature mengenai *web crawling* yang akan digunakan untuk data sistem *publish-subscribe*. Studi literatur lain yang dilakukan mengenai cosine similarity untuk melakukan pencocokan query dari *subscriber* terhadap isi dari pesan. Sehingga nantinya dapat dibuat sistem yang dapat melakukan *content-filtering* pada *topic-based publish-subscribe*.

2. Analisis dan Desain Perangkat Lunak
   Pada tahap ini disusun rancang bangun dari sistem yang akan dibangun. Pengguna memasukkan *query* yang diinginkan. *Query* yang dimasukkan oleh pengguna nantinya akan dicocokan dengan isi dari pesan yang dikirim oleh *publisher* dengan menggunakan metode *cosine-similarity*. Hasil dari pencocokan nantinya akan diurutkan berdasarkan nilai *cosine-similarity* yang paling tinggi sehingga pengguna mendapatkan hasil yang paling cocok berdasarkan *query* yang dimasukkan.
3. Implementasi Perangkat Lunak
   Sistem yang akan dibuat menggunakan HTML dan python. HTML akan digunakan sebagai penghubung antara *web-client* dengan sistem *content-filtering* yang dibangun. Penerapan sistem *publish-subscribe* akan menggunakan RabbitMQ. Sistem *content-filtering* yang dibangun akan menggunakan bahasa pemrograman python. Penerapan *content-filtering* menggunakan *cosine-similarity* akan memanfaatkan *library* scipy.
4. Uji Coba dan Evaluasi
   Beberapa hal yang akan diuji coba pada performa sistem yang akan dibuat yaitu kecepatan, ketahanan, sumber daya dan uji keakuratan *query* pencarian.

1.7 Sistematika Laporan

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna bagi pembaca yang berminat melakukan pengambangan lebih lanjut. Secara garis besar, buku tugas akhir ini terdiri atas beberapa bagian seperti berikut:

Bab I Pendahuluan

Bab yang berisi latar belakang, tujuan, manfaat,

permasalahan, batasan masalah, metodologi yang digunakan dan sistematika penulisan.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang yang digunakan dalam pembuatan tugas akhir ini.

Bab II Analisis dan Perancangan

Bab ini berisi tentang analisis dan perancangan sistem *content-filtering* yang akan dibuat.

Bab IV Implementasi

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa kode program yang digunakan dalam implementasi.

Bab V Uji Coba dan Evaluasi

Bab ini membahas tahap-tahap uji coba serta melakukan evaluasi terhadap sistem yang dibuat.

Bab VI Kesimpulan dan Saran

Bab ini merupakan bab terakhir yang memberikan kesimpulan dari hasil percobaan dan evaluasi yang telah dilakukan. Pada bab ini juga terdapat saran bagi pembaca yang berminat untuk melakukan pengembangan lebih lanjut.

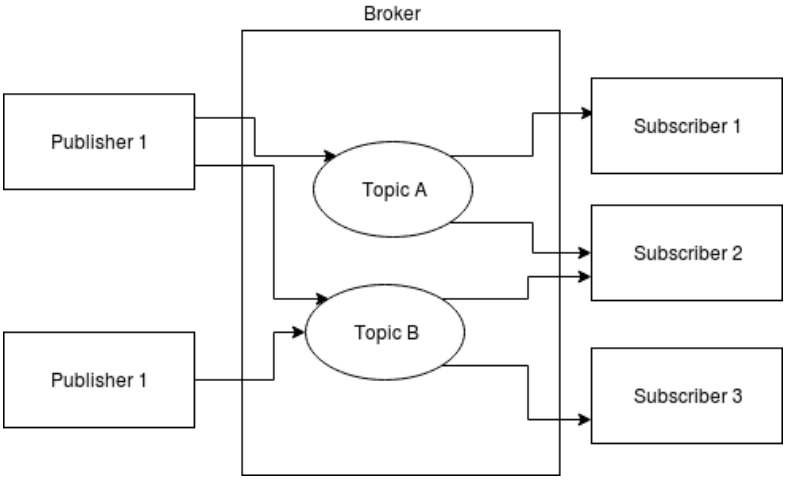
(Halaman ini sengaja dikosongkan)

BAB 2

LANDASAN TEORI

2.1 *Publish-Subscribe*

Publish-subscribe merupakan pola pengiriman pesan dimana pengirim pesan (*publisher*) tidak mengirimkan pesan secara langsung kepada penerima pesan (*subscriber*). Perantara yang menangani pengiriman pesan disebut *broker*. *Broker* berperan dalam menentukan suatu pesan dari *publisher* akan dikirim kepada satu *subscriber* saja atau bahkan lebih. Biasanya pesan yang dikirim oleh *publisher* akan dikelompokan berdasarkan topik atau kelas. *Subscriber* nantinya akan memberitahu *broker* topik apa saja yang diminati. Pada pola pengiriman pesan *publish-subscribe* ini pengirim pesan (*publisher*) tidak mengetahui siapa saja yang akan menerima pesan yang dikirimnya, atau mungkin saja pesan yang dikirimnya tidak akan diterima oleh siapapun. Penerima pesan (*subscriber*) hanya akan menerima pesan berdasarkan satu atau lebih topik yang diminati. Terdapat dua jenis *publish-subscribe*, *topic based publish-subscribe* dan *content based publish-subscribe*. *Topic based publish-subscribe* merupakan pola pengiriman pesan *publish-subscribe* dimana *publisher* menyampaikan pesan ke *subscriber* berdasarkan topik yang dipilihnya. *Content based publish-subscribe* merupakan pola pengiriman pesan *publish-subscribe* dimana *publisher* menyampaikan pesan ke *subscriber* berdasarkan isi dari pesan yang ada. Pada umumnya pola pengiriman pesan *publish-subscribe* merupakan *topic based publish-subscribe*. Untuk lebih jelasnya dapat dilihat gambar 2.1 dibawah.



Gambar 2.1: Pola Pengiriman Pesan *Topic Based Publish-Subscribe*

Kelebihan dari pola pengiriman pesan *publish-subscribe* ini terletak pada kelonggaran hubungan antara *publisher* dengan *subscriber*. Hal tersebut dikarenakan semakin berkembangnya teknologi menuntut adanya suatu model komunikasi yang lebih fleksibel[12]. Terdapat beberapa pemisahan (*decoupling*) pada pola pengiriman pesan *publish-subscribe* yang menghasilkan kelonggaran hubungan antara *publisher* dan *subscriber*, pemisahan tersebut antara lain :

1. **Pemisahan ruang (*space decoupling*)** : pihak-pihak yang berkomunikasi tidak perlu saling mengetahui satu sama lain. Seperti yang sudah dibahas pada penjelasan tentang pola pengiriman pesan *publish-subscribe* diatas, bahwa *publisher* tidak mengetahui siapa yang akan mendapatkan pesan yang dibuatnya. *Broker* yang akan mengatur *subscriber* yang mendapatkan pesan yang dibuat oleh *publisher*.
2. **Pemisahan waktu (*time decoupling*)** : pihak-pihak yang

berkomunikasi tidak harus aktif pada waktu yang bersamaan. *Publisher* tidak harus menunggu *subscriber* dalam keadaan aktif untuk mengirim pesan, karena pesan yang dibuat akan dikirim ke *broker*. *Publisher* mungkin saja mengirim pesan ketika *subscriber* yang bersangkutan sedang tidak aktif. *Subscriber* juga mungkin juga mendapatkan pesan ketika *publisher* asli sedang tidak aktif.

3. **Pemisahan sinkronisasi (*synchronization decoupling*)** : *subscriber* tidak diblokir ketika menerima pesan yang diproduksi *publisher*, dan begitu juga sebaliknya. Ketika *subscriber* menerima pesan dari *broker*, *subscriber* masih dapat melakukan proses lainnya.

2.2 RabbitMQ

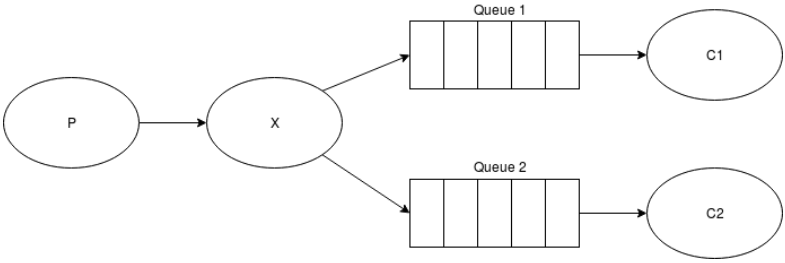
RabbitMQ[1] merupakan perantara (*broker*) pada pertukaran pesan[13]. RabbitMQ juga biasa disebut *message-oriented middleware*. RabbitMQ mendukung pola pengiriman pesan *publish-subscribe*.

Pada arsitektur pola pengiriman pesan *publish-subscribe* yang terdapat di RabbitMQ akan sering ditemui istilah-istilah seperti *exchange*, *queue*. *Exchange* dapat diartikan sebagai persimpangan sedangkan *queue* dapat diartikan sebagai tempat penyimpanan. RabbitMQ menggunakan istilah *producer* untuk pihak yang membuat pesan untuk nantinya dikirim ke *broker* dan *consumer* untuk pihak yang nantinya akan menerima pesan dari *broker* yang dibuat oleh *producer*. *Producer* pada pola pengiriman pesan *publish-subscribe* dikenal dengan istilah *publisher* sedangkan *consumer* dikenal dengan istilah *subscriber*.

Pada arsitektur pola pengiriman pesan *publish-subscribe* pesan yang dibuat oleh *publisher* mungkin saja di konsumsi oleh

[1]RabbitMQ dapat di unduh di https://www.rabbitmq.com/download.html

beberapa *subscriber*. *Publisher* nantinya akan mengirim pesan ke suatu *exchange* yang berada pada broker, akan tetapi *exchange* tidak dapat menyimpan pesan. Ketika suatu *exchange* yang dikirimi pesan tidak memiliki *queue* yang tersambung, maka pesan tersebut akan hilang. Pada kasus ini masing-masing *subscriber* akan membuat *queue* yang berbeda untuk menyimpan pesan yang dibuat oleh *publisher*. Oleh RabbitMQ pesan yang sudah pernah dipakai (*consume*) akan langsung dihapus dari *queue*. Untuk mendapatkan gambaran yang lebih jelas dari *producer* (P), *exchange* (X), *queue* dan *consumer* (C) dapat dilihat gambar 2.2 dibawah.



Gambar 2.2: Ilustrasi RabbitMQ *Publish-Subscribe*

RabbitMQ menerapkan AMQP 0-9-1. AMQP merupakan standar protokol lapisan aplikasi yang terbuka untuk *message-oriented middleware*. Penggunaan RabbitMQ didukung oleh banyak bahasa pemrograman, salah satunya python. Pika merupakan python *client* yang sama-sama menerapkan AMQP 0-9-1 yang dapat digunakan untuk memprogram RabbitMQ.

2.3 *Information Retrieval*

Information retrieval (IR) adalah menemukan materi (biasanya dokumen) dari data yang bersifat tidak terstruktur (biasanya teks) yang memenuhi kebutuhan dari koleksi yang

besar [11].

Information retrieval sudah menjadi bagian dalam kehidupan sehari-hari. Ketika seseorang mencari informasi menggunakan mesin pencari maupun ketika seseorang mencari suatu email dari tumpukan email yang ada, itu semua dapat disebut sebagai *information retrieval*.

Istilah data tidak terstruktur sebenarnya mengacu pada data yang memiliki struktur kurang jelas, tidak seperti database relasional yang biasanya digunakan perusahaan-perusahaan untuk menyimpan data tentang inventaris maupun karyawan[11].

Dengan perkembangan teknologi yang cepat seperti saat ini, pencarian suatu *query* biasa pada suatu dokumen bisa dilakukan dengan sangat cepat. Akan tetapi untuk dapat melakukan hal yang lebih dalam melakukan *information retrieval* terdapat beberapa hal yang perlu disoroti, beberapa hal tersebut antara lain:

1. Memproses data yang sangat banyak dengan cepat.
2. Melakukan proses pencocokan *query* yang lebih fleksibel.
3. Melakukan pengurutan berdasarkan peringkat kecocokan.

Sastrawi merupakan *library* yang digunakan untuk mengubah kata dalam bahasa Indonesia menjadi kata dasarnya. Ketika berurusan dengan dokumen-dokumen teks tentu saja diperlukan adanya *library* yang digunakan untun mengubah kata-kata yang berimbuhan menjadi kata dasarnya supaya mendapatkan hasil yang lebih akurat.

Teks dokumen dan *query* yang akan dihitung kemiripannya perlu diubah kedalam *vector* untuk nantinya dihitung menggunakan *cosine similarity*. Dalam mengubah teks menjadi *vector* pengubahan menjadi kata dasar sangat penting untuk mendapatkan hasil yang maksimal.

2.4 *Term Frequency–Inverse Document Frequency* (TF-IDF)

Term frequency–inverse document frequency (Tf-idf) adalah cara yang digunakan untuk menghitung bobot dari istilah-istilah yang terdapat pada kumpulan dokumen[10]. Tf-idf dapat digunakan untuk merepresentasikan kata-kata kedalam angka-angka sehingga nantinya akan dapat dilakukan perhitungan.

Term frequency–inverse document frequency (Tf-idf) terdiri dari dua istilah *term frequency* dan *inverse document frequency*. Jumlah suatu kata dalam satu dokumen disebut *term frequency*. *Term frequency* (tf) digunakan untuk mengetahui seberapa sering suatu kata digunakan dalam satu dokumen. Ketika suatu kata sering digunakan dalam suatu dokumen ada kemungkinan kata tersebut merupakan kata yang penting sehingga diulang-ulang dalam dokumen tersebut. Akan tetapi kata yang sering diulang-ulang dalam suatu dokumen tidak dapat dijadikan sebagai pembeda pada masing-masing dokumen, maka dari itu terdapat *inverse document frequency* (idf) untuk menangani hal tersebut. *Inverse document frequency* merupakan jumlah suatu kata yang ditemukan dalam berapa dokumen dalam suatu korpus. *Inverse document frequency* dapat memperlihatkan seberapa unik suatu kata tersebut dalam sebuah korpus. Ketika suatu kata jarang digunakan dalam sebuah korpus, dapat disimpulkan bahwa kata tersebut dapat digunakan sebagai pembeda antar dokumen.

Persamaan 2.1 dibawah ini merupakan persamaan yang digunakan untuk penghitungan *term frequency* (pada scikit-learn)[4].

$$idf(t) = \log \frac{1+n}{1+df(d,t)} + 1 \tag{2.1}$$

n = jumlah dokumen pada korpus
$df(d,t)$ = jumlah dokumen yang mengandung kata t

2.5 *Cosine Similarity*

Cosine similarity adalah ukuran kesamaan antara dua *vector* yang tidak nol dari *inner product* yang mengukur cosinus dari sudut antara kedua *vector* tersebut. *Cosine similarity* merupakan salah satu cara yang digunakan untuk mengukur kesamaan suatu *query* dengan dokumen-dokumen pada korpus. *Cosine similarity* dinilai lebih baik dibandingkan beberapa cara pengukuran kesamaan lainnya untuk pengukuran kesamaan terutama pada dokumen teks[9].

