PENGEMBANGAN METRIK KINERJA PADA *HYBRID RESOURCE SCHEDULER* UNTUK PELATIHAN *DEEP LEARNING* TERDISTRIBUSI

Laporan Tugas Akhir

Disusun sebagai syarat kelulusan tingkat sarjana

Oleh

PANDYAKA APTANAGI

NIM : 13517003

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

Juni 2021

PENGEMBANGAN METRIK KINERJA PADA *HYBRID RESOURCE SCHEDULER* UNTUK PELATIHAN *DEEP LEARNING* TERDISTRIBUSI

Laporan Tugas Akhir

Oleh

PANDYAKA APTANAGI

NIM : 13517003

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Telah disetujui dan disahkan sebagai Laporan Tugas Akhir

di Bandung, pada tanggal 4 Juni 2021

Pembimbing,

Achmad Imam Kistijantoro, S.T., M.Sc., Ph.D.

NIP. 197308092006041001

LEMBAR PERNYATAAN

Dengan ini saya menyatakan bahwa:

1. Pengerjaan dan penulisan Laporan Tugas Akhir ini dilakukan tanpa menggunakan bantuan yang tidak dibenarkan.
2. Segala bentuk kutipan dan acuan terhadap tulisan orang lain yang digunakan di dalam penyusunan laporan tugas akhir ini telah dituliskan dengan baik dan benar.
3. Laporan Tugas Akhir ini belum pernah diajukan pada program pendidikan di perguruan tinggi mana pun.

Jika terbukti melanggar hal-hal di atas, saya bersedia dikenakan sanksi sesuai dengan Peraturan Akademik dan Kemahasiswaan Institut Teknologi Bandung bagian Penegakan Norma Akademik dan Kemahasiswaan khususnya Pasal 2.1 dan Pasal 2.2.

Bandung, 4 Juni 2021

Pandyaka Aptanagi

NIM 13517003

ABSTRAK

PENGEMBANGAN METRIK KINERJA PADA *HYBRID RESOURCE SCHEDULER* UNTUK PELATIHAN *DEEP LEARNING* TERDISTRIBUSI

Oleh

PANDYAKA APTANAGI

NIM : 13517003

Pembelajaran mesin dengan menggunakan teknik *deep learning* terdistribusi merupakan teknik yang digunakan untuk ekstraksi fitur yang kompleks dan membutuhkan waktu yang lama. Salah satu kerangka kerja yang digunakan untuk melakukan pembelajaran mesin terdistribusi adalah AdaptDL. AdaptDL menjalankan proses pembelajaran mesin di atas *cluster* Kubernetes dengan menggunakan sistem penjadwal Pollux. Dalam menentukan keputusan penjadwalan, Pollux hanya menyediakan satu metrik kinerja, yaitu metrik Goodput, dan tidak memberikan opsi lain. Selain itu, Pollux juga memiliki potensi untuk memaksimalkan kecepatan pelatihan melalui pengubahan nilai Goodput, serta potensi untuk mengefisienkan sumber daya melalui pengubahan nilai ambang batas untuk penentuan mekanisme *scaling*. Pada tugas akhir ini, dilakukan pengembangan terhadap AdaptDL dan Pollux dengan menambahkan pemilihan metrik kinerja, metrik untuk memaksimalkan kecepatan, dan metrik untuk mengefisiensikan sumber daya. Pengembangan opsi metrik kinerja dilakukan pada kerangka kerja AdaptDL, metrik kecepatan dilakukan dengan melakukan modifikasi pada persamaan Goodput, dan metrik efisiensi dilakukan dengan melakukan modifikasi pada Pollux. Berdasarkan hasil pengujian dengan menggunakan model klasifikasi pengenalan citra pada *dataset* MNIST, pengembangan tidak memengaruhi akurasi model yang dihasilkan namun memengaruhi aspek kinerja lain. Pengembangan opsi metrik tidak memengaruhi kinerja pembelajaran secara keseluruhan. Pengembangan metrik kecepatan memengaruhi kecepatan pelatihan sehingga melambat sebesar 16,099%. Sedangkan pengembangan metrik efisiensi memengaruhi kecepatan pelatihan sehingga melambat sebesar 106,977%, waktu pembangkitan sumber daya meningkat sebesar 80%, dan penggunaan sumber daya meningkat sebesar 19,31%.

Kata kunci: Kubernetes, AdaptDL, Pollux, *deep learning*, *hybrid resource scheduler*.

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa, karena atas rahmat dan karunia-Nya, penulis dapat menyelesaikan laporan tugas akhir yang berjudul "Pengembangan Metrik Kinerja pada *Hybrid Resource Scheduler* untuk Pelatihan *Deep Learning* Terdistribusi" ini dengan baik. Tugas akhir ini disusun sebagai persyaratan untuk memperoleh gelar sarjana pada program studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung. Penulisan tugas akhir ini tidak lepas dari dukungan dan bantuan dari berbagai pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Bapak Achmad Imam Kistijantoro S.T., M.Sc., Ph.D selaku dosen pembimbing tugas akhir yang telah membimbing dan mengarahkan penulis dalam proses pengerjaan Tugas Akhir dari awal sampai akhir.
2. Bapak Dr. Judhi Santoso M.Sc. dan penguji 2 selaku dosen penguji tugas akhir.
3. Bapak Adi Mulyanto S.T., M.T., Ibu Ginar Santika Niwanputri S.T., M.Sc., Ibu Dr. Fazat Nur Azizah S.T., M.Sc., Bapak Nugraha Priya Utama S.T., M.A., Ph.D., dan Ibu Latifa Dwiyanti S.T., M.T. selaku dosen pengampu mata kuliah Tugas Akhir yang telah memberi arahan selama proses pengerjaan tugas akhir.
4. Ibu Dessi Puji Lestari S.T., M.Eng., Ph.D selaku dosen wali penulis yang telah mendampingi penulis selama berada di Program Studi Teknik Informatika.
5. Bapak Agung Budi Cahyono dan Ibu Isti Purwaningsih beserta kedua adik penulis yang telah memberikan dukungan tanpa henti kepada penulis.
6. Izdihar Ramadhanty Abiyasa Putri selaku teman dekat penulis yang telah memberikan dukungan dan motivasi penulis dari awal perkuliahan hingga akhir.

7. I Putu Gede Wirasuta, M. Rifky Indraputra Bariansyah, dan Gardahadi selaku teman seperjuangan penulis yang telah memberikan dukungan pada banyak hal, mulai dari perkuliahan, pekerjaan, hingga kehidupan.
8. Farhan Ramadhan Syah Khair, Syaiful Anwar, M. Ivan Rahmansyah Maulana, dan teman-teman "Internship Google" lainnya yang telah menjadi tempat penulis untuk berdiskusi banyak hal di luar kuliah.
9. Teman-teman Departemen Kesekjenan, "Sugar Baby", dan Badan Pengurus HMIF ITB 2020/2021 yang telah memberikan penulis pengalaman untuk dikenang dan dukungan untuk menyelesaikan Tugas Akhir.
10. Seluruh pihak lain yang tidak dapat penulis sebutkan satu persatu yang telah membantu dan mendukung penulis dalam pengerjaan Tugas Akhir.

Laporan tugas akhir ini jauh dari kata sempurna, oleh karena itu penulis sangat terbuka terhadap kritik dan saran yang membangun. Akhir kata, semoga laporan tugas akhir ini dapat memberikan dampak dan manfaat yang baik kepada seluruh pihak yang memerlukan.

Bandung, 4 Juni 2021

Penulis

DAFTAR ISI

DAFTAR LAMPIRAN

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR ISTILAH

Istilah	Deskripsi Istilah
Auto-scaling	Proses meningkatkan atau menurunkan, baik jumlah maupun kapasitas, *worker* pada suatu Job.
Hybrid Resource Scheduler	Sistem yang digunakan untuk melakukan penjadwalan pada kerangka kerja pembelajaran *deep learning* terdistribusi. Penjadwalan dapat dilakukan dengan optimasi pada level individual (Job) maupun kelompok (klaster).
Job (Kubernetes)	Proses pembelajaran mesin yang direpresentasikan sebagai suatu komponen pada Kubernetes.
Layer (Deep Learning)	Suatu struktur dari arsitektur model *deep learning* yang melakukan pemrosesan informasi.
Metrik Kinerja	Suatu elemen individual yang dapat digunakan untuk mengukur kinerja pada sistem.
Worker Node	Representasi mesin pekerja yang melakukan proses pembelajaran *deep learning*.

DAFTAR KODE

BAB I

PENDAHULUAN

Bab Pendahuluan menjelaskan tentang landasan dan arah kerja tugas akhir. Bab ini terdiri dari latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, serta sistematika pembahasan tugas akhir.

I.1 Latar Belakang

Teknologi pembelajaran mesin merupakan aspek penting dalam kehidupan modern. Dengan adanya pembelajaran mesin, teknologi seperti sistem pencarian, sistem rekomendasi, hingga robot yang mampu melakukan pekerjaan secara otomatis dapat diciptakan. Salah satu teknik yang digunakan pada teknologi pembelajaran mesin adalah teknik *deep learning*. Pembelajaran mesin dengan teknik *deep learning* menggunakan satu atau lebih lapisan *layer* yang digunakan untuk memproses informasi, baik secara *supervised* maupun *unsupervised*. Teknik *deep learning* sendiri digunakan untuk mengeliminasi kelemahan pembelajaran mesin konvensional, yaitu keterbatasan kemampuan untuk memproses data dalam bentuk aslinya (LeCun dkk., 2014). Dengan menggunakan teknik *deep learning*, mesin dapat mempelajari fungsi yang lebih kompleks dibandingkan dengan menggunakan pembelajaran mesin konvensional.

Banyaknya lapisan *layer* dan penggunaan data yang kompleks pada proses pembelajaran *deep learning* mengakibatkan teknik ini membutuhkan waktu yang lama dalam melakukan pelatihan. Semakin banyak data dan lapisan *layer* yang digunakan, semakin lama waktu yang dibutuhkan untuk proses pelatihan (Hegde & Usmani, 2018). Selain jumlah data dan lapisan *layer* yang digunakan, waktu yang dibutuhkan dalam proses pelatihan *deep learning* juga dipengaruhi oleh jumlah iterasi atau *epoch*. Untuk melakukan ekstraksi fitur yang kompleks, dibutuhkan *epoch* yang banyak sehingga waktu yang dibutuhkan oleh proses pelatihan *deep learning* semakin lama.

Salah satu pendekatan untuk mengatasi permasalahan lamanya proses pelatihan *deep learning* adalah dengan melakukan pelatihan *deep learning* secara terdistribusi. Pelatihan *deep learning* terdistribusi melakukan pembagian tugas pelatihan menjadi tugas-tugas kecil yang didistribusikan ke banyak mesin sehingga mempercepat waktu yang dibutuhkan. Semakin banyak mesin yang digunakan, semakin mahal biaya yang harus dikeluarkan sehingga penggunaan mesin harus digunakan seefisien mungkin. Dalam proses pelatihan *deep learning* terdistribusi, terdapat suatu sistem *scheduler* yang digunakan untuk menentukan banyak mesin yang digunakan dan melakukan penjadwalan pendistribusian tugas ke mesin tersebut. Sistem *scheduler* sendiri memiliki metrik kinerja yang sudah terdefinisi untuk menentukan keputusan penentuan banyak mesin dan pendistribusian tugas ke mesin pekerja. Umumnya, sistem *scheduler* hanya menggunakan satu metrik kinerja saja, seperti prediksi intra Job pada *scheduler* Gandiva (Xiao dkk., 2007) dan 2DAS pada *scheduler* Tiresias (Gu dkk., 2019), untuk melakukan penjadwalan.

Pada tugas akhir ini, dikembangkan sistem *scheduler* untuk pelatihan *deep learning* terdistribusi yang dapat menggunakan metrik kinerja sesuai dengan masukan pengguna. Selain itu, dikembangkan juga metrik kinerja untuk memaksimalkan kecepatan dan metrik kinerja untuk melakukan efisiensi sumber daya sebagai pilihan metrik yang dapat digunakan. Sistem merupakan pengembangan lebih lanjut dari sistem yang sudah dikembangkan bernama Pollux.

Pollux adalah *scheduler* untuk pembelajaran *deep learning* yang dapat melakukan alokasi sumber daya secara adaptif dan secara bersamaan melakukan *tuning* untuk setiap tugas pelatihan sehingga sumber daya yang diberikan dapat dimanfaatkan dengan maksimal (Qiao dkk., 2020). Karena melakukan *scheduling* berdasarkan keadaan lingkungan dan keadaan tugas itu sendiri, Pollux disebut juga sebagai *scheduler* yang *hybrid*. Pollux merupakan sistem *scheduler* untuk kerangka kerja *deep learning* yang bernama AdaptDL. Dalam penggunaannya, AdaptDL hanya menyediakan satu opsi metrik kinerja saja untuk Pollux, yaitu metrik kinerja Goodput. Pengembangan lebih lanjut dilakukan dengan mengimplementasikan opsi metrik kinerja tambahan untuk Pollux pada kerangka kerja AdaptDL. Selain itu,

dilakukan modifikasi terhadap metrik Goodput dan nilai ambang batas pada penentuan keputusan *scaling* untuk mencapai tujuan dari opsi metrik kinerja tambahan, yaitu kecepatan maksimal dan efisiensi sumber daya.

I.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan pada Subbab 1.1, terdapat beberapa permasalahan yang dapat diselesaikan pada pelatihan *deep learning* terdistribusi yang sudah ada. Rumusan masalah untuk diselesaikan pada tugas akhir ini adalah sebagai berikut:

1. Bagaimana pengaruh metrik kinerja yang dapat berubah sesuai masukan pengguna terhadap kecepatan proses pelatihan dan akurasi model yang dihasilkan dalam proses *deep learning* terdistribusi?
2. Bagaimana pengaruh metrik kinerja kecepatan pada *hybrid resource scheduler* terhadap kecepatan proses pelatihan dan akurasi model yang dihasilkan dalam proses *deep learning* terdistribusi?
3. Bagaimana pengaruh metrik kinerja sumber daya pada *hybrid resource scheduler* terhadap penggunaan sumber daya, kecepatan proses pelatihan, dan akurasi model yang dihasilkan dalam proses *deep learning* terdistribusi?

I.3 Tujuan

Tujuan pembuatan tugas akhir ini adalah sebagai berikut:

1. Menentukan pengaruh *hybrid resource scheduler* yang dapat menerima masukan metrik kinerja terhadap kecepatan proses pelatihan *deep learning* terdistribusi dan akurasi model yang dihasilkan.
2. Menentukan pengaruh metrik kinerja kecepatan pada *hybrid resource scheduler* terhadap kecepatan proses pelatihan *deep learning* dan akurasi model yang dihasilkan.

3. Menentukan pengaruh metrik kinerja sumber daya pada *hybrid resource scheduler* terhadap penggunaan sumber daya dalam proses pelatihan *deep learning*, kecepatan proses pelatihan, dan akurasi model yang dihasilkan.

I.4 Batasan Masalah

Batasan masalah dari tugas akhir ini adalah sebagai berikut:

1. Metrik diimplementasikan pada *scheduler* yang dijalankan oleh kerangka kerja AdaptDL.
2. Model *deep learning* yang digunakan dalam pengembangan dan pengujian solusi adalah model untuk melakukan klasifikasi untuk pengenalan citra (*image recognition*).
3. *Dataset* yang digunakan dalam proses pengembangan dan pengujian solusi adalah *dataset* MNIST.

I.5 Metodologi

Dalam pengerjaan tugas akhir ini, metodologi yang digunakan adalah sebagai berikut:

1. Analisis Permasalahan

   Pada tahap ini dilakukan identifikasi dan analisis terhadap permasalahan pada proses pelatihan *deep learning* tedistribusi yang sudah ada. Hasil dari tahap ini berupa permasalahan yang menjadi landasan dalam pengerjaan Tugas Akhir.

2. Analisis dan Desain Solusi

   Berdasarkan hasil dari analisis permasalahan, dilakukan analisis dan pembangunan desain terhadap solusi yang digunakan untuk menyelesaikan permasalahan. Analisis dilakukan terhadap kerangka kerja dan sistem *scheduler* untuk *deep learning* terdistribusi yang sudah ada. Kemudian, dilakukan pengembangan desain solusi sesuai dengan hasil analisis sesuai dengan tujuan dari tugas akhir.

3. Implementasi dan Pengembangan Solusi

   Pada tahap ini dilakukan implementasi dan pengembangan solusi berdasarkan hasil analisis dan desain solusi. Hasil dari tahap ini adalah kerangka kerja dan *scheduler* yang sudah dimodifikasi sesuai dengan tujuan tugas akhir.

4. Pengujian

   Pada tahap ini dilakukan pengujian terhadap solusi yang sudah diimplementasikan dengan menggunakan data dan model yang sudah ditentukan, yaitu *dataset* MNIST dan model klasifikasi untuk pengenalan citra.

