

Penerapan Metode *Hierarchical Clustering* untuk Dekomposisi *Microservice* dari Monolitik pada *Enterprise Resource Planning*

Albertus Septian Angkuw¹, Hans Christian Kurniawan²

Program Studi Informatika, Institut Teknologi Harapan Bangsa
Jl. Dipati Ukur No. 80, Dago, Bandung, Indonesia

¹albertusangkuw.contact@gmail.com

²hans_christian@ithb.ac.id

Abstract— *Enterprise Resource Planning (ERP) applications can be built using Monolith Architecture, Service-Oriented Architecture, and Microservice. The Monolith architecture is simple but lacks scalability and sustainability in development, while Microservice architecture is a modern approach suitable for vertically and horizontally scaled enterprise applications. The benefits of Microservice architecture have led companies to migrate from Monolith architecture to Microservices. However, this migration process has proven to be hard and costly, particularly in identifying the components of the Monolith application. Component identification can be semi-automated using clustering algorithms. The clustering algorithm used is Hierarchical Clustering which has linkages such as single linkage, complete linkage, and average linkage. In this research, Odoo, an ERP application, is decomposed from Monolith to Microservices by analysing the program's code graph then inserting the graph into Hierarchical Clustering. The results of the clustering are tested by evaluating cohesion and coupling for each linkage, followed by selecting the parts to be implemented. Based on the testing, it was found that the Average linkage is suitable for creating service clusters with good coupling and cohesion. The ideal number of services in the Average linkage ranges from 175 to 245 services. From the table structure that is formed when implementation shows Hierarchical Clustering can separate unconnected modules and identify services with strong relationships, as in case in the 10th partition with Module Product and Module Point of Sale and on case on the 17th partition with Module Calendar.*

Keywords— *Decomposition, Hierarchical Clustering, Microservice, Monolith, Enterprise Resource Planning, Odoo*

Abstrak—Aplikasi *Enterprise Resource Planning (ERP)* dapat dibangun dengan arsitektur Monolitik, *Service Oriented Architecture*, dan *Microservice*. Arsitektur monolitik merupakan arsitektur sederhana namun monolitik tidak mudah dilakukan *scaling* dan sulit dikembangkan secara berkelanjutan sedangkan *Microservice* merupakan arsitektur modern yang cocok pada aplikasi perusahaan yang telah tumbuh dengan skala secara vertikal maupun horizontal. Manfaat dari *microservice* membuat perusahaan melakukan migrasi aplikasi berarsitektur monolitik menjadi arsitektur *microservice*. Namun proses ini terbukti sulit dan mahal, salah satu tantangan adalah bagaimana mengidentifikasi komponen dari aplikasi monolitik. Identifikasi dapat dilakukan secara semi-otomatis yang menggunakan algoritma *clustering*. Algoritma *clustering* yang digunakan yaitu *Hierarchical Clustering* dimana terdapat *linkage* seperti *single*

linkage, *complete linkage*, dan *average linkage*. Penelitian ini menggunakan Odoo sebagai aplikasi ERP yang dilakukan dekomposisi dari monolitik menjadi *Microservice* dengan pendekatan menganalisis *graph* dari kode program kemudian memasukkan *graph* ke *Hierarchical Clustering*. Hasil dari pengelompokan diuji dengan melihat *cohesion* dan *coupling* untuk setiap *linkage* kemudian dilakukan pemilihan bagian yang di implementasikan. Berdasarkan pengujian ditemukan *linkage* yang cocok untuk membuat kelompok *service* yang memiliki *coupling* dan *cohesion* yang baik adalah *Average linkage*. Pemilihan jumlah *service* yang ideal pada *Average linkage* dimulai dari 175-245 *service*. Dari struktur tabel yang terbentuk ketika implementasi menunjukan *Hierarchical Clustering* dapat memisahkan *module* yang tidak terhubung dan *service* yang memiliki hubungan kuat, seperti pada kasus di partisi ke-10 dengan *Module Product* dan *Module Point of Sale* dan pada kasus di partisi ke-17 dengan *Module Calendar*.

Kata Kunci— *Dekomposisi, Hierarchical Clustering, Microservice, Monolitik, Enterprise Resource Planning, Odoo*

I. PENDAHULUAN

Aplikasi *Enterprise Resource Planning (ERP)* memiliki peran penting dalam industri dan bisnis saat ini. Tujuan dari implementasi ERP di perusahaan yaitu untuk meningkatkan efisiensi dengan cara mengintegrasikan dan mengotomatisasi aktivitas bisnisnya [1]. Selain itu diharapkan juga ERP memiliki skalabilitas dan fleksibilitas terhadap operasi bisnis baik yang diperlukan dalam jangka panjang atau jangka pendek [2]. Dalam membangun aplikasi ERP dapat dibangun dengan arsitektur seperti monolitik, *Service-Oriented Architecture (SOA)*, dan *Microservice (MSA)* [3].