Dari penjelasan diatas, *cosine similarity* digunakan untuk menghitung kesamaan/kemiripan antara dua *vector*. Sebagai contoh *vector* yang dibandingkan adalah *vector* dari *query* pencarian (q) dengan dokumen A, maka persamaan *cosine similarity*[3] dapat ditulis menjadi seperti berikut.

$$cos(q, A) = \frac{\vec{q} \cdot \vec{A}}{\|\vec{q}\| \left\|\vec{A}\right\|} \tag{2.2}$$

$\vec{q}$ = *vector* dari *query*
$\vec{A}$ = *vector* dari dokumen A
$\|\vec{q}\|$ = *Euclidean lengths* dari *query*
$\left\|\vec{A}\right\|$ = *Euclidean lengths* dari dokumen A

Pada persamaan 2.2, pembilang merupakan *dot product* dari kedua *vector* yang dibandingkan, dalam contoh kasus berarti

vector dari *query* dengan dokumen A. Persamaan *dot product*[3] ditulis pada persamaan 2.3 berikut.

$$\vec{q} \cdot \vec{A} = \sum_{i=1}^{n} q_i A_i \tag{2.3}$$

q_i = indeks ke i dari *query*
A_i = indeks ke i dari dokumen A

Penyebut dari penghitungan *cosine similarity* merupakan *euclidean length* dari *vector* yang dibandingkan. Persamaan dari *euclidean length*[3] dari contoh kasus dapat ditulis pada persamaan 2.4 berikut.

$$\|\vec{q}\| = \sqrt{\sum_{i=1}^{n} q_i^2} \tag{2.4}$$

$$\left\|\vec{A}\right\| = \sqrt{\sum_{i=1}^{n} A_i^2} \tag{2.5}$$

q_i = indeks ke i dari *query*
A_i = indeks ke i dari dokumen A

Pada bahasa pemrograman sudah terdapat banyak *library* yang mendukung penghitungan *cosine-similarity*. Pada python terdapat *library* yang disebut scikit-learn. Scikit-learn merupakan *machine learning library* pada bahasa python. Scikit-learn memiliki banyak fitur, salah satunya yaitu penghitungan *cosine-similarity*.

2.6 *Web Socket*

Web socket merupakan protokol komunikasi yang memungkinkan komunikasi dua arah antara *web client* dengan

server[8]. Penerapan komunikasi dua arah ini dapat digambarkan oleh pengguna telepon. Kedua orang yang sedang bertelepon dapat berbicara dan mendengarkan secara bersamaan dan *real-time*. Protokol *web socket* memungkinkan komunikasi dua arah antara *client* dengan *server* secara *real-time*.

Pada umumnya *web client* mendapatkan satu kali tanggapan dari *server* untuk setiap satu perminataan. Alur tersebut kurang tepat apabila digunakan untuk menerapkan pola pengiriman pesan *publish-subscribe* dimana *subscriber* biasanya akan menunggu terus menerus terhadap pesan yang mungkin oleh *publisher*. Sedangkan *publisher* mungkin saja mengirimkan pesan lagi setelah 5 menit. Maka dari itu *web socket* digunakan untuk menampilkan secara *real-time* pesan yang diperoleh dari *publisher* walaupun harus menunggu dalam jangka waktu yang tidak pasti.

2.7 Tornado

Tornado[2] merupakan *non-blocking web server* dan *web framework* yang ditulis dalam bahasa pemrograman python. *Non-blocking* berarti dapat mengerjakan proses lain. Layaknya yang terjadi pada *subscriber* yang masih dapat mengerjakan proses lain ketika menunggu pesan dari *publisher*. Tornado sangat cocok digunakan untuk melakukan proses yang membutuhkan koneksi dalam jangka waktu lama[7] karena merupakan *asynchronous server*. *Asynchronous* berarti Tornado melayani permintaan akan tetapi tidak menjalankan semua proses sampai selesai melainkan akan kembali kepada proses utama sehingga nantinya menyebabkan permintaan yang belum diselesaikan akan dikerjakan di *background* [6].

Selain tornado cocok digunakan pada proses yang membutuhkan koneksi dalam jangka waktu lama, tornado juga

[2]Tornado dapat diunduh di https://github.com/tornadoweb/tornado

ideal digunakan untuk *web socket* karena dapat membuka hingga puluhan ribu koneksi. Penerapan tornado dengan *web socket* cocok digunakan pada pola pengiriman pesan *publish-subscribe* yang tentunya akan membutuhkan koneksi yang banyak serta koneksi dalam jangka waktu lama.

2.7.1 *Asynchronous I/O*

Seperti yang sudah disinggung diatas bahwa *asynchronous* berarti fungsi akan kembali sebelum proses yang dijalankan selesai sehingga terdapat proses yang dijalankan di *background*. *Asynchronous I/O* dapat digambarkan sebagai buka tutup sumber daya untuk masing-masing proses. Penggunaan *asynchronous* ini sangat berguna untuk koneksi yang membutuhkan waktu hidup dalam jangka waktu yang lama. Penggunaan *asynchronous* cocok digunakan pada pola pengiriman pesan *publish-subscribe* karena *subscriber* akan menunggu secara terus menerus untuk mendapatkan pesan dari *publisher*. Kita ketahui bahwa *publisher* dapat mengirimkan pesan sewaktu-waktu, bahkan mungkin akan terdapat jeda waktu yang sangat panjang antara satu pesan dengan pesan yang lainnya.

2.7.2 *Thread Pool*

Thread pool merupakan kumpulan *thread* pekerja yang secara efisien memproses *asynchronous callback* atas nama aplikasi utama [5]. *Thread pool* bermanfaat untuk menjalankan proses yang membutuhkan waktu dan sumber daya yang banyak sehingga tidak mencegah *server* untuk melayani permintaan baru. *Thread pool* sangat berguna pada tornado karena tornado merupakan *web framework* yang *single thread*. Apabila terdapat proses yang membutuhkan waktu dan sumber daya yang banyak maka *server* akan terhalangi kinerjanya karena hanya ada satu proses yang dapat berjalan pada waktu yang sama. *Thread pool*

layaknya daftar tunggu suatu *thread* untuk mendapatkan giliran dalam menggunakan sumber daya yang ada. *Thread pool* juga berguna untuk membatasi jumlah *thread* yang ada sehingga tidak memakan *memory* yang terlalu banyak.

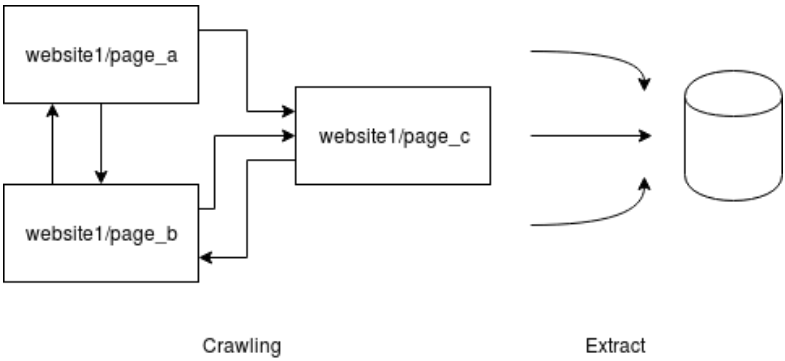
2.7.3 *Decorator*

Decorator merupakan sebuah fungsi yang mengambil fungsi lain dan memperluas perilaku dari fungsi terakhir tanpa secara eksplisit mengubah fungsi tersebut. *Decorator* dapat digunakan untuk menyederhanakan penulisan sebuah fungsi yang memiliki memiliki tingkah laku yang hampir sama.

2.8 *Web Crawling*

Web crawling biasa disebut *web spider* atau *web robot* juga merupakan proses pengambilan data dari website secara otomatis. Data yang dikumpulkan lalu akan diekstrak dan dilakukan pengindeksan yang nantinya akan diolah untuk keperluan tertentu. Scrapy[3] merupakan salah satu *open source web crawling framework* yang ditulis dalam bahasa pemrograman python. Gambar 2.3 merupakan ilustrasi dari *web crawling*.

[3]Scrapy dapat diunduh di https://scrapy.org/download/



Gambar 2.3: *Web Crawling*

BAB 3

DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai dasar perancangan sistem yang akan dibuat. Secara khusus akan dibahas mengenai deskripsi umum sistem, perancangan skenario dan arsitektur sistem.

3.1 Deskripsi Umum Sistem

Sistem yang akan dibuat merupakan sistem yang menjembatani antara pengguna dengan pola pengiriman pesan *topic based publish-subscribe*. Pengguna adalah orang yang mengirimkan *query* pencarian untuk nantinya dicocokan dengan pesan yang diperolah dari *broker*. *Broker* merupakan perantara yang menghubungkan *publisher* dengan *subscriber*. *Publisher* merupakan pihak yang membuat pesan yang kemudian dikirim ke *broker* untuk nantinya dikonsumsi oleh *subscriber* yang membutuhkan.

Setiap pengguna akan dibuatkan *subscriber* dan *filter* masing-masing. *Filter* merupakan proses yang menghitungkan kemiripan antara *query* pencarian dengan pesan yang diperolah dari *broker*. Setelah penghitungan kemiripan selesai dilakukan, pesan yang memiliki hasil penghitungan yang lebih besar dari nilai *threshold* akan dikirimkan ke pengguna. *Threshold* merupakan batas minimal nilai kemiripan untuk dapat dianggap mirip. Pesan yang dikirim ke pengguna akan diurutkan berdasarkan nilai kemiripan dari yang besar ke kecil.

3.2 Perancangan Skenario

Skenario yang berawal dari pengguna yang mengirimkan *query* pencarian ke *server*. *Server* bertugas menangani setiap *query* pencarian yang dikirim oleh pengguna. Oleh *server* setiap pengguna nantinya akan dibuatkan *subscriber* dan *filter*

masing-masing. Selain itu *server* juga bertugas membuatkan *web socket* yang digunakan untuk menghubungkan antara *web client* pengguna dengan *subscriber*. *Web socket* memungkinkan dua pihak yang terhubung untuk berkomunikasi secara dua arah, jadi antara *web client* pengguna dan *subscriber* akan dapat mengirimkan pesan dan menerima pesan secara bersamaan.

Subscriber akan mendapatkan pesan yang dikirim dari *broker*. Pesan yang diterima oleh *subscriber* merupakan hasil *crawling* yang nantinya di-*publish* ke *broker*. *Subscriber* akan menghitung nilai kemiripan antara *query* pencarian dengan pesan yang diterima menggunakan *filter* yang dibuatkan oleh *server*. Pesan yang hasil penghitungan kemiripannya memiliki nilai yang lebih besar atau sama dengan dari *threshold* akan dirimkan ke *web client* pengguna melalui *web socket*. Pesan yang ditampilkan di *web client* pengguna akan diurutkan berdasarkan nilai kemiripannya, mulai dari yang paling besar ke yang paling kecil.

3.3 Arsitektur Sistem

Pada Sub-bab ini, dibahas mengenai tahap analisis arsitektur, analisis teknologi dan desain sistem yang akan dibangun.

3.3.1 Desain Umum Sistem

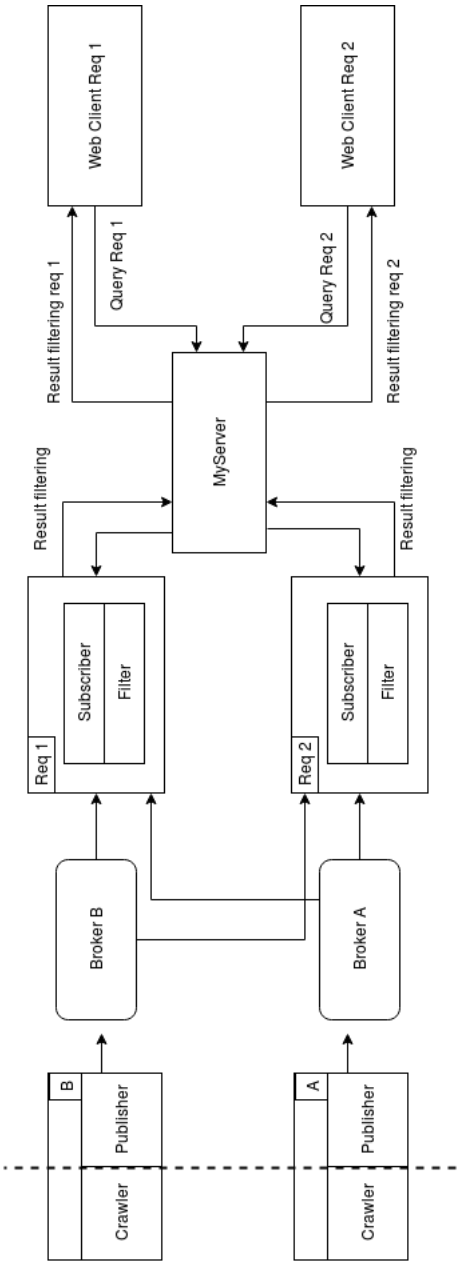
Sistem yang akan dibangun membutuhkan bagian-bagian agar dapat berjalan dengan baik. Bagian-bagian dari sistem tersebut antara lain:

1. *Web Server*
   Web server merupakan perangkat lunak yang melayani setiap permintaan yang dikirimkan olah *web client*.
2. *Broker*
   Pada pola pengiriman pesan *publish-subscribe*, *broker* berperan sebagai perantara antara *publisher* dengan *subscriber*. Pesan yang dari *publisher* tidak dapat dikirim

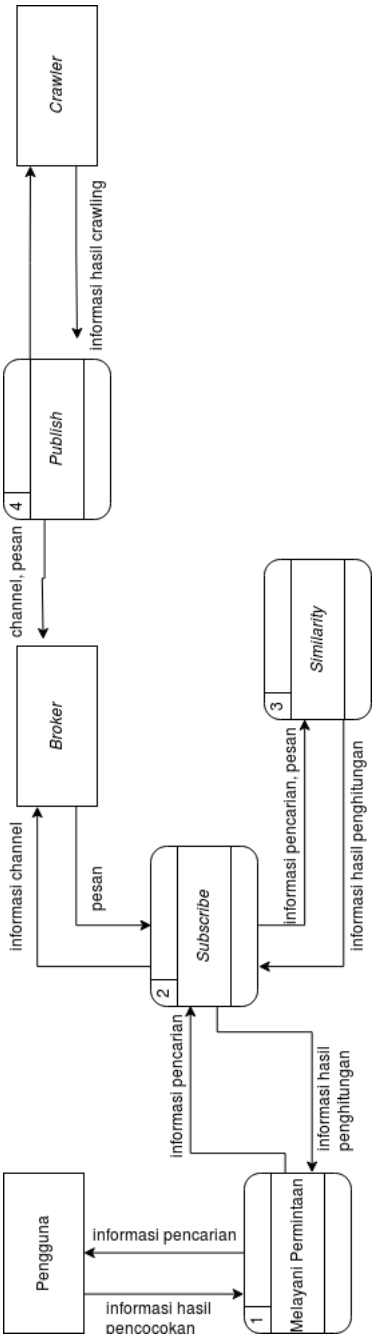
secara langsung ke *subscriber*. Pada pola pengiriman pesan *publish-subscribe* diperlukan perantara karena ada kemungkinan pesan dari *publisher* akan digunakan oleh beberapa *subscriber*.