5. Analisis Hasil dan Evaluasi

   Pada tahap ini, dilakukan analisis terhadap hasil pengujian solusi yang sudah dibangun sebagai evaluasi terhadap solusi tugas akhir. Evaluasi dilakukan untuk mengukur kesesuaian dan ketercapaian dari tugas akhir. Evaluasi dilakukan dengan menggunakan pendekatan kuantitatif.

I.6 Sistematika Pembahasan

Laporan tugas akhir dibagi menjadi enam bab, yaitu:

1. Pendahuluan

   Bab Pendahuluan berisi pengantar sekaligus gambaran tugas akhir secara garis besar. Bab ini terdiri dari enam subbab, yaitu subbab latar belakang, subbab rumusan masalah, subbab tujuan, subbab batasan masalah, subbab metodologi, dan subbab sistematika pembahasan tugas akhir.

2. Studi Literatur

   Bab Studi Literatur berisi pembahasan dasar teori dan literatur-literatur yang terkait dengan sistem *deep learning* terdistribusi pada Kubernetes. Bab ini terdiri atas kajian tentang *deep learning*, perangkat, kakas, dan pustaka yang digunakan dalam pengerjaan tugas akhir, data yang digunakan untuk

implementasi dan pengujian, serta penelitian yang terkait dengan topik tugas akhir.

3. Analisis Permasalahan dan Rancangan Solusi

   Bab Analisis Permasalahan dan Rancangan Solusi terdiri dari tiga subbab, yaitu analisis permasalahan, analisis solusi, dan deskripsi umum solusi. Subbab analisis permasalahan berisi pembahasan tentang masalah yang diselesaikan pada tugas akhir, sedangkan subbab analisis dan deskripsi solusi umum berisi pembahasan tentang pendekatan yang digunakan untuk menyelesaikan permasalahan untuk kemudian menjadi solusi tugas akhir.

4. Implementasi

   Bab Implementasi berisi pembahasan mengenai implementasi rancangan solusi berdasarkan analisis dan desain yang diusulkan. Bab ini terdiri dari tiga subbab, yaitu subbab implementasi pemilihan metrik kinerja, subbab implementasi metrik untuk memaksimalkan kecepatan, dan subbab implementasi metrik untuk mengefisienkan sumber daya.

5. Pengujian

   Bab Pengujian berisi pengujian dan hasil pengujian dari solusi yang sudah diimplementasi. Bab ini terdiri dari enam subbab, yaitu subbab tujuan pengujian, subbab lingkungan pengujian, subbab metode pengujian, subbab skenario pengujian, subbab hasil pengujian, dan subbab analisis hasil pengujian.

6. Simpulan dan Saran

   Bab Simpulan dan Saran berisi simpulan yang menjawab rumusan masalah tugas akhir dan hal-hal yang dapat dikembangkan lebih lanjut untuk topik tugas akhir. Bab ini terdiri dari dua subbab, yaitu subbab simpulan dan subbab saran.

BAB II

STUDI LITERATUR

Pada Bab Studi Literatur dijelaskan mengenai dasar teori dan literatur yang berkaitan dengan tugas akhir. Kajian literatur berisikan teori dan pekerjaan yang berkaitan dengan persoalan tugas akhir, informasi apa saja yang sudah ada dan berkaitan dengan persoalan tugas akhir, serta masalah yang belum terselesaikan.

II.1 Deep Learning

Pembelajaran mesin dengan *deep learning* merupakan bagian dari pembelajaran mesin (*machine learning*). Pembelajaran dengan *deep learning* menggunakan model yang terdiri dari satu atau lebih lapisan *layer* non-linear. Lapisan *layer* tersebut digunakan untuk melakukan pembelajaran secara *supervised* maupun *unsupervised* terhadap representasi fitur pada *layer* yang abstrak dan berturut-turut (Deng & Yu, 2014). Pembelajaran mesin dengan menggunakan *deep learning* dapat diimplementasikan pada banyak aspek seperti *computer vision*, pemrosesan suara, pemrosesan bahasa alami, robotik, pemrosesan audio, dan pemrosesan citra.

Lapisan *layer* yang ada pada model *deep learning* pada umumnya terdiri dari banyak parameter dan membutuhkan banyak data untuk dapat memenuhi tujuan dari model. Semakin banyak data dan lapisan *layer*, semakin kompleks fitur yang dapat diekstraksi oleh model, meskipun dengan risiko waktu yang dibutuhkan semakin tinggi (Hegde & Usmani, 2016). Untuk mengatasi permasalahan tersebut, digunakan pembelajaran *deep learning* secara terdistribusi yang membagi tugas-tugas dalam pembelajaran mesin ke beberapa mesin pekerja yang saling terhubung dan sinkron satu sama lain.

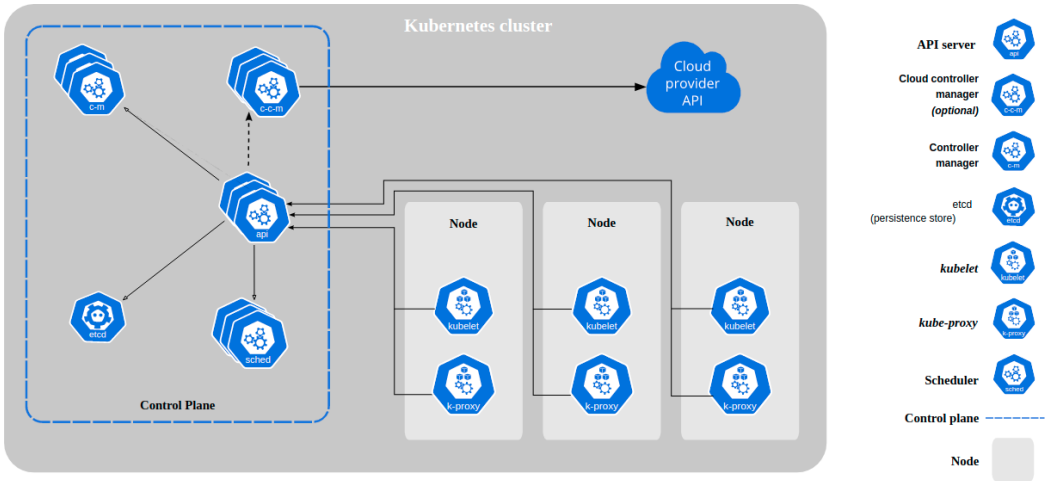
II.2 Kubernetes

Kubernetes merupakan platform *open-source* yang awalnya dikembangkan oleh Google pada tahun 2014 dan digunakan untuk mengelola beban kerja dan layanan yang dikemas (*containerized*) serta memfasilitasi konfigurasi deklaratif dan

automasi (Kubernetes, 2020). Kubernetes didesain untuk mengelola siklus hidup dari aplikasi dan layanan yang dikemas menggunakan kumpulan layanan sehingga kelangsungan sistem dapat diprediksi, *scalable*, serta memiliki ketersediaan yang tinggi. Beberapa layanan yang disediakan oleh Kubernetes adalah *service discovery* dan *load balancing*, orkestrasi untuk tempat penyimpanan, automasi untuk *rollouts* dan *rollbacks* sistem, mekanisme pemulihan terhadap sistem, serta manajemen konfigurasi untuk informasi sensitif.

II.2.1 Arsitektur Kubernetes

Pada dasarnya, Kubernetes mengumpulkan beberapa mesin/server menjadi sebuah klaster. Klaster pada Kubernetes merupakan tempat dimana semua komponen, kapabilitas, dan beban kerja pada Kubernetes dikonfigurasi. Dalam satu klaster Kubernetes terdapat satu server atau satu kelompok server yang menjadi *control plane* dan server atau kelompok sisanya menjadi *node*. Server atau kelompok yang menjadi *control plane* bertugas untuk mengatur server lain yang menjadi *node*, seperti melakukan *health checking* serta penjadwalan kerja (*scheduling*), sedangkan server atau kelompok yang menjadi *node* bertugas untuk menerima dan melakukan pekerjaan dengan menggunakan sumber daya yang ada. Arsitektur pada klaster Kubernetes dapat dilihat pada Gambar II.2.1.



Gambar II.2.1 Arsitektur Kubernetes (Kubernetes, 2020)

II.2.1.1 Kubernetes Control Plane

Pada Kubernetes, server yang menjadi *control plane* bertugas untuk membuat keputusan secara global seperti *scheduling* dan mendeteksi serta merespon *event* pada klaster. *Control plane* pada Kubernetes sendiri terdiri beberapa komponen, yaitu etcd, kube-apiserver, kube-controller-manager, kube-scheduler dan cloud-controller-manager.

i. Komponen etcd merupakan komponen yang digunakan oleh Kubernetes untuk menyimpan data konfigurasi yang dapat diakses oleh seluruh klaster. Selain itu, komponen ini juga dapat digunakan untuk melakukan *service discovery* dan sebagai data untuk melakukan konfigurasi sesuai dengan keadaan terbaru klaster.
ii. Komponen kube-apiserver merupakan komponen dari Kubernetes *control plane* yang digunakan sebagai antarmuka untuk melakukan interaksi antar komponen. Antarmuka yang digunakan pada kube-apiserver berbentuk RESTful. Selain itu, kube-apiserver juga dapat diakses secara lokal dengan menggunakan suatu *client* yang bernama kubectl.
iii. Komponen kube-controller-manager merupakan komponen pada Control Plane yang menjalankan proses *controller*. Proses *controller* yang dijalankan berupa Node *controller*, Replication *controller*, Endpoints *controller*, serta Service Account dan Token *controller*.
iv. Komponen kube-scheduler merupakan komponen yang mendistribusikan beban kerja untuk node secara spesifik pada suatu klaster. Faktor yang digunakan untuk melakukan distribusi beban kerja adalah kebutuhan sumber daya secara individu dan kolektif, batasan perangkat, afinitas dan non-afinitas dari kebutuhan, *data locality*, interferensi antar beban kerja, serta tenggat waktu dari beban kerja.
v. Komponen cloud-controller-manager merupakan komponen yang digunakan klaster untuk berhubungan dengan API dari penyedia layanan awan dan memisahkan antara komponen yang berhubungan dengan

penyedia layanan awan dengan komponen yang tidak. Komponen ini hanya menjalankan *controller* secara spesifik terhadap penyedia layanan awan.

II.2.1.2 Kubernetes Node

Pada klaster Kubernetes, server selain *control plane* akan menjadi server *node*. *Node* terdiri dari beberapa komponen, yaitu kubelet, kube-proxy, dan *container runtime*.

i. Komponen kubelet merupakan komponen yang berjalan pada setiap *node* yang ada pada klaster. Komponen ini memastikan *container* berjalan pada setiap Pod yang ada. Selain itu, kubelet juga berfungsi untuk menyampaikan pesan dari dan ke *control plane* serta berinteraksi dengan komponen etcd untuk membaca detail konfigurasi dan menuliskan nilai baru ke etcd.
ii. Komponen kube-proxy merupakan komponen untuk mengelola jaringan yang ada pada *node*. Jaringan pada *node* digunakan untuk melakukan komunikasi dari dalam dan luar klaster menuju Pod. Kube-proxy memastikan *request* diteruskan menuju ke container yang tepat serta bertanggung jawab atas jaringan yang ada supaya bersifat dapat diprediksi dan diakses, namun tetap dapat diisolasi ketika dibutuhkan.
iii. Komponen *container runtime* merupakan komponen yang bertanggung jawab untuk memulai dan mengelola *container*, yaitu aplikasi yang dikemas dalam suatu lingkungan tertentu. Kubernetes mendukung beberapa *container runtime* seperti Docker, containerd, dan *container* yang mengimplementasi Kubernetes Container Runtime Interface (CRI).

II.2.2 Objek pada Kubernetes

Objek pada Kubernetes merupakan entitas persisten yang ada pada sistem Kubernetes dan digunakan untuk merepresentasikan keadaan dari klaster. Keadaan yang direpresentasikan oleh objek Kubernetes adalah aplikasi apa yang sedang berjalan serta berada pada *node* mana, sumber daya yang tersedia untuk aplikasi, serta pengaturan kebijakan tentang aplikasi tersebut. Subbab ini menjelaskan beberapa objek umum yang ada pada Kubernetes.

II.2.2.1 Pod

Pod adalah unit terkecil yang dapat dibuat dan dikelola dalam Kubernetes. Pod adalah kumpulan dari satu atau lebih *container* yang saling berbagi alamat IP dan volume serta pengaturan tentang bagaimana *container* berjalan. Umumnya Pod terdiri dari *container* utama yang bertanggung jawab terhadap beban kerja pada Pod dan *container* opsional lain yang membantu pekerjaan dari *container* utama. Pod dapat dikonfigurasi dengan menggunakan Podspec yang memiliki ekstensi .yaml.

II.2.2.2 Replication Controllers dan ReplicaSet

Replication Controllers adalah objek yang mendefinisikan sebuah *pod template* dan parameter kontrol untuk melakukan *scaling* secara horizontal dengan menambahkan atau mengurangi jumlah dari Pod yang sedang berjalan. Replication Controllers bertanggung jawab untuk memastikan jumlah Pod yang ada pada klaster sesuai dengan jumlah Pod yang didefinisikan pada konfigurasi.

ReplicaSet merupakan versi lebih baru dari Replication Controllers yang bertujuan untuk mengelola kumpulan Pod yang stabil dan berjalan pada suatu waktu. Sama seperti Replication Controller, ReplicaSets dapat melakukan *self-healing* untuk menjalankan kembali Pod secara otomatis apabila terjadi kegagalan pada Pod. Selain itu, ReplicaSet juga dapat menambahkan atau mengurangi jumlah dari Pod sesuai dengan kebutuhan.

II.2.2.3 Job

Job merupakan beban kerja pada Kubernetes yang berbasis tugas. Job akan menciptakan satu atau lebih Pod untuk menyelesaikan tugas Job tersebut. Ketika tugas tersebut selesai, maka Pod yang menjalankan tugas akan berhenti. Ketika tugas belum selesai namun Pod mengalami kegagalan, maka Job dapat menciptakan Pod baru untuk menyelesaikan tugasnya. Job dapat menciptakan lebih dari satu Pod untuk menjalankan tugas secara paralel.

II.2.2.4 Service

Service merupakan abstraksi dari Pod yang menyediakan sebuah alamat IP dan DNS yang digunakan untuk mengakses Pod tersebut. Service pada Kubernetes

merupakan objek yang mengimplementasikan arsitektur REST. Terdapat beberapa tipe dari Service, yaitu tipe ClusterIP, NodePort, LoadBalancer, dan ExternalName.

i. ClusterIP merupakan tipe yang melakukan ekspos Service pada Cluster-internal IP. Dengan menggunakan tipe ini, Service hanya bisa diakses dari dalam klaster. Tipe ini merupakan tipe Service *default*.

ii. NodePort merupakan tipe yang melakukan ekspos Service pada alamat IP dan *port* statik setiap *node*. Dengan menggunakan tipe ini, Service dapat diakses dari luar dengan melakukan *request* pada NodeIP dan NodePort.

iii. LoadBalancer merupakan tipe melakukan ekspos Service secara eksternal dengan menggunakan Load Balancer dari penyedia layanan awan. NodePort dan NodeIP akan secara otomatis dibuat dan mengarah pada Load Balancer yang digunakan.

iv. ExternalName merupakan tipe yang memetakan Service dengan *externalName* dengan cara mengembalikan catatan CNAME bersama dengan nilainya.

II.2.3 Horizontal Pod Autoscaler

Horizontal Pod Autoscaler merupakan layanan dari Kubernetes yang dapat melakukan *scaling* jumlah Pod yang ada pada Replication Controller dan ReplicaSets berdasarkan metrik tertentu, seperti penggunaan CPU dan memori, secara otomatis. Horizontal Pod Autoscaler diimplementasikan dalam bentuk sumber daya pada Kubernetes API dan *controller*. *Controller* secara periodik akan menyesuaikan jumlah dari replika pada Replication Controller sesuai dengan metrik penggunaan sumber daya dan target yang didefinisikan oleh pengguna. Cara kerja dari Horizontal Pod Autoscaler ditunjukkan pada Gambar II.2.2.

