Jurnal Nasional Teknologi dan Aplikasinya
Volume 1, Nomor 1, November 2022

Analisis Performa Kecepatan Pemrosesan *Big Data* Dengan Menggunakan *Micro Batching* Dalam Tahap *Load* Pada *ETL* Untuk *Event-Data*

Putu Audi Pasuatmadi[a1], Ida Bagus Made Mahendra[a2]

[a]Informatics Department, Udayana University
Bali, Indonesia
[1]audipasuatmadi@gmail.com
[2]ibm.mahendra@unud.ac.id

Abstract

Speed performance in Big Data Processing is one of the key aspect of presenting a real-time data for a real-time analysis in various use cases. One of such data is an Event-Data or an Activity Data that is often used for making analysis and business decisions in real-time. This research focuses on analyzing the speed performance of Big Data Processing in the Load* stage *of Extract, Transform, Load using Micro-Batching. The speed performance is evaluated using Load-Testing with K6 and the configuration of 10 VUs and 10000 data. The Load-Testing results of the speed performance shows that implementing Micro-Batching results in 63.63% faster in performing all the request, 59.03% faster in HTTP Request Duration, and 62% faster in HTTP Request Waiting duration.

Keywords: *Big Data, ETL, Micro-Batching, Event-Data, Load-Testing, Stream-Processing*

1. Pendahuluan

Dewasa ini, data merupakan peranan yang sangat penting dalam membuat suatu keputusan. Salah satu dari data ini merupakan *Event-Data* atau data aktivitas baik aktivitas pengguna atau aktivitas lainnya yang berkaitan dengan waktu dan kejadian. Data aktivitas dapat meliputi pengguna menekan suatu tombol, ataupun aktivitas pemanggilan fungsi pada program. Data ini cenderung menghasilkan jumlah yang sangat besar dan dapat dimasukkan ke kategori *Big Data* atau Data Besar. *Big Data* merupakan sebuah data yang berjumlah banyak yang dapat menjadi tolak ukur inovasi dan juga strategi bisnis [1]. Memproses *Big Data* tidak dapat dilakukan secara tradisional [1], melainkan harus menggunakan konsep seperti *Extract, Transform, Load* atau ETL [2]. ETL merupakan sebuah teknik yang digunakan untuk mengintegrasikan data dari berbagai tempat ke dalam gudang data atau *Data Warehouse* [2] [3].

Data yang diproses dapat berupa sebuah data aktivitas, di mana data aktivitas ini salah satu kegunaanya adalah dapat digunakan di bidang sains dan teknologi [4]. Data aktivitas ini pula dapat digunakan di industri yaitu sebagai tolak ukur dan juga basis dari analisis proses bisnis atau *business process analysis* [5]. Data – data aktivitas pada umumnya merepresentasikan sebuah proses atau aktivitas dengan deskripsi dari aktivitas dan juga waktu aktivitas atau *timestamp* saat aktivitas tersebut terjadi [5]. Data aktivitas ini dapat berupa banyak hal, salah satunya adalah data *logging* yang mencatat kejadian *error* atau kegagalan pada internet yang. Data ini merupakan salah satu data yang penting karena dapat membantu *programmer* dalam melakukan *debugging* [6].

Dalam melakukan pemrosesan *Big Data*, latensi atau performa kecepatan menjadi salah satu tantangan yang besar terutama pada data waktu nyata atau *real-time data* [7]. Sistem yang tidak cepat dapat membuat pengguna yang terlibat dalam sistem tidak memiliki pengalaman terbaik dalam sistem. Untuk itu, optimisasi dibutuhkan untuk menghindari permasalahan ini terjadi.

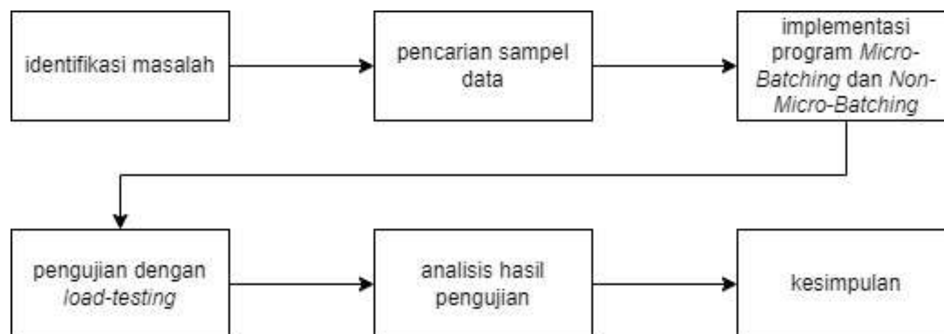
Salah satu optimisasi yang dapat dilakukan adalah dengan mengimplementasikan *Micro-Batching*. *Micro-Batching* merupakan metode dimana data dibagi dan dikelompokkan menjadi kumpulan data atau *batch* dan diproses secara bersamaan untuk satu kumpulan data [8]. *Micro-Batching* diyakini dapat meningkatkan performa sistem dalam *ETL Big Data* untuk *Event-Data* pada tahapan *Load* karena