3. *Publisher*
   Publisher merupakan pihak yang mengirimkan pesan ke *broker* untuk nantinya digunakan oleh *subscriber* yang bersangkutan.
4. *Subscriber*
   Subscriber merupakan pihak yang menggunakan pesan yang dikirim oleh *broker* yang dibuat *publisher*.
5. *Web Client*
   Web Client merupakan tampilan web yang berhubungan langsung dengan pengguna.
6. *Web Socket*
   Web socket merupakan protokol komunikasi yang memungkinkan kedua pihak yang berkomunikasi agar dapat saling mendengarkan dan mengirimkan pesan satu sama lain secara bersamaan.
7. *Filter*
   Filter merupakan proses penghitungan kemiripan antara *query* pencarian yang dikirimkan oleh pengguna dengan pesan yang didapat oleh *subscriber*.
8. *Crawler*
   Crawler merupakan pihak yang mengambil data dari internet. Data yang diambil nantinya akan digunakan sebagai pesan yang dikirim oleh *publisher* ke *broker* yang kemudian akan digunakan oleh *subscriber* yang bersangkutan.

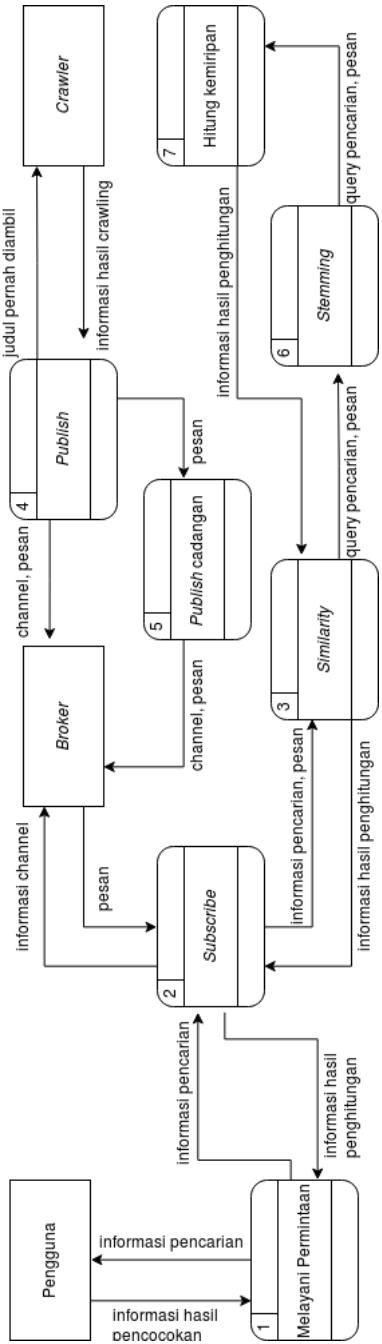
Gambar 3.1 merupakan gambaran umum dari sistem yang akan dibuat. Gambar 3.2 dan 3.3 merupakan gambar dari diagram alur data yang digunakan untuk mendapatkan gambaran yang lebih jelas terhadap sistem yang akan dibuat.



Gambar 3.1: Arsitektur Umum Sistem



Gambar 3.2: Diagram Alur Data Level 1



Gambar 3.3: Diagram Alur Data Level 2

3.3.2 Perancangan *Broker*

Pola pengiriman pesan *publish-subscribe* yang akan digunakan pada sistem yang akan dibangun membutuhkan sebuah *broker* yang berperan sebagai perantara antara *publisher* dan *subscriber*. Pola pengiriman pesan *publish-subscribe* yang akan digunakan pada sistem memanfaatkan RabbitMQ. RabbitMQ merupakan perantara (*broker*) pada pertukaran pesan. RabbitMQ juga biasa disebut *message-oriented middleware*. RabbitMQ mendukung pola pengiriman pesan *publish-subscribe*. Pemilihan RabbitMQ sebagai *broker* yang digunakan pada pola pengiriman pesan *publish-subscribe* ini karena RabbitMQ mendukung banyak bahasa pemrograman yang banyak, terutama python. Penggunaan RabbitMQ pada bahasa pemrograman python sudah banyak dilakukan sehingga memperbanyak refrensi yang diperlukan dalam pembangunan sistem. Pada python terdapat *client* yang digunakan untuk berkomunikasi dengan RabbitMQ yaitu pika.

3.3.3 Perancangan *Web Server*

Web server merupakan perangkat lunak yang melayani setiap permintaan yang dikirim oleh *web client*. Pada sistem yang akan dibangun, *web client* perlu untuk mengirim *query* pencarian ke *server*. Pada pola pengiriman pesan *publish-subscribe* alur pesan dari *publisher* menuju *subscriber* terjadi secara *real-time*. Yang dimaksud dengan *real-time* disini berarti *web client* tidak perlu mengirim permintaan secara terus menerus untuk menerima pesan yang berasal dari *publisher*. Setiap *web client* yang mengirimkan permintaan ke *web server* akan dibuatkan *subscriber*, *filter* dan *web socket* masing-masing.

Tornado merupakan *web framework* yang ditulis menggunakan bahasa pemrograman python. Penggunaan *web framework* tornado sangat cocok dalam pembuatan sistem karena

tornado merupakan *non-blocking web server* dan tornado dapat membuka koneksi yang banyak yang membutuhkan waktu hidup lama[7]. Selain hal tersebut *web framework* tornado juga cocok dikombinasikan dengan *web socket* yang nantinya diperlukan untuk menampilkan pesan secara *real time* kepada *web client*.

Tornado yang merupakan *single thread asynchronous I/O* cocok digunakan untuk *web socket* yang memakan waktu proses singkat. Akan tetapi untuk melakukan penghitungan kemiripan suatu *query* dengan pesan yang diterima membutuhkan waktu yang cukup lama. Selain itu pada penghitungan tersebut juga melakukan penutupan I/O yang dapat menyebabkan *server* tidak dapat melayani permintaan baru. Permasalahan tersebut dapat diatasi dengan cara memproses penghitungan tersebut pada *thread* baru. Untuk menerapkan pemrosesan *multi-thread* pada Tornado yang merupakan *single-thread* diperlukan adanya penjadwalan pemrosesan terhadap masing-masing *thread*. *Thread pool* merupakan cara yang dapat diterapkan pada Tornado. *Thread pool* merupakan kumpulan *thread* pekerja yang secara efisien memproses *asynchronous callback* atas nama aplikasi utama [5].

3.3.4 Perancangan *Publisher*

Publisher merupakan pihak yang mengirimkan pesan ke *broker* untuk nantinya digunakan oleh *subscriber* yang bersangkutan. Perancangan *publisher* lebih menekankan pada manipulasi tipe karena *publisher* akan mendapat data hasil *crawling* untuk dikirimkan ke *broker*.

Pada pola pengiriman pesan *publish-subscribe* terdapat kemungkinan *subscriber* sedang tidak aktif ketika *publisher* mengirimkan pesan. Diperlukan cara agar semua pesan yang pernah dikirim oleh *publisher* dapat diterima *subscriber* semua. *Publisher* bayangan, merupakan salah satu cara untuk mengatasi hal tersebut. *Publisher* bayangan akan mengirimkan semua

pesan yang pernah dikirimkan oleh *publisher* asli. Adanya *publisher* bayangan juga mengatasi masalah ketika terdapat *subscriber* yang terlambat *subscribe* karena ketika pesan yang dikirim oleh *publisher* oleh sudah digunakan oleh satu atau lebih *subscriber* maka *broker* akan langsung menghapus pesan tersebut.

3.3.5 Perancangan *Subscriber*

Subscriber merupakan pihak yang menggunakan pesan yang dikirim oleh *broker* yang dibuat *publisher*. *Subscriber* yang dibuat nantinya akan menerima semua pesan yang dikirim oleh *broker*. Setiap *subscriber* mendapatkan pesan dari *broker*, *subscriber* akan menghitung nilai kemiripan antara pesan yang diterima dengan *query* pencarian yang dikirim oleh *web client*. Proses penghitungan nilai kemiripan ini akan dilakukan oleh *filter*. Hasil penghitungan tersebut akan digunakan untuk menentukan apakah pesan yang diterima sesuai dengan permintaan pengguna. Pesan yang memiliki nilai kemiripan lebih besar atau sama dengan nilai *threshold* akan dikirimkan ke *web client*.

Subscriber perlu melakukan *subscribe* terhadap pesan yang dikirimkan oleh *publisher* asli maupun *publisher* bayangan. Ketika pesan yang diterima oleh *subscriber* merupakan pesan yang sudah pernah dihitung nilai kemiripannya maka *subscriber* perlu memutuskan hubungan terhadap *publisher* bayangan supaya tidak mendapatkan pesan yang sama berkali-kali.

3.3.6 Perancangan *Web Client*

Web client merupakan tampilan web yang berhubungan langsung dengan pengguna. Pada perancangan *web client* membutuhkan beberapa *field*, yaitu *field* yang digunakan untuk memasukkan *query* pencarian, *field* yang digunakan untuk

menampilkan pesan yang dikirim oleh *web server*, *field* yang digunakan untuk memilih sumber, *field* yang digunakan untuk memilih batas kemiripan dan sebuah tombol yang digunakan untuk mengirimkan *query* pencarian yang ke *web server*.

Penggunaan *web socket* harus diterapkan pada *client* dan *server*. Pada *client* menggunakan *web-socket.js*[1] yang nantinya dihubungkan dengan permintaan *web socket* yang berada di *server*.

3.3.7 Perancangan *Web Socket*

Web socket merupakan protokol komunikasi yang memungkinkan kedua pihak yang berkomunikasi agar dapat saling mendengarkan dan mengirimkan pesan satu sama lain secara bersamaan. Penggunaan *web socket* diperlukan untuk menyediakan pesan secara *real time*. Pada sistem yang dibuat, *web socket* digunakan untuk menghubungkan *subscriber* yang dibuat oleh *web server* dengan *web client* yang merupakan tampilan yang berhubungan secara langsung dengan pengguna. Pesan diterima oleh *web client* merupakan pesan yang sudah melalui penghitungan nilai kemiripan yang dilakukan oleh *filter*. *Web socket* akan ditutup koneksinya ketika *web client* ditutup. Ketika *web socket* ditutup maka koneksi terhadap *broker* juga perlu ditutup agar penyimpanan tidak dipenuhi koneksi yang sudah tidak terpakai.

3.3.8 Perancangan *Filter*

Filter bertugas untuk menghitung nilai kemiripan antara *query* pencarian yang dikirim oleh pengguna dengan pesan yang diterima oleh *subscriber*. Penghitungan kemiripan ini menggunakan *cosine-similarity*. *Cosine similarity* dipilih karena menurut penelitian[9] bahwa *cosine similarity* cocok digunakan

[1] web-socket.js dapat diunduh di https://github.com/gimite/web-socket-js

pada proses pencarian informasi (*information retrieval*).

Pada bahasa pemrograman python terdapat *library* yang mudahkan dalam melakukan penghitungan *cosine-similarity* yaitu scikit-learn. Scikit-learn merupakan *machine learning library* pada bahasa python. Scikit-learn memiliki banyak fungsi, salah satunya yaitu penghitungan *cosine-similarity*. Setelah didapatkan nilai kemiripan antara *query* pencarian dengan pesan yang diterima, sistem harus menentukan apakah pesan tersebut sesuai/cocok dengan *query* pencarian yang dikirim oleh pengguna atau tidak. Untuk menentukan kecocokan tersebut digunakan *threshold*. *Threshold* merupakan nilai yang menjadi batas minimal agar dapat dikelompokkan kedalam kategori yang ada, dalam hal ini sesuai/cocok atau tidak dengan *query* pencarian. Belum ada nilai *threshold* yang pasti untuk digunakan pada semua permasalahan yang menggunakan penghitungan *cosine similarity* pada proses pencarian informasi (*information retrieval*).

3.3.9 Perancangan *Crawler*

Crawler merupakan pihak yang mengambil data dari internet. Data yang diambil nantinya akan digunakan sebagai pesan yang dikirim oleh *publisher* ke *broker* yang kemudian akan digunakan oleh *subscriber* yang bersangkutan. Penggunaan crawler ditujukan untuk penerapan pada data yang berada pada dunia nyata supaya mendapatkan pesan-pesan yang sedang menjadi tren atau menjadi bahasan di khalayak umum. Data yang pernah diambil harus didata sehingga *crawler* tidak perlu mengambil data yang sama berkali-kali.

Pada bahasa pemrograman python terdapat perangkat lunak yang berfokus pada ekstraksi data dari internet, yaitu scrapy. Scrapy merupakan *web crawling framework* yang ditulis dalam bahasa pemrograman python. Scapy memiliki banyak fungsi yang ditawarkan. Pada scrapy juga sudah terdapat fungsi yang

memungkinkan pengguna untuk mengambil isi dari suatu halaman *website* berdasarkan elemen html sehingga memudahkan untuk penyusunan isi yang akan dikirim ke *broker*.

BAB 4

IMPLEMENTASI

Pada bab ini dibahas mengenai implementasi pembangunan sistem. Pada bab ini akan dijelaskan secara lebih detail mengenai implementasi *web server*, implementasi *publisher*, implementasi *subscriber*, implementasi *web client*, implementasi *websocket*, implementasi *filter* dan implementasi *crawler*.

4.1 Lingkungan Implementasi

4.1.1 Perangkat Keras

Implementasi pembangunan sistem diterapkan pada perangkat keras dengan spesifikasi sebagai berikut:

1. Intel Core i5-4210U 1.70GHz
2. RAM DDR3 4 GB

4.1.2 Perangkat Lunak

Implementasi pembangunan sistem diterapkan pada sistem operasi Ubuntu 16.04 LTS 64 bit.

4.2 Implementasi *Broker*

Implementasi pada *broker* hanya perlu membuat akun yang memungkinkan *broker* untuk dapat diakses dari perangkat lain menggunakan IP. RabbitMQ sebenarnya mempunyai akun guest, akan tetapi akun guest tidak dapat digunakan lagi untuk mengakses RabbitMQ dari perangkat lain. Pembuatan akun baru untuk mengakses RabbitMQ dari perangkat lain dapat dilihat pada kode sumber 4.1 berikut.

```
1 rabbitmqctl add_user akun_baru kata_kunci_akun
2 rabbitmqctl set_user_tags akun_baru administrator
3 sudo rabbitmqctl set_permissions -p / akun_baru ".*"
    ↪ ".*" ".*"
```

Kode Sumber 4.1: Pembuatan Akun Baru pada RabbitMQ

4.3 Implementasi *Web Server*

Untuk implementasi *web server* pertama perlu melakukan instalasi *Tornado*. Instalasi *Tornado* dapat dilakukan dengan menjalankan perintah `pip3 install tornado` pada terminal. Tornado merupakan python *web server*, sehingga semua *routing* yang akan digunakan perlu dipetakan. Pada pembuatan sistem ini *web server* hanya membutuhkan dua *routing*, *routing* untuk menuju ke halaman yang digunakan untuk memasukkan *query* pencarian dan *routing* untuk membuka koneksi *websocket*. Kode sumber 4.2 merupakan potongan kode sumber yang digunakan pada proses *routing*.

```
1 class TornadoWebServer(tornado.web.Application):
2  def __init__(self):
3   handlers = [(r"/ws_channel",
      ↪ WebSocketServer),(r"/MainPage", Filter)]
4   settings = dict(
5    cookie_secret=options.cookie_secret,
6     login_url="/signin",
7     template_path=os.path.join(os
      ↪ .path.dirname(__file__), "templates"),
8     static_path=os.path.join(os
      ↪ .path.dirname(__file__), "static"),
9     xsrf_cookies=True,
10    debug=True)
11  tornado.web.Application.__init__(self, handlers,
      ↪ **settings)
```

Kode Sumber 4.2: Potongan Kode Sumber *Routing*

MainPage merupakan sebuah kelas yang melayani permintaan untuk masuk ke halaman *web* yang digunakan untuk memasukkan *query* pencarian. WebSocketServer merupakan sebuah kelas yang menangani pembuatan koneksi *websocket* baru. Pada kelas WebSocketServer akan dibuat kelas baru ke yang bertugas untuk berhubungan dengan *broker* dan bertindak sebagai *subscriber*.