Arsitektur monolitik merupakan arsitektur paling sederhana dalam membangun aplikasi karena pengembangan yang mudah selama aplikasi berbentuk sederhana, walaupun demikian monolitik tidak mudah dilakukan *scaling* dan sulit dikembangkan secara berkelanjutan. SOA bisa membantu menyelesaikan permasalahan tersebut dengan sistem terdistribusi akan tetapi SOA memiliki kekurangan sama seperti monolitik [3]. SOA sendiri dapat disamakan sebagai bentuk monolitik terdistribusi karena sistem memiliki banyak *service* tapi seluruh sistem harus dilakukan *deployment* bersamaan. Monolitik terdistribusi memiliki *coupling* yang

tinggi dan bila dilakukan perubahan pada satu bagian dapat menyebabkan kerusakan pada bagian lain [4].

Microservice merupakan arsitektur modern yang cocok pada aplikasi perusahaan yang telah tumbuh dengan skala secara vertikal maupun horizontal, terdistribusi, dapat dikembangkan secara berkelanjutan, dan memiliki performa yang baik. Akan tetapi perlu diketahui bahwa tidak semua perusahaan harus menggunakan arsitektur *microservice* [3].

Manfaat dari *microservice* membuat banyak perusahaan melakukan migrasi aplikasi berarsitektur monolitik menjadi arsitektur *microservice* seperti Netflix, eBay, Amazon, IBM, dan lainnya. Namun proses perubahan ini terbukti sulit dan mahal, salah satu tantangan terbesar yang harus dihadapi adalah bagaimana mengidentifikasi dan membagi komponen dari aplikasi monolitik. Komponen aplikasi ini kerap kali sangat *cohesif* dan *coupled* karena sifat desain arsitektur monolitik [5].

Untuk itu ada beberapa pendekatan untuk membagi komponen atau bisa disebut dekomposisi yaitu secara manual atau secara semi-otomatis [6]. Pembagian secara manual dapat menggunakan konsep *Domain Driven Design*, dekomposisi berdasarkan kemampuan bisnis, dan menggunakan pendekatan campuran lainnya [8]. Pendekatan dekomposisi dengan cara manual tidak mudah karena mudah terjadi kesalahan dan membutuhkan banyak waktu [6]. Oleh sebab itu dikembangkan otomatisasi untuk dapat mengenali komponen dari aplikasi monolitik untuk membentuk *microservice*. Pengenalan komponen ini dapat diselesaikan dengan menggunakan algoritma *clustering*. Sebelum melakukan pengelompokan yaitu membuat *call graph* yang mengkodekan interaksi antara class dari kode program. *Graph* tersebut diolah menjadi matriks kemiripan sebelum dimasukkan ke dalam algoritma *clustering* [5].

Algoritma *clustering* yang umum digunakan dan terbukti dapat melakukan modularisasi pada perangkat lunak yaitu *Hierarchical Clustering*. *Hierarchical Clustering* memiliki kompleksitas waktu yang lebih sedikit dibandingkan algoritma lainnya seperti algoritma *hill-climbing* dan algoritma genetik. *Hierarchical Clustering* mengelompokkan objek yang memiliki kesamaan ke dalam suatu partisi (*cluster*) [9]. Dalam membentuk partisi ini terdapat beberapa metode yang disebut linkage untuk menentukan partisi terdekat dengan sebuah objek yaitu jarak maksimum (*complete linkage*), jarak minimum (*single linkage*) dan nilai rata-rata jarak (*average linkage*) [11].

Pada penelitian ini akan melakukan dekomposisi aplikasi ERP yang memiliki arsitektur monolitik menjadi arsitektur *microservice* dengan pendekatan menganalisis *graph* yang dihasilkan dari kode program kemudian dilakukan pengelompokan secara *Hierarchical Clustering*. Hasil dari pengelompokan akan diuji dengan melihat *cohesion* dan *coupling* kemudian dilakukan pemilihan bagian yang di implementasikan. Dengan ini diharapkan bisa menyelesaikan permasalahan yang terjadi ketika migrasi aplikasi seperti mengenali komponen dan dapat dikembangkan secara berkelanjutan.