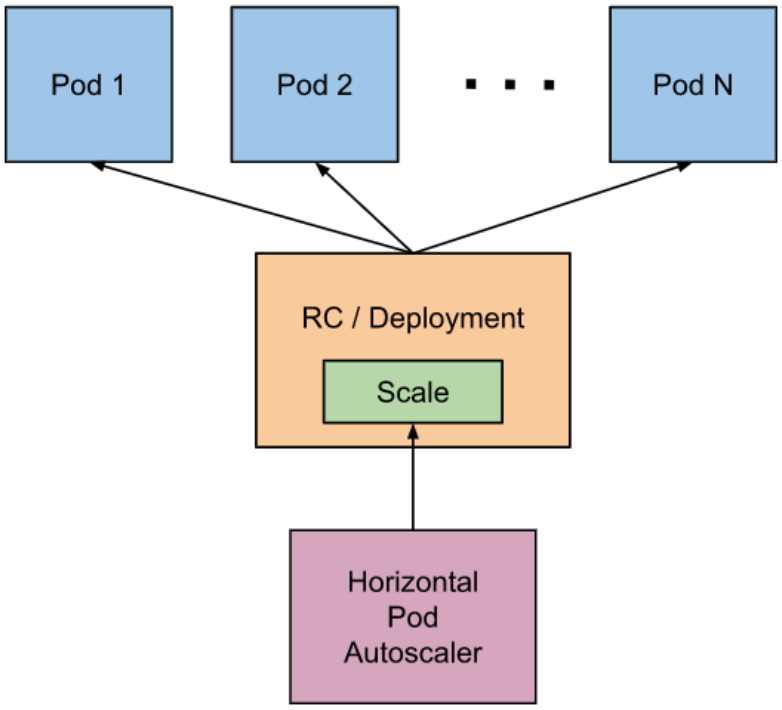
Horizontal Pod Autoscaler diimplementasikan sebagai sebuah *control loop* yang secara periodik akan mengambil data penggunaan sumber daya sesuai dengan metrik tertentu yang telah didefinisikan untuk setiap Horizontal Pod Autoscaler. Metrik yang digunakan bisa berupa Kubernetes Metrics API atau metrik lainnya. Secara umum, Horizontal Pod Autoscaler akan mengambil nilai pemanfaatan

sumber daya setiap Pod yang ditargetkan berdasarkan metrik yang telah didefinisikan, kemudian hasilnya digunakan untuk menentukan jumlah replika yang akan diperbanyak.

Horizontal Pod Autoscaler menggunakan perbandingan dari metrik yang diinginkan dengan keadaan metrik saat ini, yang hasilnya digunakan untuk menentukan jumlah replika yang akan diperbanyak. Persamaan untuk Horizontal Pod Autoscaler dapat dilihat pada persamaan (II-1).

$$desiredReplicas = ceil[currentReplicas * (\frac{currentMetricValue}{desiredMetricValue})] \quad \text{(II-1)}$$

Nilai dari *currentMetricValue* juga dapat dihitung berdasarkan rata-rata nilai sasaran pada seluruh Pod tujuan apabila parameter *targetAverageValue* atau *targetAverageUtilization* ditentukan.



Gambar II.2.2 Horizontal Pod Autoscaler pada Kubernetes (Kubernetes, 2020)

Sebelum menentukan nilai toleransi dan nilai akhir, kesiapan Pod dan nilai metrik dari Pod yang hilang akan dipertimbangkan terlebih dahulu. Semua Pod yang sedang dalam proses untuk dimatikan dan Pod yang gagal akan dihilangkan dari perhitungan, sedangkan Pod dengan nilai metrik yang hilang juga tidak masuk perhitungan namun akan digunakan untuk pertimbangan nilai akhir yang didapatkan. Pod yang sedang mempersiapkan diri juga tidak akan dihitung dan akan menjadi pertimbangan nilai akhir.

Sebelum Horizontal Pod Autoscaler melakukan *scaling* dan replikasi berdasarkan perhitungan yang sudah dilakukan, rekomendasi dari *scaling* akan dicatat. *Controller* akan mempertimbangkan semua rekomendasi dalam suatu rentang waktu untuk memilih rekomendasi tertinggi. Dengan adanya rentang waktu, maka *scaling* dan replikasi akan dilakukan secara bertahap dan mengurangi dampak dari perubahan nilai metrik yang cepat (Kubernetes, 2020).

II.2.4 Kubernetes Cluster Autoscaler

Kubernetes Cluster Autoscaler merupakan kakas yang digunakan untuk melakukan *autoscaling* ukuran dari suatu klaster Kubernetes secara otomatis (Kubernetes, 2020). Berbeda dengan *autoscaling* menggunakan Horizontal Pod Autoscaler (HPA) dan Vertical Pod Autoscaler (VPA) yang melakukan *autoscaling* pada level *container* dan *node*, Kubernetes Cluster Autoscaler melakukan autoscaling pada level infrastruktur dengan cara menambahkan atau mengurangi jumlah *node* pada suatu klaster (Tamiru dkk., 2020). Cara kerja dari Kubernetes Cluster Autoscaler dapat dilihat pada Gambar II.3.1.

Kubernetes Cluster Autoscaler melakukan penambahan jumlah *node* pada suatu klaster ketika terjadi tiga keadaan, yaitu:

- Terdapat Pod yang gagal dijadwalkan oleh *node* yang ada pada klaster karena kekurangan sumber daya yang dibutuhkan
- Menambahkan *node* yang mirip dengan *node* yang ada saat ini membantu untuk menyelesaikan permasalahan kekurangan sumber daya, dan

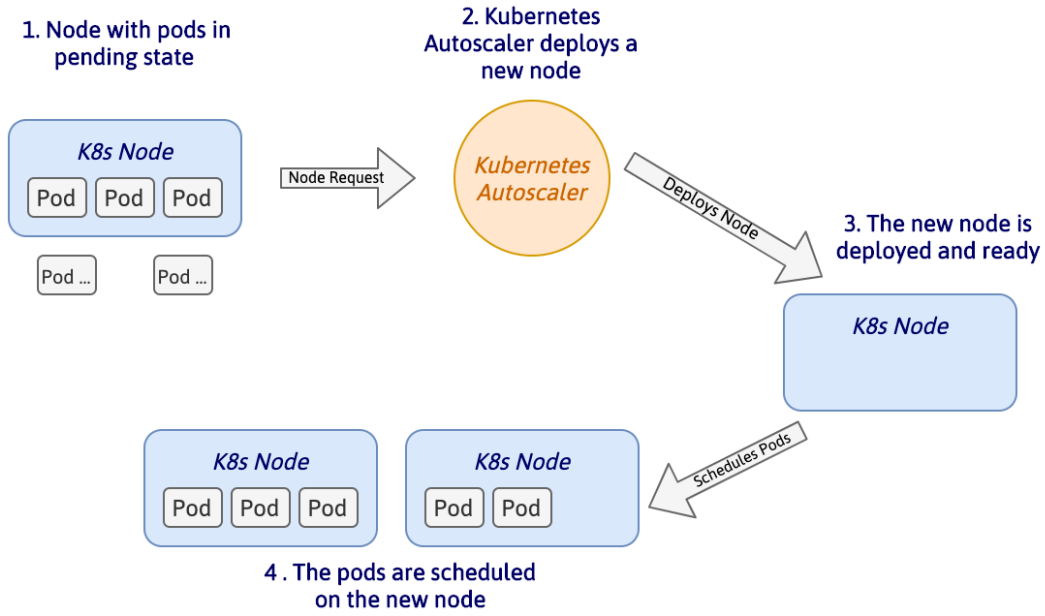
- Jumlah *node* yang ada pada klaster masih belum mencapai jumlah maksimum

Ketika *node* baru berhasil ditambahkan oleh Kubernetes Cluster Autoscaler, maka Pod yang memiliki status *pending* akan dijalankan pada *node* baru tersebut.

Selain menambah jumlah *node*, Kubernetes Cluster Autoscaler juga dapat mengurangi jumlah *node* yang ada pada klaster apabila terdapat *node* yang tidak dibutuhkan dalam jangka waktu tertentu. Suatu *node* dikategorikan sebagai *node* yang tidak dibutuhkan apabila memiliki nilai *utilization* yang rendah dan Pod yang ada pada *node* tersebut dapat dipindah ke *node* lain secara bebas.

II.3 PyTorch

PyTorch adalah *package* untuk bahasa Python yang menyediakan dua fitur *high-level* yaitu komputasi Tensor dengan akselerasi Graphical Processor Unit (GPU) dan *deep neural network* yang dibangun di atas sistem *autograd* berbasis *tape* (PyTorch, 2020).



Gambar II.3.1 Diagram Alir Kubernetes Auto Scaler (Rollik, 2020)

Pada umumnya, PyTorch digunakan sebagai pengganti kakas NumPy sehingga dapat menggunakan GPU untuk melakukan *deep learning*. Selain itu, juga dapat digunakan sebagai platform yang fleksibel dan cepat untuk melakukan *deep learning*.

Pembelajaran *deep learning* terdistribusi pada PyTorch menggunakan *package* Distributed dari Torch. *Package* tersebut memiliki tiga komponen utama, yaitu Distributed Data-Parallel Training (DDP) yang mengadopsi prinsip *single-program multiple-data*, RPC-Based Distributed Training (RPC) yang dikembangkan untuk mendukung pelatihan yang tidak bisa menggunakan paralelisasi data, dan Collective Communication (c10d) yang digunakan untuk mengirim Tensor antar proses pada satu grup yang sama.

Salah satu metode untuk melakukan *deep learning* terdistribusi adalah dengan metode paralelisasi data. Pada PyTorch, *package* Distributed Data-Parallel Training dapat digunakan untuk melakukan pelatihan terdistribusi dengan metode paralelisasi data. Paralelisasi data dilakukan dengan cara membagi data sejumlah GPU yang tersedia, kemudian setiap GPU akan melakukan pelatihan dengan data yang sudah dibagikan.

Terdapat dua opsi paralelisasi data pada PyTorch, yaitu dengan menggunakan DataParallel dan DistributedDataParallel (DDP). DataParallel relatif lebih mudah untuk digunakan jika dibandingkan dengan DDP, namun kinerja yang diberikan relatif tidak terlalu baik. Hal ini disebabkan oleh untuk setiap *forward pass*, DataParallel akan melakukan replikasi pada model dan dapat mengakibatkan Global Interpreter Lock (GIL) oleh karena prinsip *single-process multi-thread* yang dimiliki Python. Sedangkan DistributedDataParallel memiliki kinerja yang lebih baik dikarenakan model akan didistribusikan pada saat objek DDP dibuat dan bebas dari GIL dikarenakan penggunaan prinsip *multi-process parallelism*.

II.4 AdaptDL

AdaptDL merupakan kerangka kerja *open-source* yang bersifat adaptif terhadap sumber daya pada proses pembelajaran *deep learning*. AdaptDL dikembangkan

oleh Petuum dan diumumkan pada tahun 2020. AdaptDL memiliki tujuan untuk membuat *deep learning* terdistribusi menjadi mudah dan efisien pada lingkungan yang memiliki sumber daya dinamis seperti pada komputasi awan dan klaster bersama (AdaptDL, 2020). Dengan menggunakan AdaptDL, dapat ditentukan berapa banyak sumber daya yang optimal untuk suatu Job dengan cara menambahkan dan mengurangi sumber daya secara dinamis. Fitur utama dari AdaptDL adalah melakukan *scheduling* secara elastis untuk Job *deep learning* terdistribusi pada klaster bersama, membuat ukuran *batch* pada saat pembelajaran *deep learning* menjadi adaptif, dan melakukan *tuning* secara *hyper-parameter* terhadap *learning rate* pembelajaran.

Pembelajaran *deep learning* dengan menggunakan AdaptDL dilakukan dengan menggunakan AdaptDL *command line interface* pada kode pembbelajaran untuk menciptakan AdaptDL Job pada klaster Kubernetes. Setelah Job tercipta, maka *scheduler* milik AdaptDL akan melakukan penjadwalan terhadap Job untuk kemudian dieksekusi. AdaptDL *scheduler* akan membangkitkan *worker* Pod untuk Job yang akan dieksekusi. Saat status dari Pod sudah *running*, maka dapat dilakukan pengawasan terhadap kode pembelajaran yang berjalan dalam *worker* Pod dengan cara melihat *log* dari Pod tersebut. Ketika status dari Pod sudah *completed*, maka proses pelatihan *deep learning* pada Pod tersebut sudah selesai dieksekusi dan dapat model hasil pelatihan dapat diperoleh.

AdaptDL memiliki dua komponen yang dapat digunakan baik secara terpisah maupun secara bersamaan. Komponen pertama adalah suatu pustaka yang dinamakan adaptdl, digunakan sebagai kakas untuk membuat ukuran *batch* menjadi adaptif dan dapat di-*scale* secara terdistribusi ke banyak *worker node*. Sedangkan komponen kedua dinamakan adaptdl-sched dan digunakan sebagai *cluster scheduler* pada klaster Kubernetes untuk mengoptimasi sumber daya yang tersedia pada klaster.

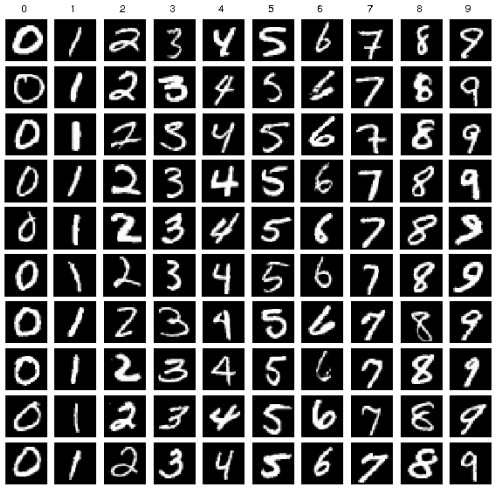
II.5 Dataset MNIST

Dataset Modified National Institute of Standards and Technology (MNIST) merupakan koleksi data citra angka yang ditulis dengan menggunakan tangan. *Dataset* MNIST sendiri merupakan modifikasi dan pengembangan lebih lanjut dari *dataset* NIST. Koleksi data yang dimiliki oleh *dataset* MNIST berjumlah 60.000 citra untuk pelatihan dan 10.000 citra untuk pengujian, dengan jumlah distribusi citra untuk pelatihan dan pengujian yang sama (Deng, 2012). Setiap citra yang ada pada *dataset* MNIST memiliki warna hitam dan putih, dengan tulisan angka yang terletak pada tengah citra. Masing-masing citra memiliki ukuran 28 x 28-*binary pixel* dengan total ukuran dimensi 784-*binary pixel*.

Pada tugas akhir ini, *dataset* MNIST digunakan sebagai data untuk melakukan pelatihan dan pengujian solusi yang telah dikembangkan. *Dataset* MNIST digunakan karena koleksi yang dimiliki sudah seragam sehingga tidak perlu lagi dilakukan data *preprocessing* dan data *formatting* sebelum pembelajaran dimulai untuk mendapatkan hasil klasifikasi yang maksimal. Contoh data yang dimiliki oleh *dataset* MNIST dapat dilihat pada Gambar II.6.1.

II.6 Penelitian Terkait

Terdapat penelitian terkait dengan tugas akhir ini, yaitu penelitian tentang Pollux, *scheduler* untuk klaster yang adaptif dengan menggunakan metrik kinerja Goodput.



Gambar II.6.1 Contoh Data pada *Dataset* MNIST (Lim dkk., 2017)

II.6.1 Pollux: *Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning*

Pollux adalah *scheduler* untuk pembelajaran *deep learning* yang dapat melakukan alokasi sumber daya secara adaptif dan secara bersamaan melakukan *tuning* untuk setiap tugas pelatihan sehingga sumber daya yang diberikan dapat dimanfaatkan dengan maksimal (Qiao dkk., 2020). Dalam melakukan *scheduling*, Pollux melakukan adaptasi pada dua tingkat granularitas, yaitu pada tingkat Job dan pada tingkat klaster. Pada tingkat granularitas Job, Pollux secara dinamis melakukan *tuning* terhadap ukuran *batch* dan *learning rate* untuk mendapatkan hasil yang maksimal dengan sumber daya yang sudah dialokasikan. Kemudian pada tingkat granularitas klaster, Pollux secara dinamis melakukan realokasi terhadap sumber daya berdasarkan perhitungan prediksi kinerja dari semua Job yang ada pada klaster. Untuk mencapai tingkat adaptif yang *scalable*, Pollux memanfaatkan dua komponen yaitu PolluxAgent dan PolluxSched. Arsitektur Pollux dibandingkan dengan sistem *scheduler* lain dapat dilihat pada Gambar II.6.2.

II.6.1.1 Goodput untuk Pembelajaran Deep Learning

Goodput merupakan metrik kinerja yang digunakan oleh Pollux dan digunakan untuk memprediksi kinerja suatu Job pada pembelajaran *deep learning*. Nilai perhitungan Goodput kemudian digunakan untuk melakukan optimasi alokasi sumber daya dan ukuran *batch*. Nilai Goodput dari sebuah Job *deep learning* adalah hasil dari perkalian *system throughput* dengan *statistical efficiency* untuk Job tersebut. Persamaan nilai Goodput ditunjukkan pada persamaan (II-2) dengan deskripsi notasi persamaan tercantum pada Tabel II.6.1.

$$GOODPUT_t(a,m) = THROUGHPUT(a,m) \times EFFICIENCY_t(m) \qquad \text{(II-2)}$$

Pollux akan memulai setiap Job *deep learning* dengan menggunakan sebuah GPU, ukuran *batch* m_0, dan *learning rate* awal η_0. Dalam prosesnya, Pollux akan mempelajari kinerja dari Job dan menggunakan hasilnya untuk mengembangkan model untuk prediksi *throughput* dan *efficiency*. Dengan menggunakan model prediksi tersebut, Pollux secara periodik akan melakukan *tuning* terhadap alokasi

Tabel II.6.1 Deskripsi Notasi Persamaan Goodput

Notasi	Deskripsi
$GOODPUT_t(a,m)$	Nilai Goodput Job pada iterasi ke-*t* dengan alokasi sumber daya *a* dan ukuran *batch m*
$THROUGHPUT(a,m)$	Nilai *system throughput* Job dengan alokasi sumber daya *a* dan ukuran *batch m*
$EFFICIENCY_t(m)$	Nilai *statistical efficiency* Job dengan ukuran *batch m*

sumber daya dan ukuran *batch* untuk setiap Job berdasarkan sumber daya yang tersedia secara *cluster-wide* dan kinerja dari Job tersebut.