Pada Tornado proses yang menutup I/O dapat mencegah *server* untuk melayani permintaan baru, maka perlu adanya *thread* yang menangani setiap proses yang dapat menutup I/O dalam waktu lama. Pada sistem yang dibuat penghitungan kemiripan antara *query* dengan pesan merupakan proses yang memakan waktu yang lama dan munutup I/O. Hal tersebut dikarenakan dalam menghitung nilai kemiripan terdapat proses *stemming* yang membutuhkan sumber daya yang banyak. Penghitungan kemiripan ini perlu untuk diproses pada *thread* terpisah dari *thread* utama dan diproses secara *asynchronous* sehingga tidak mencegah *server* untuk melayani permintaan baru. Pemrosesan pada *thread* baru tersebut menerapkan *thread pool*. Kode sumber 4.3 merupakan potongan kode sumber dari penerapan *lst:thread pool*.

```
1  def unblock(f):
2    @tornado.web.asynchronous
3    @wraps(f)
4    def wrapper(*args, **kwargs):
5      self = args[0]
6      def callback(future):
7        tupleCosineDocument = future.result()
8        if tupleCosineDocument is not None:
9          self.list_checked_document
       ↪ .append(tupleCosineDocument[0])
10         resultCosine = tupleCosineDocument[-1]
11         if float(resultCosine) > self.threshold:
12           if tupleCosineDocument not in
```

```
      ↪ self.list_similar_document:
13              self.list_similar_document
      ↪ .append(tupleCosineDocument)
14              sorted_similar_documents =
      ↪ sorted(self.list_similar_document, key = lambda
      ↪ x : x[5], reverse=True)
15              del sorted_similar_documents[self.n_result:]
16              del sorted_similar_documents[self.n_result:]
17              del self.list_similar_document[:]
18              self.list_similar_document =
      ↪ sorted_similar_documents
19
20              json_documents =
      ↪ demjson.encode(sorted_similar_documents)
21              self.websocket.write_message(json_documents)
22
23      EXECUTOR.submit(
24        partial(f, *args, **kwargs)
25          ).add_done_callback(lambda future:
26          tornado.ioloop.IOLoop.instance().add_callback(
27          partial(callback, future)))
28  return wrapper
```

Kode Sumber 4.3: Potongan Kode Sumber *Thread Pool*

Fungsi callback nantinya akan menangani hasil perhitungan kemiripan yang diproses secara *asynchronous*. Fungsi callback ini juga merupakan fungsi yang bertugas untuk membandingkan nilai perhitungan kemiripan dengan nilai *threshold* yang ada. Pesan yang nilai kemiripannya lebih besar dari nilai *threshold* akan dikirimkan ke *web client* melalui *web socket*. Sebelum dikirimkan ke *web client* pesan akan diurutkan berdasarkan nilai kemiripan yang paling tinggi ke paling rendah, hal tersebut dapat dilihat pada baris 14.

Ketika *web client* mengirimkan *query* baru *server* perlu menerima lagi pesan dari *publisher* bayangan sehingga *query* baru ini akan dibandingkan dengan semua pesan yang pernah dikirimkan oleh *publisher*. Selain itu *server* juga perlu

menghapus variabel yang menyimpan daftar pesan yang sudah dibandingkan dan menyimpan pesan yang sesuai dengan *query* pencarian yang sebelumnya. Kode sumber 4.4 merupakan potongan kode sumber dari penangannan *query* baru.

```
1  def request_query(self, rq_query):
2    tmp_rq_query = rq_query.split('^')
3    self.query = tmp_rq_query[0]
4    self.threshold = float(tmp_rq_query[1])
5    self.selectedSources = tmp_rq_query[2].split('_')
6
7    del self.list_similar_document[:]
8    del self.list_checked_document[:]
9
10   if self.connection is not None and self.connection2
       ↪ is not None:
11     if not self.bound_bc:
12       self.bound_bc =
       ↪ self.conn_b1.bind_queue(self.exchange_name_bc,
       ↪ self.queue_name)
13     if not self.starting_consume:
14       self.starting_consume =
       ↪ self.conn_b1.start_consume(self.on_pika_message)
15     if not self.bound_bc2:
16       self.bound_bc2 =
       ↪ self.conn_b2.bind_queue(self.exchange_name_bc2,
       ↪ self.queue_name2)
17     if not self.starting_consume2:
18       self.starting_consume2 =
       ↪ self.conn_b2.start_consume(self.on_pika_message)
19
20   else:
21     self.connect()
22     self.request_query(rq_query)
```

Kode Sumber 4.4: Potongan Kode Sumber *Query* Baru

Pada baris 7 dan 8 dilakukan penghapusan terhadap variabel yang menyimpan daftar pesan yang mirip dengan *query* pencarian dan daftar pesan yang pernah dihitung nilai kemiripannya pada permintaan sebelumnya. Pada baris 12 dan

16 menghubungkan kembali koneksi terhadap *publisher* bayangan. Menghubungkan kembali koneksi ke *publisher* bayangan penting untuk dilakukan agar *subscriber* dapat membandingkan *query* pencarian dengan semua pesan yang pernah dikirim oleh *publisher* asli. Pada baris 14 dan 18 memanggil proses untuk mulai menerima pesan dari *publisher*. Fungsi yang akan menangani pesan yang dikirim oleh *publisher* adalah parameter yang terdapat pada fungsi yang memanggil tersebut. Penghitungan nilai kemiripan melibatkan proses *stemming* yang memakan waktu yang lama. *Stemming* merupakan proses pengubahan suatu kata menjadi kata dasarnya. Proses *stemming* tersebut dapat dipercepat dengan cara menyimpan *cache* proses *stemming* pada sesi sebelumnya untuk digunakan pada sesi selanjutnya. Penyimpanan *cache* ini dapat memangkas waktu sangat banyak karena sebelum proses *stemming* dilakukan program akan memeriksa apakah suatu kata yang akan di *stemming* berada pada *cache* atau tidak. Kode sumber 4.5 dibawah menunjukan proses penyimpanan *cache stemming* dapat dilihat dibawah ini.

```
1  class PikaClient(Sim, tornado.web.RequestHandler):
2    def on_close(self):
3      if self.pika_client.connection is not None and
       ↪ self.pika_client.connection.is_open:
4        self.pika_client.connection.close()
5      if self.pika_client.connection2 is not None and
       ↪ self.pika_client.connection2.is_open:
6        self.pika_client.connection2.close()
7
8  class Sim(object):
9    def save_new_stemmer_cache(self):
10     with open("arrayStemmerList.pal") as f:
11       arrayStemmerDictionary =
       ↪ ast.literal_eval(f.read())
12     if len(self.arrayStemmer.data) >
       ↪ len(arrayStemmerDictionary):
13       fileOp = open("arrayStemmerList.pal",'w')
```

```
14        stemmerCache = str(self.arrayStemmer.data)
15        fileOp.write(stemmerCache.replace(", ",",\n"))
```

Kode Sumber 4.5: Potongan Kode Sumber Simpan *Cache*

4.4 Implementasi *Publisher*

Publisher merupakan program yang dijalankan secara terpisah dari *server*. *Publisher* bertugas mengirimkan pesan ke *broker*. Pesan yang dikirim oleh *publisher* merupakan hasil *crawling* yang dilakukan oleh *crawler*. *Publisher* perlu membuat *publisher* bayangan yang bertugas mengirim ulang semua pesan yang pernah dikirim oleh *publisher* asli. Hal tersebut dikarenakan terdapat kemungkinan bahwa suatu *subscriber* terlambat melakukan *subscribe*. Kode sumber 4.6 merupakan potongan kode sumber *publisher*.

```
 1 #!/usr/bin/python
 2 import pika
 3 import time
 4 import subprocess
 5 import ast
 6
 7 credentials = pika.PlainCredentials('guest', 'guest')
 8 param = pika.ConnectionParameters(host='localhost',
 9   port=5672,
10   virtual_host="/",
11   credentials=credentials,retry_delay=1,
       ↪ heartbeat_interval=0)
12 connection = pika.BlockingConnection(param)
13 channel = connection.channel()
14
15 exchangeName = 'tornado'
16 exchangeNameBc = 'detik-bc'
17 pathCrawler = '/opt/lampp/htdocs/playground8/
       ↪ test/scrapy/twitter/'
18 outputCrawlingKompas = 'twitter.json'
```

```
19
20 channel.exchange_declare(exchange=exchangeName,
       ↪ type='fanout', auto_delete=False)
21
22 channel.exchange_declare(exchange=exchangeNameBc,
       ↪ type='fanout', auto_delete=False)
23
24 listTitle = []
25 listAllMsg = []
26
27 while True:
28   fileKompasInit =
       ↪ open(pathCrawler+outputCrawlingKompas, 'w+')
29   fileKompasInit.seek(0)
30   fileKompasInit.truncate()
31   fileKompasInit.close()
32
33   listTitleStr = ""
34   for line in listTitle:
35     listTitleStr += line + "_"
36   cmd = 'cd %s && scrapy crawl twitter -a
       ↪ listTitle="%s" -o %s' % (pathCrawler,
       ↪ listTitleStr, outputCrawlingKompas)
37
38   crawling = subprocess.Popen(cmd, shell=True)
39   while crawling.poll() is None:
40     connection.process_data_events()
41     time.sleep(5)
42
43   fileKompasCrawl =
       ↪ open(pathCrawler+outputCrawlingKompas, 'r')
44   lines = fileKompasCrawl.readlines()
45   for line in lines:
46     if len(line) > 2:
47       lineTmp = ""
48       if line[-2] == ",":
49         lineTmp = line[0:-2]
50       else:
51         lineTmp = line
52       if lineTmp.find("null") == -1:
53         lineDict = ast.literal_eval(lineTmp)
```

```
54
55          title = lineDict['title']
56          content =
        ↪ lineDict['content'].decode('unicode_escape').
        ↪ encode('ascii','ignore')
57          topics = lineDict['topics']
58          author = lineDict['author']
59          source = lineDict['source']
60
61          lineTuple = (title, content, author, topics,
        ↪ source)
62          listTitle.append(title)
63          listAllMsg.append(lineTuple)
64          channel.basic_publish(exchange=exchangeName,
        ↪ routing_key='', body=str(lineTuple))
65          time.sleep(0.05)
66    interval = 60
67    while interval > 0:
68      connection.process_data_events()
69      time.sleep(5)
70      interval-=5
71    for line in listAllMsg:
72      channel.basic_publish(exchange=exchangeNameBc,
        ↪ routing_key='', body=str(line))
73      time.sleep(0.05)
74
75 connection.close()
```

Kode Sumber 4.6: Potongan Kode Sumber *Publisher*

Pada baris 44 pada kode sumber diatas *publisher* membaca berkas yang merupakan hasil *crawling* yang dilakukan oleh *crawler*. Pada baris 56 dilakukan penghapusan karakter-karakter *unicode* karena data yang diperoleh dari proses *crawling* terdapat banyak karakter *unicode*.

Proses *crawling* dapat memakan waktu yang lama ketika kecepatan internet sedang lambat. Hal tersebut dapat menyebabkan koneksi terhadap *broker* terputus. Terputusnya koneksi dikarenakan *broker* tidak mendapatkan tanggapan dari

koneksi *publisher* yang bersangkutan. Terdapat istilah *heartbeat* pada RabbitMQ *broker*. *Heartbeat* merupakan proses dimana RabbitMQ *server* memerlukan tanggapan dari koneksi *publisher* maupun *subscriber* yang bersangkutan setiap kurun waktu tertentu. Untuk mengatasi permasalah tersebut *publisher* dapat mengirimkan tanggapan ke *server* agar koneksi ke *broker* tidak terputus. Potongan kode sumber pada permasalah *heartbeat* tersebut dapat dilihat pada kode sumber diatas pada baris 67 sampai 70.

Ketika terdapat *web client* yang mengirimkan permintaan setelah *publisher* asli mengirimkan pesan, *web client* akan tetap mendapatkan pesan yang pernah dikirimkan oleh *publisher* asli. Pengiriman pesan tersebut dilakukan oleh *publisher* bayangan yang dilakukan setiap kurun waktu tertentu. Potongan kode sumber *publisher* bayangan dapat dilihat pada kode sumber diatas pada baris 71 sampai 73 yang akan dijalankan selama interval tertentu yang terdapat pada baris 66.

4.5 Implementasi Subscriber

Implementasi *subscriber* terdapat pada kelas PikaClient yang akan dibuat ketika terdapat koneksi baru pada *websocket*. Penerapan pola pengiriman pesan *publish-subscribe* menggunakan RabbitMQ sebagai *broker* memerlukan sebuah *queue* untuk setiap *subscriber*. Hal tersebut dikarenakan apabila dua *subscriber* mengambil pesan dari *queue* yang sama maka pesan akan diterima secara bergiliran yang menyebabkan *subscriber* hanya akan mendapatkan sebagian dari pesan yang dikirim. *Subscriber* yang menerima pesan akan langsung melakukan penghitungan kemiripan pesan tersebut terhadap *query* pencarian yang dikirim oleh *web client*. Kode sumber 4.7 merupakan kode sumber *subscriber* pada kelas Connect dan PikaClient.

```
class PikaClient(Sim, tornado.web.RequestHandler):
  def __init__(self):
    Sim.__init__(self)
    self.conn_b1 = Connect_B1()
    self.conn_b2 = Connect_B2()
    self.list_checked_document = []
    self.list_similar_document = []
    self.threshold = 0.1 #threshold for similarity
    self.n_result = 10 #max number of message to save
    ↪ as result
    self.connection = None
    self.bound_bc = True
    self.starting_consume = False
    self.connection2= None
    self.bound_bc2= True
    self.starting_consume2= False
    self.websocket = None
  def connect(self):
    if self.connection is None:
      result = self.conn_b1.connect(self.on_closed)
      self.connection = result[0]
      self.queue_name = result[1]
      self.exchange_name = result[2]
      self.exchange_name_bc = result[3]
    if self.connection2 is None:
      result2= self.conn_b2.connect(self.on_closed)
      self.connection2= result2[0]
      self.queue_name2= result2[1]
      self.exchange_name2= result2[2]
      self.exchange_name_bc2= result2[3]

  @unblock
  def on_pika_message(self, channel, method, header,
    ↪ body):
    tupleDocument =
    ↪ ast.literal_eval(body.decode("utf-8"))
    if tupleDocument[0] not in
    ↪ self.list_checked_document:
      result = self.similar(self.query, body,
```

```
      ↪ self.selectedSources)
36      return result
37    else:
38      if self.bound_bc:
39        self.bound_bc =
      ↪ self.conn_b1.unbind_queue(self.exchange_name_bc,
      ↪ self.queue_name)
40      if self.bound_bc2:
41        self.bound_bc2 =
      ↪ self.conn_b2.unbind_queue(self.exchange_name_bc2,
      ↪ self.queue_name2)
42      return None
43
44 class Connect_B1(object):
45 def connect(self, on_close_callback):
46   if self.connecting:
47     return
48   self.connecting = True
49   credentials = pika.PlainCredentials('pal2', 'pass2')
50   param = pika.ConnectionParameters(host=self.host,
51     port=5672,
52     virtual_host="/",
53     credentials=credentials, retry_delay=1,
       ↪ heartbeat_interval=0)
54   self.connection = TornadoConnection(param,
55     on_open_callback=self.on_connected)
56   self.connection.add_on_close_callback
       ↪ (on_close_callback)
57   result = (self.connection, self.queue_name,
       ↪ self.exchange_name, self.exchange_name_bc)
58   return result
```

Kode Sumber 4.7: Potongan Kode Sumber *Subscriber*

Dapat dilihat dari potongan kode sumber diatas bahwa penghitungan nilai kemiripan dilakukan oleh *thread* secara *asynchronous* pada baris 31. Potongan kode sumber *decorator* unblock dapat dilihat pada subbab implementasi *web server*.