membutuhkan inisialisasi atau persiapan dalam pemrosesan yang lebih sedikit, yaitu satu persiapan proses seperti inisialisasi koneksi untuk satu kumpulan data di mana satu kumpulan data ini akan memiliki banyak data. *Micro-Batching* pada penelitian lain berhasil menginatkan performa sebanyak 1.63x lebih cepat dalam menjalankan *Deep Learning* [8], dan juga telah berhasil menginkatkan performa *throughput* atau performa kecepatan jumlah data yang diproses sebanyak 200% lebih baik pada pemrosesan *real-time* untuk data *Tweet* yang merupakan *tweets* pengguna Twitter dan *SynD* yang merupakan data sintesis untuk pengujian [9]. Dengan mengetahui hasil dua penelitian tersebut, peneliti memiliki tujuan untuk melakukan penelitian ini demi mengetahui apakah implementasi *Micro-Batching* dapat meningkatkan performa kecepatan dalam *Event-Data* pada tahap *Load* dalam *ETL*.

2. Metodologi Penelitian

2.1. Alur Penelitian

Penelitian ini mengikuti sebuah alur dan juga tahapan untuk memastikan kelancaran dari penelitian. Kerangka penelitian tersebut dapat dilihat pada **Gambar 1** di bawah.



Gambar 1. Alur Penelitian

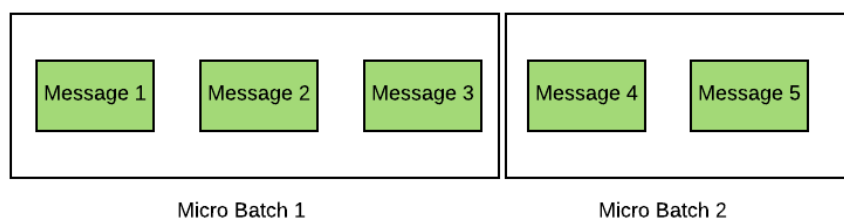
Alur kerangka penelitian di atas memiliki penjelasan sebagai berikut:

a. Identifikasi Masalah: tahapan awal dari penelitian, di mana peneliti mengidentifikasikan masalah yang ada di lapangan dan juga studi literatur penyokong dari masalah yang ditemukan. Permasalahan yang didapatkan dari penelitian ini adalah performa kecepatan dalam fase *Load* pada *ETL* untuk *Event-Data* atau data aktivitas. Pada tahap ini pula peneliti melakukan proses studi literatur sebagai sumber informasi untuk melakukan penelitian.
b. Pencarian Sampel Data: pada tahapan ini, data dicari dengan melakukan survey terhadap perusahaan teknologi. Dilakukan wawancara dan observasi untuk mencari tahu seperti apa format *logging* atau format dari *Event-Data* yang dapat digenerasikan ulang nantinya dalam penelitian. Hal ini dilakukan agar pengujian performa kecepatan tetap relevan terhadap bentuk dari data yang digunakan.
c. Implementasi Program *Micro-Batching* dan tanpa *Micro-Batch*: pada tahapan ini, peneliti melakukan implementasi dari program dengan dua metode. Hal ini bertujuan untuk dapat membandingkan performa kecepatan dari dua metode tersebut. Implementasi dilakukan dengan menggunakan bahasa pemrograman *Go* dan juga *Visual Studio Code*.
d. Pengujian dengan *Load-Testing*: pada tahapan ini, peneliti melakukan pengujian terhadap dua metode tersebut dengan metode *Load-Testing* menggunakan *K6*. Pengujian dilakukan dengan sampel data yang ditemukan dan digenerasikan ulang.
e. Analisis Hasil Pengujian: pada tahapan ini, peneliti melakukan analisis dari hasil yang didapatkan dari pengujian terhadap dua metode *Micro-Batching* dan tanpa *Micro-Batching*.
f. Kesimpulan: pada tahapan ini, peneliti menyimpulkan hasil dari performa kecepatan dari penggunaan *Micro-Batching* dan juga tanpa *Micro-Batching*.