II.6.1.1.1 Pemodelan *Statistical Efficiency*

Nilai *statistical efficiency* dari sebuah Job adalah tingkat kemajuan yang dapat dibuat pada *single update* menggunakan ukuran *batch* m, relatif terhadap tingkat kemajuan Job *deep learning* dengan ukuran *batch* awal m_0. Persamaan nilai *statistical efficiency* ditunjukkan pada persamaan (II-3) dan (II-4) dengan deskripsi notasi persamaan tercantum pada Tabel II.6.2.

$$EFFICIENCY_t(m) = r_t m_0/m = (\varphi_t + m_0) / (\varphi_t + m) \quad \text{(II-3)}$$

$$\varphi_t = m_0 \sigma_t^2 / \mu_t^2 \quad \text{(II-4)}$$

Nilai $EFFICIENCY_t(m)$ merupakan nilai kontribusi dari setiap *training example* terhadap keseluruhan proses pembelajaran. Nilai tersebut memiliki nilai $0 < EFFICIENCY_t(m) \leq 1$ dan menunjukkan bahwa pembelajaran dengan menggunakan ukuran *batch m* harus memproses $1/EFFICIENCY_t(m)$ kali *training examples* untuk mencapai nilai *efficiency* sebesar pembelajaran dengan menggunakan ukuran *batch* m_0.

Tabel II.6.2 Deskripsi Notasi Persamaan Statistical Efficiency

Notasi	Deskripsi
$EFFICIENCY_t(m)$	Nilai *statistical efficiency* untuk Job pada iterasi ke-*t* dengan ukuran *batch m*
φ_t	*Gradient noise scale*, merepresentasikan tingkat kemajuan pembelajaran dalam *single update*
m_0	Ukuran *batch* awal
m	Ukuran *batch* pada proses *training*
σ_t^2	Variansi *gradient* pada iterasi ke-*t* menggunakan ukuran *batch* m_0
μ_t^2	Nilai *squared norm* dari *gradient* pada iterasi ke-*t* menggunakan ukuran *batch* m_0

II.6.1.1.2 Pemodelan *System Throughput*

Untuk memodelkan dan memprediksi nilai *system throughput* untuk Job *deep learning* yang menggunakan *data-parallel*, dibutuhkan pemodelan terhadap waktu yang dibutuhkan per iterasi untuk menghitung estimasi *local gradient* (T_{grad}) dan waktu yang dibutuhkan per iterasi untuk menghitung nilai rata-rata estimasi *gradient* sekaligus sinkronisasi parameter model untuk seluruh *resources* (T_{sync}).

Nilai estimasi T_{grad} dihitung menggunakan *back-propagation* dimana waktu yang dibutuhkan akan linear dengan ukuran *batch* local untuk setiap proses. Persamaan untuk menghitung T_{grad} ditunjukkan pada persamaan (II-5).

$$T_{grad}(a,m) = \alpha_{grad} + \beta_{grad} \cdot m / K \qquad \text{(II-5)}$$

Nilai estimasi T_{sync} dihitung berdasarkan banyaknya alokasi *resource* yang diberikan dan banyaknya *node* yang digunakan oleh setidaknya satu replika. Persamaan untuk menghitung T_{sync} ditunjukkan pada persamaan (II-6).

$$T_{sync}(a,m) = \begin{cases} 0, & K = 1 \\ \alpha_{sync}^{local} + \beta_{sync}^{local} . (K-2), & N = 1 \; and \; K \geq 2 \\ \alpha_{sync}^{node} + \beta_{sync}^{node} . (K-2), & otherwise \end{cases} \quad \text{(II-6)}$$

Setelah didapatkan nilai estimasi T_{grad} dan T_{sync}, kedua nilai tersebut dapat digabungkan dengan memperhitungkan dua kemungkinan ekstrim yaitu tidak adanya *gradient overlapping* dengan jaringan komunikasi dan adanya *perfect gradient overlapping* dengan jaringan komunikasi. Perhitungan nilai tersebut ditunjukkan pada persamaan (II-7).

$$T_{iter}(a, m) = (T_{grad}(a,m)^{\gamma} + T_{sync}(a)^{\gamma})^{1/\gamma} \quad \text{(II-7)}$$

Terakhir, pemodelan untuk memprediksi *system throughput* dapat dihitung dengan menggunakan persamaan (II-8). Deskripsi notasi untuk persamaan (II-5), (II-6), (II-7), dan (II-8) dapat dilihat pada Tabel II.6.3.

$$Throughput(a, m) = m \,/\, Ti_{ter}(a, m) \quad \text{(II-8)}$$

II.6.1.1.3 Pemodelan *Utility*

Nilai *utility* merupakan nilai yang digunakan untuk mengukur tingkat utilitas dari *resource* yang dialokasikan pada suatu klaster. Untuk menghitung nilai *utility* dari tersebut, dibutuhkan perbandingan nilai maksimum Goodput yang dihasilkan oleh Job yang memliki alokasi GPU sebanyak 1 dengan Job yang memiliki alokasi GPU sebanyak *A*. Nilai perbandingan tersebut dinamakan nilai *speedup*. Persamaan untuk nilai *speedup* ditunjukkan oleh persamaan (II-9) dengan deskripsi persamaan pada Tabel II.6.4 dan persamaan untuk nilai *utility* ditunjukkan oleh persamaan (II-10) dengan deskripsi persamaan pada Tabel II.6.5.

$$Speedup_j(A_j) = \frac{max_m GOODPUT_j(A_j,m)}{max_m GOODPUT_j(1,m)} \quad \text{(II-9)}$$

$$Utility(A) = \frac{\sum_j Speedup_j(A_j)}{TOTAL\_GPUS} \quad \text{(II-10)}$$

Tabel II.6.3 Deskripsi Notasi Persamaan System Throughput

Notasi	Deskripsi
Throughput(a, m)	Nilai *system throughput* untuk Job dengan ukuran *batch m* dan alokasi *resource a*
$Ti_{ter}(a, m)$	Nilai realistis gabungan dari T_{grad} dan T_{sync}
T_{grad}	Waktu yang dibutuhkan per iterasi untuk menghitung estimasi *local gradient*
T_{sync}	Waktu yang dibutuhkan per iterasi untuk menghitung nilai rata-rata estimasi *gradient* sekaligus sinkronisasi parameter model untuk seluruh *resources*
γ	Nilai *learnable parameter*
α	Nilai *learnable parameter*
β	Nilai *learnable parameter*
N	Jumlah *physical nodes* yang ditempati oleh setidaknya satu replika
K	Jumlah *resource* yang dialokasikan pada suatu Job

II.6.1.2 PolluxAgent

PolluxAgent dijalankan bersama dengan setiap Job. PolluxAgent bertugas untuk menghitung nilai skala *gradient noise* dan *system throughput* untuk Job yang dijalankan, serta melakukan *tuning* pada ukuran *batch* dan *learning rate* untuk mengefisiensikan penggunaan dari sumber daya yang sudah dialokasikan. PoluxAgent secara periodik akan mengirimkan hasil dari fungsi Goodput dari Job ke PolluxSched.

II.6.1.3 PolluxSched

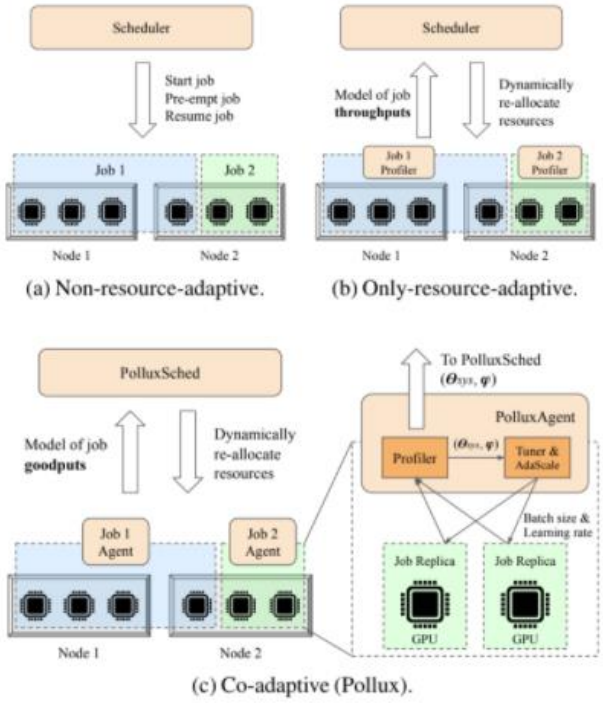
PolluxSched secara periodik melakukan optimasi alokasi sumber daya untuk Job pada klaster berdasarkan *statistical efficiency* untuk setiap Job. PolluxSched menggunakan fungsi Goodput untuk memprediksi kinerja dari Job ketika diberikan penambahan ataupun pengurangan sumber daya.

Tabel II.6.4 Deskripsi Notasi Persamaan Speedup

Notasi	Deskripsi
$Speedup_j(A_j)$	Nilai *speedup* untuk Job *j* dengan jumlah *resource* A
$max_m GOODPUT_j(A_j,m)$	Nilai Goodput maksimum untuk Job *j* dengan jumlah *resource* A dan ukuran *batch m*
$max_m GOODPUT_j(1,m)$	Nilai Goodput maksimum untuk Job *j* dengan alokasi *resource* 1 dan ukuran *batch m*

Tabel II.6.5 Deskripsi Notasi Persamaan Utility

Notasi	Deskripsi
Utility(A)	Nilai utilitas dari *resource* A
$\Sigma_j Speedup_j(A_j)$	Total nilai *speedup* untuk Job *j* dengan jumlah *resource* A
TOTAL_GPUS	Total jumlah GPU sebagai *resource* yang dimiliki
A	Matriks yang menunjukkan alokasi *resource* untuk setiap *job*



Gambar II.6.2 Arsitektur Pollux (Qiao dkk., 2020)

BAB III

ANALISIS PERMASALAHAN DAN RANCANGAN SOLUSI

Pada bab ini dijelaskan tentang deskripsi dan analisis persoalan, analisis solusi, serta rancangan solusi yang akan dikembangkan.

III.1 Analisis Permasalahan

Proses pelatihan *deep learning* merupakan pekerjaan yang membutuhkan waktu yang lama dikarenakan kompleksnya data dan lapisan *layer* yang digunakan (Hegde & Usmani, 2016). Salah satu pendekatan untuk menyelesaikan permasalahan waktu pelatihan *deep learning* adalah dengan melakukan proses pelatihan *deep learning* secara terdistribusi. Proses pelatihan *deep learning* dijalankan secara terdistribusi melalui tugas yang dikerjakan oleh mesin yang berbeda namun terhubung satu sama lain. Semakin banyak mesin yang digunakan, semakin mahal biaya yang harus dikeluarkan untuk menggunakan mesin-mesin tersebut. Hal ini menyebabkan pengguna harus mengetahui jumlah mesin yang dibutuhkan dan dapat menggunakannya mesin tersebut secara efisien. Umumnya, pengguna harus mencari konfigurasi parameter pelatihan, seperti ukuran *batch* dan nilai *learning rate*, yang tepat supaya dapat menggunakan mesin dengan efisien. Untuk mengatasi permasalahan tersebut, dibutuhkan sistem *scheduler* yang mampu melakukan penjadwalan tugas sedemikian rupa sehingga dapat mengalokasikan dan menggunakan sumber daya secara maksimal.

Salah satu sistem *scheduler* yang digunakan untuk melakukan penjadwalan Job *deep learning* adalah Pollux. Pollux (Qiao dkk., 2020) merupakan sistem *hybrid resource scheduler* yang melakukan penjadwalan Job pada klaster *deep learning* dengan cara melakukan optimasi pada level Job dan level *cluster-wide*. Pollux menggunakan metrik kinerja Goodput yang merupakan gabungan nilai *system throughput* dan *statistical efficiency* untuk melakukan prediksi terhadap kinerja Job pada proses pelatihan *deep learning*. Dengan menggunakan nilai *system throughput* dan *statistical efficiency*, Pollux dapat menghasilkan nilai yang tinggi untuk Job

yang cepat diselesaikan dan memiliki *statistical efficiency* yang tinggi. Hanya saja, penggunaan kedua nilai tersebut mengakibatkan nilai Goodput dari Job yang memiliki *system throughput* tinggi namun memiliki *statistical efficiency* rendah akan bernilai kecil dan tidak menjadi prioritas untuk dikerjakan. Hal ini dapat mengurangi potensi kecepatan dalam menyelesaikan proses pelatihan *deep learning*.

Dalam menentukan keputusan *scaling* pada suatu klaster, Pollux menggunakan pemodelan nilai *utility* yang berbasis pada nilai Goodput. Nilai *utility* tersebut dibandingkan dengan nilai ambang batas (*threshold*) untuk menentukan keputusan *scaling up* atau *scaling down* terhadap ukuran klaster. Apabila nilai *utility* bernilai tinggi, maka PolluxSched akan meminta tambahan sumber daya (*scaling up*) dan apabila nilai *utility* bernilai rendah, PolluxSched akan melepaskan sumber daya yang ada (*scaling down*). Secara *default*, nilai ambang batas bawaan Pollux adalah 0.35 untuk ambang batas minimum (*minimum threshold*) dan 0.65 untuk ambang batas maksimal (*maximum threshold*). Dengan nilai ambang batas maksimal yang cukup rendah, yaitu 0.65, dapat menyebabkan PolluxSched sering meminta tambahan sumber daya. Hal ini dapat menyebabkan ketidakefisienan penggunaan sumber daya akibat pengalokasian sumber daya yang kurang selektif.

Berdasarkan analisis permasalahan di atas, dapat disimpulkan bahwa diperlukan pengembangan lebih lanjut terhadap sistem *hybrid resource scheduler* Pollux. Sistem perlu menyediakan pelatihan *deep learning* terdistribusi dengan opsi kecepatan maksimal, opsi efisiensi sumber daya, serta opsi bawaan yang dapat disesuaikan dengan kebutuhan pengguna. Selain itu, sistem *scheduler* juga perlu menyediakan mekanisme pilihan metrik sehingga pengguna mampu melakukan pelatihan secara fleksibel pada satu sistem *scheduler* yang sama.

III.2 Analisis Solusi

Solusi yang dapat diterapkan untuk memberikan fleksibilitas pengguna dalam menggunakan metrik adalah dengan memberikan pilihan metrik sebelum proses pelatihan *deep learning* dimulai. Pilihan metrik yang disediakan adalah metrik

untuk kecepatan pelatihan maksimal, efisiensi sumber daya, dan metrik bawaan dari *scheduler*. Metrik untuk kecepatan pelatihan yang maksimal dikembangkan dengan melakukan modifikasi perhitungan metrik Goodput, sedangkan metrik untuk efisiensi sumber daya dikembangkan dengan melakukan modifikasi nilai ambang batas (*threshold*). Modifikasi dilakukan berdasarkan tujuan dari metrik baru, yaitu memaksimalkan kecepatan pelatihan dan mengefisienkan sumber daya pada proses pelatihan *deep learning*.