Subscriber yang terlambat melakukan *subscriber* tetap akan mendapatkan pesan sebelumnya yang pernah dikirimkan oleh

publisher asli. Pesan yang didapat tersebut dikirimkan oleh *publisher* bayangan. Setelah *subscriber* tidak membutuhkan lagi *publihser* bayangan maka *subscriber* akan memutuskan hubungan dengan *subscriber* bayangan. Potongan kode sumber dari proses tersebut dapat dilihat pada kode sumber diatas pada baris 39 sampai 41.

4.6 Implementasi *WebSocket*

Web Client berfungsi mengirimkan *query* pencarian yang dimasukkan pengguna dan menampilkan pesan yang sesuai dengan *query* tersebut. Selain itu pada *web client* juga terdapat *web socket* yang bertugas untuk menjaga koneksi antara *web client* dengan *server* selama proses penghitungan nilai kemiripan maupun ketika menunggu pesan dari *publisher*. Kode sumber 4.8 merupakan implementasi *web socket* pada *web client*.

```
1 var timeLog;
2 function log(msg){
3   document.getElementById('result').style.display =
      ↪ "inline-block";
4   document.getElementsByClassName('sk-circle')[0]
      ↪ .style.display = "none";
5   document.getElementById('result').innerHTML = msg ;
6   document.getElementById('time-update').style.display
      ↪ = "block";
7   timeLog = new Date().getTime();
8   timer = setInterval(beginTimer, 10000);
9 }
10 function ws_init(url) {
11   ws = new WebSocket(url);
12   ws.onopen = function(){
13     document.getElementById('sendbtn').disabled = false;
14     document.getElementById('result').innerHTML = "";
15     var sourceClass =
      ↪ document.getElementsByClassName('source');
```

```
16      for (var ite=0; ite< sourceClass.length; ite++){
17        console.log('ws_onopen source');
18        sourceClass[ite].checked = true;
19      }
20      var ledBox =
        ↪ document.getElementsByClassName("led-box");
21      for (var index in ledBox){
22        if (index % 2 == 0){
23          ledBox[index].style.display = "block";
24        }
25        else{
26          ledBox[index].style.display = "none";
27        }};
28    };
29    ws.onmessage = function(msg){
30      document.getElementById('result').innerHTML = "";
31      var objMsg = JSON.parse(msg.data);
32      var result = "";
33      for (var index in objMsg){
34        result += "<li>" +
35          "<a class=\"toggle\"
        ↪ href=\"javascript:void(0)\">" + objMsg[index][0]
        ↪ + "</a>" +
36          "<div class=\"inner\">" +
37          "<p> <b>Isi </b>: " + objMsg[index][1] +
38          "<br><b>Penulis </b>: " + objMsg[index][2] +
39          "<br><b>Topik </b>: " + objMsg[index][3] +
40          "<br><b>Kemiripan </b>: " + objMsg[index][5] +
41          "<br><b>Sumber </b>: " + objMsg[index][4] +
        ↪ "</p></div>" + "</li>";
42      }
43      result = "<ul class=\"accordion\">" + result +
        ↪ "</ul>";
44      log(result);
45    };
46  ...
47  }
48  function ws_send(msg){
49    var sourceClass =
        ↪ document.getElementsByClassName('source');
50    var listSources = "";
```

```
51    for(var ite=0; ite<sourceClass.length; ite++){
52      if(sourceClass[ite].checked){
53        listSources += sourceClass[ite].value + "_";
54      }
55    }
56    if(listSources.length > 0){
57      if(msg.match(/^[a-zA-Z 0-9]+$/i) && msg.length >
        ↪ 2){
58        if(timer !== undefined){
59          clearInterval(timer);
60          document.getElementById('time-update')
        ↪ .style.display = "none";
61        }
62        document.getElementById('result').style.display =
        ↪ "none";
63        document.getElementsByClassName('sk-circle')[0]
        ↪ .style.display = "block";
64        document.getElementById('result').innerHTML = "";
65        var threshold =
        ↪ document.getElementById('threshold').value;
66        ws.send(msg+"^"+threshold+"^"+
        ↪ listSources.slice(0,-1));
67      }
68      else{
69        alert("Hanya gunakan alfabet dan angka dengan
        ↪ panjang lebih dari 2 karakter");
70      }
71    }
72    else{
73      alert("Silakan pilih minimal satu sumber.");
74    }
75  }
76  $(document.getElementById("result")).ready(function(){
77    ws_init('ws://localhost:8888/ws_channel');
78  });
```

Kode Sumber 4.8: Kode Sumber *Web Socket* pada *Client*

Pada baris 76 *web socket* pada *web client* dibuat. *Web socket* pada *web client* ini nantinya akan menghubungkan *web client* dengan *server* agar ketika terdapat pesan yang memiliki nilai

kemiripan sesuai dengan permintaan pesan tersebut akan diterima oleh *web client* tanpa harus mengirimkan permintaan baru ke *server*. Pesan yang dikirimkan oleh *server* diproses pada baris 29 pada kode sumber diatas. Pesan yang diterima oleh *web client* nantinya akan diubah formatnya supaya dapat lebih mudah dibaca. Pengubahan format pesan tersebut diproses pada baris 33 sampai 41. Pada kode sumber diatas juga terdapat bagian yang memeriksa masukan oleh pengguna. Masukan yang diperiksa adalah *query* pencarian dan daftar sumber yang dipilih. Pada baris 57 *query* yang dimasukan oleh pengguna diperiksa. *Query* yang dimasukan harus berupa huruf dan angka yang panjangnya lebih dari dua karakter. Daftar sumber yang dipilih diproses pada baris 53 dan akan diperiksa pada baris 56. Pengguna harus memilih paling sedikit satu daftar sumber.

Potongan-potongan kode sumber diatas merupakan potongan kode sumber penerapan *web socket* pada sisi *client*. Pada sisi *server* juga terdapat proses yang menangani permintaan yang berasal dari *web socket*. Untuk lebih jelasnya dapat dilihat pada kode sumber 4.9.

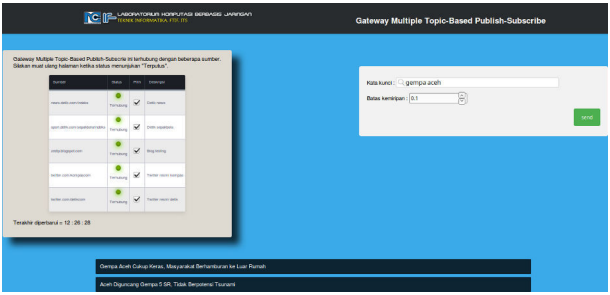
```
1 class WebSocketServer(tornado.websocket
      ↪ .WebSocketHandler):
2   def open(self):
3     self.pika_client = PikaClient()
4     self.pika_client.websocket = self
5     ioloop.add_timeout(1000, self.pika_client.connect)
6   def on_message(self, msg):
7     self.pika_client.request_query(msg)
8   def on_close(self):
9     if self.pika_client.connection is not None and
      ↪ self.pika_client.connection.is_open:
10      self.pika_client.connection.close()
11    if self.pika_client.connection2 is not None and
      ↪ self.pika_client.connection2.is_open:
12      self.pika_client.connection2.close()
```

Kode Sumber 4.9: Potongan Kode Sumber *Web Socket* pada *Server*

Pada bagian implementasi *web server* dapat dilihat bahwa kelas WebSocketServer merupakan kelas yang akan dijalankan ketika *web client* mengakses *routing* /ws_channel. Untuk setiap *web socket* oleh *server* akan dibuatkan kelas PikaClient hal tersebut dapat dilihat pada baris 3. Pada kelas PikaClient ini terdapat *subscriber* yang menerima berita yang dibuat oleh *publisher*. Pada *server* terdapat fungsi yang menerima *query* pencarian dari *web client*, fungsi tersebut dapat dilihat pada baris 6 pada kode sumber diatas. Pada baris 8 merupakan fungsi yang akan dikerjakan ketika *web socket* ditutup. Ketika *web socket* ditutup maka koneksi ke *broker* juga akan ditutup.

4.7 Implementasi *Web Client*

Implementasi web client terdiri dari beberapa bagian, yaitu bagian yang digunakan pengguna memasukan *query* pencarian, memasukan batas minimal kemiripan, menampilkan status sumber, memilih sumber, menampilkan terakhir kali data diperbarui dan bagian yang menampilkan pesan yang dikirimkan oleh *server*. Gambar 4.1 merupakan tampilan dari implementasi web client.



Gambar 4.1: Implementasi *Web Client*

4.8 Implementasi *Filter*

Implementasi *filter* pada sistem yang dibangun dimulai ketika *subscriber* mendapatkan pesan dari *broker*. Pesan tersebut akan dihitung nilai kemiripannya menggunakan *cosine similarity*. Kode sumber 4.10 merupakan penerapan *filter* dapat dilihat dibawah ini.

```
class Sim(object):
 def __init__(self):
   self.factory = StemmerFactory()
   self.stemmer = self.factory.create_stemmer()
   self.arrayStemmer = self.stemmer.get_cache()
   self.listCheckedDocument = []
   self.load_stopwords()
   self.load_prev_stemmer_cache()
  def load_prev_stemmer_cache(self):
    with open("arrayStemmerList.pal") as f:
      arrayStemmerDictionary =
    ↪ ast.literal_eval(f.read())
    for key in arrayStemmerDictionary:
      self.arrayStemmer.set(key,
    ↪ arrayStemmerDictionary[key])
  def get_stemmed_query(self, query):
    stemmed_query = self.stemmer.stem(query)
    yield stemmed_query
  def load_stopwords(self):
    self.stopwordsList = []
    with open("stopwords.txt") as f:
      for line in f:
        self.stopwordsList.append(line.strip('\n'))
  def get_stemmed_document(self, document):
    documentTmp =
    ↪ document.replace("(","").replace(")","")
    ↪ .replace("'","").replace(",","").strip('\n')
    ↪ .strip('\'').strip('\r').strip('\n')
    stemmedDocument = self.stemmer.stem(documentTmp)
    yield stemmedDocument
def calculate_similarity(self, query, document, source):
```

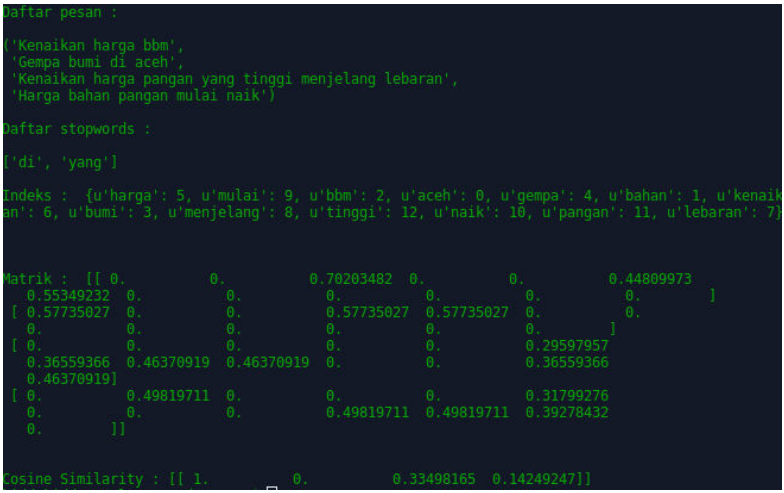
```
27    tupleDocument = ast.literal_eval(document)
28    if tupleDocument[-1] in source:
29      stemmed_query = self.get_stemmed_query(query)
30      stemmed_document =
        ↪ self.get_stemmed_document(document)
31      tfidf_vectorizer = TfidfVectorizer(stop_words =
        ↪ set(self.stopwordsList))
32      tfidf_matrix_documents = tfidf_vectorizer.
        ↪ fit_transform(stemmed_document)
33      tfidf_matrix_dummy_query = tfidf_vectorizer.
        ↪ transform(stemmed_query)
34      resultCosineSimilarity =
        ↪ cosine_similarity(tfidf_matrix_dummy_query,
        ↪ tfidf_matrix_documents)
35      tupleCosineDocument = ast.literal_eval(document)
36      tupleCosineDocument +=
        ↪ (str(resultCosineSimilarity[0][0]),)
37      return tupleCosineDocument
38    else:
39      tupleCosineDocument = ast.literal_eval(document)
40      tupleCosineDocument += (str(0),)
41      return tupleCosineDocument )
```

Kode Sumber 4.10: Potongan Kode Sumber *Filter*

Pada penghitungan nilai kemiripan proses pertama yang dilakukan adalah memeriksa apakah pesan yang akan dihitung nilai kemiripannya ini berasal dari sumber yang dipilih oleh penggua. Apabila pesan berasal dari sumber yang tidak dipilih pengguna maka nilai kemiripan akan langsung dianggap 0. Proses tersebut berapa pada baris 28.

Pada baris 32 dan 33 terdapat perbedaan penangan terhadap pesan dan *query* pencarian. Pada proses pengubahan pesan kedalam matriks TF-IDF pada baris 32 menggunakan fungsi fit_transform sedangkan pada *query* pencarian pada baris 33 menggunakan fungsi transform. Penggunaan fungsi fit_transform akan memasukkan semua kata pada pesan untuk dijadikan *indexing* dan mengubah pesan tersebut kedalam

matriks TF-IDF. Penggunaan transform hanya akan mengubah *query* pencarian kedalam matriks TF-IDF berdasarkan *indexing* yang telah dilakukan. Untuk mendapatkan gambaran yang lebih jelas terhadap data pada proses penghitungan nilai kemiripan ini dapat dilihat pada gambar 4.2 dibawah.



Gambar 4.2: Contoh Data Penghitungan Nilai Kemiripan

Pada gambar 4.2 dapat dilihat bentuk data pada proses penghitungan nilai kemiripan yang dilakukan. Pada gambar diatas dapat dilihat bahwa terdapat indeks yang seharusnya sama, yaitu naik dan kenaikan. Hal tersebut terjadi karena belum dilakukan proses *stemming* pada pesan yang digunakan. Proses *stemming* bertugas untuk mengubah suatu kata menjadi kata dasarnya. Jika dibandingkan dengan kode sumber 4.10, proses *stemming* dilakukan pada baris 14 untuk *query* pencarian dan baris 22 untuk pesan yang dibandingkan. Proses *stemming* dilakukan agar mendapatkan indeks yang lebih akurat. Selain proses *stemming* terdapat proses yang memuat daftar *stopwords*

yang digunakan untuk menghilangkan kata yang dianggap tidak penting. Proses tersebut terdapat pada baris 17 pada kode sumber diatas.