2.2. Micro-Batching

Penelitian ini menggunakan metode *Micro-Batching* sebagai metode yang akan diuji performa kecepatannya. *Micro-Batching* merupakan sebuah konsep dimana data yang banyak atau datang secara kontinu dikelompokan menjadi beberapa kelompok untuk nantinya pemrosesan dilakukan per-kelompok data tersebut sebagai kesatuan [8]. *Micro-Batching* mengambil konsep dari *Batch Procecssing*, dimana *Batch Processing* merupakan pemrosesan sekumpulan data secara periodik yang biasanya dijalankan pada waktu atau kebutuhan tertentu [10]. Perbedaan dari *Micro-Batching* dan *Batch Processing* adalah, *Micro-Batching* menggunakan konsep *Batch Processing* pada halnya

memproses sekumpulan data secara bersamaan, namun *Micro-Batching* melakukannya dengan jumlah data yang jauh lebih sedikit dan juga sering digunakan pada *Stream-Processing* atau pemrosesan waktu nyata karena dapat dikatakan mengumpulkan data – data yang datang dan memprosesnya kumpulan data – data tersebut secara langsung di waktu tersebut. Konsep *Micro-Batching* kemudian diturunkan dari konsep *Windowing*. *Windowing* merupakan metode yang bekerja dengan cara membagi data – data yang datang secara kontinu dan menerus ke dalam segmen – segmen [11] [12]. Salah satu keunggulan dari menggunakan *Micro-Batching* berada di sisi jaringan. Dengan menggunakan *Micro-Batching*, kita dapat menghindari *Repeated TCP*. *Repeated TCP* adalah sebuah inisiasi komunikasi data pada jaringan yang di mana pada setiap komunikasi tersebut berjalan dengan singkat dan juga berulang [13]. Untuk mendukung konsep ini, terdapat sebuah riset yang menyebutkan bahwa meminimalisir penggunaan koneksi *TCP* berulang atau *Repeated TCP* dapat menghasilkan latensi 19 milidetik pada persentil ke-99 untuk 2000 data dengan mengurangi *Repeated TCP* dengan metode *Database Pooling* [14].



Gambar 2. Ilustrasi *Micro-Batching*. Sumber: packtpub.com

2.3. Extract, Transform, Load (ETL)

Penelitian ini beradap pada lingkup proses *Extract, Transform, Load* atau *ETL* khususnya pada tahapan *Load*. *ETL* merupakan sebuah metode untuk melakukan pemrosesan data besar atau *Big Data* yang melibatkan tiga tahap, yaitu pengambilan data dari sumber data, transformasi data atau pemrosesan data, dan juga penyimpanan hasil pemrosesan data pada gudang data [2]. Konsep *ETL* merupakan sebuah pilihan standar yang digunakan untuk mengolah datab esar atau aset – aset *Big Data* yang berharga dan variatif [7]. Di penelitian kali ini, berfokus pada tahapan *Load* dalam *ETL* yaitu proses penyimpanan data yang telah dilakukan proses transormasi ke dalam gudang data atau basis data lainnya.

2.4. Load-Testing

Pengujian dari metode yang diimplementasikan akan menggunakan *Load-Testing* dengan *K6*. *Load-Testing* merupakan sebuah metode pengujian untuk performa sebuah sistem dengan mengimitasikan pengguna dengan mengirim *request* sebanyak konfigurasi yang diberikan [15]. Salah satu alat yang dapat digunakan untuk melakukan *Load-Testing* adalah *K6*. *K6* merupakan sebuah alat *open-source* yang digunakan untuk *Load-Testing* sehingga dapat mengevaluasikan performa kecepatan sebuah sistem [16]. Menggunakan *Load-Testing* dengan *K6* menghasilkan beberapa metris yang dapat digunakan sebagai alat ukur. Beberapa metris tersebut dapat dilihat pada tabel di bawah.