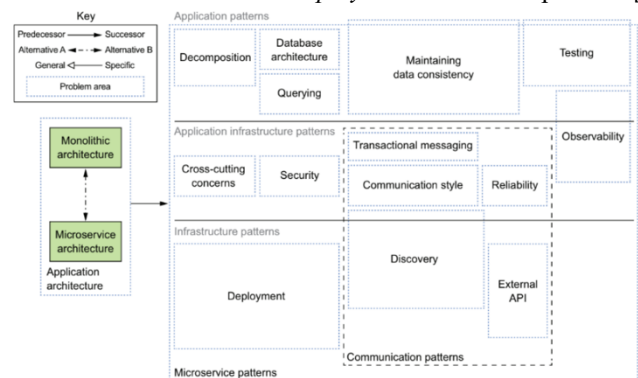
II. METODOLOGI

A. Monolitik

Monolitik yaitu suatu cara untuk melakukan penyebaran. Ketika semua fungsi dalam sistem harus disebar secara bersama-sama, maka itu merupakan sebuah monolitik. Setidaknya terdapat 3 jenis monolitik yang memiliki struktur yang berbeda masing-masing namun masih merupakan arsitektur monolitik [4]. *Single Process Monolith*, *Distributed Monolith*, Sistem *Black-Box* oleh Pihak Ketiga. Masing-masing jenis monolitik memiliki keuntungan yang sama seperti: Sederhana dalam melakukan pengembangan, mudah untuk melakukan perubahan secara radikal di aplikasi, pengujian dilakukan pada satu aplikasi. Namun monolitik memiliki tantangan seperti: sulit dikembangkan secara berkelanjutan, memiliki reliabilitas yang rendah, tidak mudah untuk melakukan skalabilitas, terkunci pada teknologi lama [4, 10].

B. Microservice

Microservice adalah beberapa *service* yang bisa di-*deploy* secara independen yang dimodelkan berdasarkan bisnis domain. *Service* ini berkomunikasi satu sama lain melalui jaringan komputer dan bisa dibangun dengan berbagai macam teknologi. *Microservice* adalah salah tipe dari *service oriented architecture* (SOA) meskipun ada perbedaan dalam membuat batasan antara *service* dan *deployment* secara independen [4].



Gambar 1 Pola dalam menyelesaikan masalah di arsitektur *Microservice* [10]

Hal yang membedakan arsitektur *microservice* dengan arsitektur lainnya yaitu kecil dan berfokus pada satu hal dengan baik, otonomi / bisa berdiri sendiri, data yang dikelola masing-masing *service*. Keuntungan dari menggunakan arsitektur *Microservice*: memudahkan pengembangan aplikasi kompleks dan fleksibel, bisa dilakukan *scaling* secara independen, mudah melakukan percobaan dan penggunaan teknologi baru. Tantangan dari menggunakan arsitektur *Microservice*: Menemukan *service* yang tepat itu sulit, memiliki kompleksitas karena merupakan suatu sistem terdistribusi, *deployment* yang melibatkan beberapa *service* [10, 5, 4].

C. Dekomposisi

Dekomposisi merupakan proses pemisahan bagian dari aplikasi, proses dekomposisi sendiri dapat menyebabkan masalah dengan latensi, penyelesaian masalah, integritas, kegagalan bersamaan, dan hal lainnya. Terdapat beberapa cara untuk melakukan dekomposisi aplikasi monolitik, dekomposisi ini menentukan bagaimana bagian aplikasi menjadi *Service*:

Kemampuan Bisnis [10], Domain Driver Design (DDD), dan Analisis Kode [5, 12].

Ketika sudah menentukan *service* yang ingin didekomposisi diperlukan pola dan strategi untuk memisahkan bagian yang ingin didekomposisi. Setiap pola memiliki kekurangan dan kelebihan masing-masing: Pola *Strangle Fig*, Pola *UI Composition*, *Branch By Abstraction*, *Parallel Run*, *Decorating Collaborator*, *Change Data Capture* [4]. Terdapat tantangan dan hambatan ketika melakukan proses dekomposisi. Tantangan dan hambatan dapat terjadi karena proses dekomposisi itu sendiri seperti latensi jaringan, menjaga konsistensi data antar *service*, adanya *God Class* yang mencegah dekomposisi [10].

D. Clustering

Clustering yaitu suatu proses untuk melakukan pengelompokan atau klasifikasi objek. Objek bisa ditentukan dari pengukuran atau berdasarkan hubungan antar objek lainnya. Tujuan dari *clustering* yaitu untuk menemukan struktur data yang valid. *Cluster* terdiri dari sejumlah objek serupa yang dikumpulkan / dikelompokkan bersama [7].