Telah dijelaskan pada bagian implementasi *subscriber* bahwa penghitungan nilai kemiripan dilakukan oleh *thread* secara *asynchronous*. Pada implementasi *web server* juga sudah dijelaskan bahwa hasil penghitungan akan ditangani oleh fungsi callback pada *decorator* unblock. Nilai kembalian pada baris 41 diatas akan ditangani oleh fungsi pada *decorator* unblock pada kode sumber 4.11.

```
1  def unblock(f):
2    @tornado.web.asynchronous
3    @wraps(f)
4    def wrapper(*args, **kwargs):
5    self = args[0]
6    def callback(future):
7      tupleCosineDocument = future.result()
8      self.list_checked_document.
       ↪ append(tupleCosineDocument[0])
9      resultCosine = tupleCosineDocument[-1]
10     if float(resultCosine) > self.threshold:
11       if tupleCosineDocument not in
         ↪ self.list_similar_document:
12         self.list_similar_document.
         ↪ append(tupleCosineDocument)
13         sorted_similar_documents =
         ↪ sorted(self.list_similar_document, key = lambda
         ↪ x : x[5], reverse=True)
14         json_documents =
         ↪ demjson.encode(sorted_similar_documents)
15       self.websocket.write_message(json_documents)
16     ....
```

Kode Sumber 4.11: Potongan Kode Sumber *Decorator unblock*

Data hasil nilai kembalian dari penghitungan nilai kemiripan

berisi tentang pesan beserta nilai kemiripannya. Ketika terdapat pesan yang nilai kemiripannya lebih besar dari nilai *threshold* dan belum terdapat pada daftar berita yang sesuai dengan *query* pencarian maka akan dimasukkan ke daftar tersebut. Setelah itu kumpulan pesan yang sesuai dengan *query* pencarian akan diurutkan berdasarkan nilai kemiripan yang paling besar baru kemudian dikirimkan ke *web client* melalui *web socket*.

4.9 Implementasi *Crawler*

Implementasi *crawler* pada sistem menggunakan scrapy. Scrapy merupakan *web crawling framework*. Scrapy mudah untuk digunakan dan memiliki banyak fungsi. Kode sumber 4.12 merupakan potongan kode sumber pada Scrapy yang bertugas melakukan *crawling* data pada *web* berita.

```
1  class DetikCrawler(scrapy.Spider):
2    name = 'detik'
3    start_urls = ['http://news.detik.com/indeks']
4    def parse(self, response):
5      firstTitle = response.css('ul#indeks-container
       ↪ h2::text') .extract_first()
6      listTitleStr = []
7      for title in listTitleResponse:
8        listTitleStr.append(title.encode('utf8'))
9      listTitle = self.listTitle.split('_')[:-1]
10     for ite in range(0, len(listTitleStr)):
11       if listTitleStr[ite] not in listTitle:
12         link = response.css('article
       ↪ a::attr(href)').extract()[ite]
13         yield scrapy.Request(url=link,
       ↪ callback=self.getArticles,
       ↪ errback=self.errorArticle)
14         ...
```

Kode Sumber 4.12: Potongan Kode Sumber *Crawler*

Crawler tidak akan langsung menyimpan semua data pada halaman yang terdapat pada variable start_urls. *Crawler* hanya akan mulai menyimpan artikel ketika terdapat artikel baru. Fungsi yang bertugas menyimpan kedalam berkas keluaran dapat dilihat pada potongan kode sumber 4.13 dibawah ini.

```
1 class DetikCrawler(scrapy.Spider):
2   ...
3   def getArticles(self, response):
4   title = response.css('div.jdl
      ↪ h1::text').extract_first()
5   content = response.css('div.detail_text').extract()
6   tags = response.css('div.breadcrumb
      ↪ a::text').extract()
7   editor =
      ↪ response.css('div.author::text').extract_first()
8   contentString = " ".join(content[0:])
9   contentNoTags = remove_tags(contentString)
10  tagsString = ", ".join(tags[0:1])
11  tmpUrl = response.url.split('.')
12  source = tmpUrl[0].split('/')[-1] + "." + tmpUrl[1]
13  yield {
14    'title' : title,
15    'content' : contentNoTags,
16    'topics' : tagsString,
17    'author' : editor,
18    'source' : source,}
19
```

Kode Sumber 4.13: Potongan Kode Sumber Penyimpanan Berkas Keluaran

Data yang diambil dari artikel berita pada halaman yang dituju adalah judul berita, isi berita, topik berita dan penulis berita. Berkas keluaran hasil *crawling* ini merupakan kumpulan pesan yang nantinya akan dikirimkan oleh *publisher* ke *broker*.

Penerapan proses *crawling* pada sistem yang terdapat didalam *publisher*. Kode sumber 4.14 merupakan potongan kode

sumber yang menampilkan pemanggilan proses *crawling* yang terdapat pada *publisher*.

```
while True:
 ....
 cmd = 'cd %s && scrapy crawl detik -a
   listTitle="%s" -o %s' % (pathCrawler,
   listTitleStr, outputCrawlingKompas)
 crawling = subprocess.Popen(cmd, shell=True)
 while crawling.poll() is None:
  connection.process_data_events()
  time.sleep(5)

```

Kode Sumber 4.14: Potongan Kode Sumber Pemanggilan *Crawler* pada *Publisher*

Pada *publisher* pemanggilan proses *crawling* dilakukan dengan `subprocess.Popen`. `subprocess.Popen` memanggil program *subprocess* tanpa melakukan penutupan I/O. Penggunaan `subprocess.Popen` diperlukan untuk menjada koneksi terhadap *broker* agar tidak terputus. Seperti yang telah dijelaskan pada implementasi *publisher* bahwa baris dibawahnya menunjukkan cara yang digunakan untuk menjaga koneksi dengan *broker* tidak terputus.

BAB 5

PENGUJIAN DAN EVALUASI

5.1 Lingkungan Uji Coba

5.1.1 Perangkat Keras

Implementasi pengujian sistem diterapkan pada perangkat keras dengan spesifikasi sebagai berikut:

1. *Web server* dan *broker* pertama : Asus Intel Core i5-4210U 1.70GHz, RAM DDR3 4 GB
2. *Broker* kedua : Lenovo Intel Core i3-3240 3.4GHz, RAM DDR3 8GB

5.1.2 Perangkat Lunak

Implementasi pengujian sistem diterapkan pada sistem operasi Ubuntu 16.04 LTS 64 bit untuk *server* dan *broker* pertama. Untuk *broker* kedua menggunakan Windows 8.1 pro.

5.2 Skenario Uji Coba

5.2.1 Skenario Uji Fungsionalitas

5.2.1.1 Memilih Batas Kemiripan

Uji coba batas kemiripan dilakukan untuk mengetahui apakah fungsi pemilihan batas kemiripan berjalan dengan benar. Uji coba batas kemiripan dilakukan dengan cara membandingkan nilai kemiripan yang dipilih pengguna dengan hasil yang diperolah dari *server*.

5.2.1.2 Penanganan Masukan

Uji coba penaganan masukan dilakukan untuk mengetahui apakah *error handling* terhadap masukan oleh pengguna berjalan dengan baik. Uji coba penanganan masukan dilakukan dengan cara memberikan masukan yang dianggap salah oleh *server*.

Ketika terdapat kesalahan pada masukan yang dilakukan oleh *server* apakah nantinya *error handling* dapat berjalan dengan baik atau tidak.

5.2.1.3 Menampilkan Status Sumber

Uji coba penampilan status sumber lebih condong kearah status koneksi terhadap *web socket* dari *web client* ke *server*. Uji coba status sumber ini dapat dilakukan dengan cara mematikan *server* ketika terdapat *web client* yang masih aktif. Ketika *server* dimatikan maka koneksi *web socket* juga akan terputus.

5.2.1.4 Memilih Sumber

Uji coba memilih sumber dilakukan untuk mengetahui apakah fungsi untuk memilih sumber yang diinginkan pengguna dari daftar sumber yang ada berjalan dengan baik. Uji coba ini dilakukan dengan membandingkan antara sumber yang dipilih dengan hasil yang diberikan oleh *server*.

5.2.1.5 Menampilkan Waktu Terakhir Diperbarui

Uji coba menampilkan waktu terakhir diperbarui dilakukan untuk mengetahui apakah fungsi menampilkan waktu terakhir kali pesan diperbarui berjalan dengan baik. Uji coba ini dilakukan dengan cara melihat pada bagian penampilan waktu terakhir diperbarui ketika mulai terdapat pesan yang diterima dari *server*.

5.2.2 Skenario Uji Performa

Uji coba performa yang dilakukan untuk mengetahui kemampuan dari sistem yang dibangun.

5.2.2.1 Kecepatan

Uji coba kecepatan bertujuan untuk mengetahui seberapa cepat sistem yang dibuat dalam menghitung nilai kemiripan antara *query* pencarian dengan pesan. Uji coba kecepatan terhadap sistem yang dibuat dilakukan dengan cara menghitung waktu yang diperlukan oleh sistem untuk menghitung nilai kemiripan antara *query* pencarian dengan semua pesan yang diterima. *Publisher* hanya akan mengirimkan pesan satu kali tanpa adanya pengiriman pesan berulang yang dikirim oleh *publisher* bayangan. Pengujian kecepatan yang dilakukan akan menggunakan jumlah sumber dan jumlah permintaan yang berbeda-beda. Penghitungan waktu dilakukan ketika sistem mulai menghitung nilai kemiripan antara *query* pencarian dengan pesan. Uji performa kecepatan akan dilakukan 10 kali untuk masing-masing jumlah permintaan. 10 kali percobaan ini ditujukan untuk mendapatkan data yang lebih akurat.

5.2.2.2 Ketahanan

Uji coba ketahanan bertujuan untuk mengetahui kemampuan sistem dalam melayani permintaan dari pengguna. Uji coba terhadap ketahanan ini lebih fokus terhadap berapa banyak koneksi yang dapat dibuka oleh sistem. Uji coba ketahanan dilakukan dengan menggunakan perangkat lunak artillery [1]. Artillery merupakan perangkat lunak yang Penghitungan ketahanan yang dilakukan akan mengirimkan permintaan dengan jumlah yang banyak dan melihat jumlah permintaan yang berhasil dan gagal.

[1] Artillery dapat diinstal dengan perintah npm install -g artillery

5.2.2.3 Sumber Daya

Uji coba sumber daya bertujuan untuk mengetahui seberapa banyak sumber daya yang digunakan oleh sistem. Uji coba ini akan menggunakan perangkat lunak htop [2]. Htop merupakan perangkat lunak yang digunakan untuk memantau proses yang berjalan di sistem Unix. Pengujian sumber daya yang digunakan sistem akan dimulai ketika sistem melayani permintaan. Beberapa parameter yang dirubah pada pengujian sumber daya ini adalah jumlah permintaan dan banyak pesan. Layaknya pada pengujian kecepatan, *publisher* hanya akan mengirimkan pesan satu kali tanpa adanya pengiriman pesan berulang yang dikirim oleh *publisher* bayangan. Uji performa sumber daya akan dilakukan 10 kali untuk masing-masing jumlah permintaan. 10 kali percobaan ini ditujukan untuk mendapatkan data yang lebih akurat.

5.2.3 Keakuratan *Query* Pencarian

Uji keakuratan *query* pencarian bertujuan untuk mengetahui seberapa akurat sistem dalam memberikan hasil penghitungan nilai kemiripan dari *query* pencarian yang dimasukan pengguna. Uji keakuratan dilakukan dengan cara membandingkan pesan yang ditampilkan oleh sistem dengan pesan yang dianggap mirip oleh pengguna secara langsung terhadap suatu *query* pencarian. Nantinya akan dihitung nilai *precision* dan *recall* dari hasil yang diberikan oleh sistem dengan data dari pengguna sebagai *ground truth*.

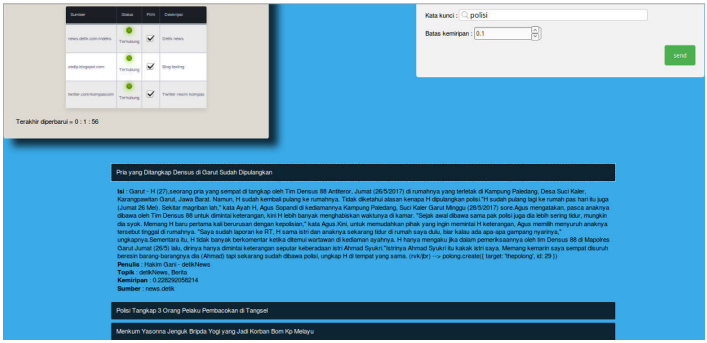
[2]htop dapat diunduh di http://hisham.hm/htop/index.php?page=downloads

5.3 Hasil Uji Coba dan Evaluasi

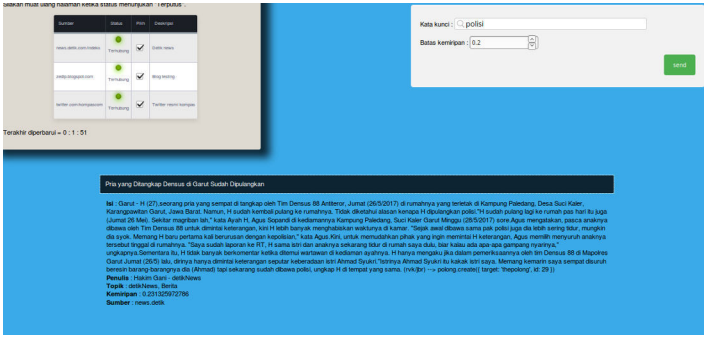
5.3.1 Uji Fungsionalitas

5.3.1.1 Memilih Batas Kemiripan

Uji coba pemilihan batas kemiripan bertujuan untuk mengetahui apakah fungsi pemilihan batas kemiripan berjalan dengan baik. Fungsi pemilihan batas kemiripan ini dapat memberikan keleluasaan terhadap pengguna untuk menentukan seberapa mirip suatu pesan terhadap kata kunci yang dimasukkan. Uji coba ini dilakukan dengan membandingkan dari dua hasil batas kemiripan yang berbeda dengan menggunakan *query* pencarian yang sama. Berikut merupakan gambar uji coba *query* pencarian dengan 2 batas kemiripan yang berbeda dan sumber yang sama.



Gambar 5.1: Hasil Uji Coba Batas Kemiripan dengan Batas Kemiripan 0.1



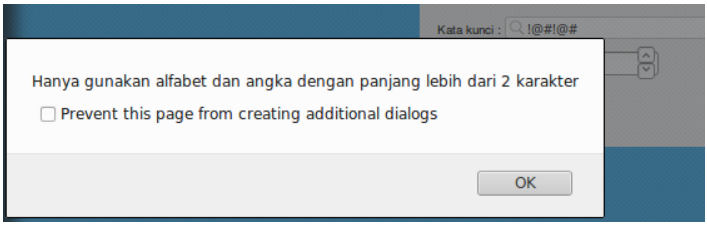
Gambar 5.2: Hasil Uji Coba Batas Kemiripan dengan Batas Kemiripan 0.2

Dari hasil uji coba fungsional batas kemiripan pada gambar 5.1 dan 5.2 dapat dilihat bahwa fungsi batas kemiripan berjalan dengan baik. Hal tersebut dikarenakan ketika diuji dengan *query* pencarian dan sumber yang sama, pesan yang nilai kemiripannya kurang dari batas minimal tidak muncul pada gambar 5.2.