Tabel 1. Metris Utama *K6 Load-Testing*

Key	Deskripsi
total duration	Total waktu yang dibutuhkan untuk mengirim seluruh data dan menerima respons sesuai konfigurasi.
http_req_duration	Waktu yang dibutuhkan untuk mengirim data dan menerima respon pada. Dihitung oleh library dengan cara menambahkan lama data dikirim ke tujuan, lama menunggu data diproses oleh tujuan, dan lama data kembali ke pengirim.
http_req_waiting	Waktu yang dibutuhkan untuk mendapatkan bit pertama respon data setelah mengirim *request.* Dihitung oleh library dengan menghitung lamanya data saat seluruh data tepat diterima oleh tujuan

	sampai dengan bit pertama data tersebut dikembalikan ke pengirim. Penghitungan ini dapat dikatakan merepresentasikan seberapa lama proses tujuan melakukan operasi pemrosesan data.

3. Hasil dan Pembahasan

Berdasarkan alur penelitian dan juga studi literatur yang telah dijelaskan, maka didapatkan hasil penelitian sebagai berikut.

3.1. Pencarian Sampel Data

Penelitian ini melakukan pencarian sampel data ke perusahaan teknologi. Sampel data dicari dengan cara melakukan survey dan juga observasi terhadap pelaku perusahaan. Didapatkan format *logging* atau format dari *Event-Data* yang digunakan oleh perusahaan. Format data tersebut dapat dilihat pada **Tabel 2** di bawah.

Tabel 2. Struktur *Event-Data*

Key	Tipe Data	Deskripsi
level	String	Kategori dari *Event-Data*
message	String	Pesan atau intisari dari *Event-Data*
timestamp	Date	Waktu kejadian
project_name	String	Nama proyek yang menghasilkan *Event-Data*

Kemudian, *Key* pada *Event-Data* pada bagian *level* tersebut yang merepresentasikan kategori dari Event-Data dapat dibagi lagi pada **Tabel 3** di bawah.

Tabel 3. Penjabaran Kategori *Event-Data*

Level	Deskripsi
info	Jika yang berkaitan dengan informasi seperti *logging* pada umumnya.
warn	Jika yang berkaitan dengan informasi seperti peringatan yang tidak fatal seperti *error*.
error	Jika berkaitan dengan *error* yang fatal.

Peneliti berhasil mengumpulkan 10 data sebagai sampel dari perusahaan teknologi di Denpasar tersebut. Kemudian data ini dijadikan oleh peneliti sebagai basis untuk menguji penelitian terhadap performa *Micro-Batching*. Hal ini dilakukan dengan melakukan generasi sebanyak 10.000 data berdasarkan 10 sampel data yang dimiliki, dilakukan seperti demikian karena hal yang diuji nantinya adalah kecepatan dari pemrosesan sistem yang memiliki dampak terhadap jumlah data dan besarnya data di lapangan.

3.2. Implementasi Program *Micro-Batching* dan Tanpa *Micro-Batching*

Proses implementasi dari *Micro-Batching* dan juga tanpa *Micro-Batching* dilakukan pada satu basis kode atau *codebase* dengan menggunakan bahasa pemrograman *Go*. Intisari dari implementasi program dapat dilihat pada penggalan kode di **Tabel 4** di bawah.

Tabel 4. Penggalan Intisari Kode

Baris	Kode
1	`package service`
2	
3	`import (`
4	`      "time"`
5	
6	

```
 7   	"github.com/audi-
     skripsi/snatia_audi_ingestor_be/internal/util/converterutil"
 8   	"github.com/audi-skripsi/snatia_audi_ingestor_be/pkg/dto"
 9   )
10   
11   func (s *service) StoreEvent(event dto.EventLog, microBatch bool)
     (err error) {
12   	if microBatch {
13   		err = s.processMicrobatch(event)
14   	} else {
15   		err = s.processNonMicrobatch(event)
16   	}
17   
18   	if err != nil {
19   		s.logger.Errorf("error processing data of %+v: %+v",
     event, err)
20   	}
21   
22   	return
23   }
24   
25   func (s *service) processNonMicrobatch(event dto.EventLog) (err
     error) {
26   	eventModel := converterutil.EventDtoToModel(event)
27   	err = s.repository.InsertEvent(eventModel, "non_microbatch")
28   	return
29   }
30   
31   func (s *service) processMicrobatch(event dto.EventLog) (err
     error) {
32   	eventModel := converterutil.EventDtoToModel(event)
33   	s.EventBatch.BatchEventData =
     append(s.EventBatch.BatchEventData, eventModel)
34   
35   	if len(s.EventBatch.BatchEventData) == 500 {
36   		s.EventBatch.Mu.Lock()
37   		err = s.repository.MicrobatchInsertEvent(s.EventBatch,
     "microbatch")
38   		s.EventBatch.BatchEventData = nil
39   		s.EventBatch.Mu.Unlock()
40   	}
41   
42   	return
43   }
44   
45   func (s *service) initBatchCron() {
46   	go func() {
47   		for {
48   			if len(s.EventBatch.BatchEventData) > 0 {
49   				s.EventBatch.Mu.Lock()
50   				err :=
     s.repository.MicrobatchInsertEvent(s.EventBatch, "microbatch")
51   				if err != nil {
52   					s.logger.Errorf("error batch
     insert: %+v", err)
53   				}
54   				s.EventBatch.BatchEventData = nil
55   				s.EventBatch.Mu.Unlock()
56   			}
57   			time.Sleep(2 * time.Second)
58   		}
```