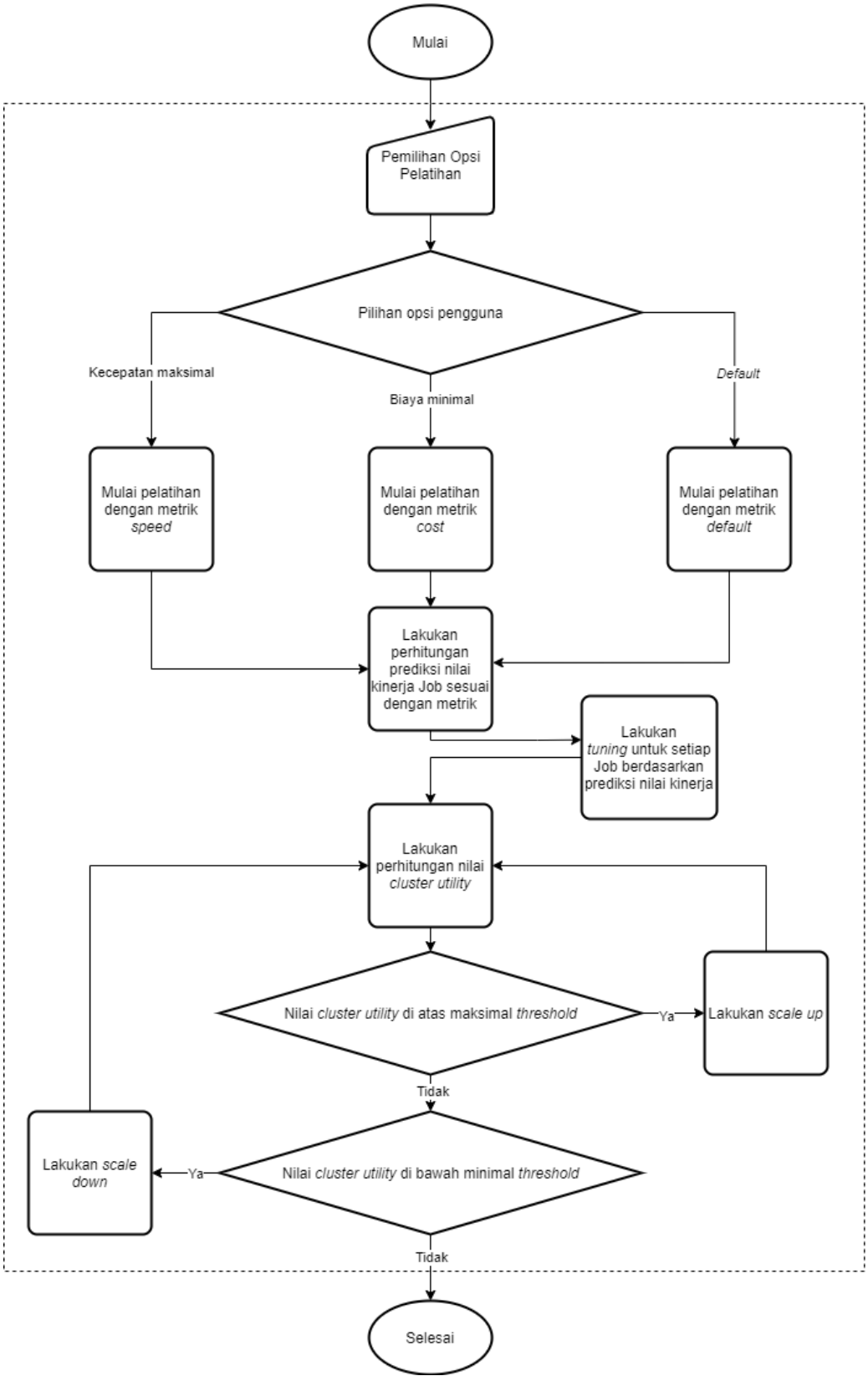
Untuk tujuan memaksimalkan kecepatan pelatihan, dibutuhkan metrik yang mampu memaksimalkan potensi kecepatan pelatihan. Kecepatan pelatihan dapat dimaksimalkan dengan cara meningkatkan ukuran *batch* sehingga proses pelatihan cepat selesai. Hanya saja, dengan menggunakan metrik kecepatan maksimal, model yang dilatih dapat memiliki akurasi yang lebih rendah dibandingkan dengan model dengan metrik *default*. Hal ini dapat terjadi karena metrik kecepatan hanya memaksimalkan ukuran *batch* saja tanpa memedulikan tingkat kegunaan data dengan ukuran *batch* tersebut.

Untuk tujuan melakukan efisiensi sumber daya, dibutuhkan metrik yang mampu meminimalkan penggunaan sumber daya pada saat pelatihan. Hal ini dapat dilakukan dengan cara mengubah batasan maksimum dari jumlah sumber daya yang dapat digunakan dan meningkatkan nilai ambang batas untuk penentuan *scaling*. Pengubahan batas maksimum jumlah sumber daya dilakukan secara manual oleh *cluster operator* dan bergantung pada seberapa banyak jumlah sumber daya yang disediakan oleh pengguna sehingga sumber daya yang digunakan tidak akan melebihi kapasitas maksimal. Kemudian, peningkatan nilai ambang batas (*threshold*) dilakukan dengan cara mengubah konstanta nilai ambang batas yang sudah tersedia. Dengan meningkatnya nilai ambang batas, alokasi sumber daya untuk penentuan *scaling* pada kluster dapat dilakukan dengan lebih selektif sesuai dengan tingkat kenaikan nilai ambang batas.

III.3 Rancangan Solusi

Solusi sistem yang dirancang terdiri dari proses pemilihan metrik kinerja, metrik untuk memaksimalkan kecepatan, dan metrik untuk mengefisienkan sumber daya. Sebelum proses pelatihan *deep learning* dimulai, pengguna dapat memilih tiga opsi pelatihan, yaitu pelatihan dengan kecepatan maksimal, pelatihan dengan efisiensi sumber daya, dan pelatihan secara *default*. Untuk pelatihan dengan kecepatan maksimal, metrik yang digunakan oleh *scheduler* adalah metrik untuk kecepatan maksimal. Untuk pelatihan dengan efisiensi sumber daya, metrik yang digunakan adalah metrik untuk efisiensi sumber daya. Dan untuk pelatihan secara bawaan, metrik yang digunakan adalah metrik *default* yaitu Goodput.

Setelah pengguna memilih, proses pelatihan akan berjalan dengan menggunakan metrik pilihan pengguna. Selama proses pelatihan berjalan, kerangka kerja AdaptDL akan menghasilkan Job yang kemudian dijadwalkan pengerjaannya oleh *scheduler* Pollux. Untuk setiap Job yang dihasilkan oleh AdaptDL, *scheduler* Pollux akan menghitung nilai Goodput dari Job tersebut. Nilai Goodput dari Job digunakan untuk melakukan utilisasi ukuran *batch* dan *learning rate* sehingga Job dapat memaksimalkan sumber daya yang diberikan. Selain itu, nilai Goodput dari seluruh Job juga digunakan untuk menghitung nilai utilitas klaster. Jika nilai utilitas klaster berada di bawah nilai ambang batas minimum, maka Pollux akan mengurangi jumlah sumber daya yang ada. Jika nilai utilitas klaster berada di atas nilai ambang batas maksimum, maka Pollux akan menambahi jumlah sumber daya. Dan jika nilai utilitas klaster berada di antara nilai ambang batas minimum dan nilai ambang batas maksimum, maka Pollux tidak melakukan baik penambahan maupun pengurangan sumber daya. Penambahan dan pengurangan sumber daya terus berjalan hingga nilai utilitas klaster sudah berada di antara nilai ambang batas maksimum dan minimum. Proses ini berjalan pada setiap *epoch* dan berakhir apabila semua *epoch* telah selesai. Diagram alir dari rancangan solusi sebagai solusi tugas akhir dapat dilihat pada Gambar III.3.1.



Gambar III.3.1 Diagram Alir Rancangan Solusi

III.3.1 Pemilihan Metrik Kinerja

Mekanisme pemilihan metrik kinerja dilakukan sebelum pelatihan *deep learning* dengan menggunakan *flag* pada perintah *command line argument* (CLI). Pilihan metrik yang dapat dipilih adalah *speed* untuk kecepatan, *cost* untuk efisiensi sumber daya, dan *default*.

III.3.2 Metrik untuk Memaksimalkan Kecepatan

Metrik untuk memaksimalkan kecepatan merupakan pengembangan dari metrik Goodput. Perhitungan metrik Goodput dimodifikasi sehingga hanya mempertimbangkan nilai *system throughput* saja tanpa mempertimbangkan nilai *statistical efficiency* dari Job. Dengan demikian, perhitungan nilai Goodput hanya bergantung pada nilai *system throughput* saja sehingga bisa menghasilkan nilai *throughput* yang tinggi, walaupun nilai tersebut tidak memiliki tingkat kegunaan yang tinggi terhadap proses pelatihan. Dengan nilai *throughput* yang tinggi, dapat dilakukan *tuning* terhadap ukuran *batch* sehingga ukuran *batch* menjadi lebih tinggi. Selain itu, sistem juga tidak perlu menghitung nilai *statistical efficiency* dari Job sehingga mempercepat proses pembelajaran.

III.3.3 Metrik untuk Mengefisienkan Sumber Daya

Metrik untuk mengefisienkan sumber daya merupakan pengembangan dari mekanisme *scaling up* dan *scaling down* pada sistem *scheduler* Pollux. *Scaling up* dilakukan dengan melakukan penambahan *worker node* pada *cluster* apabila nilai *cluster utility* berada di atas nilai ambang batas maksimum. Sedangkan apabila nilai *cluster utility* berada di bawah nilai ambang batas minimum, *scaling down* akan dilakukan dengan melakukan pengurangan *worker node* pada *cluster*. *Scaling* sumber daya pada *cluster* akan terus dilakukan hingga nilai prediksi *cluster utility* berada di atas nilai ambang batas minimum dan di bawah nilai ambang batas maksimum. Pengembangan mekanisme *scaling* sumber daya dilakukan dengan meningkatkan nilai ambang batas (*threshold*) untuk penentuan keputusan *scaling*. Dengan meningkatnya nilai *threshold,* proses *scaling up* hanya akan dilakukan ketika nilai *cluster utility* sangat tinggi.

BAB IV

IMPLEMENTASI

Pada bab ini dijelaskan tentang implementasi dari rancangan solusi berdasarkan analisis dan rancangan solusi pada Bab III.

IV.1 Implementasi Pemilihan Metrik Kinerja

Pemilihan metrik kinerja diimplementasikan sebagai konfigurasi pembelajaran dan dapat diubah oleh pengguna melalui argumen pada *command line interface* (CLI). Dengan memanfaatkan kakas *argument parser* yang dimiliki oleh bahasa Python, pengguna dapat memilih metrik yang akan digunakan dalam proses pembelajaran *deep learning*. Pilihan yang dapat dipilih adalah *speed* dan *cost*, serta *default* apabila pengguna tidak memberikan pilihan metrik secara spesifik. Metrik yang dipilih akan menentukan konfigurasi kakas dari awal pembelajaran hingga akhir. *Pseudocode* untuk implementasi pemilihan kinerja pada saat inisiasi pembelajaran dapat dilihat pada Kode IV.1.1.

```
USERINPUT
     Arguments A;

1.  Initialize args = parse A
2.  Initialize metrics_options = metric from args
3.  Initialize kwargs = empty set
4.
5.  if metrics_options <> null do
6.      add metrics_options to kwargs
7.  else do
8.      add default to kwargs
9.  end if
10.
11. Initialize train_dataset = MNIST train dataset
12. Initialize test_dataset = MNIST test dataset
13. run training and test function using train_dataset and
    kwargs
```

Kode IV.1.1. *Pseudocode* Pemilihan Metrik Kinerja (inisiasi)

Pada saat fungsi *training* dan *test* dijalankan, fungsi tersebut akan memanggil fungsi lain dengan menggunakan opsi metrik dari masukan pengguna sebagai parameter. Fungsi yang dipanggil adalah fungsi untuk melakukan evaluasi nilai Goodput dan fungsi konstruktor untuk kelas PolluxPolicy. *Pseudocode* untuk fungsi evaluasi nilai Goodput dapat dilihat pada Kode IV.1.2 dan fungsi konstruktor PolluxPolicy dapat dilihat pada Kode IV.1.3. Kode gabungan untuk inisiasi, fungsi evaluasi nilai Goodput, dan fungsi konstruktor PolluxPolicy dapat dilihat pada Lampiran A.

```
INPUT
    Metric option M;

1.  if M = "speed" do
2.      Initialize throughput_result = call throughput function
3.  else do
4.      Initialize throughput_result = call throughput function
5.      Initialize efficiency_result = call efficiency function
6.      Calculate throughput result * efficiency result
7.  end if
```

Kode IV.1.2. *Pseudocode* Pemilihan Metrik Kinerja (Goodput)

```
INPUT
    Metric option M;

1.  Initialize min_util_threshold = 0.35
2.
3.  if M = "cost" do
4.      Initialize max_util_threshold = 0.8
5.  else do
6.      Initialize max_util_threshold = 0.65
7.  end if
```

Kode IV.1.3. *Pseudocode* Pemilihan Metrik Kinerja (PolluxPolicy)

IV.2 Implementasi Metrik untuk Memaksimalkan Kecepatan Pelatihan

Metrik untuk memaksimalkan kecepatan pelatihan diimplementasikan dengan melakukan modifikasi dari metrik Goodput. Sebelumnya, metrik Goodput ditunjukkan pada persamaan (IV-1) dengan deskripsi notasi pada Tabel IV.2.1. Modifikasi dilakukan dengan menghilangkan nilai *statistical efficiency* dan menyisakan nilai *system throughput* pada persamaan nilai Goodput, dapat dilihat pada persamaan (IV-2) dan deskripsi notasi pada Tabel IV.2.2.

$$GOODPUT_t(a,m) = THROUGHPUT(a,m) \times EFFICIENCY_t(m) \quad \text{(IV-1)}$$

$$GOODPUT_t(a,m) = THROUGHPUT(a,m) \quad \text{(IV-2)}$$

Dengan menggunakan persamaan (IV-2), nilai Goodput akan bernilai linear dengan nilai *system throughput*. Apabila nilai *system throughput* tinggi, maka nilai Goodput juga tinggi dan sebaliknya. *Pseudocode* untuk implementasi metrik kecepatan dapat dilihat pada Kode IV.2.1. Kode selengkapnya tercantum pada Lampiran B.

```
INPUT
    Metric option M, number of nodes N, number of replicas R,
    atomic batch size A, accumulation steps AS;
OUTPUT
    Evaluation result ER;

1.  Initialize result = 0
2.  Initialize batch_size = R * A * (AS + 1)
3.
4.  if M = "speed" do
5.      result = call throughput function using N, R, A, and AS
6.  else do
7.      result = (call throughput function using N, R, A, and
    AS) * (call efficiency function using batch_size)
8.  end if
9.
10. Initialize ER = result
11. return ER
```

Kode IV.2.1. *Pseudocode* Implementasi Metrik Kecepatan

Tabel IV.2.1. Deskripsi Notasi Persamaan Awal Goodput

Notasi	Deskripsi
$GOODPUT_t(a,m)$	Nilai Goodput Job pada iterasi ke-*t* dengan alokasi sumber daya *a* dan ukuran *batch m*
$THROUGHPUT(a,m)$	Nilai *system throughput* Job dengan alokasi sumber daya *a* dan ukuran *batch m*
$EFFICIENCY_t(m)$	Nilai *statistical efficiency* Job dengan ukuran *batch m*

Tabel IV.2.2. Deskripsi Notasi Persamaan Modifikasi Goodput

Notasi	Deskripsi
$GOODPUT_t(a,m)$	Nilai Goodput Job pada iterasi ke-*t* dengan alokasi sumber daya *a* dan ukuran *batch m*
$THROUGHPUT(a,m)$	Nilai *system throughput* Job dengan alokasi sumber daya *a* dan ukuran *batch m*

IV.3 Implementasi Metrik untuk Mengefisienkan Sumber Daya

Metrik untuk mengefisienkan sumber daya merupakan pengembangan dari mekanisme penentuan *scaling up* dan *scaling down* pada sistem *scheduler* Pollux. Pengembangan mekanisme dilakukan dengan cara mengubah nilai ambang batas (*threshold*), sehingga memengaruhi keputusan *scaling up* dan *scaling down* pada saat proses perbandingan nilai ambang batas dengan nilai *cluster utility*. Nilai ambang batas bawaan (*default*) dari Pollux sendiri bernilai 0,35 untuk ambang batas minimum dan 0,65 untuk ambang batas maksimum. Pada tugas akhir ini, nilai ambang batas maksimum dinaikkan menjadi 0,8, sedangkan nilai ambang batas minimum tetap. Nilai tersebut dipilih dikarenakan memiliki perbedaan yang cukup tinggi dengan nilai sebelumnya. *Pseudocode* untuk implementasi metrik efisiensi dapat dilihat pada Kode IV.3.1. Kode selengkapnya tercantum pada Lampiran C.

```
INPUT
     Metric option M, utilities U, values V, nodes N
OUTPUT
     Best nodes BN

1.   Initialize min_util_threshold = 0.35
2.
3.   if M = "cost" do
4.       Initialize max_util_threshold = 0.8
5.   else
6.       Initialize max_util_threshold = 0.65
7.   end if
8.
9.   Initialize idx = call select_result function using V and
     length N
10.
11.  if idx <> null and min_util_threshold <= utilities[idx] <=
     max_util_threshold do
12.      return length of N
13.  end if
14.
15.  Initialize target_util = (max_util_threshold +
     min_util_threshold) / 2
16.  Initialize best_util = infinity
17.  Initialize best_nodes = length of N
18.
19.  for util, (_, num_nodes) in (U, V) do
20.      if util < min_util_threshold:
21.          continue
22.      end if
23.
24.       if util is close with best_util and num_nodes >
          best_nodes do
25.          best_nodes = num_nodes
26.      end if
27.
28.       if absolute of (util - target_util) < absolute of
          (best_util - target_util) do
29.          best_util = util
30.          best_nodes = num_nodes
31.      end if
32.  end for
33.
34.  Initialize BN = best_nodes
35.  return BN
```

Kode IV.3.1. *Pseudocode* Implementasi Metrik Efisiensi

BAB V

PENGUJIAN

Pada bab ini dijelaskan tentang pengujian untuk solusi yang diimplementasikan pada Bab IV. Bab ini terdiri dari tujuan pengujian, lingkungan pengujian, metode pengujian, skenario pengujian, serta hasil pengujian.