5.3.1.2 Penanganan Masukan

Uji coba penganganan masukan bertujuan untuk mengetahui apakah fungsi untuk menangani masukan yang salah dari pengguna berjalan dengan baik. Penanganan masukan penting untuk dilakukan untuk mencegah adanya pesan yang tidak valid dikirim ke *server*. Data yang tidak valid dapat berupa sumber yang kosong maupun *query* pencarian dengan menggunakan karakter selain huruf dan angka. Berikut merupakan hasil uji coba penanganan masukan.

(a)



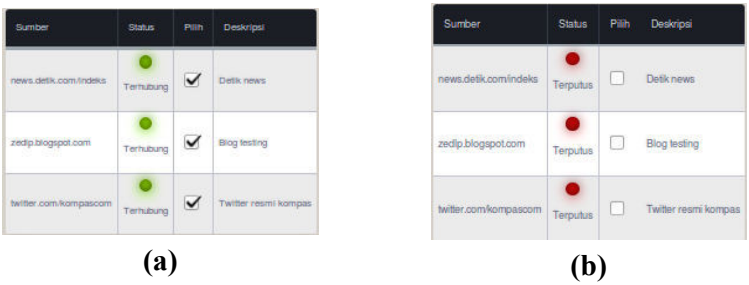
(b)

Gambar 5.3: Hasil Uji Coba Penanganan Masukan (a) Tidak Memilih Sumber, (b) *Query* Pencarian Selain Huruf dan Angka

Dari gambar 5.3 dapat dilihat bahwa fungsi yang menangani masukan pengguna yang salah berjalan dengan baik.

5.3.1.3 Menampilkan Status Sumber

Uji coba menampilkan status sumber bertujuan untuk mengetahui fungsi untuk menampilkan status tersambung atau terputus dengan *web socket* yang mengarah ke sumber pesan berjalan dengan baik atau tidak. Uji coba dilakukan dengan membandingkan status sumber ketika *server* berjalan dan ketika *server* dimatikan ketika terdapat koneksi. Dibawah ini merupakan hasil dari uji coba.

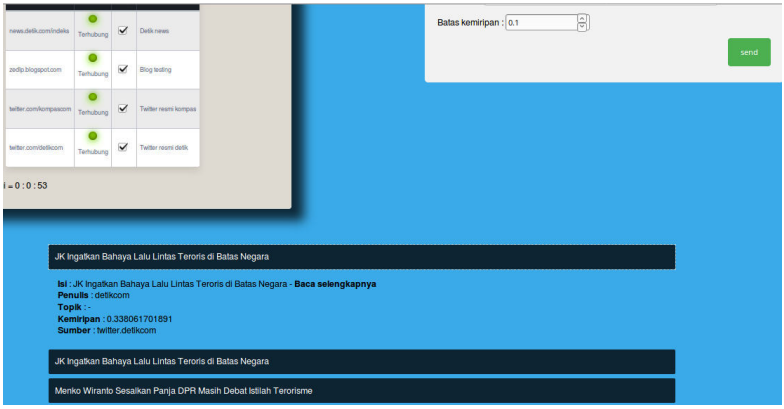


Gambar 5.4: Hasil Uji Coba Menampilkan Status Sumber (a) *Server* Menyala, (b) *Server* Mati

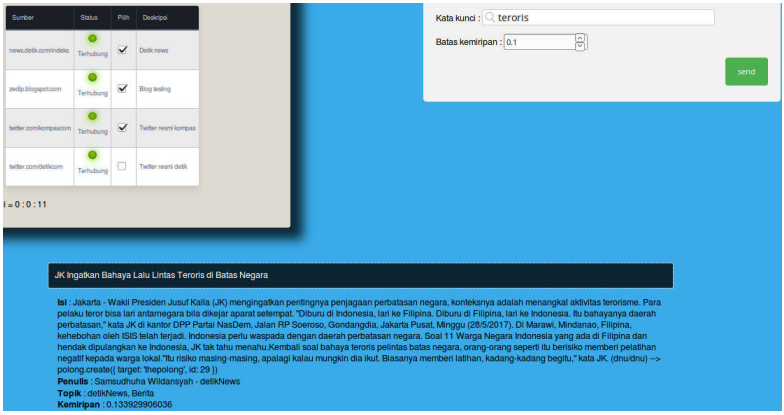
Dari gambar 5.4 dapat dilihat bahwa fungsi menampilkan status sumber berjalan dengan baik. Hal tersebut dikarenakan ketika *server* menyala status sumber menunjukan status terhubung sedangkan ketika *server* mati status sumber menunjukan status terputus.

5.3.1.4 Memilih Sumber

Uji coba memilih sumber bertujuan untuk mengetahui apakah fungsi memilih sumber dapat berjalan dengan baik. Uji coba dilakukan dengan membandingkan permintaan dengan *query* pencarian dan batas minimal kemiripan yang sama akan tetapi memilih sumber yang berbeda. Berikut merupakan hasil dari uji coba memilih sumber.



(a)



(b)

Gambar 5.5: Hasil Uji Coba Memilih Sumber (a) Memilih Twitter Detik, (b) Tanpa Twitter Detik

Dari gambar 5.5 dapat dilihat bahwa fungsi memilih sumber berjalan dengan baik. Hal tersebut dikarenakan pesan yang berasal dari sumber yang tidak dipilih tidak akan diolah oleh *server*.

5.3.1.5 Menampilkan Waktu Terakhir Diperbarui

Uji coba menampilkan waktu terakhir diperbarui bertujuan untuk mengetahui apakah fungsi untuk menampilkan waktu terakhir kali pesan diperbarui berjalan dengan baik atau tidak. Uji coba dilakukan dengan melihat pada bagian yang menampilkan waktu terakhir kali diperbarui ketika mulai mendapat pesan dari *server*. Berikut merupakan hasil dari uji coba.

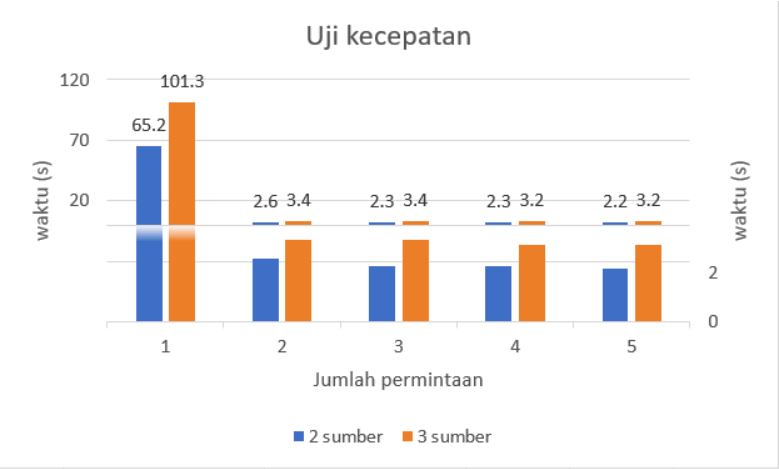
Gambar 5.6: Hasil Uji Coba Menampilkan Waktu Terakhir Diperbarui

Dari gambar 5.6 dapat dilihat bahwa fungsi menampilkan waktu terakhir kali diperbarui berjalan dengan baik. Waktu yang ditunjukan merupakan jam, menit dan detik terakhir kali pesan diperbarui oleh *server*.

5.3.2 Uji Performa

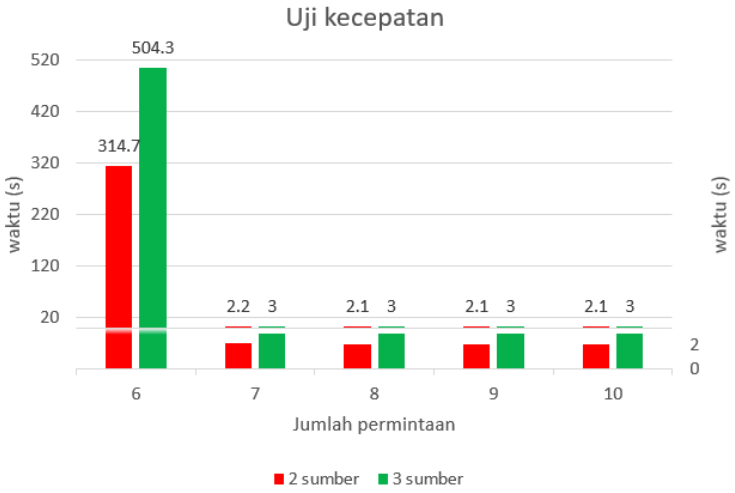
5.3.2.1 Kecepatan

Uji coba kecepatan sistem bertujuan untuk mengetahui seberapa cepat sistem menghitung nilai kemiripan antara *query* pencarian dengan pesan. Uji coba kecepatan ini juga ditujukan untuk mengetahui kecepatan sistem dalam menghitung nilai kemiripan ketika melayani lebih dari satu permintaan. Dibawah ini merupakan hasil uji coba kecepatan sistem dalam menghitung nilai kemiripan.



Gambar 5.7: Hasil Uji Coba Kecepatan 1 sampai 5 Permintaan

Gambar 5.7 merupakan grafik uji coba kecepatan untuk 1 permintaan sampai 5 permintaan. Uji coba pada gambar 5.7 dan gambar 5.8 dilakukan 10 kali untuk masing-masing jumlah permintaan. Untuk mendapatkan gambaran yang lebih jelas, gambar 5.8 merupakan hasil uji coba kecepatan untuk 6 sampai 10 permintaan.



Gambar 5.8: Hasil Uji Coba Kecepatan 6 sampai 10 Permintaan

Pada gambar 5.7 dan 5.8, uji coba dengan jumlah permintaan 1 sampai 5 menggunakan sumber yang berbeda dibanding dengan uji coba dengan jumlah permintaan 6 sampai 10. Dapat dilihat pada gambar 5.7 dan 5.8 bahwa pada jumlah permintaan 1 dan 6 memerlukan waktu yang sangat lama dibandingkan dengan uji coba pada jumlah permintaan lainnya. Hal tersebut dikarenakan ketika uji coba pada jumlah permintaan 1 dan 6, pesan yang digunakan belum pernah diproses oleh *server*. Penghitungan kemiripan *query* pencarian dengan pesan terdapat proses *stemming* yang memerlukan waktu yang lama. Ketika suatu kata dikenakan proses *stemming* oleh *server* maka kata tersebut akan disimpan pada sebuah *cache*. Kata yang akan dikenakan proses *stemming* akan dicocokan dahulu dengan *cache* yang ada. Hal tersebut juga terlihat pada uji coba dengan jumlah permintaan 1 dan 6 ketika dilakukan penambahan jumlah sumber dari 2 menjadi 3. Waktu yang diperlukan untuk melakukan

penghitungan pada jumlah sumber 3 tersebut membutuhkan waktu yang lebih banyak dibandingkan pada jumlah sumber 2 karena terdapat lebih pesan baru untuk diproses.

Dari hasil uji coba terlihat bahwa setelah pesan pernah diproses oleh *server*, jumlah permintaan yang dilayani *server* tidak terlalu berpengaruh terhadap kecepatan penghitungan yang dilakukan *server*. Kecepatan penghitungan nilai kemiripan yang dilakukan oleh *server* sangat dipengaruhi oleh pesan yang diproses. Ketika terdapat pesan yang memiliki kata-kata yang belum ada pada *cache* maka proses penghitungan akan memakan waktu yang cukup lama. Akan tetapi ketika kata-kata yang terdapat pada pesan sudah terdapat pada *cache* maka kecepatan penghitungan nilai kemiripan yang dilakukan oleh *server* akan sangat cepat. Terdapat peningkatan kecepatan rata-rata sebesar 98% ketika pesan belum pernah diproses oleh *server* debandingkan dengan sesudah.

5.3.2.2 Ketahanan

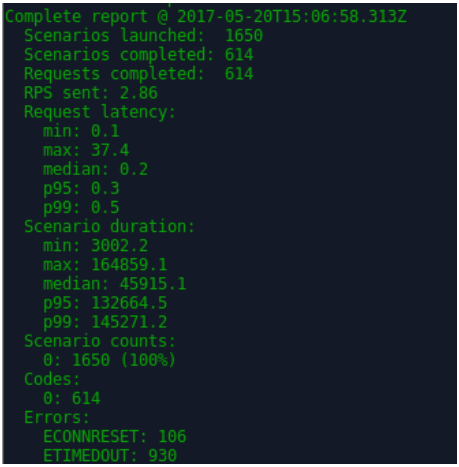
Uji coba ketahanan sistem bertujuan untuk mengetahui kemampuan sistem dalam melayani permintaan dari pengguna. Pengujian dilakukan dengan menghitung seberapa besar permintaan yang berhasil dilayani. Uji ketahanan sistem dilakukan menggunakan perangkat lunak artillery. Kode sumber 5.1 merupakan kode sumber yang digunakan untuk melakukan uji ketahanan sistem menggunakan artillery.

```
1 {"config": {
2  "target": "ws://localhost:8888/ws_channel",
3  "phases": [
4   {"duration": 30, "arrivalRate": 5, "name": "low"},
5   {"pause": 10},
6   {"duration": 30, "arrivalRate": 50, "name":
     ↪ "medium"},
```

```
 7    {"pause": 10}]
 8  },
 9  "scenarios": [{
10   "engine": "ws",
11   "flow": [
12    {"think": 3},
13    {"send": "polisi^.1"}]}
14  ]}
```

Kode Sumber 5.1: Uji Ketahanan Sistem

Yang perlu diperhatikan pada kode sumber uji ketahanan diatas adalah `duration` dan `arrivalRate`. `duration` merupakan waktu untuk tiap gelombang dalam detik. `arrivalRate` merupakan jumlah permintaan yang dikirimkan setiap detik selama satu gelombang tersebut. Uji coba diatas akan mengirimkan 5 permintaan setiap detik selama 30 detik pada gelombang pertama dan akan mengirimkan 50 permintaan setiap detik selama 30 detik. Jadi pada uji coba diatas akan mengirimkan permintaan sebanyak 1650. Hasil uji coba ketahanan dapat dilihat pada gambar 5.9 dibawah.