59	`}()` `}`

Implementasi dari *Micro-Batch* dapat dilihat pada kode baris ke-30 sampai dengan baris ke-42. Dapat dilihat bahwa pada bagian tersebut terdapat sebuah penyimpanan data terlebih dahulu ke dalam sebuah *array* sampai data berjumlah sekiranya 500. Kemudian, setelah data mencapai angka 500, maka akan dilakukan penyimpanan seluruh data yang telah dilakukan *batching* terhadap basis data pada baris ke-36. Hal serupa dilakukan pula pada baris ke-44 sampai dengan baris ke-59 dimana hal tersebut dilakukan lagi setiap 2 detik untuk memastikan tidak ada data yang tertinggal. Dapat dilihat pula pada baris ke-35 dan juga pada baris ke-38 bahwa terdapat *sync.Mutex* untuk memastikan tidak terdapat *racing condition* atau data sama yang diproses oleh dua *thread* berbeda. Kemudian implementasi dari metode tanpa *Micro-Batch* dapat dilihat pada baris kode ke-24 sampai dengan baris kode ke-28. Pada bagian kode tersebut dapat dilihat bahwa data yang diterima langsung saja dikirim ke dalam basis data tanpa dilakukan *Batching*. Pemilihan kedua metode ini ditentukan pada baris ke-10 sampai dengan baris ke-22. Dilakukan pemisahan apakah menggunakan *Micro-Batch* atau tanpa *Micro-Batch* dari argumen yang dikirim.

3.3. Pengujian Menggunakan *Load-Testing*

Pengujian dari sistem dapat dilakukan *Load-Testing* menggunakan *K6*. Program untuk melakukan *Load-Testing* ditulis dengan bahasa pemrograman *JavaScript*. Penggalan kode dari *Load-Testing* menggunakan *K6* dapat dilihat pada penggalan kode di **Tabel 5** di bawah.

Tabel 5. Penggalan Intisari Kode *Load-Testing*

Baris	Kode
1	`import http from "k6/http";`
2	
3	`export const options = {`
4	`  vus: 10,`
5	`  iterations: 10000,`
6	`};`
7	
8	`const rawData = open("data-sample.json");`
9	`const testData = JSON.parse(rawData);`
10	
11	`export default function () {`
12	`  const mode = __ENV.MODE;`
13	`  let reqPath = "";`
14	`  if (mode == "microbatch") {`
15	`    reqPath = "/v1/microbatch";`
16	`  } else if (mode == "non-microbatch") {`
17	`    reqPath = "/v1/non-microbatch";`
18	`  } else {`
19	`    console.error("ENV should be microbatch or non-microbatch");`
20	`    return;`
21	`  }`
22	
23	`  const randInt = Math.floor(Math.random() * testData.length);`
24	
25	`  http.post(`
26	`    "http://localhost:8080" + reqPath,`
27	`    JSON.stringify(testData[randInt]),`
28	`    {`
29	`      headers: {`
30	`        "Content-Type": "application/json",`
31	`      },`
32	`    }`
33	`  );`
34	`}`