V.1 Tujuan Pengujian

Pengujian dilakukan untuk menentukan pengaruh solusi terhadap kinerja pembelajaran *deep learning*. Untuk setiap solusi yang dikembangkan, terdapat aspek yang diuji dan hasilnya akan dibandingkan dengan hasil dari sistem pada kondisi standar. Hasil perbandingan tersebut kemudian akan dianalisis dan dijadikan landasan untuk mengetahui pengaruh dari solusi terhadap sistem pada kondisi standar. Pengujian dilakukan dengan menggunakan beberapa skenario sehingga pengaruh solusi dapat ditentukan dengan tepat.

V.2 Lingkungan Pengujian

Pengujian dilakukan pada *platform* Amazon Elastic Kubernetes Service (EKS) dengan spesifikasi yang tertera pada Tabel V.2.1. Spesifikasi Lingkungan Pengujian. *Worker node* yang digunakan merupakan *instance* dari Amazon Elastic Computing Cloud (EC2). Jumlah *worker node* yang digunakan selama pengujian berjumlah dua buah agar mekanisme *scaling* dari solusi dapat diuji. Versi Kubernetes yang digunakan adalah versi 1.18.0 yang merupakan versi Kubernetes bawaan dari Amazon EKS. Kode untuk membangkitkan *cluster* sebagai tempat pengujian dapat dilihat pada Lampiran D.

Tabel V.2.1. Spesifikasi Lingkungan Pengujian

Jenis	Nilai
Jenis *instance*	Amazon EKS dan Amazon EC2 g4dn.xlarge
Jumlah *cluster*	Satu (1)
Versi Kubernetes	1.18.0
Jumlah *node*	Dua (2)
Sistem Operasi pada *node*	Amazon Linux 2 AMI
CPU per *node*	Empat (4) vCPU
Memory per *node*	16 GB
Jumlah *node* awal	Satu (1)

V.3 Metode Pengujian

Pada solusi pemilihan metrik kinerja dan metrik kecepatan, pengukuran dilakukan terhadap kecepatan pelatihan dan akurasi model yang dihasilkan. Sedangkan pada metrik efisiensi, pengukuran dilakukan terhadap selisih waktu pembangkitan berdasarkan waktu hidup *worker nodes*, kecepatan pelatihan, dan akurasi model yang dihasilkan.

Untuk setiap pengujian solusi, hasil yang diperoleh akan dibandingkan dengan hasil kerangka kerja AdaptDL yang menggunakan *scheduler* Pollux tanpa adanya modifikasi. Hal ini bertujuan untuk membandingkan pengaruh modifikasi terhadap kondisi awal dari kerangka kerja dan *scheduler* yang menjadi *baseline* dari tugas akhir.

Proses pengujian dilakukan dengan menjalankan beberapa Job secara bersamaan pada *cluster* Kubernetes. Job yang dijalankan pada sistem merupakan Job pembelajaran *deep learning* dengan tujuan klasifikasi pengenalan citra pada *dataset* MNIST. Terdapat beberapa skenario pengujian yang dibuat untuk proses pengujian, dengan setiap skenario memiliki konfigurasi yang berbeda.

Tabel V.3.1. Arsitektur Model Pengujian

Layer	Ukuran Layer
ConvNet (2D)	1 * 32 * 3 * 1
ConvNet (2D)	32 * 64 * 3 * 1
Dropout (0.25)	1
Dropout (0.25)	1
Fully Connected (Linear)	9216 * 128
Fully Connected (Linear)	128 * 10

Konfigurasi yang diubah untuk setiap skenario adalah jenis metrik yang akan dibandingkan. Sedangkan konfigurasi yang tetap untuk setiap skenario adalah nilai *learning rate* awal dan arsitektur model pembelajaran. Nilai *learning rate* awal yang digunakan adalah nilai *default* yaitu 0,7. Arsitektur model yang dipilih merupakan arsitektur *default* untuk klasifikasi pengenalan gambar pada *dataset* MNIST yang disediakan oleh kerangka kerja AdaptDL. Arsitektur model dapat dilihat pada Tabel V.3.1.

V.4 Skenario Pengujian

Terdapat tiga skenario pengujian yang dilakukan, yaitu skenario untuk menguji solusi pemilihan metrik kinerja, skenario untuk menguji metrik untuk memaksimalkan kecepatan, dan skenario untuk menguji metrik untuk mengefisienkan sumber daya Untuk setiap skenario pengujian, terdapat lima Job yang dijalankan secara bersamaan pada *cluster* Kubernetes. Terdapat dua Job yang memiliki jumlah *epoch* rendah dan ukuran *batch* bawaan, dua Job yang memiliki jumlah *epoch* tinggi dan ukuran *batch* bawaan, dan satu Job yang memiliki jumlah *epoch* tinggi dan ukuran *batch* tinggi. Konfigurasi Job untuk pengujian dapat dilihat pada Tabel V.4.1.

Tabel V.4.1. Konfigurasi Job untuk Skenario Pengujian

Job	**Jumlah *Epoch***	**Ukuran *Batch***
Job 1	10	64
Job 2	15	64
Job 3	40	64
Job 4	45	64
Job 5	45	128

V.4.1 Skenario Pengujian Pemilihan Metrik Kinerja

Pada proses pengujian pemilihan metrik kinerja, dilakukan skenario pengujian yang bertujuan untuk menentukan pengaruh modifikasi kerangka kerja AdaptDL terhadap kecepatan pelatihan dan akurasi model. Pilihan metrik yang digunakan untuk pengujian pemilihan metrik adalah metrik *default*.

V.4.2 Skenario Pengujian Metrik untuk Memaksimalkan Kecepatan

Pada proses pengujian metrik untuk memaksimalkan kecepatan, dilakukan skenario pengujian yang bertujuan untuk menentukan pengaruh metrik kecepatan terhadap kecepatan proses pelatihan dan akurasi model yang dihasilkan. Proses pembelajaran dilakukan menggunakan dua metrik untuk dibandingkan, yaitu metrik *default* dan metrik *speed*.

V.4.3 Skenario Pengujian Metrik untuk Mengefisienkan Sumber Daya

Pada proses pengujian metrik untuk mengefisienkan sumber daya, dilakukan skenario pengujian yang bertujuan untuk menentukan pengaruh metrik efisiensi terhadap penggunaan *resources* sistem dalam melakukan proses pembelajaran. Aspek yang diuji adalah selisih waktu pembangkitan berdasarkan waktu hidup *worker nodes*, akurasi model, dan waktu pelatihan. Proses pembelajaran dilakukan menggunakan dua metrik untuk dibandingkan, yaitu metrik *default* dan metrik *cost*.

V.5 Hasil Pengujian

Pada subbab ini dijelaskan mengenai hasil pengujian dari setiap skenario yang sudah dijelaskan pada subbab sebelumnya.

V.5.1 Hasil Pengujian Pemilihan Metrik Kinerja

Hasil pengujian pemilihan metrik kinerja menunjukkan kerangka kerja AdaptDL tanpa modifikasi memiliki kecepatan pelatihan dan akurasi model yang relatif sama dibandingkan dengan kerangka kerja AdaptDL dengan modifikasi. Rincian hasil pengujian kecepatan dan akurasi model dapat dilihat pada Tabel V.5.1 dan Tabel V.5.2.

V.5.2 Hasil Pengujian Metrik untuk Memaksimalkan Kecepatan

Hasil pengujian metrik untuk memaksimalkan kecepatan menunjukkan *scheduler* yang menggunakan metrik kecepatan memiliki kecepatan pelatihan yang relatif lebih lambat dan akurasi model yang relatif sama dibandingkan dengan *scheduler* yang menggunakan metrik *default*. Rincian hasil pengujian kecepatan dan akurasi model dapat dilihat pada Tabel V.5.3 dan Tabel V.5.4.

V.5.3 Hasil Pengujian Metrik untuk Mengefisienkan Sumber Daya

Hasil pengujian metrik untuk mengefisienkan sumber daya menunjukkan *scheduler* dengan menggunakan metrik *cost* memiliki selisih waktu pembangkitan *worker nodes* yang relatif lebih tinggi, akurasi model yang relatif sama, dan kecepatan pelatihan yang relatif lebih lambat dibandingkan dengan *scheduler* yang menggunakan metrik *default*. Rincian hasil pengujian kecepatan dan akurasi model dapat dilihat pada Tabel V.5.5 dan Tabel V.5.6. Rincian hasil pengujian waktu pembangkitan *worker nodes* dapat dilihat pada Tabel V.5.7 dan Tabel V.5.8.

Tabel V.5.1. Hasil Pengujian 1 dengan AdaptDL

Kategori	Waktu yang Dibutuhkan (detik)	Akurasi Model (%)
Job 1	368,481	99,15
Job 2	124,483	99,21
Job 3	743,627	99,2
Job 4	646,017	99,17
Job 5	312,535	99,18
Mean	439,029 ± 257,755	99,182 ± 0,024

Tabel V.5.2. Hasil Pengujian 1 dengan AdaptDL Modifikasi

Kategori	Waktu yang Dibutuhkan (detik)	Akurasi Model (%)
Job 1	168,471	99,16
Job 2	268,42	99,2
Job 3	472,3	99,17
Job 4	761,12	99,16
Job 5	457,68	99,19
Mean	425,598 ± 227,2689	99,176 ± 0,018

Tabel V.5.3. Hasil Pengujian 2 dengan Metrik *Default*

Kategori	**Waktu yang Dibutuhkan (detik)**	**Akurasi Model (%)**
Job 1	368,481	99,15
Job 2	124,483	99,21
Job 3	743,627	99,2
Job 4	646,017	99,17
Job 5	312,535	99,18
Mean	439,029 ± 257,755	99,182 ± 0,024

Tabel V.5.4. Hasil Pengujian 2 dengan Metrik Kecepatan

Kategori	**Waktu yang Dibutuhkan (detik)**	**Akurasi Model (%)**
Job 1	226,115	99,17
Job 2	106,022	99,1
Job 3	472,3	99,11
Job 4	1168,12	99,11
Job 5	575,98	99,18
Mean	509,707 ± 413,124	99,134 ± 0,038

Tabel V.5.5. Hasil Pengujian 3 dengan Metrik *Default*

Kategori	Waktu yang Dibutuhkan (detik)	Akurasi Model (%)
Job 1	368,481	99,15
Job 2	124,483	99,21
Job 3	743,627	99,2
Job 4	646,017	99,17
Job 5	312,535	99,18
Mean	439,029 ± 257,755	99,182 ± 0,024

Tabel V.5.6. Hasil Pengujian 3 dengan Metrik Efisiensi

Kategori	Waktu yang Dibutuhkan (detik)	Akurasi Model (%)
Job 1	284,303	99,13
Job 2	908,11	99,17
Job 3	451,8	99,12
Job 4	184,57	99,13
Job 5	2694,9	99,21
Mean	904,7366 ± 1038,439	99.152 ± 0,037

Tabel V.5.7. Hasil Pengujian 3 Waktu Hidup dengan Metrik *Default*

Worker Node	**Waktu Hidup (menit)**
Node 1	75
Node 2	70
Mean	72,5 ± 3,535

Tabel V.5.8. Hasil Pengujian 3 Waktu Hidup dengan Metrik Efisiensi

Worker Node	**Waktu Hidup (menit)**
Node 1	91
Node 2	82
Mean	86,5 ± 6,364

V.6 Analisis Hasil Pengujian

V.6.1 Analisis Hasil Pengujian Pemilihan Metrik Kinerja

Rangkuman hasil pengujian untuk pengujian pemilihan metrik kinerja dapat dilihat pada Tabel V.6.1 untuk kecepatan proses pelatihan dan Tabel V.6.2 untuk akurasi model. Persentase selisih merupakan nilai perbandingan antara hasil kerangka kerja AdaptDL tanpa modifikasi terhadap hasil kerangka kerja AdaptDL dengan modifikasi. Untuk setiap tabel, terdapat ilustrasi dengan grafik yang dapat dilihat pada Gambar V.6.1 untuk grafik kecepatan proses pelatihan dan Gambar V.6.2 untuk grafik akurasi model. Pada grafik kecepatan, nilai yang rendah menandakan hasil yang lebih baik. Sedangkan pada grafik akurasi model, hasil yang lebih baik ditandai dengan nilai yang tinggi.

Berdasarkan hasil pengujian untuk skenario pengujian pemilihan metrik kinerja, terlihat bahwa kerangka kerja AdaptDL modifikasi memiliki hasil pengujian yang bervariasi untuk setiap Job. Namun, secara keseluruhan modifikasi tidak memengaruhi kecepatan proses pelatihan. Hal ini ditunjukkan dengan selisih rata-rata kecepatan pelatihan yang hanya berbeda 3,06%. Selain itu, modifikasi juga tidak memengaruhi akurasi model yang dihasilkan. Hal ini ditunjukkan dengan selisih rata-rata akurasi model yang hanya 0,006%.

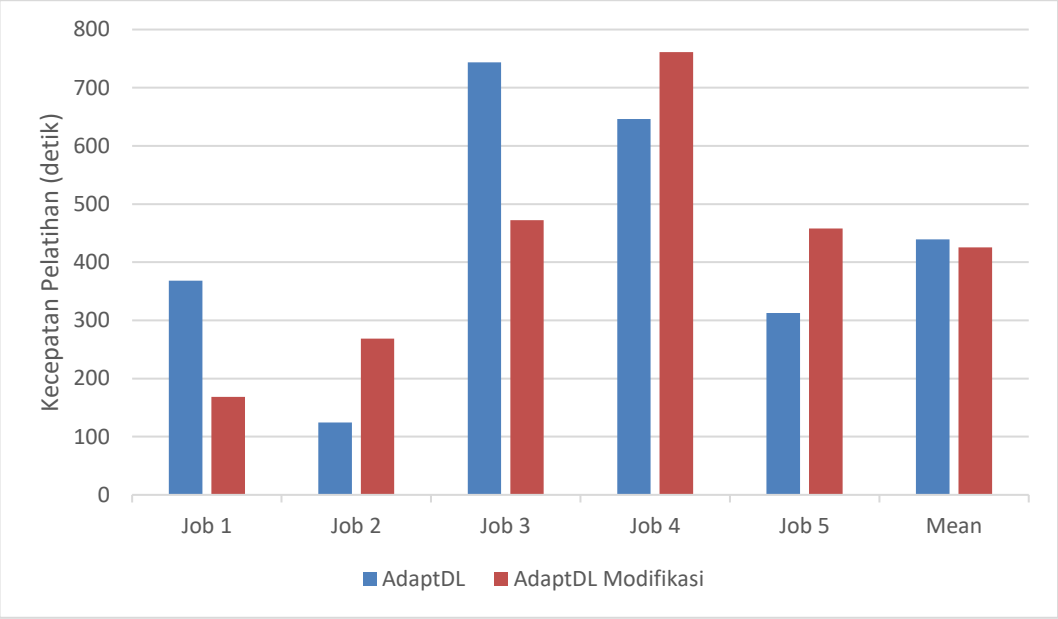
Pada solusi pemilihan metrik kinerja, implementasi yang dilakukan tidak berdampak secara langsung pada proses pembelajaran. Masukan metrik yang diberikan oleh pengguna hanya dijadikan sebagai parameter pada fungsi untuk menentukan konfigurasi pembelajaran. Apabila konfigurasi pembelajaran pada kerangka kerja modifikasi dibuat sama dengan kerangka kerja tanpa modifikasi, maka proses pembelajaran tidak akan berbeda. Oleh karena itu, nilai kecepatan proses pelatihan dan akurasi model pada kerangka kerja modifikasi relatif sama dengan kerangka kerja tanpa modifikasi.