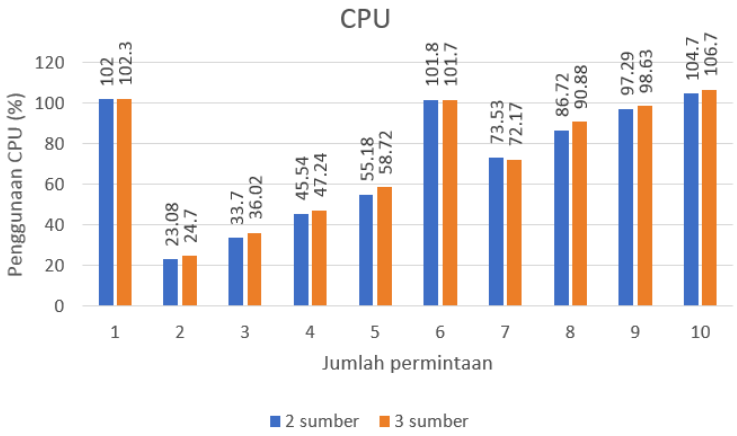


Gambar 5.9: Hasil Uji Coba Ketahanan Sistem

Dari gambar 5.9, dapat dilihat bahwa dari 1650 permintaan yang dikirim terdapat 614 permintaan yang berhasil dan 1036 permintaan yang gagal. Salah satu kemungkinan penyebab kegagalan sistem dalam melayani permintaan adalah terjadinya *race condition* ketika pembuatan *channel* pada *broker*. Adakalanya terdapat kondisi dimana pembuatan *channel* untuk sebuah permintaan memakan waktu yang lama, akan tetapi *channel* tersebut sudah mulai digunakan untuk menerima pesan dari *publisher*. Kemungkinan penyebab kegagalan yang lain adalah adanya permintaan yang perlu dilayani oleh sistem dalam jumlah yang banyak dalam waktu bersamaan. Dari berkas detail hasil uji coba menggunakan perangkat lunak artillery, diketahui bahwa permintaan paling banyak yang dilayani oleh sistem secara bersamaan sebanyak 1417.

5.3.2.3 Sumber Daya

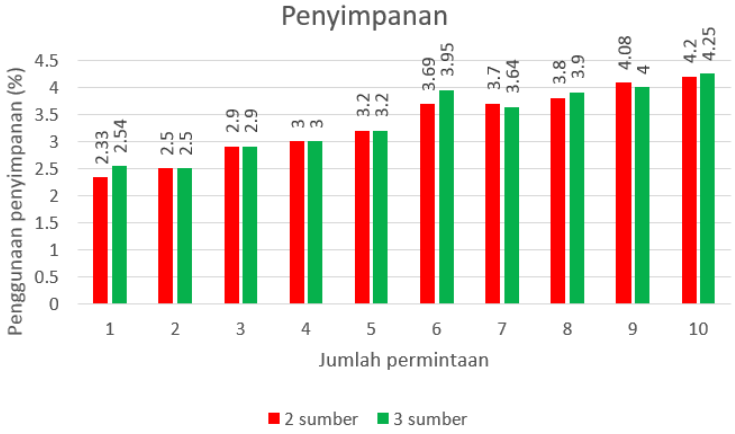
Uji sumber daya dilakukan untuk mengetahui seberapa banyak sumber daya yang digunakan oleh sistem ketika melayani permintaan. Sumber daya yang dimaksud berupa memori dan CPU. Uji coba dilakukan menggunakan perangkat lunak htop. Uji coba dilakukan dengan mengubah jumlah permintaan dan jumlah pesan. Dibawah ini merupakan hasil uji coba sumber daya menggunakan perangkat lunak htop. Uji coba pada gambar 5.10 dilakukan 10 kali untuk masing-masing jumlah permintaan.



Gambar 5.10: Uji Sumber Daya CPU

Dari gambar 5.10 tersebut dapat dilihat bahwa penggunaan CPU cenderung meningkat ketika terdapat penambahan jumlah permintaan. Pesan yang digunakan pada jumlah permintaan 1 sampai 5 berbeda dengan pesan yang digunakan pada jumlah permintaan 6 sampai 10. Terjadi peningkatan penggunaan CPU yang signifikan ketika terdapat pesan yang belum pernah diproses oleh *server* jika dibandingkan pada jumlah permintaan

lainnya. Apabila dilihat nilai persentase penggunaan CPU melebihi 100%. Hal tersebut dikarenakan pada perangkat lunak htop menghitung persentase pada masing-masing *core* yang pada perangkat keras. Ketika terdapat lebih dari satu *core* maka nilai persentase lebih dari 100% merupakan hal yang wajar. Hal itu menandakan proses yang diukur nilai persentasenya menggunakan lebih dari satu *core*. Peningkatan jumlah pesan yang diproses menyebabkan penggunaan CPU yang cenderung meningkat juga. Rata-rata peningkatan penggunaan CPU ketika jumlah permintaan bertambah 1 permintaan sebesar 23% dari permintaan sebelumnya. Untuk hasil uji sumber daya pada penyimpanan dapat dilihat pada gambar 5.11 berikut. Uji coba penyimpanan dilakukan 10 kali untuk masing-masing jumlah permintaan.



Gambar 5.11: Uji Sumber Daya Penyimpanan

Dari gambar 5.11 diatas dapat dilihat bahwa jumlah penyimpanan yang dibutuhkan oleh *server* cenderung meningkat seiring dengan bertambahnya jumlah permintaan. Rata-rata

peningkatan jumlah penyimpanan ketika terdapat satu permintaan baru adalah sebesar 6.7% dibanding permintaan sebelumnya.

5.3.3 Keakuratan *Query* Pencarian

Uji keakuratan *query* pencarian bertujuan untuk mengetahui seberapa akurat sistem dalam memberikan hasil penghitungan nilai kemiripan dari *query* pencarian yang dimasukan pengguna. Uji keakuratan query pencarian ini akan menghitung nilai *precision* dan *recall* dari hasil yang diberikan oleh sistem. Hasil yang diberikan oleh sistem akan dibandingkan dengan data yang diperoleh dari beberapa responden. Data dari responden yang dikumpulkan berupa pesan yang dianggap mirip dengan *query* pencarian tertentu. Tabel 5.1 merupakan hasil *query* pencarian dari sistem dengan menggunakan batas kemiripan 0.3.

Tabel 5.1: Data Hasil *Query* Pencarian Sistem

Nomor	Kata kunci	Ground truth
1	Gempa bumi	d11, d7, d4, d5
2	Motor	d41, d50,d32, d23, d21, d40
3	Mobil listrik	d46, d47, d30, d48, d42, d31, d33, d22, d50, d43

Tabel 5.2 merupakan data yang diperoleh dari responden yang akan digunakan untuk menghitung *precision* dan *recall* dari hasil *query* pencarian oleh sistem.

Tabel 5.2: Data Responden *Query* Pencarian

Responden	Kata kunci	Ground truth
1	Gempa bumi	d5, d7, d11
2	Motor	d21, d23, d37, d39, d40, d41, d50
3	Mobil listrik	d46, d35
4	Mobil listrik	d46, d39, d35
5	Mobil listrik	d35, d46
6	Mobil listrik	d35, d46
7	Mobil listrik	d35, d46
8	Gempa bumi	d5, d7, d11

Dari data-data pada tabel 5.1 dan 5.2 dapat dilakukan penghitungan terhadap *precision* dan *recall* dari proses penghitungan nilai kemiripan yang digunakan pada sistem yang dibangun. Untuk melakukan penghitungan *precision* dan *recall* dibawah ini merupakan *confusion matrix* yang digunakan untuk melakukan visualisasi performa suatu algoritma[1].

		System	
		True	*False*
Ground truth	*True*	TP	FN
	False	FP	TN

Gambar 5.12: Confusion Matrix

Gambar 5.12 memberikan gambaran untuk persamaan 5.1 yang digunakan untuk menghitung *precision* dan persamaan 5.2 yang digunakan untuk menghitung *recall*[2].

$$precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$recall = \frac{TP}{TP + FN} \tag{5.2}$$

$$TP = \textit{true positive}$$
$$FP = \textit{false positive}$$
$$FN = \textit{false negative}$$

Dari persamaan 5.1 dan 5.2 diatas masih terdapat variabel yang masih kurang jelas penggunaannya, yaitu TP, FP dan FN. TP(*true positive*) merupakan data yang oleh sistem dianggap benar dan oleh *ground truth* dianggap benar. FP(*false positive*) merupakan data yang oleh sistem dianggap benar akan tetapi oleh *ground truth* dianggap salah. FN(*false negative*) merupakan data yang oleh sistem dianggap salah akan tetapi oleh *ground truth* dianggap benar. Dibawah ini merupakan hasil dari penghitungan *precision* dan *recall* terhadap masing-masing data responden.

Tabel 5.3: Penghitungan *Precision* dan *Recall*

Responden	TP	FP	FN	Precision	Recall
1	3	1	0	0.75	1
2	5	2	1	0.83	0.71
3	1	9	1	0.1	0.5
4	1	9	2	0.1	0.3
5	1	9	1	0.1	0.5
6	1	9	1	0.1	0.5
7	1	9	1	0.1	0.5
8	3	1	0	0.75	1

Dari tabel 5.3 didapatkan rata-rata dari *precision* adalah 0.35 dan rata-rata dari *recall* adalah 0.64. Agar lebih mudah memahami, *recall* merupakan perbandingan dari jumlah pesan yang dianggap benar oleh sistem dan *ground truth* dibandingkan dengan jumlah semua *ground truth* sedangkan *precision* merupakan perbandingan dari jumlah pesan yang yang dianggap benar oleh sistem dan *ground truth* dibandingkan dengan jumlah semua pesan yang diberikan oleh sistem.

Kecilnya nilai *precision* dan *recall* dapat disebabkan oleh dua hal, yaitu cara pengumpulan data *ground truth* yang kurang tepat atau terdapat kesalahan pada penerapan penghitungan kemiripan yang dilakukan oleh sistem. Data *ground truth* mungkin saja kurang tepat karena responden yang mengisi data untuk digunakan *ground truth* akan menganggap *query* pencarian yang lebih dari satu kata sebagai suatu kesatuan sedangkan untuk penghitungan dianggap terpisah atau responden memilih pesan tidak berdasarkan ada tidaknya *query* pencarian pada pesan tersebut. Penghitungan kemiripan yang dilakukan oleh sistem mungkin terdapat kekurangan karena setiap terdapat pesan yang datang akan langsung dihitung nilai kemiripannya tanpa menunggu semua pesan yang selesai dikirim.

(Halaman ini sengaja dikosongkan)

BAB 6

PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Dari proses perancangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. Berhasil dibuat sistem yang dapat menjembatani antara *topic-based publish-subscribe* dan pengguna dengan mekanisme *content-filtering*.
2. Mengurutkan peringkat pada hasil *content filtering* dapat dilakukan dengan mengurutkan hasil *content filtering* berdasarkan nilai kemiripan dari besar ke kecil yang diperoleh dari penghitungan *cosine similarity*.
3. Berdasarkan uji ketahanan, penggunaan *web socket* untuk membuat *gateway client* yang menghubungkan beberapa *topic-based publish-subscribe* RabbitMQ dapat melayani permintaan hingga 600 permintaan.
4. Berdasarkan hasil uji keakuratan *query* pencarian, penggunaan *cosine similarity* untuk melakukan *content filtering* pada pola pesan *topic-based publish-subscribe* menghasilkan rata-rata *recall* sebesar 64%.
5. Dari hasil uji coba kecepatan, penggunaan *cache* pada *library* sastrawi memberikan peningkatan kecepatan rata-rata sebesar 98% dibandingkan ketika pesan belum pernah diproses oleh *server*.
6. Dari hasil uji coba sumber daya, penerapan *single thread asynchronous I/O web framework* pada pola pesan *topic-based publish-subscribe* memiliki pengingkatan

jumlah penyimpanan rata-rata sebesar 6.7% dan 23% peningkatan penggunaan CPU dari sebelumnya.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

- Penggunaan *cache stemming* pada *library* sastrawi secara *real-time* untuk semua permintaan sehingga semakin mempercepat proses pengubahan kata menjadi kata dasar.
- Pembatasan jumlah proses yang dilayani pada masing-masing permintaan ketika terdapat banyak permintaan supaya permintaan yang datang terlambat akan tetap dilayani secara cepat tanpa harus menunggu semua proses dari permintaan yang lebih dulu datang selesai diproses.
- Pengubahan cara penghitungan nilai kemiripan untuk mendapatkan nilai *precision* dan *recall* yang lebih baik.

DAFTAR PUSTAKA

[1] Howard Hamilton, "*Confusion matrix*", *Department of Computer Science*, *University of* Regina, 8 Juni 2012. [Online]. Available : http://www2.cs.uregina.ca/ dbd/cs831/notes/confusion_matrix /confusion_matrix.html. [Diakses 3 Juni 2017]

[2] "*Chapter* 7", *Information Retrieval Group, University of* Glasgow. [Online]. Available : http://www.dcs.gla.ac.uk/Keith/Chapter.7/Ch.7.html. [Diakses 3 Juni 2017]

[3] "*Dot products*", Stanford, 7 April 2009. [Online]. Available : https://nlp.stanford.edu/IR-book/html/htmledition/dot-products-1.html. [Diakses 19 Mei 2017]

[4] "*Feature extraction*", Scikit-learn. [Online]. Available: http://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weighting. [Diakses 19 Mei 2017]

[5] "*Thread Pools Windows*", Microsoft. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms686760(v=vs.85).aspx. [Diakses 13 Mei 2017]

[6] "*Asynchronous and non-Blocking I/O*", Tornado. [Online]. Available: http://www.tornadoweb.org/en/stable/guide/async.html. [Diakses 12 Mei 2017]

[7] "Tornado *Web Server*", Tornadoweb. [Online]. Available: http://www.tornadoweb.org/en/stable/. [Diakses pada 7 Mei 2017]

[8] I. Fette, A. Melnikov, "Web Socket Protokol", Internet Engineering Task Force, RFC 6455, December 2011.

[9] Subhashini, R. and Kumar, V. J. S., ”*Evaluating the Performance of Similarity Measures Used in Document Clustering and Information Retrieval*”, 2010 *First International Conference on Integrated Intelligent Computing*, pp. 27-31, Agustus 2010.

[10] ”*An Introduction to Information Retrieval*”, Cambridge University Press, pp. 199-133, April 2009.

[11] Manning, Christopher D. and Raghavan, Prabhakar and Schütze, Hinrich, ”*Boolean retrieval*”, *Introduction to information retrieval*, pp. 1-18, 2008.

[12] Eugster, Patrick Th. and Felber, Pascal A. and Guerraoui, Rachid and Kermarrec, Anne-Marie, ”*The Many Faces of Publish/Subscribe*”, ACM Comput. Surv., pp. 114-131, Juni 2003.

[13] ”RabbitMQ”, RabbitMQ. [Online]. Available: https://www.rabbitmq.com/features.html. [Diakses 3 Juni 2017]

BIODATA PENULIS

Naufal Fakhri Muhammad, lahir di Klaten, pada tanggal 21 Juni 1995. Penulis menempuh pendidikan mulai dari SDN 2 PUNDUNGSARI (2001-2007), SMPN I CAWAS (2007-2010), SMAN 1 KLATEN (2010-2013) hingga terakhir Institut Teknologi Sepuluh Nopember Surabaya (2013-2017) di jurusan Teknik Informatika, Fakultas Teknologi Informasi angkatan tahun 2013.

Selama berada di kampus penulis aktif dalam organisasi keagaaman Keluarga Muslim Informatika. Pada organisasi keagamaan ini penulis berkesempatan untuk menjadi anggota dan menjadi pengurus inti. Selain itu penulis juga aktif dalam unit kegiatan mahasiswa (UKM) Jiu-Jitsu. Selama belajar di kampus Teknik Informatika, penulis mengembangkan minat pada bidang Managemen Informasi (MI) dan Komputasi Berbasis Jaringan (KBJ). Komunikasi dengan penulis dapat melalui email: naufal.fakhri.m@gmail.com

(Halaman ini sengaja dikosongkan)