BAB 6 PENGUJIAN DAN ANALISIS

Bab ini bertujuan untuk membahas pengujian yang dilakukan dan menganalisis hasil dari pengujian tersebut. Adapun pada bab ini akan disajikan data yang didapatkan dalam bentuk tabel. Kemudian data tersebut dirubah menjadi suatu grafik dan diagram untuk dianalisis dan melihat pola – pola yang dibentuk.

6.1 Pengujian waktu untuk melakukan penyebaran pesan antar *broker* menggunakan *bridge*

6.1.1 Tujuan Pengujian

Mendapatkan nilai waktu yang dibutuhkan oleh *broker* yang menerima pesan dari *publisher* untuk dibagikan pesan ke *broker* lainnya yang terhubung dengan *bridge*. Sehingga didapatkan nilai waktu yang dibutuhkan *broker* untuk menyebarkan pesan.

6.1.2 Langkah – langkah

1. *Broker* 1, *broker 2,* dan *broker 3* dalam kondisi aktif.
2. *Publisher* dibuat menggunakan program *Python* yang memanfaatkan *multi thread*. Dimana jumlah *thread* sebanyak 25, 50, 75, 100, dan 125 *thread* dimana setiap *thread* merepresentasikan *publisher* yang mengirimkan 1 pesan.
3. *Thread* mengirimkan pesan ke *broker 1*.
4. Untuk pengambilan data digunakan *Wireshark*.
5. Pengujian dilakukan sebanyak 5 kali untuk setiap jumlah *thread* yang ada.

6.1.3 Data yang didapatkan

Tabel 6.1 adalah hasil percobaan ketika jumlah pesan yang dikirimkan ke *broker 1* sebanyak 25 pesan. Diperlukan waktu yang tertera pada tabel 6.1 untuk bisa menyebarkan pesan ke *broker 2* dan *broker 3*.

Tabel 6.1 Delay Sinkronisasi dengan jumlah *Publish 25 Pesan*

Percobaan	Delay Pengiriman (s)		RATA - RATA
	Broker 2	*Broker* 3	
1	0.304	0.348	0.326
2	0.324	0.356	0.340
3	0.446	0.488	0.467
4	0.303	0.382	0.343
5	0.308	0.363	0.335
Rata - Rata	0.337	0.387	0.362

Tabel 6.2 adalah hasil percobaan ketika jumlah pesan yang dikirimkan ke *broker 1* sebanyak 50 pesan. Diperlukan waktu yang tertera pada tabel 6.1 untuk bisa menyebarkan pesan ke *broker 2* dan *broker 3*.

Tabel 6.2 Delay Sinkronisasi dengan jumlah *Publish 50 Pesan*

Percobaan	Delay Pengiriman (s)		RATA - RATA
	Broker 2	*Broker* 3	
1	0.844	0.779	0.811
2	0.672	0.828	0.750
3	0.701	0.797	0.749
4	0.530	0.755	0.643
5	0.403	0.561	0.482
Rata - Rata	0.630	0.744	0.687

Tabel 6.3 adalah hasil percobaan ketika jumlah pesan yang dikirimkan ke *broker 1* sebanyak 75 pesan. Diperlukan waktu yang tertera pada tabel 6.1 untuk bisa menyebarkan pesan ke *broker 2* dan *broker 3*.

Tabel 6.3 Delay Sinkronisasi dengan jumlah *Publish 75 Pesan*

Percobaan	Delay Pengiriman (s)		RATA - RATA
	Broker 2	*Broker* 3	
1	0.622	0.928	0.775
2	0.687	0.982	0.835
3	0.711	1.031	0.871
4	0.826	1.255	1.040
5	0.757	1.110	0.933
Rata - Rata	0.721	1.061	0.891

Tabel 6.4 adalah hasil percobaan ketika jumlah pesan yang dikirimkan ke *broker 1* sebanyak 100 pesan. Diperlukan waktu yang tertera pada tabel 6.1 untuk bisa menyebarkan pesan ke *broker 2* dan *broker 3*.

Tabel 6.4 Delay Sinkronisasi dengan jumlah *Publish 100 Pesan*

Percobaan	Delay Pengiriman (s)		RATA - RATA
	Broker 2	*Broker* 3	
1	1.003	1.514	1.258
2	1.072	1.449	1.260
3	0.737	1.345	1.041
4	1.246	1.560	1.403
5	0.810	1.257	1.033
Rata - Rata	0.973	1.425	1.199

Tabel 6.5 adalah hasil percobaan ketika jumlah pesan yang dikirimkan ke *broker 1* sebanyak 125 pesan. Diperlukan waktu yang tertera pada tabel 6.1 untuk bisa menyebarkan pesan ke *broker 2* dan *broker 3*.

Tabel 6.5 Delay Sinkronisasi dengan jumlah *Publish 125 Pesan*

Percobaan	Delay Pengiriman (s)		RATA - RATA
	Broker 2	*Broker* 3	
1	1.569	2.214	1.891
2	1.523	1.843	1.683
3	1.590	1.850	1.720
4	1.134	1.769	1.451
5	1.122	1.602	1.362
Rata - Rata	1.388	1.856	1.622

6.1.4 Analisis

Gambar 6.1 Grafik waktu penyebaran pesan

Dari hasil data tabel yang berhasil di dapatkan dari pengujian ini, maka bisa dilihat ketika pesan yang dikirimkan semakin banyak. Maka waktu dibutuhkan oleh *broker* untuk meneruskan pesan ke *broker* lainnya semakin lama. Terlihat dari gambar 6.1, ketika *broker 1* hanya meneruskan 25 pesan, waktu yang dibutuhkan berada di sebesar 0.362 detik. Akan tetapi ketika jumlah pesan yang harus diteruskan bertambah, maka waktu yang diperlukan juga bertambah seperti pengujian dengan 50 pesan, 75 pesan, 100 pesan, dan 125 pesan. Terjadi kenaikkan waktu yang dibutuhkan untuk meneruskan pesan yang ada. Hal ini dipengaruhi oleh utilisasi *CPU* di *broker 1*. *Broker 1* memerlukan waktu untuk untuk memproses setiap pesan yang masuk. Semakin banyak pesan yang masuk,

maka semakin besar waktu yang diperlukan oleh *broker 1* untuk mengirimkan pesan tersebut ke *broker* lainnya untuk melakukan sinkronisasi pesan.

6.2 Pengujian CPU *usage* ketika melakukan penyebaran pesan antar *broker* menggunakan *bridge*

6.2.1 Tujuan Pengujian

Mendapatkan nilai besar utilisasi CPU setiap *broker* ketika menyebarkan dan menerima pesan. Dimana pengujian ini akan lebih menitik beratkan pada *broker* yang berperan sebagai *bridge* dan harus meneruskan pesan ke semua *broker* yang ada.

6.2.2 Langkah – langkah

1. *Broker* 1 dalam kondisi aktif
2. *Publisher* dibuat menggunakan program *Python* yang memanfaatkan *multi thread*. Dimana jumlah *thread* sebanyak 25, 50, 75, 100, dan 125 *thread* dimana setiap *thread* merepresentasikan *publisher* yang mengirimkan 1 pesan.
3. *Thread* mengirimkan pesan ke *broker 1*.
4. Untuk pengambilan data menggunakan *tools pidstat* pada setiap *broker* untuk mencatat utilisasi *CPU.*
5. *Interval* pencatatan adalah setiap 2 detik dengan 3 kali pencatatan. Sehingga proses pencatatan utilisasi CPU dilakukan selama 6 detik, kemudian semua hasil utilisasi CPU yang dicatat dibagi dengan jumlah pencatatan.
6. Ulangi langkah 1 hingga 5 dengan menambahkan 1 *broker* aktif sampai semua *broker* menjadi aktif.
7. Percobaan dilakukan sebanyak 5 kali untuk setiap pesan dan jumlah *broker* aktif.

6.2.3 Data yang didapatkan

Tabel 6.6 adalah hasil percobaan ketika mengirimkan 25 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* menerima pesan dari *publisher* dan tidak melakukkan *bridge* ke *broker* lainnya.

Tabel 6.6 Penggunaan *Resource* CPU *Broker* 1 aktif dengan 25 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	5.45	5.83	6.00	6.20	5.30	5.76
Broker 2	0	0	0	0	0	0

Broker 3	0	0	0	0	0	0

Tabel 6.7 adalah hasil percobaan ketika mengirimkan 50 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* menerima pesan dari *publisher* dan tidak melakukkan *bridge* ke *broker* lainnya.

Tabel 6.7 Penggunaan *Resource* CPU *Broker* 1 aktif dengan 50 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	10.30	9.36	11.11	11.82	11.62	10.84
Broker 2	0.00	0.00	0.00	0.00	0.00	0.00
Broker 3	0.00	0.00	0.00	0.00	0.00	0.00

Tabel 6.8 adalah hasil percobaan ketika mengirimkan 75 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* menerima pesan dari *publisher* dan tidak melakukkan *bridge* ke *broker* lainnya.

Tabel 6.8 Penggunaan *Resource* CPU *Broker* 1 aktif dengan 75 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	16.40	16.37	17.21	16.73	15.26	16.39
Broker 2	0.00	0.00	0.00	0.00	0.00	0.00
Broker 3	0.00	0.00	0.00	0.00	0.00	0.00

Tabel 6.9 adalah hasil percobaan ketika mengirimkan 100 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* menerima pesan dari *publisher* dan tidak melakukkan *bridge* ke *broker* lainnya.

Tabel 6.9 Penggunaan *Resource* CPU *Broker* 1 aktif dengan 100 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	22.61	21.25	22.19	21.86	21.31	21.84
Broker 2	0.00	0.00	0.00	0.00	0.00	0.00
Broker 3	0.00	0.00	0.00	0.00	0.00	0.00

Tabel 6.10 adalah hasil percobaan ketika mengirimkan 125 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* menerima pesan dari *publisher* dan tidak melakukkan *bridge* ke *broker* lainnya.

Tabel 6.10 Penggunaan *Resource* CPU *Broker* 1 aktif dengan 125 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	28.84	32.56	24.86	27.78	32.16	29.24
Broker 2	0.00	0.00	0.00	0.00	0.00	0.00
Broker 3	0.00	0.00	0.00	0.00	0.00	0.00

Tabel 6.11 adalah hasil percobaan ketika mengirimkan 25 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* melakukan *bridge* ke *broker 2* agar pesan yang diterima *broker* 1 bisa diterima oleh *broker* 2.

Tabel 6.11 Penggunaan *Resource* CPU *Broker* 1 & 2 aktif dengan 25 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	8.54	6.86	7.59	7.75	6.24	7.40
Broker 2 (Aktif)	6.13	4.55	5.63	5.82	6.30	5.69
Broker 3	0.00	0.00	0.00	0.00	0.00	0.00

Tabel 6.12 adalah hasil percobaan ketika mengirimkan 50 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* melakukan *bridge* ke *broker 2* agar pesan yang diterima *broker* 1 bisa diterima oleh *broker* 2.

Tabel 6.12 Penggunaan *Resource* CPU *Broker* 1 & 2 aktif dengan 50 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	14.32	13.76	13.56	14.46	14.05	14.03
Broker 2 (Aktif)	10.80	19.56	9.26	10.80	8.19	11.72
Broker 3	0.00	0.00	0.00	0.00	0.00	0.00

Tabel 6.13 adalah hasil percobaan ketika mengirimkan 75 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* melakukan *bridge* ke *broker 2* agar pesan yang diterima *broker* 1 bisa diterima oleh *broker* 2.

Tabel 6.13 Penggunaan *Resource* CPU *Broker* 1 & 2 aktif dengan 75 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	16.72	17.84	19.06	18.02	18.63	18.05
Broker 2 (Aktif)	13.93	13.81	14.39	16.27	12.65	14.21
Broker 3	0.00	0.00	0.00	0.00	0.00	0.00

Tabel 6.14 adalah hasil percobaan ketika mengirimkan 100 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* melakukan *bridge* ke *broker 2* agar pesan yang diterima *broker* 1 bisa diterima oleh *broker* 2.