Tabel V.6.1. Rangkuman Hasil Pengujian Kecepatan pada Pemilihan Metrik

Kategori	AdaptDL (detik)	AdaptDL Modifikasi (detik)	Selisih Kecepatan (detik)
Job 1	368,481	168,471	200,01
Job 2	124,483	268,42	-143,937
Job 3	743,627	472,3	271,327
Job 4	646,017	761,12	-115,103
Job 5	312,535	457,68	-145,145
Mean	439,029	425,598	13,4304

Persentase Selisih Kecepatan

$$\frac{selisih\ rata - rata\ kecepatan}{rata - rata\ AdaptDL} * 100\% = 3{,}06\%$$

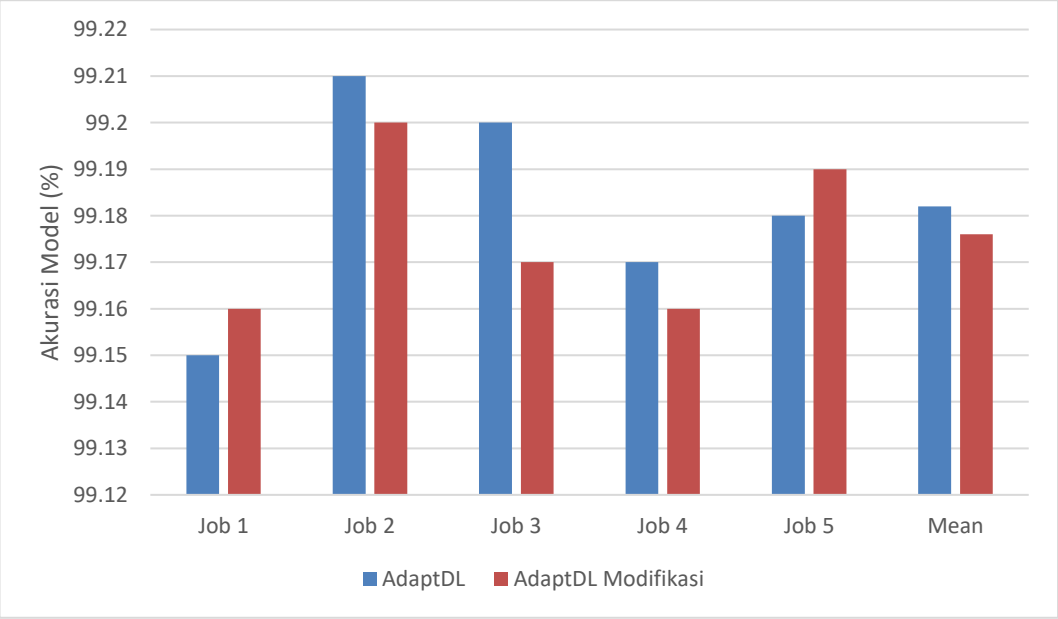


Gambar V.6.1. Grafik Hasil Pengujian Kecepatan pada Pemilihan Metrik

Tabel V.6.2. Rangkuman Hasil Pengujian Akurasi pada Pemilihan

Kategori	AdaptDL (%)	AdaptDL Modifikasi (%)	Selisih Akurasi (%)
Job 1	99,15	99,16	-0,01
Job 2	99,21	99,2	0,01
Job 3	99,2	99,17	0,03
Job 4	99,17	99,16	0,01
Job 5	99,18	99,19	-0,01
Mean	99,182	99.176	0,006

Persentase Selisih Akurasi $\frac{selisih\ rata - rata\ akurasi}{rata - rata\ AdaptDL} * 100\% = 0{,}006\%$



Gambar V.6.2. Grafik Hasil Pengujian Akurasi pada Pemilihan Metrik Kinerja

V.6.2 Analisis Hasil Pengujian Metrik untuk Memaksimalkan Kecepatan

Rangkuman hasil pengujian metrik untuk memaksimalkan kecepatan dapat dilihat pada Tabel V.6.3 untuk kecepatan proses pelatihan dan Tabel V.6.4 untuk akurasi model. Persentase selisih merupakan nilai perbandingan antara hasil *scheduler* dengan metrik kinerja *default* terhadap hasil *scheduler* dengan metrik kinerja *speed*. Untuk setiap tabel, terdapat ilustrasi dengan grafik yang dapat dilihat pada Gambar V.6.3 untuk grafik kecepatan proses pelatihan dan Gambar V.6.4 untuk grafik akurasi model. Pada grafik kecepatan, nilai yang rendah menandakan hasil yang lebih baik. Sedangkan pada grafik akurasi model, hasil yang lebih baik ditandai dengan nilai yang tinggi.

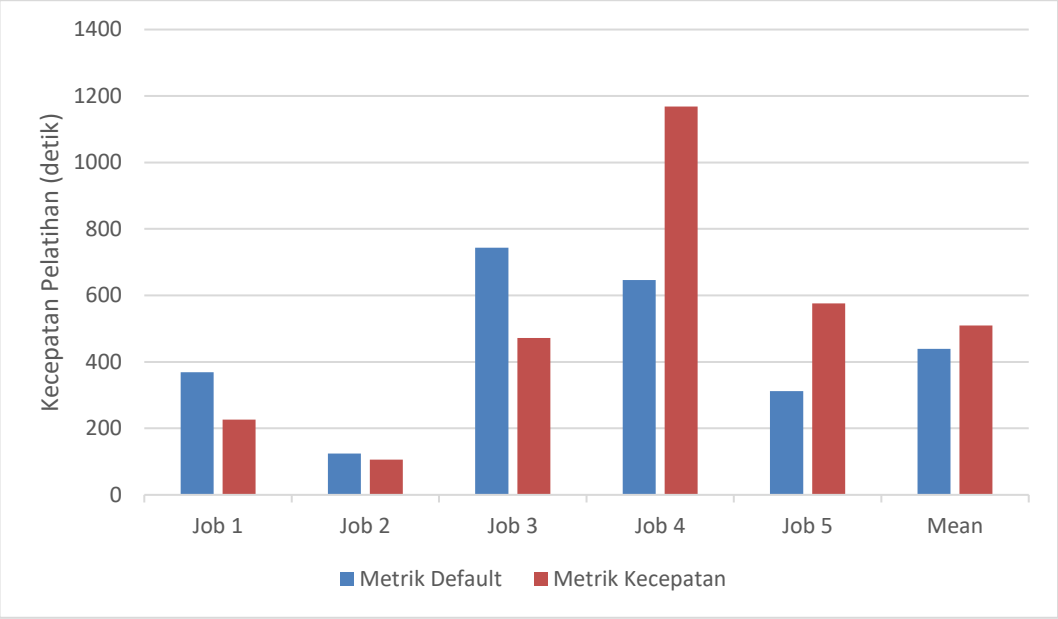
Berdasarkan hasil pengujian untuk skenario pengujian metrik untuk memaksimalkan kecepatan, terlihat bahwa metrik kecepatan menghasilkan kecepatan proses pelatihan yang lebih lambat dibandingkan dengan metrik *default*. Hal ini ditunjukkan dengan selisih rata-rata kecepatan pelatihan pada metrik kecepatan yang lebih lambat 16,099% dibandingkan dengan metrik *default*. Namun, akurasi yang dihasilkan relatif sama dengan metrik *default* dengan perbedaan yang hanya 0,048%.

Pada solusi metrik untuk memaksimalkan kecepatan, implementasi yang dilakukan memiliki dampak secara langsung pada proses pembelajaran dan membuat kecepatan pelatihan menjadi melambat. Hal ini diakibatkan karena dengan hilangnya nilai *statistical efficiency* pada perhitungan nilai Goodput, maka nilai Goodput yang dihasilkan oleh metrik kecepatan berbeda dengan metrik *default*. Pollux, melalui PolluxSched, menggunakan nilai Goodput untuk menentukan pengalokasian *worker* pada Job yang dijalankan. Dari hasil pengujian dengan menggunakan metrik kecepatan, tiga Job yang berukuran kecil diselesaikan dengan lebih cepat namun dua Job yang berukuran besar diselesaikan lebih lama dibandingkan dengan metrik *default*. Hal ini diperkirakan terjadi karena metrik kecepatan tidak memberikan nilai Goodput yang optimal sehingga pengalokasian *worker* juga menjadi tidak optimal dan mengakibatkan kecepatan rata-rata pelatihan menjadi lebih lama.

Tabel V.6.3. Rangkuman Hasil Pengujian Kecepatan pada Metrik Kecepatan

Kategori	Metrik *Default* (detik)	Metrik Kecepatan (detik)	Selisih Kecepatan (detik)
Job 1	368,481	226,115	142,366
Job 2	124,483	106,022	18,461
Job 3	743,627	472,3	271,327
Job 4	646,017	1168,12	-522,103
Job 5	312,535	575,98	-263,445
Mean	439,029	509,707	-70,679

Persentase Selisih Kecepatan $\frac{selisih\ rata - rata\ kecepatan}{rata - rata\ metrik\ default} * 100\% = -16{,}099\%$

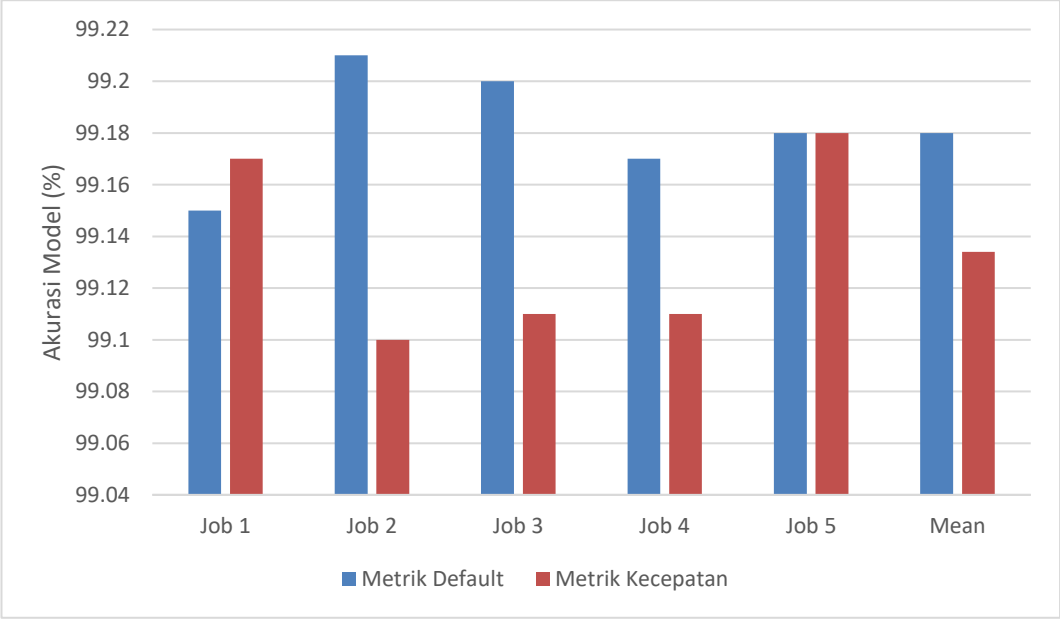


Gambar V.6.3. Grafik Hasil Pengujian Kecepatan pada Metrik Kecepatan

Tabel V.6.4. Rangkuman Hasil Pengujian Akurasi pada Metrik Kecepatan

Kategori	Metrik *Default* (%)	Metrik Kecepatan (%)	Selisih Akurasi (%)
Job 1	99,15	99,17	-0.02
Job 2	99,21	99,1	0.11
Job 3	99,2	99,11	0.09
Job 4	99,17	99,11	0.06
Job 5	99,18	99,18	0
Mean	99,182	99,134	0,048

Persentase Selisih Akurasi $\frac{selisih\ rata - rata\ akurasi}{rata - rata\ metrik\ default} * 100\% = 0{,}048\%$



Gambar V.6.4. Grafik Hasil Pengujian Akurasi pada Metrik Kecepatan

V.6.3 Analisis Hasil Pengujian Metrik untuk Mengefisienkan Sumber Daya

Rangkuman hasil pengujian metrik untuk mengefisienkan sumber daya dapat dilihat pada Tabel V.6.5 untuk kecepatan proses pelatihan, Tabel V.6.6 untuk akurasi model, dan Tabel V.6.7 untuk selisih waktu pembangkitan *worker node*. Persentase selisih merupakan nilai perbandingan antara hasil *scheduler* dengan metrik kinerja *default* terhadap hasil *scheduler* dengan metrik kinerja *cost*. Untuk setiap tabel, terdapat ilustrasi dengan grafik yang dapat dilihat pada Gambar V.6.5 untuk grafik kecepatan proses pelatihan, Gambar V.6.6 untuk grafik akurasi model, dan Gambar V.6.7 untuk grafik waktu hidup. Pada grafik kecepatan, nilai yang rendah menandakan hasil yang lebih baik. Pada grafik akurasi model, hasil yang lebih baik ditandai dengan nilai yang tinggi. Sedangkan pada grafik waktu hidup, nilai yang rendah untuk waktu hidup dan nilai yang tinggi untuk selisih waktu hidup menandakan hasil yang lebih baik.

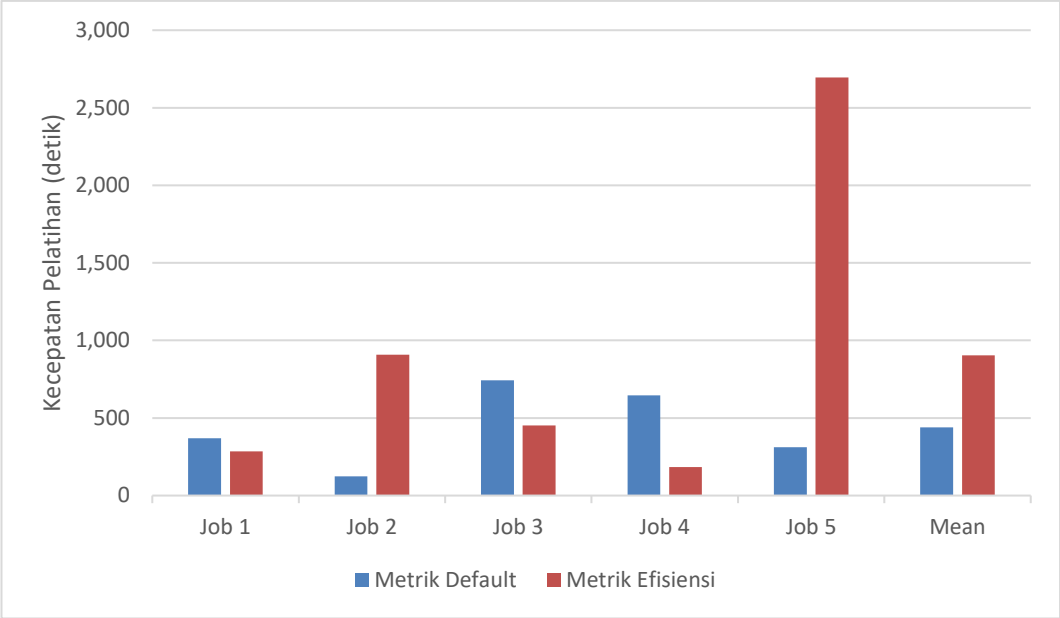
Berdasarkan hasil pengujian untuk skenario pengujian metrik untuk mengefisienkan sumber daya, terlihat bahwa metrik efisiensi menghasilkan kecepatan proses pelatihan yang lebih tinggi dibandingkan dengan metrik *default*. Hal ini ditunjukkan dengan selisih rata-rata kecepatan pelatihan pada metrik efisiensi yang melambat hingga 106,077% dibandingkan dengan metrik *default*. Kemudian, selisih waktu hidup antar *worker node* pada metrik efisiensi meningkat 80% dibandingkan dengan metrik *default*. Selain itu, rata-rata waktu hidup *worker node* meningkat sebesar 19,31%. Namun, akurasi yang dihasilkan relatif sama dengan metrik *default* dengan perbedaan hanya 0,03%.

Pada solusi metrik untuk mengefisienkan sumber daya, implementasi yang dilakukan memiliki dampak secara langsung pada proses pembelajaran. Dengan berubahnya nilai ambang batas atas menjadi 0,8 membuat mekanisme *scaling up* menjadi lebih lama. Hal ini ditunjukkan pada waktu pembangkitan *worker node* kedua yang lebih tinggi hingga 80% pada metrik efisiensi. Namun, akibat dari pembangkitan *worker node* yang lebih lama, waktu pelatihan yang diperlukan dan waktu hidup *worker node* juga menjadi lebih lama. Waktu pelatihan melambat

hingga 106,77% dikarenakan terdapat Job yang harus menunggu sampai *worker node* dibangkitkan untuk kemudian dikerjakan oleh *worker node* tersebut.