Untuk membuat kedekatan antar objek perlu diketahui tipe data yang digunakan agar kedekatan yang dihasilkan relevan. Berikut adalah metode mencari nilai kedekatan untuk pengelompokan objek: *Jaccard Coefficient* [7], *Structural Similarity* [5], dan *Simple matching coefficient* [7].

Unsupervised Clustering adalah pengelompokan di mana tidak diberikan sebuah label untuk mengetahui partisi objek sehingga pengelompokan hanya menggunakan nilai kedekatan antar objek. Salah satu bentuk *Unsupervised Clustering* yaitu *Hierarchical Clustering* dimana sebuah prosedur untuk mentransformasi sebuah *proximity matrix* menjadi beberapa partisi. Terdapat 2 pendekatan algoritma *clustering* dalam membentuk suatu partisi yaitu secara *agglomerative* dan *divisive* [7].

Untuk mengetahui apakah pengelompokan yang dihasilkan dari proses *clustering* memiliki kualitas yang tepat maka perlu dilakukan evaluasi yang mempertimbangkan aspek *microservice* seperti *Structural and Behavioral Dependencies* [6], *Data Autonomy Class* [6], *Iter-Partition Call Percentage* (ICP) [5], dan jumlah *interface* [5].

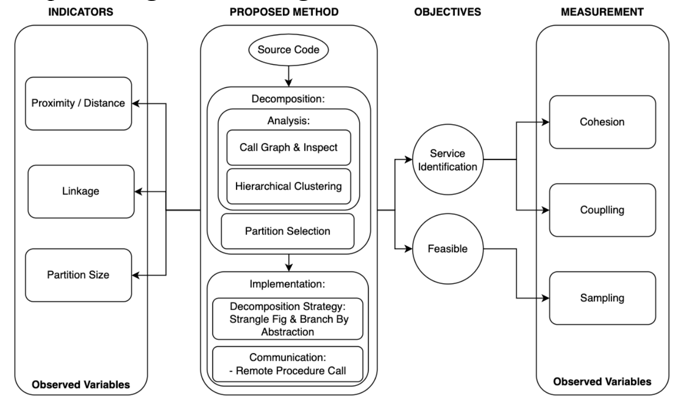
E. Perancangan Sistem

1) Kerangka Pemikiran

Penelitian dimulai dengan menggunakan kode sumber aplikasi yang dibuat dengan monolitik. Kode sumber dilakukan proses dekomposisi yaitu dengan analisis seperti mencari objek beserta atributnya, untuk mencari keterhubungan lebih lanjut tentang objek maka dilakukan pencarian pada fungsi-fungsi sehingga terbentuklah *call graph* yang menunjukkan bagaimana keterhubungan masing-masing objek di aplikasi.

Dari *graph* yang sudah dibuat dilakukan pengelompokan dengan algoritma *Hierarchical Clustering*. Pendekatan algoritma yang digunakan pada penelitian ini yaitu *agglomerative*, selain itu ditentukan cara menghitung kedekatan antara objek dan pemilihan algoritma *linkage*. Metode *linkage* yaitu menentukan jarak atau kemiripan antara

semua objek. Untuk menentukan jarak ini bisa dengan *complete*, *single*, dan *average*.

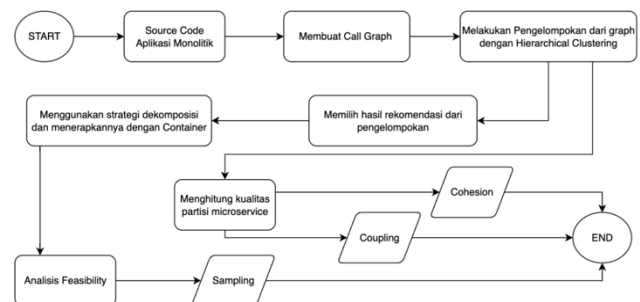


Gambar 2 Kerangka Pemikiran

Pengelompokan dari *Hierarchical Clustering* dipilih dengan mencari nilai *cohesion* terendah dan nilai *coupling* tertinggi. Di mana *coupling* mengevaluasi tingkat ketergantungan langsung dan tidak langsung antar objek. Semakin banyak dua objek menggunakan metode masing-masing semakin mereka menjadi satu kesatuan. Sedangkan *cohesion* mengevaluasi kekuatan interaksi antar objek. Biasanya, dua objek atau lebih menjadi interaktif jika metodenya menggunakan metodenya satu sama lain. Dengan membandingkan nilai *coupling*, *cohesion*, *linkage* dan ukuran partisi maka dapat diketahui kelompok *service* yang ideal.