Tabel 6.14 Penggunaan *Resource* CPU *Broker* 1 & 2 aktif dengan 100 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	26.36	26.97	24.91	26.21	25.98	26.09
Broker 2 (Aktif)	20.17	20.03	21.64	20.84	21.84	20.90
Broker 3	0.00	0.00	0.00	0.00	0.00	0.00

Tabel 6.15 adalah hasil percobaan ketika mengirimkan 125 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* melakukan *bridge* ke *broker 2* agar pesan yang diterima *broker* 1 bisa diterima oleh *broker* 2.

Tabel 6.15 Penggunaan *Resource* CPU *Broker* 1 & 2 aktif dengan 125 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	32.55	31.79	35.98	38.79	33.07	34.44
Broker 2 (Aktif)	27.56	22.48	28.39	29.70	24.95	26.62
Broker 3	0.00	0.00	0.00	0.00	0.00	0.00

Tabel 6.16 adalah hasil percobaan ketika mengirimkan 25 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* melakukan *bridge* ke *broker 2* dan *broker 3*, sehingga pesan yang diterima *broker* 1 bisa diterima didistribusikan ke semua *broker*.

Tabel 6.16 Penggunaan *Resource* CPU *Broker* 1 ,2, & 3 aktif dengan 25 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	11.07	11.71	12.99	12.85	14.43	12.61
Broker 2 (Aktif)	4.72	6.16	6.32	7.07	5.99	6.05
Broker 3 (Aktif)	4.38	5.25	4.23	4.19	4.40	4.49

Tabel 6.17 adalah hasil percobaan ketika mengirimkan 50 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* melakukan *bridge* ke *broker 2* dan *broker 3*, sehingga pesan yang diterima *broker* 1 bisa diterima didistribusikan ke semua *broker*.

Tabel 6.17 Penggunaan *Resource* CPU *Broker* 1 ,2, & 3 aktif dengan 50 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	

Broker 1 (Aktif)	25.93	24.35	25.19	24.95	24.72	25.03
Broker 2 (Aktif)	12.28	12.74	12.22	13.40	13.89	12.91
Broker 3 (Aktif)	13.27	11.58	12.64	13.46	12.87	12.76

Tabel 6.18 adalah hasil percobaan ketika mengirimkan 75 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* melakukan *bridge* ke *broker 2* dan *broker 3*, sehingga pesan yang diterima *broker* 1 bisa diterima didistribusikan ke semua *broker*.

Tabel 6.18 Penggunaan *Resource* CPU *Broker* 1 ,2, & 3 aktif dengan 75 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	43.20	35.69	36.91	46.56	35.66	39.60
Broker 2 (Aktif)	16.87	20.14	20.63	15.55	16.28	17.89
Broker 3 (Aktif)	19.35	13.16	18.81	13.25	19.00	16.71

Tabel 6.19 adalah hasil percobaan ketika mengirimkan 75 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* melakukan *bridge* ke *broker 2* dan *broker 3*, sehingga pesan yang diterima *broker* 1 bisa diterima didistribusikan ke semua *broker*.

Tabel 6.19 Penggunaan *Resource* CPU *Broker* 1 ,2, & 3 aktif dengan 100 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	69.27	58.61	60.78	66.38	56.02	62.21
Broker 2 (Aktif)	21.92	22.43	22.34	20.11	21.61	21.68
Broker 3 (Aktif)	22.22	24.68	19.47	18.88	25.55	22.16

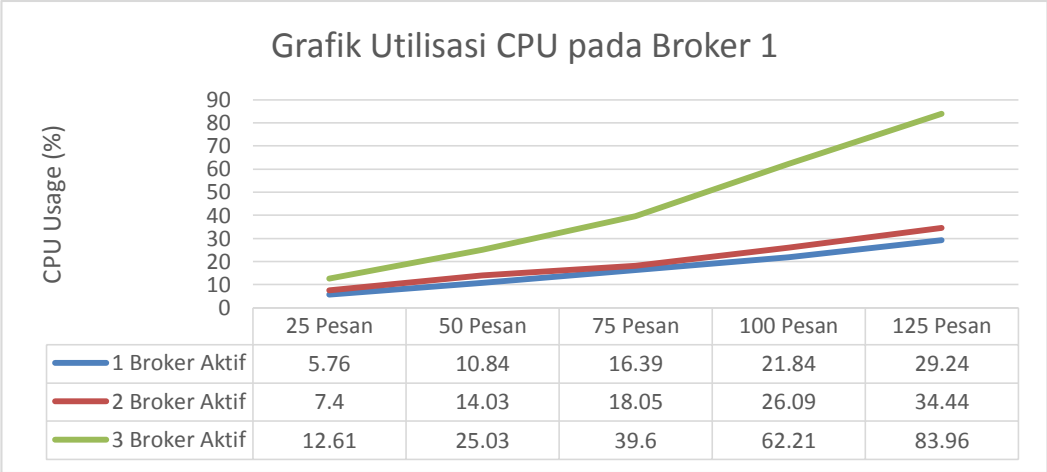
Tabel 6.20 adalah hasil percobaan ketika mengirimkan 125 pesan ke *broker 1*. Dimana tabel ini menampilkan besar *resource* yang digunakan ketika *broker 1* melakukan *bridge* ke *broker 2* dan *broker 3*, sehingga pesan yang diterima *broker* 1 bisa diterima didistribusikan ke semua *broker*.

Tabel 6.20 Penggunaan *Resource* CPU *Broker* 1 ,2, & 3 aktif dengan 125 *Publish*er

Kondisi *Broker*	Percobaan CPU Usage (%)					Rata - Rata
	I	II	III	IV	V	
Broker 1 (Aktif)	87.00	83.80	84.00	82.80	82.21	83.96
Broker 2 (Aktif)	25.32	26.57	26.46	26.82	28.36	26.71
Broker 3 (Aktif)	22.73	25.93	29.72	25.22	24.60	25.64

6.2.4 Analisis

Berdasarkan data dari tabel yang didapatkan, maka didapatkan data CPU *Usage* yang digunakan setiap *broker* untuk melakukan pemrosesan pesan. Percobaan awal menggunakan 1 *broker* yang aktif, dimana *broker* tersebut adalah *broker 1.* Ketika hanya menggunakan 1 *broker* aktif, maka *broker 1* hanya melakukan proses membuat koneksi dengan *client publisher* dan hanya menerima pesan yang dikirimkan oleh *publisher* tanpa harus menyebarkan pesan tersebut ke *broker* lainnya karena *broker* lain dalam kondisi tidak aktif (*inactive)*. Kemudian pada percobaan dengan 2 *broker* yang aktif, yaitu menggunakan *broker 1* dan *broker 2*. Ketika pesan dikirimkan ke b*roker 1,* terjadi kenaikkan penggunaan *resource CPU*, hal ini disebabkan karena pada percobaan ini, selain harus membuat koneksi dan menerima pesan dari *publisher*. *Broker 1* juga diharuskan melakukan *publish* ulang, dimana pesan *publish* tersebu dikirimkan ke *broker* lain yang tersambung dengan *broker 1* sesuai dengan arsitektur jaringan yang digunakan pada penelitian ini. Sehingga akhirnya *broker 2* akan menerima pesan yang sama seperti *broker 1*. Pada gambar 6.2, percobaan dengan 2 *broker* aktif dilambangkan dengan garis merah, yang menunjukkan terjadi kenaikkan CPU yang tidak signifikan. Akan tetapi pada garis hijau yang melambangkan 3 *broker* aktif, terjadi kenaikkan *resource* yang cukup signifikan, karena *broker 1* harus mengirimkan pesan tambahan. Semisal pada pengujian dengan semua *broker* aktif dan 125 pesan, maka ketika *broker 1* menerima 125 *request* dari publisher, selain *broker 1* harus menjaga koneksi dan menerima pesan *publish, broker 1* juga diharuskan mengirimkan kembali pesan *publish* sejumlah 125 pesan ke *broker 2* dan *broker* 3. Hal ini tentu memperberat kerja *broker 1*.

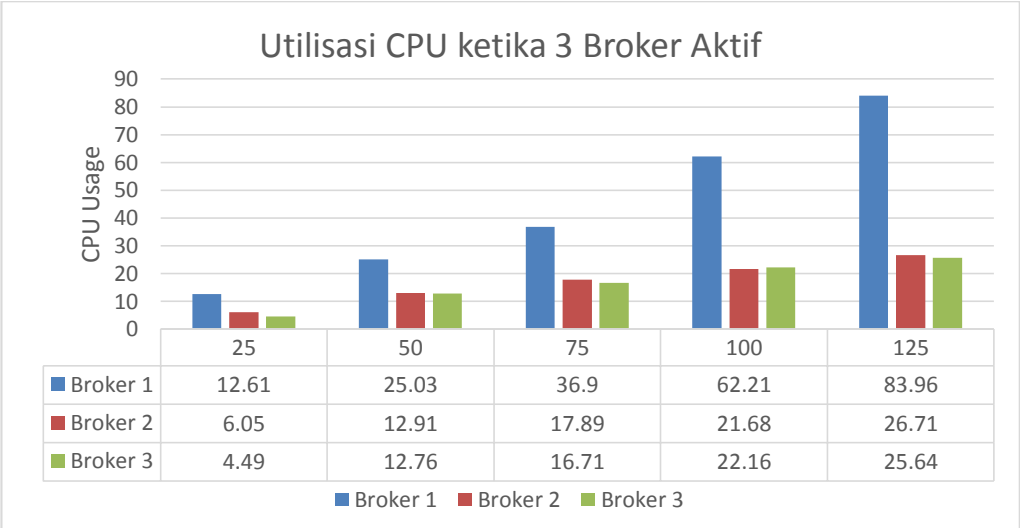


	25 Pesan	50 Pesan	75 Pesan	100 Pesan	125 Pesan
1 Broker Aktif	5.76	10.84	16.39	21.84	29.24
2 Broker Aktif	7.4	14.03	18.05	26.09	34.44
3 Broker Aktif	12.61	25.03	39.6	62.21	83.96

Gambar 6.2 Grafik Penggunaan *Resource Broker 1*

Selain itu, berdasarkan rancangan arsitektur yang ada di bab perancangan. Posisi *broker 1* berada diantara *broker 2* dan *broker 3* sehingga *broker 1* bertanggung jawab mengirimkan pesan ke *broker 2* dan *broker 3*. Gambar 6.3 akan menunjukkan penggunaan *resource* setiap *broker* ketika melakukan sinkronisasi.

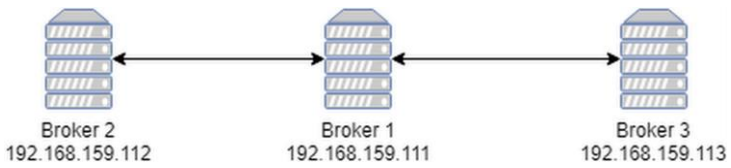
Resource CPU yang digunakan oleh *broker 1* terlihat yang tertinggi karena selain harus menerima koneksi dan pesan dari *publisher, broker 1* juga bertugas mengirimkan pesan ke *broker 2* dan *broker 3.* Dimana *broker 2* dan *broker 3* hanya menerima pesan *publish*. Sehingga penggunaan CPU *broker 1* berbeda cukup jauh dengan 2 *broker* lainnya. Berdasarkan diagram batang pada gambar 6.3, terlihat kenaikkan utilisasi CPU *pada broker 1* semakin bertambah tinggi seiring dengan bertambahnya jumlah *client publisher*.



	25	50	75	100	125
Broker 1	12.61	25.03	36.9	62.21	83.96
Broker 2	6.05	12.91	17.89	21.68	26.71
Broker 3	4.49	12.76	16.71	22.16	25.64

Gambar 6.3 Diagram Batang Penggunaan *Resource* setiap *Broker*

Berdasarkan gambar 6.3, dilihat beban *broker 1* sangat tinggi, dikarenakan desain jaringan pada gambar 6.4 yang menunjukkan *broker 1* berada diantara *broker 2* dan *broker 3*. Sehingga *broker 1* harus menyebarkan seluruh pesan ke *broker* lainnya. Jika pengujian dilakukan dengan mengirimkan pesan ke *broker 2*. Maka *broker* 2 akan menerima dan mengirimkan pesan *publish* ke *broker 1*, Setelah itu *broker 1* akan melakukan hal yang sama seperti *broker 2,* yaitu mengirimkan pesan *publish* ke *broker 3*.