Implementasi dari *Load-Testing* dapat dilihat pada penggalan kode di atas. Pertama – tama dilakukan konfigurasi dari skenario pada baris ke-3 sampai dengan baris ke-6. Kemudian sampel data di ambil dan diubah menjadi obyek *seperti* pada baris ke-8 dan baris ke-9. Proses pemilihan apakah akan melakukan *Load-Testing* terhadap *Micro-Batching* atau tidak berada pada baris ke-12 sampai dengan baris ke-21. Kemudian pemilihan sampel data dan pengiriman data dapat dijumpai pada baris ke-23 sampai dengan baris ke-33. Kemudian proses *Load-Testing* dijalankan terhadap *Micro-Batch* dan tanpa *Micro-Batch* dengan 10000 data dan juga 10 *Virtual User* untuk mengetahui performa kecepatan dari kedua metode tersebut. Sehingga didapatkan hasil dari *Load-Testing* seperti pada **Gambar 3** dan juga **Gambar 4** di bawah.

```
putuaudipasuatmadi@pop-os:~/Workspaces/SNATIA-Personal/Dev/snatia_audi_ingestor_be$ k6 run -e MODE=microbatch loadtest.js

  execution: local
     script: loadtest.js
     output: -

  scenarios: (100.00%) 1 scenario, 10 max VUs, 10m30s max duration (incl. graceful stop):
           * default: 10000 iterations shared among 10 VUs (maxDuration: 10m0s, gracefulStop: 30s)


running (00m00.4s), 00/10 VUs, 10000 complete and 0 interrupted iterations
default ✓ [======================================] 10 VUs  00m00.4s/10m0s  10000/10000 shared iters

     data_received..................: 1.5 MB 3.7 MB/s
     data_sent......................: 2.4 MB 5.8 MB/s
     http_req_blocked...............: avg=3.22µs   min=601ns   med=1.76µs   max=1.7ms    p(90)=2.98µs   p(95)=4µs
     http_req_connecting............: avg=123ns    min=0s      med=0s       max=344.78µs p(90)=0s       p(95)=0s
     http_req_duration..............: avg=290.42µs min=70.26µs med=189.71µs max=10.93ms  p(90)=513.41µs p(95)=729.17µs
       { expected_response:true }...: avg=290.42µs min=70.26µs med=189.71µs max=10.93ms  p(90)=513.41µs p(95)=729.17µs
     http_req_failed................: 0.00%  ✓ 0        ✗ 10000
     http_req_receiving.............: avg=35.96µs  min=8.91µs  med=19.42µs  max=9.51ms   p(90)=33.89µs  p(95)=63.45µs
     http_req_sending...............: avg=16.23µs  min=4.86µs  med=10.95µs  max=4.91ms   p(90)=16.92µs  p(95)=22.22µs
     http_req_tls_handshaking.......: avg=0s       min=0s      med=0s       max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=238.2µs  min=47.55µs med=152.19µs max=10.87ms  p(90)=435.47µs p(95)=634.45µs
     http_reqs......................: 10000  24306.293918/s
     iteration_duration.............: avg=395.17µs min=122.15µs med=272.09µs max=11.49ms p(90)=689.61µs p(95)=959.82µs
     iterations.....................: 10000  24306.293918/s
```

Gambar 3. Hasil *Load-Testing* dari *Micro-Batch K6*

```
putuaudipasuatmadi@pop-os:~/Workspaces/SNATIA-Personal/Dev/snatia_audi_ingestor_be$ k6 run -e MODE=non-microbatch loadtest.js

  execution: local
     script: loadtest.js
     output: -

  scenarios: (100.00%) 1 scenario, 10 max VUs, 10m30s max duration (incl. graceful stop):
           * default: 10000 iterations shared among 10 VUs (maxDuration: 10m0s, gracefulStop: 30s)


running (00m01.1s), 00/10 VUs, 10000 complete and 0 interrupted iterations
default ✓ [======================================] 10 VUs  00m01.1s/10m0s  10000/10000 shared iters

     data_received..................: 1.5 MB 1.4 MB/s
     data_sent......................: 2.4 MB 2.2 MB/s
     http_req_blocked...............: avg=4.36µs   min=651ns    med=2.33µs   max=1.68ms   p(90)=3.89µs  p(95)=4.88µs
     http_req_connecting............: avg=147ns    min=0s       med=0s       max=342.68µs p(90)=0s      p(95)=0s
     http_req_duration..............: avg=938.03µs min=342.58µs med=809.45µs max=9.54ms   p(90)=1.44ms  p(95)=1.78ms
       { expected_response:true }...: avg=938.03µs min=342.58µs med=809.45µs max=9.54ms   p(90)=1.44ms  p(95)=1.78ms
     http_req_failed................: 0.00%  ✓ 0        ✗ 10000
     http_req_receiving.............: avg=50.18µs  min=13.57µs  med=31.41µs  max=2.37ms   p(90)=58.5µs  p(95)=99.7µs
     http_req_sending...............: avg=19.42µs  min=6.85µs   med=14.69µs  max=870.46µs p(90)=21.81µs p(95)=28.71µs
     http_req_tls_handshaking.......: avg=0s       min=0s       med=0s       max=0s       p(90)=0s      p(95)=0s
     http_req_waiting...............: avg=868.42µs min=290.85µs med=745.97µs max=9.47ms   p(90)=1.35ms  p(95)=1.67ms
     http_reqs......................: 10000  9266.894816/s
     iteration_duration.............: avg=1.06ms   min=419.39µs med=919.86µs max=9.62ms   p(90)=1.59ms  p(95)=1.97ms
     iterations.....................: 10000  9266.894816/s
```