Tabel V.6.5. Rangkuman Hasil Pengujian Kecepatan pada Metrik Efisiensi

Kategori	**Metrik *Default* (detik)**	**Metrik Efisiensi (detik)**	**Selisih Kecepatan (detik)**
Job 1	368,481	284,303	-84,178
Job 2	124,483	908,11	783,867
Job 3	743,627	451,8	-291,827
Job 4	646,017	184,57	-461,447
Job 5	312,535	2694,9	2382,365
Mean	439,029	904,7366	-465,708
Persentase Selisih Kecepatan	$\frac{selisih\ rata-rata\ kecepatan}{rata-rata\ metrik\ default} * 100\% = -106{,}077\%$		

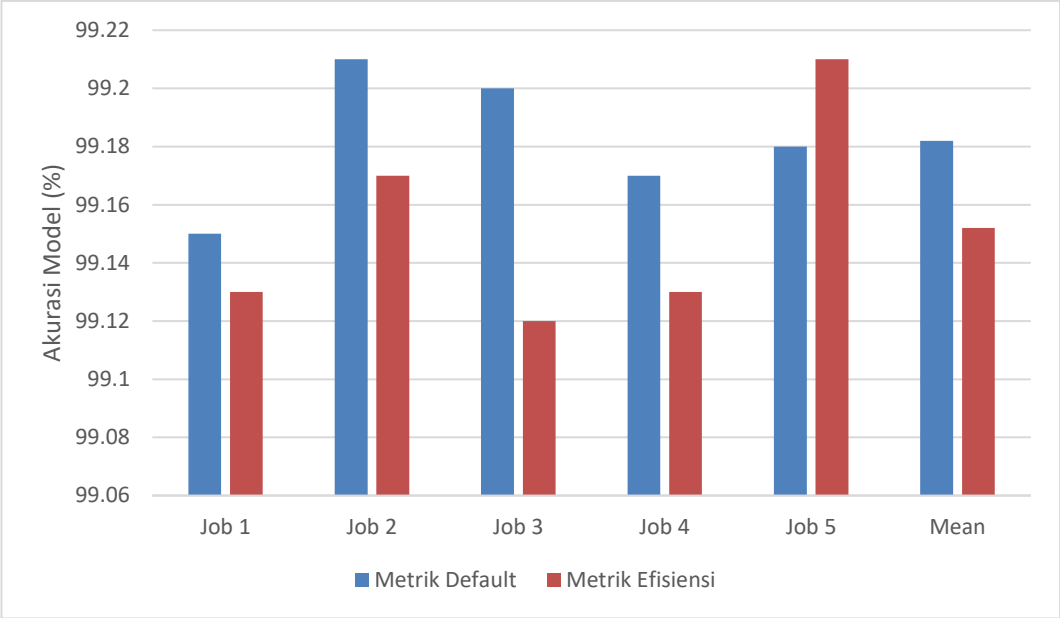


Gambar V.6.5. Grafik Hasil Pengujian Kecepatan pada Metrik Efisiensi

Tabel V.6.6. Rangkuman Hasil Pengujian Akurasi pada Metrik Efisiensi

Kategori	**Metrik *Default* (%)**	**Metrik Efisiensi (%)**	**Selisih Akurasi (%)**
Job 1	99,15	99,13	0,02
Job 2	99,21	99,17	0,04
Job 3	99,2	99,12	0,08
Job 4	99,17	99,13	0,04
Job 5	99,18	99,21	-0,03
Mean	99,182	99,152	0,03

Persentase Selisih Akurasi $\frac{selisih\ rata - rata\ akurasi}{rata - rata\ metrik\ default} * 100\% = 0{,}03\%$



Gambar V.6.6. Grafik Hasil Pengujian Akurasi pada Metrik Efisiensi

Tabel V.6.7. Rangkuman Hasil Pengujian Waktu Hidup pada Metrik Efisiensi

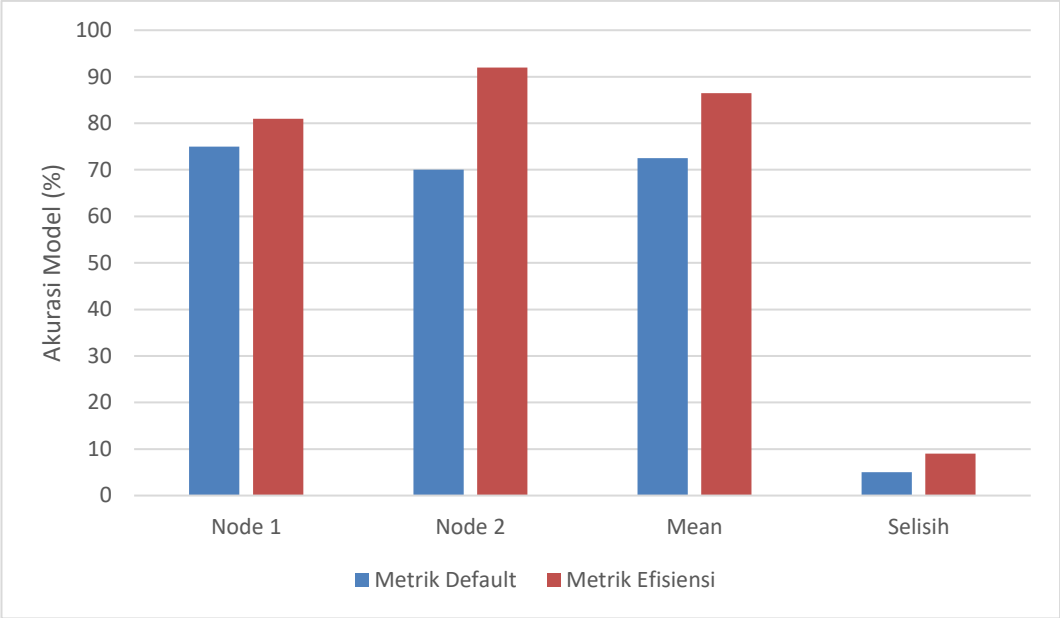
Metrik	**Waktu Hidup (menit)**		**Mean**	**Selisih Waktu Hidup (menit)**
	Node 1	**Node 2**		
Metrik *Default*	75	70	72,5	5
Metrik Efisiensi	91	82	86,5	9

Persentase Selisih Rata-rata Waktu Hidup

$$\frac{selisih\ rata - rata\ waktu\ hidup}{waktu\ hidup\ metrik\ default} * 100\% = -19{,}31\%$$

Persentase Selisih Waktu Hidup

$$\frac{selisih\ waktu\ hidup\ default - efisiensi}{selisih\ waktu\ hidup\ metrik\ default} * 100\% = -80\%$$



Gambar V.6.7. Grafik Hasil Pengujian Waktu Hidup pada Metrik Efisiensi

BAB VI

SIMPULAN DAN SARAN

Pada bab ini dijelaskan tentang simpulan dan saran berdasarkan hasil implementasi pada Bab IV dan pengujian pada Bab V.

VI.1 Simpulan

Berdasarkan analisis hasil, dapat ditarik kesimpulan sebagai berikut:

1. Penambahan opsi metrik kinerja pada *hybrid resource scheduler* tidak berpengaruh terhadap kecepatan proses pelatihan *deep learning* terdistribusi dan akurasi model yang dihasilkan. Hal ini ditunjukkan oleh perbedaan kecepatan proses pelatihan yang hanya berbeda 3,06% dan akurasi model yang hanya berbeda 0,006% pada pengujian dengan *dataset* MNIST.
2. Metrik kinerja kecepatan pada *hybrid resource scheduler* menyebabkan kecepatan proses pelatihan *deep learning* terdistribusi menjadi lebih lambat walaupun akurasi model yang dihasilkan relatif sama. Kecepatan proses pelatihan dengan menggunakan metrik kinerja kecepatan melambat sebesar 16,099%, sedangkan akurasi model relatif sama dengan perbedaan hanya 0.048% pada pengujian dengan *dataset* MNIST.
3. Metrik efisiensi sumber daya pada *hybrid resource scheduler* menyebabkan kecepatan proses pelatihan menjadi lebih lambat, meskipun akurasi model yang dihasilkan relatif sama. Selain itu, metrik efisiensi sumber daya juga menyebabkan penggunaan sumber daya yang lebih tinggi akibat proses pelatihan yang lebih lambat, walaupun waktu untuk pembangkitan *worker node* relatif lebih lama. Kecepatan proses pelatihan dengan menggunakan metrik kinerja efisiensi sumber daya melambat sebesar 106,077%, waktu pembangkitan *worker node* meningkat sebesar 80%, penggunaan sumber daya berdasarkan waktu hidup *worker nodes* meningkat sebesar 19,31%, dan akurasi model relatif sama dengan perbedaan hanya 0.03% pada pengujian dengan *dataset* MNIST.

VI.2 Saran

Saran terkait dengan topik tugas akhir untuk pengembangan lebih lanjut adalah sebagai berikut:

1. Mengembangkan skenario pengujian pembelajaran *deep learning* dengan model dan data yang lebih kompleks serta jumlah sumber daya yang lebih banyak.
2. Mengembangkan fitur *graphical user interface* (GUI) untuk melakukan *monitoring* selama proses pembelajaran *deep learning*.
3. Mengembangkan metrik baru yang merupakan gabungan dari metrik untuk memaksimalkan kecepatan dan metrik untuk mengefisienkan sumber daya.

DAFTAR REFERENSI

AdaptDL. (2020). *AdaptDL Documentation*. Diakses pada 17 Oktober, 2020, dari https://adaptdl.readthedocs.io/en/latest/.

Deng, L. (2012). *The MNIST database of handwritten digit images for machine learning research*. *IEEE Signal Processing Magazine*, *29*(6), 141–142. https://doi.org/10.1109/MSP.2012.2211477

Deng, L., & Yu, D. (2014). D*eep Learning: Methods and Applications (Issue MSR-TR-2014-21).* https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/

Ellingwood, J. (2018). *An Introduction to Kubernetes*. Diakses pada 17 Oktober, 2020, dari https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes/.

Gu, J., Chowdhury, M., Shin, K. G., Zhu, Y., Jeon, M., Qian, J., Liu, H., & Guo, C. (2019). Tiresias: A GPU cluster manager for distributed deep learning. *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019*, 485–500.

Hegde, V., & Usmani, S. (2016). *Parallel and Distributed Deep Learning*. Tech Report, 1–11.

Kubernetes. (2020). *Concepts*. Diakses pada 17 Oktober, 2020, dari https://kubernetes.io/.

Lecun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning*. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539.

Lim, S., Young, S. R., & Patton, R. M. (2016). *An analysis of image storage systems for scalable training of deep neural networks*. *The Seventh Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware (in Conjunction with ASPLOS'16)*, *April*.

PyTorch. (2020). *Learning PyTorch*. Diakses pada 18 Oktober, 2020, dari https://pytorch.org/.

Qiao, A., Neiswanger, W., Ho, Q., Zhang, H., Ganger, G. R., & Xing, E. P. (2020). Pollux: *Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning*. http://arxiv.org/abs/2008.12260

Rollik, B. (2020). *Understanding Kubernetes Autoscaling*. Diakses pada 19 Februari, 2020, dari https://blog.scaleway.com/understanding-kubernetes-autoscaling/.

Tamiru, M., Tordsson, J., Elmroth, E., Pierre, G., Tamiru, M., Tordsson, J., Elmroth, E., Pierre, G., Experimental, A., Tordsson, J., & Elmroth, E. (2020). *An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud*. CloudCom 2020 - 12th IEEE International Conference on Cloud Computing Technology and Science, Dec 2020, Bangkok, Thailand. pp.1-9. ffhal-02958916

Xiao, W., Bhardwaj, R., Ramjee, R., Sivathanu, M., Kwatra, N., Han, Z., Patel, P., Peng, X., Zhao, H., Zhang, Q., Yang, F., & Zhou, L. (2007). Gandiva: Introspective cluster scheduling for deep learning. *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018*, 595–610.

Lampiran A. Kode Implementasi Pemilihan Metrik Kinerja

```
1.  # Pemanggilan pada file driver untuk menjalankan deep learning
2.  def main():
3.      # Kode sebelumnya ...
4.      # Python parser argument, untuk mengambil value flag dari
         pengguna
5.      parser.add_argument('--metrics-options', type=str,
                            default='default', help='For choosing
                            metric options')
6.      args = parser.parse_args()
7.
8.      # Taruh opsi metrics_options sebagai keywords arguments
9.      kwargs = {'batch_size': args.batch_size, 'metrics_options':
                  args.metrics_options}
10.
11.     # Ambil dataset MNIST
12.     dataset1 = datasets.MNIST('../data', train=True,
                                  download=True, transform=transform)
13.     dataset2 = datasets.MNIST('../data', train=False,
                                  transform=transform)
14.
15.     # Passing opsi metrik dalam keywords arguments pada
         pemanggilan DataLoader untuk memulai pelatihan dan pengujian
16.     train_loader = adaptdl.torch.AdaptiveDataLoader(dataset1,
                                                        drop_last=True
                                                        , **kwargs)
17.     test_loader = adaptdl.torch.AdaptiveDataLoader(dataset2,
                                                       **kwargs)
18.
19.     # Kode setelahnya ...
20.
21. # Pemanggilan pada kelas Goodput
22. def evaluate(self, num_nodes, num_replicas, atomic_bsz,
                 accum_steps):
23.     batch_size = num_replicas * atomic_bsz * (accum_steps + 1)
24.     assert np.all(self._init_batch_size <= batch_size)
25.     result = 0
26.
27.     # Pemilihan metrik
28.     if self.metrics_options == "speed":
29.         result = self.throughput(num_nodes, num_replicas,
                                     atomic_bsz, accum_steps)
30.     else:
31.         result = self.throughput(num_nodes, num_replicas,
                                     atomic_bsz, accum_steps) *
                                     self.efficiency(batch_size)
32.     return result
33.
34. # Pemanggilan pada kelas PolluxPolicy
35. def __init__(self, metrics_options="default"):
36.     self._prev_states = None
37.     self._prev_jobs = None
```

```
38.     self._prev_nodes = None
39.     self._min_util = 0.35
40.     # Pemilihan metrik
41.     self._max_util = 0.8 if metrics_options == "cost" else 0.65
```

Lampiran B. Kode Implementasi Metrik Kecepatan

```
1.  class GoodputFunction(object):
2.
3.      def __init__(self, perf_params, grad_params, init_batch_size,
                     metrics_options="default"):
4.          # Atribut lain ...
5.          # Ambil value metrics_options dari parameter konstruktor
6.          self.metrics_options = metrics_options
7.
8.      def __call__(self, num_nodes, num_replicas, atomic_bsz,
                     accum_steps):
9.          return self.evaluate(num_nodes, num_replicas, atomic_bsz,
                     accum_steps)
10.
11.     # Method untuk melakukan evaluate terhadap Job
12.     def evaluate(self, num_nodes, num_replicas, atomic_bsz,
                     accum_steps):
13.         batch_size = num_replicas * atomic_bsz * (accum_steps +
                       1)
14.         assert np.all(self._init_batch_size <= batch_size)
15.         result = 0
16.         # Ambil value metrics_options sebagai atribut kelas
17.         if self.metrics_options == "speed":
18.             result = self.throughput(num_nodes, num_replicas,
                                         atomic_bsz, accum_steps)
19.         else:
20.             result = self.throughput(num_nodes, num_replicas,
                                         atomic_bsz,accum_steps)
21.             result *= self.efficiency(batch_size)
22.         return result
23.
24.     # Method lain ...
```

Lampiran C. Kode Implementasi Metrik Efisiensi

```
1.  class PolluxPolicy(object):
2.      def __init__(self, metrics_options="default"):
3.          self._prev_states = None
4.          self._prev_jobs = None
5.          self._prev_nodes = None
6.          self._min_util = 0.35
7.          # Pemilihan metrik untuk nilai nilai ambang batas
8.          self._max_util = 0.8 if metrics_options == "cost" else
             0.65
9.
10.     # Method untuk menentukan jumlah node paling baik
11.     # Node lebih banyak dari node sebelumnya => scaling up
12.     # Node lebih sedikit dari node sebelumnya => scaling down
13.     def _desired_nodes(self, utilities, values, nodes):
14.         idx = self._select_result(values, len(nodes))
15.         # Apabila nilai utility sudah berada di antara nilai
             ambang bawah dan ambang atas
16.         if idx is not None and \
17.                 self._min_util <= utilities[idx] <=
             self._max_util:
18.             return len(nodes)
19.         target_util = (self._min_util + self._max_util) / 2
20.         best_util = np.inf
21.         best_nodes = len(nodes)
22.         for util, (_, num_nodes) in zip(utilities, values):
23.             if util < self._min_util:
24.                 continue
25.             if np.isclose(util, best_util) and num_nodes >
             best_nodes:
26.                 best_nodes = num_nodes
27.             if abs(util - target_util) < abs(best_util -
             target_util):
28.                 best_util = util
29.                 best_nodes = num_nodes
30.         return int(best_nodes)
31.
32.     # Method lain ...
```

Lampiran D. Kode Konfigurasi *Cluster* untuk Pengujian

```
1.  apiVersion: eksctl.io/v1alpha5
2.  kind: ClusterConfig
3.
4.  metadata:
5.    name: adaptdl-eks-cluster
6.    region: us-east-1
7.
8.  nodeGroups:
9.    - name: ng-1
10.     instanceType: g4dn.xlarge
11.     maxSize: 2
12.     minSize: 1
13.     desiredCapacity: 1
14.     iam:
15.       withAddonPolicies:
16.         autoScaler: true
17.         efs: true
18.     ami: auto
19.     ssh:
20.       allow: true
21.     tags:
22.       Owner: "pandyaka"
23.       Team: "team-ta2 "
24.       k8s.io/cluster-autoscaler/enabled: "true"
25.       k8s.io/cluster-autoscaler/adaptdl-eks-cluster: "owned"
26.
27. cloudWatch:
28.     clusterLogging:
29.         enableTypes: ["*"]
```