Gambar 6.4 Arsitektur *bridge Mosquitto*

6.3 Pengujian waktu yang dibutuhkan *broker* untuk menangani *request subscribe*

6.3.1 Tujuan Pengujian

Mendapatkan nilai dari waktu yang dibutuhkan oleh *broker* untuk memproses dan menangani semua pesan *subscribe* yang ada. Pengujian ini dilakukan dengan 2 variabel pembanding, yaitu ketika sistem menggunakan *load balancer* dan tidak menggunakan *load balancer*. Dengan parameter uji yaitu jumlah *subscriber* dengan tujuan untuk mendapatkan waktu yang dibutuhkan setiap *broker* dan keseluruhan sistem untuk menangani *request* dari *subscriber*.

6.3.2 Langkah – langkah

1. *Broker* 1, *Broker 2,* dan *Broker 3* dalam kondisi aktif.
2. *Subscriber* dibuat menggunakan program *Python* yang memanfaatkan *multi thread*. Dimana jumlah *thread* sebanyak 300, 600, 900, dan 1200 *thread* dimana setiap *thread* merepresentasikan *subscriber*.
3. Setiap *broker* menerima koneksi dari *subscriber* sebanyak 100, 200, dan 300. Contoh : *Broker 1* = 100 *request*, *Broker 2 = 100 request* dan *Broker 3* = 100 *request*.
4. Sehingga pada 1 waktu akan terjadi 300 , 600, 900, dan 1200 koneksi.
5. Data diambil menggunakan *Wireshark*.
6. Percobaan dilakukan sebanyak 5 kali untuk setiap jumlah *subscriber*.
7. Pengujian diulang dengan skenario *subscriber* melakukan koneksi melalui *load balancer*.

6.3.3 Data yang didapatkan

Dari pengujian ini didapatkan tabel yang menampilkan durasi *broker* dalam menyelesaikan *request* yang ada. Tabel 6.21 berisikan waktu yang diperlukan oleh *broker* untuk menyelesaikan *request* 100 *subscriber* ke setiap *broker* yang ada.

Tabel 6.21 Delay waktu 100 *Subscribe*r pada setiap *Broker*

100 Client	Durasi menyelesaikan semua request (s)					Rata - Rata
	I	II	III	IV	V	
Broker 1	2.44	2.90	3.01	3.01	2.66	2.80
Broker 2	2.99	2.79	3.02	2.60	3.01	2.88
Broker 3	3.03	3.35	3.07	3.14	2.81	3.08
Rata - Rata	2.82	3.01	3.03	2.92	2.82	2.92

Tabel 6.22 berisikan data utilisasi CPU ketika 200 *subscriber* melakukan koneksi secara langsung ke setiap *broker*

Tabel 6.22 Delay waktu 200 *Subscribe*r pada setiap *Broker*

200 Client	Durasi menyelesaikan semua request (s)					Rata - Rata
	I	II	III	IV	V	
Broker 1	5.48	5.36	5.47	5.62	5.70	5.53
Broker 2	5.41	4.88	5.35	5.26	5.89	5.36
Broker 3	5.85	6.12	6.30	6.52	6.48	6.26
Rata - Rata	5.58	5.46	5.71	5.80	6.02	5.71

Tabel 6.23 berisikan data waktu yang diperlukan ketika 300 *subscriber* melakukan koneksi secara langsung ke setiap *broker*

Tabel 6.23 Delay waktu 300 *Subscribe*r pada setiap *Broker*

300 Client	Durasi menyelesaikan semua request (s)					Rata - Rata
	I	II	III	IV	V	
Broker 1	8.75	8.82	8.15	9.09	8.13	8.59
Broker 2	8.47	8.51	8.17	8.72	8.54	8.48
Broker 3	9.31	9.76	8.65	9.34	8.64	9.14
Rata - Rata	8.85	9.03	8.32	9.05	8.44	8.74

Tabel 6.24 berisikan data waktu yang diperlukan oleh *broker* ketika 400 *subscriber* melakukan koneksi melalui *load balancer*.

Tabel 6.24 Delay waktu 400 *Subscribe*r pada setiap *Broker*

400 Client	Durasi menyelesaikan semua request (s)					Rata - Rata
	I	II	III	IV	V	
Broker 1	10.85	10.95	11.25	12.09	12.58	11.54
Broker 2	13.65	13.51	11.47	13.14	12.54	12.86
Broker 3	14.22	14.52	13.05	13.97	13.62	13.88
Rata - Rata	12.91	12.99	11.92	13.06	12.91	12.76

2. **Data dimana sistem menggunakan *Load Balancer***

Tabel 6.25 berisikan data waktu yang diperlukan oleh broker ketika 300 subscriber melakukan koneksi melalui load balancer.

Tabel 6.25 Delay waktu kecepatan 600 *Subscribe* menggunakan *load balancer*

300 Client 100/broker	Durasi menyelesaikan semua request (s)					Rata - Rata
	I	II	III	IV	V	
Broker 1	3.03	3.20	2.93	2.87	3.14	3.03
Broker 2	3.31	3.39	2.92	2.86	2.85	3.07
Broker 3	3.03	2.92	3.09	3.03	2.95	3.00

Rata – Rata	3.12	3.17	2.98	2.92	2.98	3.03

Tabel 6.26 berisikan data waktu yang diperlukan oleh *broker* ketika 600 *subscriber* melakukan koneksi melalui *load balancer*..

Tabel 6.26 Delay waktu kecepatan 1200 *Subscribe* menggunakan *load balancer*

600 Client 200/broker	Durasi menyelesaikan semua request (s)					Rata - Rata
	I	II	III	IV	V	
Broker 1	6.09	5.46	6.14	5.95	6.16	5.96
Broker 2	5.82	5.84	5.35	6.32	6.34	5.94
Broker 3	5.99	5.51	5.76	6.16	6.39	5.96
Rata – Rata	5.97	5.60	5.75	6.14	6.30	5.95

Tabel 6.27 berisikan data waktu yang diperlukan oleh *broker* ketika 900 *subscriber* melakukan koneksi melalui *load balancer*.

Tabel 6.27 Delay waktu kecepatan 1800 *Subscribe* menggunakan *load balancer*

900 Client 300/broker	Durasi menyelesaikan semua request (s)					Rata - Rata
	I	II	III	IV	V	
Broker 1	9.48	8.65	9.14	8.78	8.88	8.98
Broker 2	8.98	8.42	9.00	9.42	8.99	8.96
Broker 3	8.63	8.75	9.89	8.14	9.57	9.00
Rata – Rata	9.03	8.60	9.34	8.78	9.15	8.98

Tabel 6.28 berisikan data waktu yang diperlukan oleh *broker* ketika 1200 *subscriber* melakukan koneksi melalui *load balancer*.

Tabel 6.28 Delay waktu kecepatan 1800 *Subscribe* menggunakan *load balancer*

1200 Client 400/broker	Durasi menyelesaikan semua request (s)					Rata - Rata
	I	II	III	IV	V	
Broker 1	12.36	13.43	12.75	12.50	12.83	12.77
Broker 2	12.46	13.03	13.78	13.13	12.91	13.06
Broker 3	13.12	12.96	13.15	13.28	13.05	13.11
Rata - Rata	12.65	13.14	13.23	12.97	12.93	12.98

Tabel 6.29 menampilkan total waktu yang diperlukan oleh ketiga *broker,* dimana untuk mendapatkan nilai tersebut menggunakan hasil dari waktu tercepat *broker* menerima pesan *connect* dan waktu terakhir ketika *broker* mengirimkan pesan *suback*. Hal ini untuk mengetahui total lama waktu pemrosesan. Untuk nilai total waktu pada kolom LB, nilai didapatkan dari waktu ketika *load balancer* menerima pesan *connect* pertama dan ketika *load balancer* mengirimkan pesan *suback* yang terakhir. Kolom “NO LB” menandakan bahwa percobaan tidak menggunakan *load balancer,*

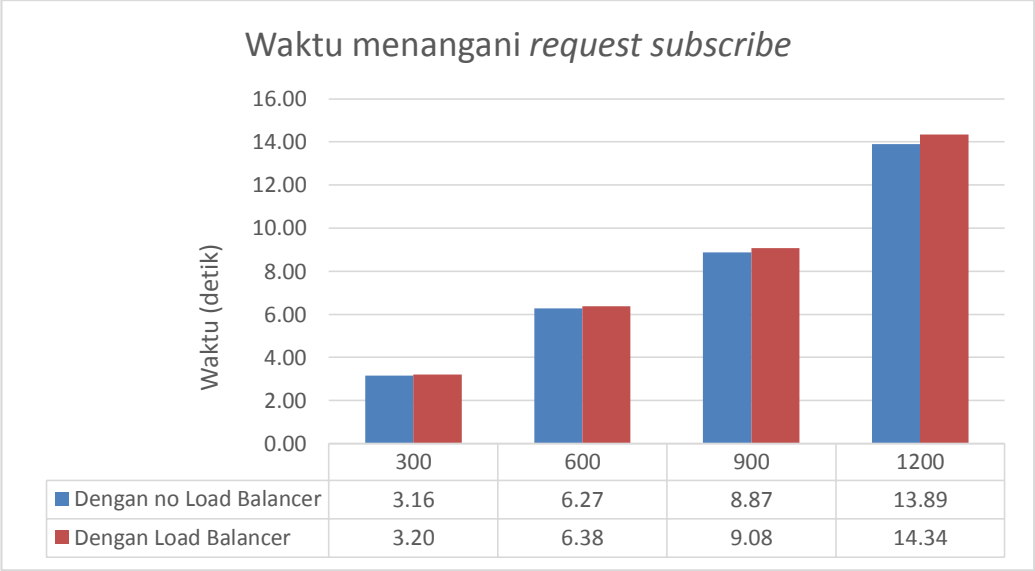
sedangkan kolom “LB” menampilkan total waktu ketika menggunakan *load balancer*.

Tabel 6.29 Perbandingan Delay waktu kecepatan *Subscribe*

PERCOBAAN	300		600		900		1200	
	NO LB	LB	NO LB	LB	NO LB	LB	NO LB	LB
1	3.04	3.31	5.85	6.09	8.90	9.20	14.26	14.14
2	3.36	3.41	6.12	6.85	9.10	8.62	14.53	14.44
3	3.09	3.11	6.30	6.22	8.53	9.36	13.05	14.78
4	3.17	3.04	6.52	6.33	9.12	8.90	13.97	14.28
5	3.16	3.14	6.54	6.42	8.70	9.31	13.62	14.07
Rata - Rata	3.16	3.20	6.27	6.38	8.87	9.08	13.89	14.34

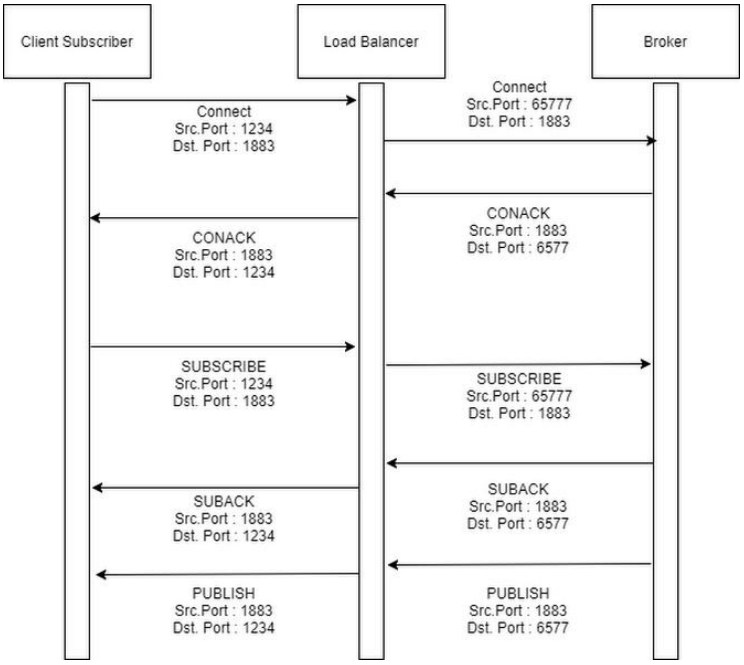
6.3.4 Analisis

Dari data tabel hasil pengujian yang ada. Maka bisa dilihat dari gambar 6.5. Terlihat ketika menggunakan *load balancer*. Waktu *request* menjadi sedikit lebih lama, dimana selisih waktu antara ketika menggunakan *load balancer* dengan tidak menggunakan *load balancer* semakin jauh ketika *subscriber* yang ada semakin banyak. Hal ini terjadi karena saat menggunakan *load balancer*. Pesan dari *subscriber* mengalami waktu *pemrosesan* di *load balancer*. Sehingga dibutuhkan tambahan waktu ketika paket masuk ke *load balancer*. Sedangkan ketika *subscriber* melakukan koneksi secara langsung dengan *broker*, tentu hasilnya akan lebih cepat karena waktu proses di *load balancer* tidak diperhitungkan.