Gambar 4. Hasil *Load-Testing* dari tanpa *Micro-Batch K6*

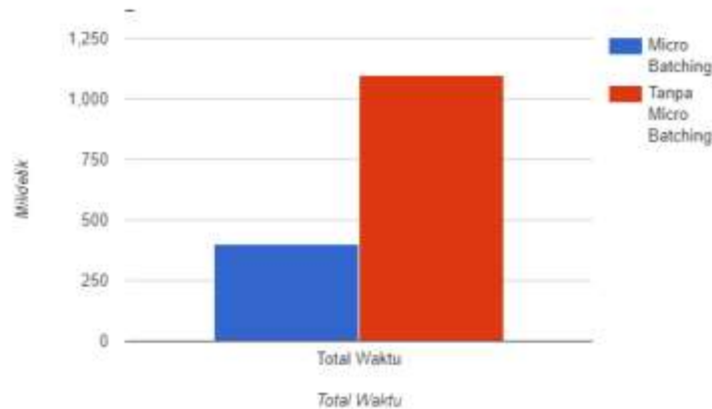
3.4. Analisis Hasil Pengujian

Dalam pengukuran kinerja dari metode *Micro-Batching* dan juga tanpa *Micro-Batching* pada penelitian ini, dapat menggunakan perbandingan dari hasil *Load-Testing* pada kedua metode tersebut. Perbandingan dapat dilihat dengan mudah pada **Tabel 6** di bawah.

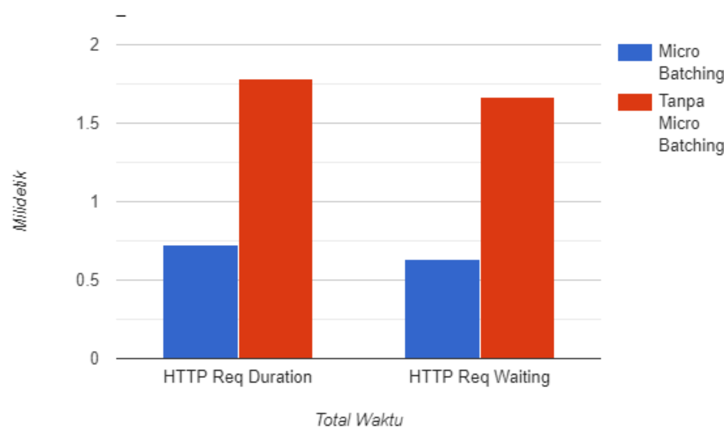
Tabel 6. Perbandingan Performa Kecepatan

Metris	Micro-Batching	Tanpa Micro-Batching	Performa Tertinggi
total waktu	0.4 detik	1.1 detik	*Micro-Batching*
http_req_duration	729.16 mikrodetik	1.78 milidetik	*Micro-Batching*
http_req_waiting	634.45 mikrodetik	1.67 milidetik	*Micro-Batching*

Pada tabel di atas, metris *http_req_duration* dan juga *http_req_waiting* menggunakan p(95) atau persentase *request* pada kecepatan yang jatuh pada persentase ke-95 agar lebih tepat dan menghindari *outlier*. Nilai – nilai metris ini didapatkan dari hasil pengujian menggunakan *Load-Testing*. Kemudian hasil penelitian divisualisasikan pada dua grafik di bawah, dan dapat terlihat dengan lebih jelas bagaimana perbandingan dari kedua metode tersebut.