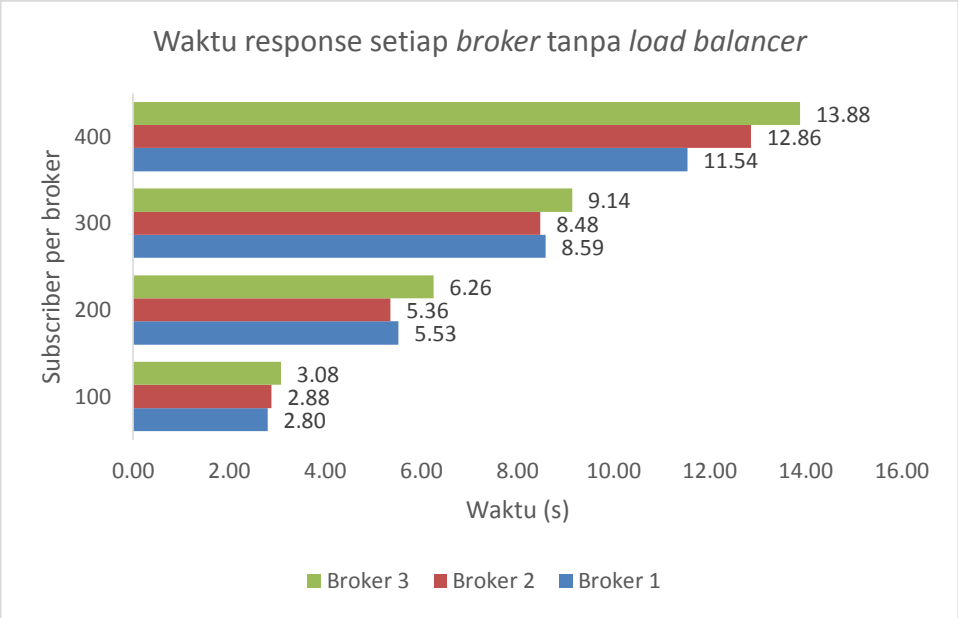
Gambar 6.5 Waktu yang dibutuhkan untuk menangani *subscriber*

Gambar 6.6 menjelaskan alur koneksi ketika melakukan proses *subscribe* dan *publish*.



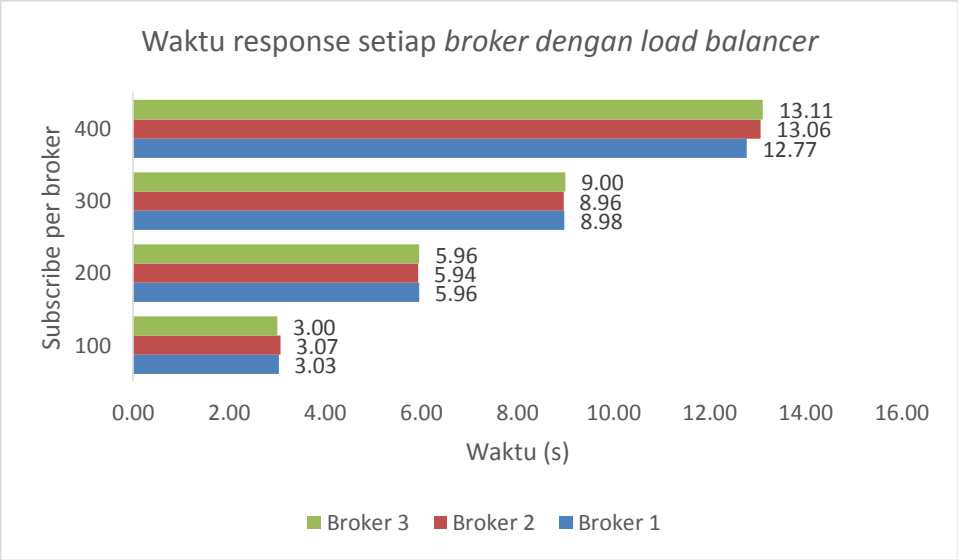
Gambar 6.6 Komunikasi antara *subscriber, load balancer, dan broker*

Gambar di atas menunjukkan, ketika *client* melakukan koneksi ke *load balancer, client* menggunakan *source port* secara acak yang menuju ke *port 1883*, dimana pada konfigurasi *HAProxy,* port tersebut berfungsi melakukan *listen* koneksi TCP yang menuju port 1883. Kemudian *load balancer* melakukan modifikasi dengan mengubah *source port* dan mengganti *destination IP* sesuai dengan alamat *broker* yang sudah terdaftar di *load balancer*. *Load balancer* mengidentifikasi setiap *client* yang terkoneksi dengan *load balancer* menggunakan nomor *port* sehingga jika ada paket untuk port tersebut, maka secara otomatis *load balancer* mengetahui kemana paket tersebut akan dilanjutkan. Kemudian untuk paket *subscribe* dan *suback* juga mengalami mekanisme yang sama.



Gambar 6.7 Waktu yang dibutuhkan *broker* untuk menangani *subscriber*

Ketika menggunakan *load balancer*. Distribusi penanganan *request* pada setiap *broker* lebih lama dibandingkan ketika tidak menggunakan *load balancer.* Tetapi waktu setiap *broker* untuk menyelesaikan setiap *request* tidak seimbang seperti pada gambar 6.7. Tapi, saat menggunakan *load balancer*, waktu penanganan setiap *broker* terlihat lebih merata dibandingkan jika melakukan koneksi secara langsung ke *broker* seperti yang ditunjukkan pada gambar 6.8.



Gambar 6.8 Waktu yang dibutuhkan *broker* untuk menangani *subscriber* ketika menggunakan *load balancer*

6.4 Pengujian CPU *usage broker* saat menangani *request subscribe*

6.4.1 Tujuan Pengujian

Mendapatkan nilai *CPU Usage broker* saat menangani pesan *subscribe* yang ada ketika *subscriber*. Dilakukan perbandingan ketika *subscriber* melakukan *request* melalui *load balancer* dan ketika tidak melakukan *load balancer* dengan pembagian jumlah *request* yang merata.

6.4.2 Langkah – langkah

1. *Broker* 1, *Broker 2,* dan *Broker 3* dalam kondisi aktif.
2. *Subscriber* dibuat menggunakan program *Python* yang memanfaatkan *multi thread*. Dimana jumlah *thread* sebanyak 300, 600, 900, dan 1200 *thread* dimana setiap *thread* merepresentasikan *subscriber*.
3. Setiap *broker* menerima koneksi dari *subscriber* sebanyak 100, 200, dan 300. Contoh : *Broker 1* = 100 *request*, *Broker 2 = 100 request* dan *Broker 3* = 100 *request*.
4. Sehingga pada 1 waktu akan terjadi 300 , 600, 900, dan 1200 koneksi.
5. Untuk pengambilan data menggunakan *tools pidstat* pada setiap *broker* untuk mencatat utilisasi *CPU.*
6. *Interval* pencatatan adalah setiap 2 detik dengan 10 kali pencatatan. Sehingga proses pencatatan utilisasi CPU dilakukan selama 20 detik, kemudian semua hasil utilisasi CPU yang dicatat dibagi dengan jumlah pencatatan.
7. Percobaan dilakukan sebanyak 5 kali untuk setiap jumlah *subscriber*.
8. Pengujian diulang dengan skenario *subscriber* melakukan koneksi melalui *load balancer*.

6.4.3 Data yang didapatkan

1. **Data dimana sistem tidak menggunakan *Load Balancer***

Tabel 6.30 menunjukkan *CPU Usage* semua *broker* ketika setiap *broker* melakukan menangani 100 *subscriber*.

Tabel 6.30 *Resource* CPU *Broker* dengan 100 *request subscribe* untuk setiap *broker*

100	CPU Load(%)					Rata - Rata
	I	II	III	IV	V	
Broker 1	12.04	12.43	12.19	12.56	11.64	12.17
Broker 2	11.73	11.78	12.42	11.71	12.25	11.98
Broker 3	14.83	15.23	14.90	13.35	14.26	14.51
Rata - Rata	12.87	13.15	13.17	12.54	12.72	12.89

Tabel 6.31 menunjukkan *resource CPU* semua *broker* ketika setiap *broker* menangani 200 *subscriber*.

Tabel 6.31 *Resource* CPU *Broker* dengan 200 *request subscribe* untuk setiap *broker*

200	CPU Load(%)					Rata - Rata
	I	II	III	IV	V	
Broker 1	23.68	26.45	24.05	23.90	23.23	24.26
Broker 2	23.88	25.38	26.45	26.88	24.82	25.48
Broker 3	30.63	28.80	29.33	29.41	28.96	29.43
Rata - Rata	26.06	26.88	26.61	26.73	25.67	26.39

Tabel 6.32 menunjukkan *resource CPU* semua *broker* ketika setiap *broker* menangani 300 *subscriber*.

Tabel 6.32 *Resource* CPU *Broker* dengan 300 *request subscribe* untuk setiap *broker*

300	CPU Load(%)					Rata - Rata
	I	II	III	IV	V	
Broker 1	41.33	42.20	41.08	43.52	43.59	42.34
Broker 2	39.95	38.92	36.70	36.93	39.89	38.48
Broker 3	43.24	44.41	41.24	45.08	43..16	43.49
Rata - Rata	41.51	41.84	39.67	41.84	41.74	41.32

Tabel 6.33 menunjukkan *resource CPU* semua *broker* ketika setiap *broker* menangani 400 *subscriber*.

Tabel 6.33 *Resource* CPU *Broker* dengan 400 *request subscribe* untuk setiap *broker*

400	CPU Load(%)					Rata - Rata
	I	II	III	IV	V	
Broker 1	56.06	46.17	51.94	56.38	51.42	52.39
Broker 2	54.34	51.68	50.21	50.42	50.98	51.53
Broker 3	61.36	60.38	60.30	64.15	60.40	61.32
Rata - Rata	57.25	52.74	54.15	56.98	54.27	55.08

2. **Data dimana sistem tidak menggunakan *Load Balancer***

Tabel 6.34 menunjukkan *resource CPU* semua *broker* ketika setiap *broker* menangani 300 *subscriber*.

Tabel 6.34 *Resource Broker* terhadap 300 *request subscribe* melalui *load balancer*

300	CPU Load(%)					Rata - Rata
	I	II	III	IV	V	

Broker 1	12.21	13.89	13.76	13.52	11.98	13.07
Broker 2	13.04	12.58	13.58	12.82	13.62	13.13
Broker 3	14.04	13.79	13.75	13.79	13.61	13.80
Rata - Rata	13.10	13.42	13.70	13.38	13.07	13.33
LB	3.02	2.96	3.62	3.24	2.95	3.16

Tabel 6.35 menunjukkan *resource CPU* semua *broker* ketika setiap *broker* menangani 600 *subscriber*.

Tabel 6.35 *Resource Broker* terhadap 600 *request subscribe* melalui *load balancer*

600	CPU Load(%)					Rata - Rata
	I	II	III	IV	V	
Broker 1	28.68	28.10	27.55	27.36	28.90	28.12
Broker 2	28.74	27.59	28.01	26.94	26.97	27.65
Broker 3	28.75	27.00	27.57	26.44	30.13	27.98
Rata - Rata	28.72	27.56	27.71	26.91	28.67	27.92
LB	6.58	6.01	5.93	6.52	6.76	6.36

Tabel 6.36 menunjukkan *resource CPU* semua *broker* ketika setiap *broker* menangani 900 *subscriber*.

Tabel 6.36 *Resource Broker* terhadap 900 *request subscribe* melalui *load balancer*

900	CPU Load(%)					Rata - Rata
	I	II	III	IV	V	
Broker 1	48.46	42.72	43.42	40.97	46.47	44.41
Broker 2	41.27	42.55	46.37	43.15	47.73	44.21
Broker 3	47.88	45.52	41.26	43.69	47.43	45.16
Rata - Rata	45.87	43.60	43.68	42.60	47.21	44.59
LB	9.32	9.69	9.50	9.36	9.28	9.43

Tabel 6.37 menunjukkan *resource CPU* semua *broker* ketika setiap *broker* melakukan koneksi dengan 1200 *subscriber*.

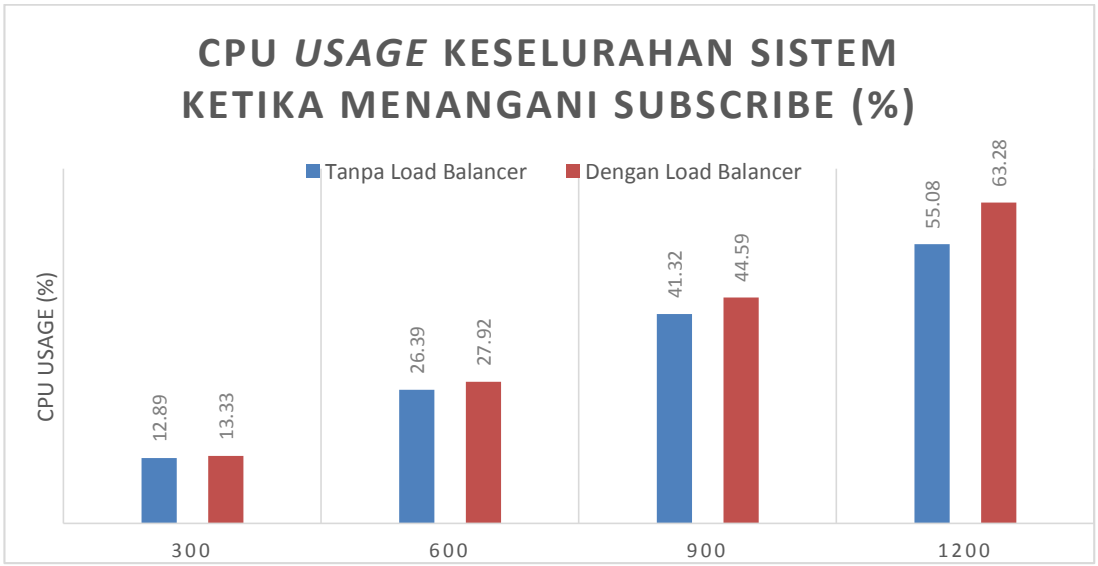
Tabel 6.37 *Resource Broker* terhadap 1200 *request subscribe* melalui *load balancer*

1200	CPU Load(%)					Rata - Rata
	I	II	III	IV	V	
Broker 1	64.88	58.95	64.32	69.27	63.65	64.21
Broker 2	65.75	62.94	63.89	65.44	63.37	64.28
Broker 3	59.35	62.74	58.56	60.68	65.38	61.34
Rata - Rata	63.33	61.54	62.26	65.13	64.13	63.28

LB	12.08	12.17	12.64	13.55	12.07	12.50

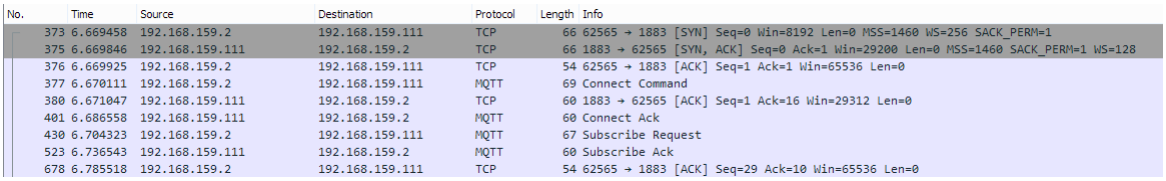
6.4.4 Analisis

Berdasarkan data – data yang berhasil diperoleh dari tabel – tabel pada bagian pengumpulan data, dibuatlah diagram batang yang bisa dilihat pada gambar 6.9. Dari gambar 6.9 bisa diketahui *resource* ketika menggunakan *load balancer* dan tanpa menggunakan *load balancer* dengan pengujian dimana setiap *broker* diberikan 100 *subscriber*, dimana jika ada 3 *broker* maka total *client*nya adalah *300*. Kemudian total *client* digunakan kepada sistem yang menggunakan *load balancer* sehingga pengujian menjadi adil dan bisa diketahui perbedaan apa yang akan dialami. Gambar 6.8 menunjukkan bahwa ketika menggunakan *load balancer*, *resource* CPU yang digunakan menjadi lebih besar. Hal ini bisa terjadi karena *broker* harus memberikan paket yang lebih besar dibandingkan dengan ketika *subscriber* melakukan koneksi langsung dengan *broker*. Karena *broker* harus memproses paket dengan data yang lebih besar serta mengirimkan paket dengan data yang lebih besar juga seperti yang akan ditunjukan pada gambar 6.10 dan 6.11.



Gambar 6.9 CPU *Usage broker* untuk menangani *subscriber* ketika menggunakan *load balancer*

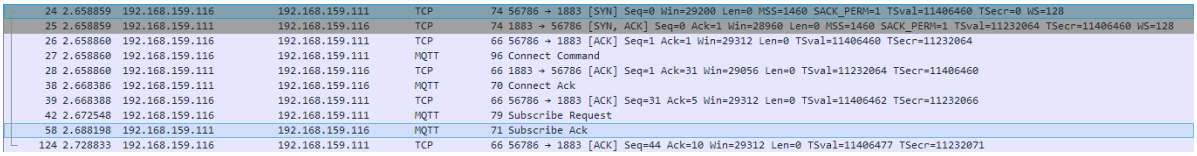
Dari gambar 6.10, bisa dilihat jika ketika *subscriber* melakukan proses untuk membuat koneksi dan melakukan *subscribe*. Maka besar paket di atas hanya sebesar 556 Bytes.



No.	Time	Source	Destination	Protocol	Length	Info
373	6.669458	192.168.159.2	192.168.159.111	TCP	66	62565 → 1883 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
375	6.669846	192.168.159.111	192.168.159.2	TCP	66	1883 → 62565 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
376	6.669925	192.168.159.2	192.168.159.111	TCP	54	62565 → 1883 [ACK] Seq=1 Ack=1 Win=65536 Len=0
377	6.670111	192.168.159.2	192.168.159.111	MQTT	69	Connect Command
380	6.671047	192.168.159.111	192.168.159.2	TCP	60	1883 → 62565 [ACK] Seq=1 Ack=16 Win=29312 Len=0
401	6.686558	192.168.159.111	192.168.159.2	MQTT	60	Connect Ack
430	6.704323	192.168.159.2	192.168.159.111	MQTT	67	Subscribe Request
523	6.736543	192.168.159.111	192.168.159.2	MQTT	60	Subscribe Ack
678	6.785518	192.168.159.2	192.168.159.111	TCP	54	62565 → 1883 [ACK] Seq=29 Ack=10 Win=65536 Len=0

Gambar 6.10 Hasil Capture Wireshark (1)

Sedangkan pada gambar 6.11, ketika menggunakan *load balancer*, maka semua paket yang diterima oleh *broker* telah disisipkan data tambahan, sehingga untuk transaksi *subscriber* yang menggunakan *load balancer*, maka besar paket totalnya adalah 728 *Bytes* untuk setiap *client*. Perbedaan 172 Bytes untuk setiap *client subscriber* ini tentu menambahkan beban kerja *broker* sehingga *broker* membutuhkan *resource* yang lebih besar.



24	2.658859	192.168.159.116	192.168.159.111	TCP	74	56786 → 1883 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=11406460 TSecr=0 WS=128
25	2.658859	192.168.159.111	192.168.159.116	TCP	74	1883 → 56786 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=11232064 TSecr=11406460 WS=128
26	2.658860	192.168.159.116	192.168.159.111	TCP	66	56786 → 1883 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=11406460 TSecr=11232064
27	2.658860	192.168.159.116	192.168.159.111	MQTT	96	Connect Command
28	2.658860	192.168.159.111	192.168.159.116	TCP	66	1883 → 56786 [ACK] Seq=1 Ack=31 Win=29056 Len=0 TSval=11232064 TSecr=11406460
38	2.668386	192.168.159.111	192.168.159.116	MQTT	70	Connect Ack
39	2.668388	192.168.159.116	192.168.159.111	TCP	66	56786 → 1883 [ACK] Seq=31 Ack=5 Win=29312 Len=0 TSval=11406462 TSecr=11232066
42	2.672548	192.168.159.116	192.168.159.111	MQTT	79	Subscribe Request
58	2.688198	192.168.159.111	192.168.159.116	MQTT	71	Subscribe Ack
124	2.728833	192.168.159.116	192.168.159.111	TCP	66	56786 → 1883 [ACK] Seq=44 Ack=10 Win=29312 Len=0 TSval=11406477 TSecr=11232071

Gambar 6.11 Hasil Capture Wireshark (2)

6.5 Pengujian waktu ketika mengirimkan pesan *publish* ke *subscriber*

6.5.1 Tujuan Pengujian

Mendapatkan nilai waktu durasi *publish* ketika *broker* mengirimkan pesan yang sesuai dengan topik yang diikuti oleh *subscriber* ketika sistem menggunakan *load balancer* dan tidak menggunakan *load balancer*.

6.5.2 Langkah – langkah

1. *Broker* 1 dalam kondisi aktif
2. *Publisher* dibuat menggunakan program *Python* yang memanfaatkan *multi thread*. Dimana jumlah *thread* sebanyak 100, 200, 300, dan 400 *thread* dimana setiap *thread* merepresentasikan *publisher* yang mengirimkan 1 pesan.
3. *Subscriber* dibuat menggunakan program *Python.* Dimana jumlah *subscriber* hanya sebanyak 1 dan terkoneksi dengan *broker 1*.
4. *Thread* mengirimkan pesan ke *broker 1*.
5. Untuk pengambilan data digunakan *Wireshark*.
6. Pengujian dilakukan sebanyak 5 kali untuk setiap jumlah *thread* yang ada.
7. Ulangi pengujian dengan mengimplementasikan *load balancer*.

6.5.3 Data yang didapatkan

Tabel 6.38 menunjukkan perbandingan waktu yang dibutuhkan agar pesan dari *publisher* bisa sampai ke *subscriber* ketika sistem menggunakan *load balancer*.

Tabel 6.38 Perbandingan Delay waktu *Publish* ketika menggunakan *load balancer*

Banyak Pesan	Delay Publish (s)					Rata - Rata
	I	II	III	IV	V	
100	1.39	1.54	1.64	1.52	1.48	1.51
200	3.00	2.85	2.70	2.70	2.86	2.82
300	4.14	4.87	4.29	4.46	4.13	4.38
400	6.25	5.95	6.12	5.93	6.01	6.05

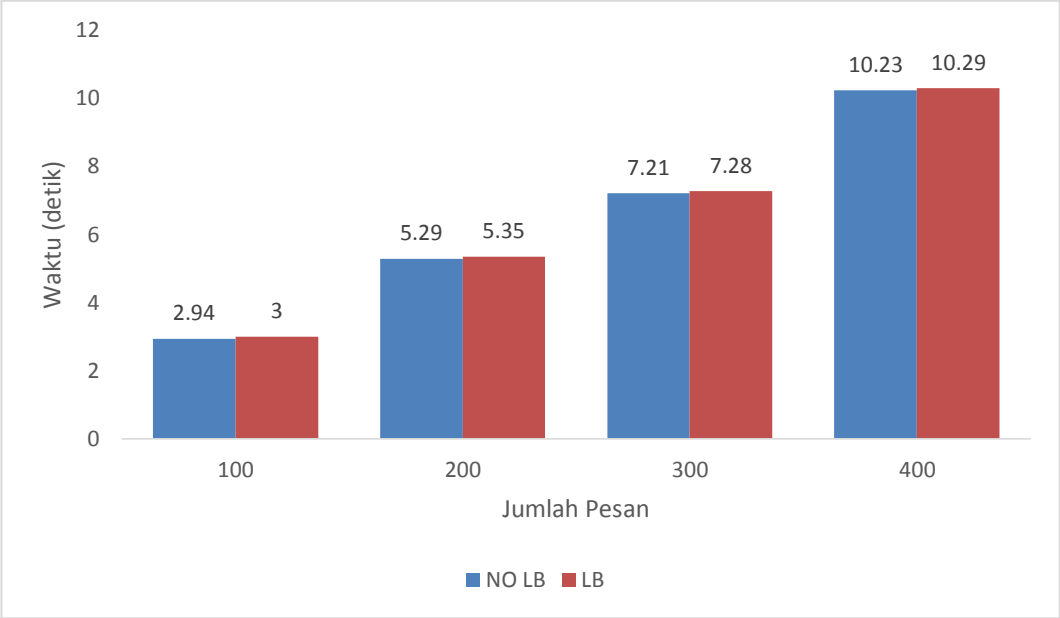
Tabel 6.38 menunjukkan perbandingan waktu yang dibutuhkan agar pesan dari *publisher* bisa sampai ke *subscriber* ketika sistem tidak menggunakan *load balancer*.