Gambar 5. Grafik visualisasi hasil pengujian total waktu



Gambar 6. Grafik visualisasi hasil pengujian Http Req Duration & Req Waiting

4. Kesimpulan

Berdasarkan penelitian, ditemui bahwa menggunakan *Micro-Batching* memiliki keunggulan performa kecepatan jika dibandingkan dengan tidak menggunakan *Micro-Batching* atau tanpa *Micro-Batching*. Didapatkan bahwa menggunakan *Micro-Batching* lebih cepat dalam hal total durasi sebesar 63.63%, *http_req_duration* sebesar 59.03%, dan *http_req_waiting* sebesar 62%. Berdasarkan performa kecepatan yang diperoleh, ditunjukkan bahwa *Micro-Batching* baik digunakan dalam proses *Load* pada *Extract, Transform*, *Load* atau ETL pada *Event-Data* karena memiliki performa kecepatan yang lebih unggul. Menggunakan *Micro-Batching* lebih unggul karena menghindari *Repeated TCP* karena inisiasi koneksi TCP dilakukan untuk setiap *Micro-Batch* dan tidak setiap data melakukan inisiasi koneksi TCP.

References

[1] N. Shakhovska, N. Boyko, Y. Zasoba e E. Benova, “Big Data Processing Technologies in Distributed Information Systems,” 2019.

[2] J. C. Nwokeji, F. Aqlan, A. Apoorva e A. Olagunju, “Big Data ETL Implementation Approaches: A Systematic Literature Review,” 2018.

[3] M. Richards e N. Ford, Fundamentals of Software Architecture: An Engineering Approach, O'Reilly Media, Incorporated, 2020.

[4] U. Kroehne e F. Goldhammer, “How To Conceptualize, Represent, and Analyze Log Data From Technology-Based Assessments? A Generic Framework and an Application to Questionnaire Items,” *Behaviormetrika,* vol. 45, nº 2, pp. 527-563, 10 2018.

[5] M. Bauer, A. Senderovich, A. Gal, L. Grunske e M. Weidlich, “How Much Event Data Is Enough? A Statistical Framework for Process Discovery,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* vol. 10816 LNCS, pp. 239-256, 2018.

[6] A. R. Chen, “An Empirical Study on Leveraging Logs for Debugging Production Failures,” 2019.

[7] A. Sabtu, N. F. M. Azmi, N. N. A. Sjarif, S. A. Ismail, O. M. Yusop, H. Sarkan e S. Chuprat, “The Challenges of Extract, Transform and Loading (ETL) System Implementation For Near Real-Time Environment,” Langkawi, Malaysia, 2017.

[8] Y. Oyama, T. Ben-Nun, T. Hoefler e S. Matsuoka, “Less is More: Accelerating Deep Neural Networks with Micro-Batching,” *IPSJ SIG Technical Report,* Vols. %1 de %22017-HPC-162, nº 22, 2017.

[9] A. S. Abdelhamid, A. R. Mahmood, A. Daghistani e W. G. Aref, “Prompt: Dynamic Data-Partitioning for Distributed Micro-batch Stream Processing Systems,” 2020.

[10] M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, 1 ed., O'Reilly Media, Inc, 2017.

[11] T. Akidau, S. Chernyak e R. Lax, Streaming Systems: The What, Where, When, and how of Large-scale Data Processing, O'Reilly, 2018.

[12] A. Bellemare, Building Event-Driven Microservices: Leveraging Organizational Data at Scale, O'Reilly Media, Incorporated, 2020.

[13] J. Lee, G. Yang, Z. Niu, P. Cheng, Y. Xiong e C. Yoo, “Enhanced control path for repeated TCP connections,” 2020.

[14] N. A. Nor Sobri, M. A. H. Abas, A. I. Mohd Yassin, M. S. A. Megat Ali, N. Md Tahir e A. Zabidi, “Database Connection Pool in Microservice Architecture,” *Journal of Electrical & Electronic Systems Research,* vol. 20, nº APR2022, pp. 29-33, 4 2022.

[15] H. Schulz, T. Angerstein e A. Van Hoorn, “Towards Automating Representative Load Testing in Continuous Software Engineering,” 2018.

[16] M. Chadha, A. Jindal e M. Gerndt, “Architecture-Specific Performance Optimization of Compute-Intensive FaaS Functions,” 2021.

Please submit paper in doc or docx format. You can directly write your paper in this document.
PDF file format is **not recommended**.