Tabel 6.39 Perbandingan Delay waktu *Publish* ketika menggunakan *load balancer*

Banyak Pesan	Delay Publish (s)					Rata - Rata
	I	II	III	IV	V	
100	1.55	1.46	1.47	1.37	1.25	1.42
200	2.89	2.79	2.65	2.70	2.72	2.75
300	3.79	4.50	4.09	3.91	4.75	4.21
400	6.79	5.15	5.44	6.71	5.14	5.84

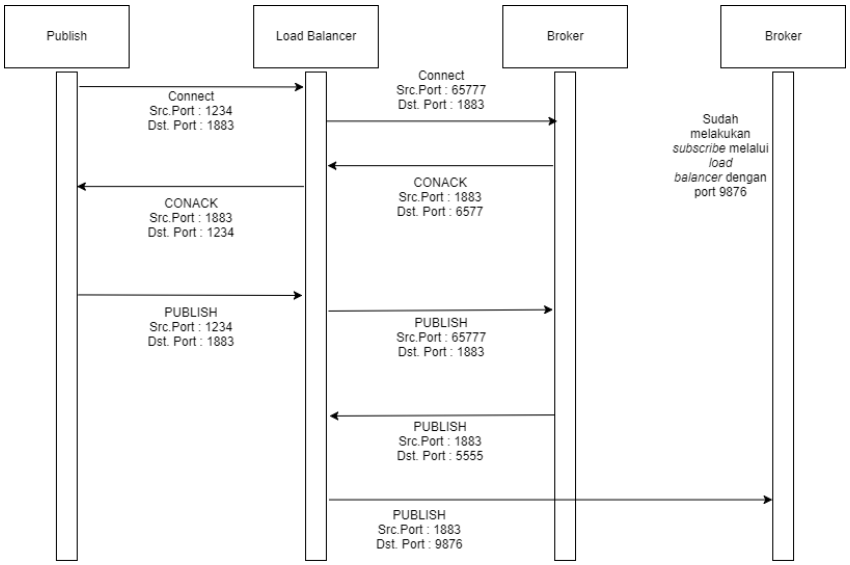
6.5.4 Analisis

Gambar 6.12 menunjukkan grafik lama waktu yang dibutuhkan oleh sistem ketika mengirimkan 100, 200, 300, dan 400 pesan ke 1 *subscriber*. Terlihat selisih ketika *broker* mengirimkan pesan *publish* secara langsung maupun melalui *load balancer* tidak signifikan. Selisih keduanya cenderung berbeda dalam satuan milidetik. Tidak seperti pada pengujian analisis 6.3 dimana selisih waktunya tidak berbeda jauh. Hal in dikarenakan ketika *broker* mengirimkan pesan *publish, load balancer* hanya mengirimkan 1 pesan yang sama, dan tidak ada koneksi seperti *connect – connack – subscrice request – suback*. Sehingga prosesnya lebih cepat.



Gambar 6.12 Selisih Waktu *publish* ketika menggunakan *load balancer*

Pada gambar 6.13, digambarkan bahwa ketika terjadi proses *publish*. Maka *client client* yang berperan sebagai *publisher* hanya cukup melakukan *connect – connect - publish*. Berbeda ketika proses *subscribe*. Dimana terjadi *connect – connack – subscribe - suback.* Pada proses *publish, publisher* membuat koneksi terlebih dahulu dengan *broker* melalui *load balancer*. Ketika sudah terjadi koneksi, maka *publisher* mengirim pesan *publish* melalui *load balancer*. Ketika *broker* memiliki *subscriber* yang berlangganan dengan topik yang sesuai, maka *broker* akan mengirimkan pesan tersebut melalui *load balancer* kembali karena *broker* tidak mengetahui alamat *IP* dari *subscriber*.



Gambar 6.13 Komunikasi antara *publisher, load balancer, dan broker*

6.6 Pengujian untuk mengetahui CPU usage *broker* dalam menangani banyak *client* secara pararel dalam melakukan proses *publish-subscribe*

6.6.1 Tujuan Pengujian

Mendapatkan nilai *CPU Usage* yang dibutuhkan *broker* ketika melakukan proses *publish – subscribe*. Kemudian dari pengujian ini bisa dibandingkan *resource CPU* ketika jumlah *broker* bertambah dan beban kerja dibagi berdasarkan jumlah *broker* aktif.

6.6.2 Langkah – langkah

1. *Subscriber* dibuat menggunakan program *Python* yang memanfaatkan *multi thread*. Dimana jumlah *thread* sebanyak 100, 200, 300, dan 400 *thread* dimana setiap *thread* merepresentasikan *subscriber*.
2. *Publisher* dibuat menggunakan program *Python* yang memanfaatkan *multi thread*. Dimana jumlah *thread* sebanyak 10 *thread* dimana setiap *thread* merepresentasikan *publisher*. Interval pengiriman pesan *publish* adalah sebesar 1 detik per pesan.
3. Kondisi awal sistem hanya menggunakan 1 *broker*.
4. Untuk pengambilan data menggunakan *tools pidstat* pada setiap *broker* untuk mencatat utilisasi *CPU.*
5. *Interval* pencatatan adalah setiap 2 detik dengan 10 kali pencatatan. Sehingga proses pencatatan utilisasi CPU dilakukan selama 20 detik, kemudian semua hasil utilisasi CPU yang dicatat dibagi dengan jumlah pencatatan.
6. Percobaan dilakukan sebanyak 5 kali untuk setiap jumlah *subscriber* dan jumlah *broker* aktif

6.6.3 Data yang didapatkan

1. **Data dimana hanya *Broker 1* yang aktif**

Tabel 6.40 menunjukkan *resource CPU broker* 1 ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani adalah 100.

Tabel 6.40 Penggunaan *resource* CPU dengan hanya 1 *Broker* dengan 100 *subscriber*

100	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	11.00	6.00	8.50
Percobaan 2	10.92	6.60	8.76
Percobaan 3	10.20	7.06	8.63
Percobaan 4	10.49	6.66	8.57
Percobaan 5	9.80	6.33	8.06

Rata - Rata	10.48	6.53	8.51

Tabel 6.41 menunjukkan *resource CPU broker* 1 ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani adalah 375.

Tabel 6.41 Penggunaan *resource* CPU dengan hanya 1 *Broker* dengan 200 *subscriber*

200	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	22.01	12.00	17.00
Percobaan 2	21.83	13.20	17.52
Percobaan 3	20.41	14.11	17.26
Percobaan 4	20.98	13.31	17.14
Percobaan 5	19.60	12.66	16.13
Rata - Rata	20.96	13.06	17.01

Tabel 6.42 menunjukkan *resource CPU broker* 1 ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani adalah 500.

Tabel 6.42 Penggunaan *resource* CPU dengan hanya 1 *Broker* dengan 300 *subscriber*

300	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	38.54	21.23	29.89
Percobaan 2	35.33	19.89	27.61
Percobaan 3	34.51	20.82	27.67
Percobaan 4	36.48	20.16	28.32
Percobaan 5	36.60	19.71	28.16
Rata - Rata	36.29	20.36	28.33

Tabel 6.43 menunjukkan *resource CPU broker* 1 ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani adalah 400.

Tabel 6.43 Penggunaan *resource* CPU dengan hanya 1 *Broker* dengan 400 *subscriber*

400	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	51.39	28.31	39.85
Percobaan 2	47.10	26.52	36.81
Percobaan 3	46.02	27.77	36.89
Percobaan 4	48.64	26.88	37.76
Percobaan 5	48.80	26.28	37.54
Rata - Rata	48.39	27.15	37.77

2. **Data dimana terdapat 2 *broker* yang aktif, yaitu *broker 1* & *broker 2***

Tabel 6.44 menunjukkan *resource CPU broker* 1, *broker 2,* dan *load balancer* ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani secara keselurahan adalah 100.

Tabel 6.44 Penggunaan *resource* CPU dengan hanya 2 *Broker* dengan 100 *subscriber*

100	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	6.19	4.79	5.49
Percobaan 2	6.04	4.89	5.47
Percobaan 3	6.39	4.30	5.35
Percobaan 4	6.30	4.27	5.28
Percobaan 5	6.21	4.01	5.11
Rata - Rata	6.23	4.45	5.34
100	CPU Load(%) (Broker 2)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	6.05	4.35	5.20
Percobaan 2	6.28	4.47	5.38
Percobaan 3	6.17	4.07	5.12
Percobaan 4	6.14	4.27	5.20
Percobaan 5	6.01	4.15	5.08
Rata - Rata	6.13	4.26	5.20
100	CPU Load(%) (LB)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	1.10	0.99	1.05
Percobaan 2	1.04	1.02	1.03
Percobaan 3	1.13	1.24	1.18
Percobaan 4	1.23	1.15	1.19
Percobaan 5	1.15	1.09	1.12
Rata - Rata	1.13	1.10	1.11

Tabel 6.45 menunjukkan *resource CPU broker* 1, *broker 2,* dan *load balancer* ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani secara keselurahan adalah 200.

Tabel 6.45 Penggunaan *resource* CPU dengan hanya 2 *Broker* dengan 200 *subscriber*

200	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	12.38	9.58	10.98
Percobaan 2	12.09	9.78	10.93
Percobaan 3	12.78	8.61	10.70

Percobaan 4	12.59	8.54	10.57
Percobaan 5	12.42	8.02	10.22
Rata - Rata	12.45	8.91	10.68
200	CPU Load(%) (Broker 2)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	12.10	8.70	10.40
Percobaan 2	12.57	8.94	10.75
Percobaan 3	12.34	8.14	10.24
Percobaan 4	12.27	8.54	10.41
Percobaan 5	12.02	8.30	10.16
Rata - Rata	12.26	8.52	10.39
200	CPU Load(%) (LB)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	2.21	1.98	2.09
Percobaan 2	2.09	2.04	2.06
Percobaan 3	2.26	2.47	2.36
Percobaan 4	2.46	2.30	2.38
Percobaan 5	2.30	2.18	2.24
Rata - Rata	2.26	2.20	2.23

Tabel 6.46 menunjukkan *resource CPU broker* 1, *broker 2,* dan *load balancer* ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani secara keselurahan adalah 300.

Tabel 6.46 Penggunaan *resource* CPU dengan hanya 2 *Broker* dengan 300 *subscriber*

300	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	17.84	11.42	14.63
Percobaan 2	19.40	12.54	15.97
Percobaan 3	20.62	12.60	16.61
Percobaan 4	20.98	11.06	16.02
Percobaan 5	19.90	10.57	15.24
Rata - Rata	19.75	11.64	15.69
300	CPU Load(%) (Broker 2)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	16.65	9.66	13.15
Percobaan 2	18.18	10.25	14.21
Percobaan 3	19.37	11.08	15.22
Percobaan 4	19.24	11.08	15.16
Percobaan 5	18.76	11.50	15.13
Rata - Rata	18.44	10.71	14.58
300	CPU Load(%) (LB)		Rata - rata(%)

	Proses Subscribe	*Proses Publish*	
Percobaan 1	3.31	4.02	3.67
Percobaan 2	3.85	3.74	3.80
Percobaan 3	3.02	3.77	3.39
Percobaan 4	3.62	3.40	3.51
Percobaan 5	3.38	3.50	3.44
Rata - Rata	3.44	3.69	3.56

Tabel 6.47 menunjukkan *resource CPU broker* 1, *broker 2, broker 3* dan *load balancer* ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani secara keselurahan adalah 400.

Tabel 6.47 Penggunaan *resource* CPU dengan hanya 2 *Broker* dengan 400 *subscriber*

400	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	23.79	15.23	19.51
Percobaan 2	25.87	16.73	21.30
Percobaan 3	27.50	16.80	22.15
Percobaan 4	27.97	14.74	21.35
Percobaan 5	26.54	14.09	20.31
Rata - Rata	26.33	15.52	20.92
400	CPU Load(%) (Broker 2)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	22.20	12.87	17.54
Percobaan 2	24.23	13.66	18.95
Percobaan 3	25.82	14.77	20.30
Percobaan 4	25.65	14.77	20.21
Percobaan 5	25.01	15.33	20.17
Rata - Rata	24.58	14.28	19.43
400	CPU Load(%) (LB)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	4.42	5.37	4.89
Percobaan 2	5.13	4.99	5.06
Percobaan 3	4.02	5.02	4.52
Percobaan 4	4.82	4.53	4.68
Percobaan 5	4.51	4.66	4.59
Rata - Rata	4.58	4.92	4.75

Tabel 6.48 menunjukkan *resource CPU broker* 1, *broker 2, broker 3* dan *load balancer* ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani secara keselurahan adalah 100.

Tabel 6.48 Penggunaan *resource* CPU dengan hanya 3 *Broker* dengan 100 *subscriber*

100	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	4.28	3.34	3.81
Percobaan 2	4.28	3.28	3.78
Percobaan 3	4.57	3.58	4.08
Percobaan 4	3.74	3.22	3.48
Percobaan 5	3.44	2.84	3.14
Rata - Rata	4.06	3.25	3.66
100	CPU Load(%) (Broker 2)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	4.04	2.85	3.44
Percobaan 2	3.40	2.58	2.99
Percobaan 3	4.28	2.78	3.53
Percobaan 4	3.96	2.87	3.42
Percobaan 5	4.18	3.00	3.59
Rata - Rata	3.97	2.81	3.39
100	CPU Load(%) (Broker 3)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	4.76	3.30	4.03
Percobaan 2	4.56	2.84	3.70
Percobaan 3	4.64	2.80	3.72
Percobaan 4	4.56	2.70	3.63
Percobaan 5	4.10	3.05	3.57
Rata - Rata	4.52	2.94	3.73
100	CPU Load(%) (LB)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	1.14	1.06	1.10
Percobaan 2	1.02	1.09	1.05
Percobaan 3	1.23	1.13	1.18
Percobaan 4	1.03	0.84	0.94
Percobaan 5	1.03	0.92	0.98
Rata - Rata	1.09	1.01	1.05

Tabel 6.49 menunjukkan *resource CPU broker* 1, *broker 2, broker 3* dan *load balancer* ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani secara keselurahan adalah 200.

Tabel 6.49 Penggunaan *resource* CPU dengan hanya 3 *Broker* dengan 200 *subscriber*

200	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	8.56	6.67	7.62
Percobaan 2	8.56	6.56	7.56
Percobaan 3	9.14	7.17	8.16
Percobaan 4	7.49	6.44	6.96
Percobaan 5	6.88	5.67	6.28
Rata - Rata	8.13	6.50	7.31
200	CPU Load(%) (Broker 2)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	8.07	5.70	6.88
Percobaan 2	6.79	5.16	5.98
Percobaan 3	8.56	5.55	7.06
Percobaan 4	7.93	5.74	6.83
Percobaan 5	8.35	6.00	7.18
Rata - Rata	7.94	5.63	6.78
200	CPU Load(%) (Broker 3)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	9.52	6.59	8.06
Percobaan 2	9.11	5.69	7.40
Percobaan 3	9.27	5.61	7.44
Percobaan 4	9.13	5.41	7.27
Percobaan 5	8.19	6.10	7.15
Rata - Rata	9.04	5.88	7.46
200	CPU Load(%) (LB)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	2.27	2.12	2.20
Percobaan 2	2.03	2.18	2.10
Percobaan 3	2.46	2.26	2.36
Percobaan 4	2.06	1.68	1.87
Percobaan 5	2.06	1.85	1.96
Rata - Rata	2.18	2.02	2.10

Tabel 6.50 menunjukkan *resource CPU broker* 1, *broker 2, broker 3* dan *load balancer* ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani secara keselurahan adalah 300.

Tabel 6.50 Penggunaan *resource* CPU dengan hanya 3 *Broker* dengan 300 *subscriber*

300	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	

Percobaan 1	11.89	8.38	10.14
Percobaan 2	12.05	8.10	10.07
Percobaan 3	12.25	9.23	10.74
Percobaan 4	11.20	8.96	10.08
Percobaan 5	11.53	8.08	9.80
Rata - Rata	11.78	8.55	10.17
300	CPU Load(%) (Broker 2)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	13.70	7.76	10.73
Percobaan 2	12.70	7.53	10.11
Percobaan 3	13.17	7.63	10.40
Percobaan 4	12.30	8.60	10.45
Percobaan 5	12.51	6.71	9.61
Rata - Rata	12.88	7.65	10.26
300	CPU Load(%) (Broker 3)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	14.12	8.70	11.41
Percobaan 2	14.35	8.65	11.50
Percobaan 3	13.78	10.93	12.36
Percobaan 4	12.42	9.02	10.72
Percobaan 5	12.88	8.56	10.72
Rata - Rata	13.51	9.17	11.34
300	CPU Load(%) (LB)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	3.41	3.32	3.36
Percobaan 2	3.38	3.21	3.30
Percobaan 3	3.26	3.55	3.41
Percobaan 4	3.38	3.45	3.41
Percobaan 5	3.29	3.41	3.35
Rata - Rata	3.34	3.39	3.37

Tabel 6.51 menunjukkan *resource CPU broker* 1, *broker 2, broker 3* dan *load balancer* ketika menangani proses *subscribe* dan proses *publish*. Jumlah *subscriber* yang ditangani secara keselurahan adalah 400.

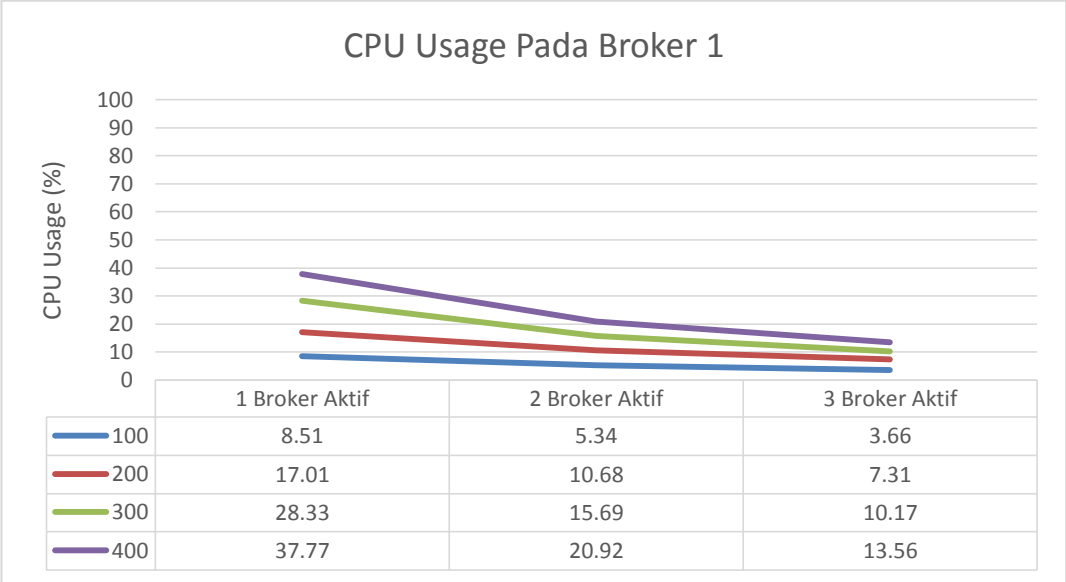
Tabel 6.51 Penggunaan *resource* CPU dengan hanya 3 *Broker* dengan 400 *subscriber*

400	CPU Load(%) (Broker 1)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	15.85	11.18	13.51
Percobaan 2	16.06	10.79	13.43
Percobaan 3	16.33	12.31	14.32
Percobaan 4	14.93	11.95	13.44

Percobaan 5	15.37	10.77	13.07
Rata - Rata	15.71	11.40	13.56
400	CPU Load(%) (Broker 2)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	18.27	10.35	14.31
Percobaan 2	16.93	10.04	13.48
Percobaan 3	17.56	10.18	13.87
Percobaan 4	16.39	11.47	13.93
Percobaan 5	16.68	8.95	12.82
Rata - Rata	17.17	10.20	13.68
400	CPU Load(%) (Broker 3)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	18.83	11.59	15.21
Percobaan 2	19.14	11.53	15.33
Percobaan 3	18.38	14.57	16.47
Percobaan 4	16.57	12.03	14.30
Percobaan 5	17.17	11.41	14.29
Rata - Rata	18.02	12.23	15.12
400	CPU Load(%) (LB)		Rata - rata(%)
	Proses Subscribe	*Proses Publish*	
Percobaan 1	4.54	4.43	4.49
Percobaan 2	4.51	4.28	4.39
Percobaan 3	4.35	4.74	4.54
Percobaan 4	4.50	4.60	4.55
Percobaan 5	4.38	4.54	4.46
Rata - Rata	4.46	4.52	4.49

6.6.4 Analisis

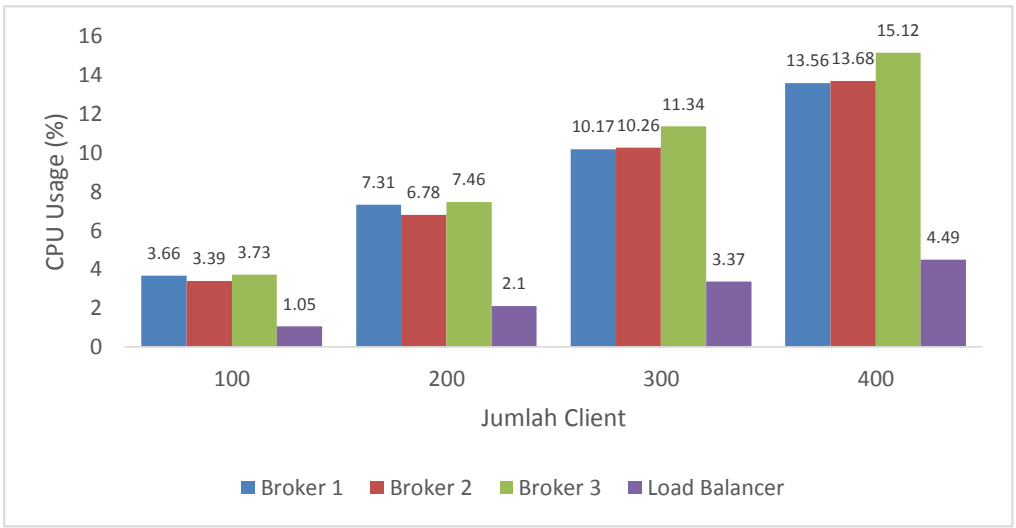
Dari data tabel yang di dapatkan, maka bisa dibentuk suatu grafik untuk mengetahui pengaruh *load* balancer terhadap CPU *Usage broker* 1. Pada pengujian ini ada 100, 200, 300, dan *400 client* yang bertindak sebagai *subscriber,* dimana pada percobaan awalnya hanya melakukan koneksi secara langsung ke *broker 1*. Utilisasi *resource CPU* ketika *broker 1* menangani 100 *client*, dimana terjadi penurunan utilisasi *resource CPU* ketika *broker* lainnya aktif, yang artinya *load balancer* sukses melakukan *load balancing*, dimana beban kerja didistribusikan ke *broker* lain sehingga secara logika, *broker 1* sekarang hanya menangani 50 *subscriber.* Untuk percobaan dengan 200, 300, dan 400 *client* juga berhasil memberikan penuruan utilisasi *resource* CPU dengan pola grafik yang hampir sama seperti yang ada pada gambar 6.14.



	1 Broker Aktif	2 Broker Aktif	3 Broker Aktif
100	8.51	5.34	3.66
200	17.01	10.68	7.31
300	28.33	15.69	10.17
400	37.77	20.92	13.56

Gambar 6.14 CPU *Usage broker 1*

Berdasarkan diagram batang pada gambar 6.15, ditunjukkan bahwa *resource CPU* yang digunakan pada setiap *broker* hampir merata dengan perbedaan penggunaan *resource* yang cukup tipis. Dimana perbedaan tersebut kurang dari 2%. Penurunan yang terjadi ketika menggunakan 2 *broker* tidak mencapai 50%. Hal ini dikarenakan pada pengujian 6.4 dan 6.3, diketahui bahwa ketika menggunakan *load balancer,* besar paket yang ada ditambahkan.



Gambar 6.15 Diagram penggunaan *resource CPU* pada setiap *devices*

6.7 Pengujian integritas data

6.7.1 Tujuan Pengujian

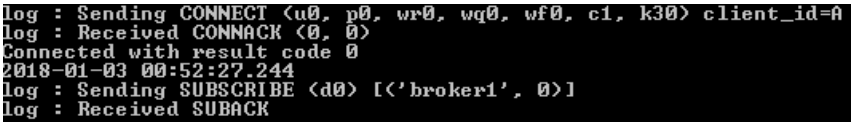
Melakukan validasi integritas data yang dikirimkan oleh *publisher* ke *subscriber.*

6.7.2 Langkah – langkah

1. *Subscriber* melakukan *subscription* ke *broker* melalui *load balancer.*
2. *Broker 1* menerima pesan *subscribe* dan membuat koneksi degnan *subscriber.*
3. *Publisher* mengirimkan pesan ke *broker 1*
4. *Broker 1* menerima pesan dan melakukkan pengecekkan, jika topik yang diterima sama, maka *broker* akan meneruskan pesan ke *subscriber* yang berlangganan topik tersebut.

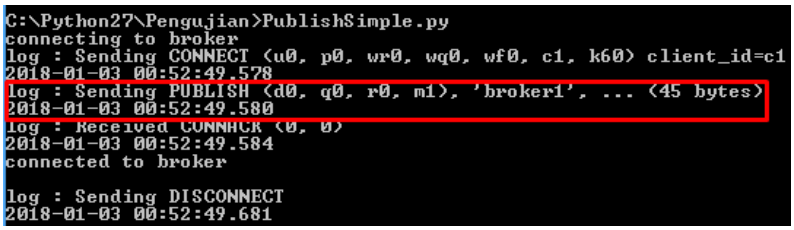
6.7.3 Data yang didapatkan

Subscriber dengan *client*_id A berlangganan topik *broker1,* gambar 6.16 menunjukkan proses *subscribe.*



Gambar 6.16 *Publish*er mengirimkan pesan ke *broker*

*Publish*er dengan *client*_id 1 mengirimkan pesan dengan topik *broker* 1 yang ditandai dengan kotak merah. Gambar 6.17 mennjukkan proses *publish.*

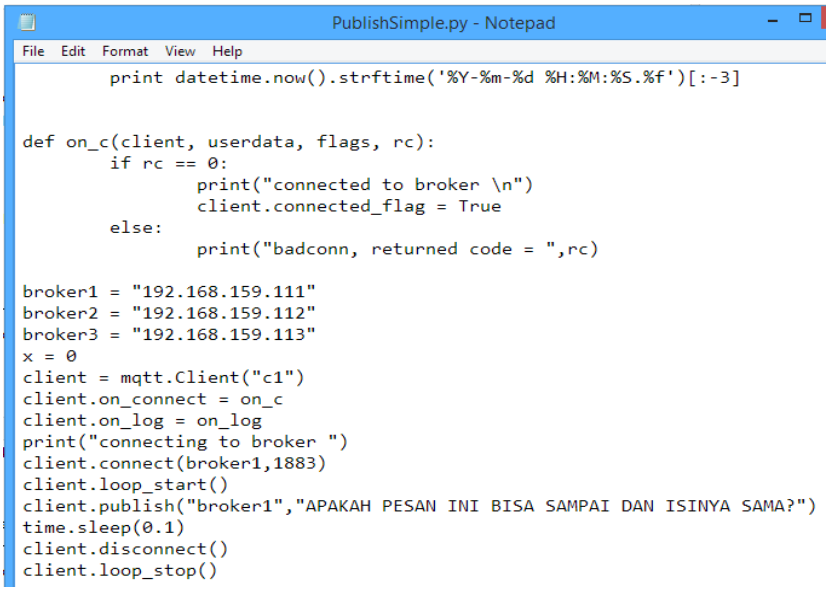


Gambar 6.17 *Publish*er mengirimkan pesan ke *broker*

Gambar 6.18 menunjukkan *subscriber* menerima pesan dari *publisher* melalui *broker* sesuai dengan topik yang dipilih.

Gambar 6.18 *Subscriber* menerima pesan

Gambar 6.19 merupakan isi baris kode dari *client publisher*. Dimana pada potongan source code yang ada digambar 6.19 diketahui bahwa pesan yang di*publish* adalah "APAKAH PESAN INI BISA SAMPAI DAN ISINYA SAMA?".



Gambar 6.19 Pesan yang dikirimkan oleh *publisher*

6.7.4 Analisis

Berdasarkan pengujian yang ada, *subscriber* berhasil menerima pesan dari *publisher*, dimana pesan yang diterima *publisher* sama dengan pesan yang dikirimkan oleh *broker*. Gambar 6.14 menunjukkan *subscriber A* berhasil berlangganan topik "*broker*1". Kemudian pada gambar 6.15, *publisher* c1 mengirimkan pesan ke *broker*. Terakhir, *subscriber A* berhasil menerima pesan seperti yang *publisher* kirimkan yang ditunjukkan pada gambar 6.16. *Subscriber A* berhasil mendapatkan pesan yang dikirimkan oleh *publisher c1* dan data yang diterima sesuai dengan data yang dikirimkan oleh *publisher c1* yang menandakan integritas data terjamin, dimana pesan yang diterima *subscribe A* adalah pesan yang dikirimkan oleh *publisher c1* dengan topik yang sesuai. Kemudian untuk waktu pengirimannya adalah 0.023 milidetik.

6.8 Pengujian rekoneksi ketika *broker* mengalami kegagalan

6.8.1 Tujuan

Menguji kemampuan sistem dalam melakukan rekoneksi ketika terjadi kegagalan pada salah satu *broker* yang sedang terkoneksi dengan *subscriber*.

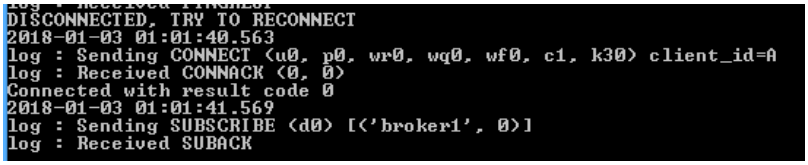
6.8.2 Langkah – langkah

1. *Subscriber* melakukan proses *subscription* melalui *load balancer.*
2. *Broker* menerima *subscriber* dan terjadi koneksi diantara *broker – load balancer – subscriber.*
3. *Service mosquitto* dimatikan pada *broker* yang sedang menangani *subscriber*.

4. Ulangi percobaan di atas sebanyak 5 kali akan tetapi pengujian dilakukan ketika *CPU Usage* di masing – masing *broker* dalam keadaan tinggi.

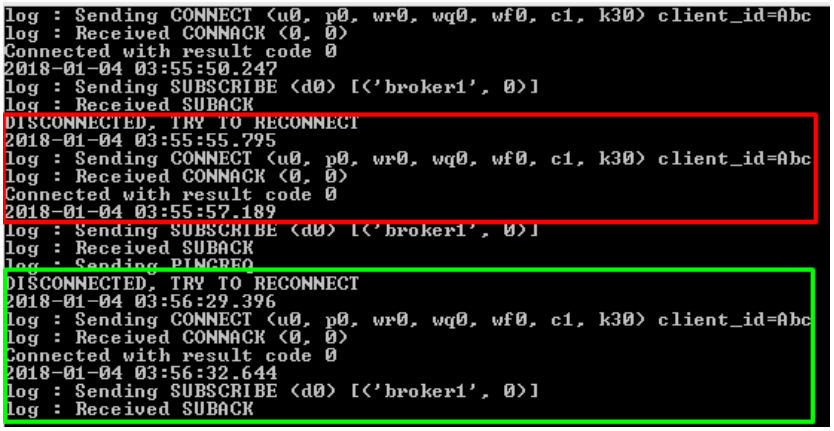
6.8.3 Data yang didapatkan

Gambar 6.20 merupakan *screenshoot* data yang menunjukkan proses *fail over* ketika *traffic* pada semua *broker* dalam keadaan idle atau tidak sibuk.



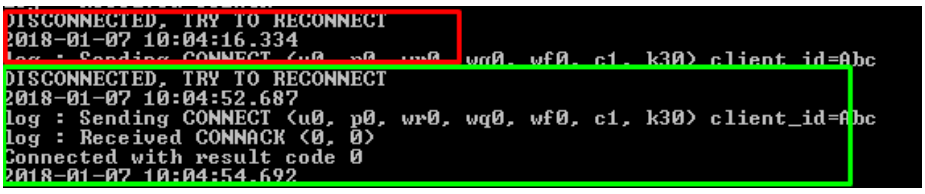
Gambar 6.20 *Subscriber* melakukan recovery ketika utilisasi CPU *broker rendah*

Gambar 6.20 merupakan screenshoot data yang menunjukkan proses *failover* ketika *CPU Usage* semua *broker* dalam keadaan tinggi.



Gambar 6.21 *Subscriber* melakukan recovery ketika utilisasi CPU *broker* tinggi

Kemudian jika kondisi *CPU Usage* benar – benar sangat tinggi, maka ada kemungkinan ketika *client* melakukan *reconnect,* pesan tersebut tidak dilayani seperti gambar 6.21.



Gambar 6.22 Rekoneksi gagal

Gambar 6.22 pada bagian yang ditandai dengan kotak merah, terlihat *client* terputus pada detik 16.334. *Client* mengirimkan pesan *connect*. Akan tetapi tidak ada respon dari *broker*. Kemudian pada bagian yang diberika kotak hijau, karena tidak ada respon, *client* kembali melakukan *reconnect* dan kemabli mencetak waktu untuk melakukan *reconnect* pada waktu 52.687, dan berhasil terkoneksi pada 54.692.

Percobaan di atas kemudian diulangi sebanyak 5 kali untuk di dapatkan data seperti pada tabel 6.52 yaitu waktu yang dibutuhkan *client subscriber* untuk melakukan *reconnect* ke *broker* lain ketika *broker* yang sedang menangani *subscriber* tersebut mengalami kegagalan dengan kondisi setiap *broker* sedang *idle*, dan tabel 6.53 ketika *broker* sedang sibuk

Tabel 6.52 Waktu yang dibutuhkan untuk melakukan reconnect saat *broker idle*

	I	II	III	IV	V
Reconnect	55.102	58.53	11.849	6.389	11.833
Disconnect	54.095	57.525	10.842	5.382	10.826
Selisih (detik)	1.007	1.005	1.007	1.007	1.007

Tabel 6.53 Waktu yang dibutuhkan untuk melakukan reconnect saat *broker* sibuk

	I	II	III	IV	V
Reconnect	51.418	40.234	11.765	0	54.692
Disconnect	50.413	37.443	7.171	16.334	52.687
Selisih (detik)	1.005	2.791	4.594	-16.334	2.005

6.8.4 Analisis

Dari tabel di 6.52 dan tabel 6.53, di dapatkan waktu bahwa ketika *broker* dalam keadaan *idle,* waktu yang dibutuhkan untuk *reconnect* sebesar 1.007 detik. Dimana kecepatan untuk melakukan rekoneksi sangatlah baik. Akan tetapi ketika CPU *Usage* di *broker* sedang tinggi, diperlukan waktu variatif bergantung pada *traffic* dan *CPU Usage* di setiap *broker*. Pada tabel 6.53 juga ditunjukkan pada percobaan ke 4, dimana *client subscriber* gagal melakukan rekoneksi karena *traffic* dan *CPU Usage* yang tinggi di *broker*. Hal ini dikarenakan kekurangan algoritma *round robin* dalam mendistribusikan beban, *client* yang melakukan rekoneksi didistribusikan ke *broker* yang kemungkinan memiliki beban yang tinggi, oleh karena itu bisa terjadi kegagalan ketika melakukan rekoneksi.