Selain itu, untuk mengetahui apakah hasil *clustering* relevan dan dapat diimplementasikan maka dilakukan *sampling* yang berjumlah 2 *service* dari kelompok *service* yang ideal. Untuk menerapkannya pemecahan dimulai dari pemecahan kode yang menggunakan strategi dekomposisi *strangle fig* dan *branch by abstraction*. Untuk metode komunikasinya antara *service* yaitu dengan *Remote Procedure Call* (RPC) dan untuk mengelola datanya, setiap *service* memiliki datanya masing-masing.

2) Flowchart Global



Gambar 3 Diagram Flowchart Proses Global

Aplikasi ERP Odoo merupakan aplikasi berlisensi *open source*, kode program dapat diunduh melalui situs *repository* Odoo. Pada tugas akhir ini menggunakan Odoo versi 16 dan menggunakan *tools* PyCG untuk menghasilkan *call graph* dalam bentuk format JSON. Terdapat target folder yang harus dianalisis oleh PyCG yaitu folder 'odoo/addons' atau *package* 'odoo.addons'. Pemilihan folder ini dikarenakan

folder/package lainnya tidak memiliki hubungan mengenai proses bisnis.

Keterbatasannya informasi *call graph* yang dihasilkan dari PyCG, sehingga tugas akhir ini menggunakan library Python yaitu 'inspect' untuk menganalisis objek secara *run-time*. Hal ini disebabkan Python adalah bahasa pemrograman dinamik di mana pengecekan tipe data dilakukan secara *'run-time'*. *Object* yang dianalisis yaitu *class* yang merupakan turunan dari *class odoo.models.MetaModel*, di mana *class MetaModel* memiliki properti seperti *name*, *inherits*, *inherit*, dan *comodel name*. Hasil ekstrasi *dependency model* digabungkan melalui nama modul PyCG.

Graph yang dihasilkan dari proses ekstrasi dipisahkan antara modul eksternal dan modul internal. Module yang digunakan untuk pengelompokan adalah modul internal, setiap *call* yang dilakukan memiliki nama *call* yang berupa gabungan antara nama fungsi / *class* / modul / file di kode program. *Graph* yang digunakan merupakan *directed* sehingga setiap pemanggilan berlaku untuk satu arah. *Graph* ini dapat diubah menjadi bentuk *dictionary* sebagai *adjacency list*, tujuannya agar pengaksesan dan pencarian node lebih cepat dan mudah dilakukan. Setiap relasi memiliki jumlah berapa kali node lain dipanggil dari node tersebut.

Graph yang berbentuk *Adjacency list* diubah menjadi *adjacency matrix*, proses normalisasi data dilakukan pada matriks. Tujuan normalisasi data agar nilai *weight*(jumlah call) dari relasi berkisar dari 1 hingga 0. Semakin banyak jumlah call dilakukan maka nilai mendekati 1, kemudian matriks tersebut dibuat menjadi *Distance Matrix*, rumus jarak yang digunakan ada 2 yaitu *Jaccard* dan *Struktural Similarity*.

Tugas akhir ini menggunakan library SciPy untuk melakukan proses clustering yang memiliki fungsi *Hierarchical Clustering*. Pemilihan pengelompokan dengan *hierarchical agglomerative clustering* dibandingkan *partition clustering* karena tidak mudah untuk mengetahui jumlah ideal cluster. Untuk menentukan metode *linkage* tugas akhir ini menggunakan *single linkage*, *average linkage*, dan *complete linkage*. Hasil dari masing-masing *linkage* dipilih jumlah partisi yang ideal untuk *microservice*.

Pemilihan jumlah partisi perlu dilakukan dengan perhitungan yang dapat menentukan jumlah *service* yang ideal. *Microservice* yang ideal memiliki nilai *coupling* yang rendah dan nilai *cohesion* yang tinggi. Untuk itu tugas akhir ini menentukan partisi dengan nilai *structural*. Pemilihan partisi harus mempertimbangkan nilai *cohesion*, nilai *coupling*, jumlah *service*, dan apakah *service* tersebut seimbang. Partisi yang akan menjadi *service* diharapkan bisa independen.

Odoo adalah aplikasi ERP berbasis web, tampilan pada Odoo bisa dibuka pada browser yang kompatibel. Proses pemisahan membutuhkan *reverse proxy* seperti *API Gateway* Kong yang dapat menghubungkan *client* dengan banyak *server*. Tujuan adanya *reverse proxy* agar *client* hanya perlu mengetahui satu pintu masuk aplikasi. Tugas akhir ini juga menggunakan 2 strategi yaitu pola *Strangle* dan pola *Branch by Abstraction* dalam pemisahan kode aplikasi. Proses komunikasi antar *service* dilakukan melalui JSON-RPC.

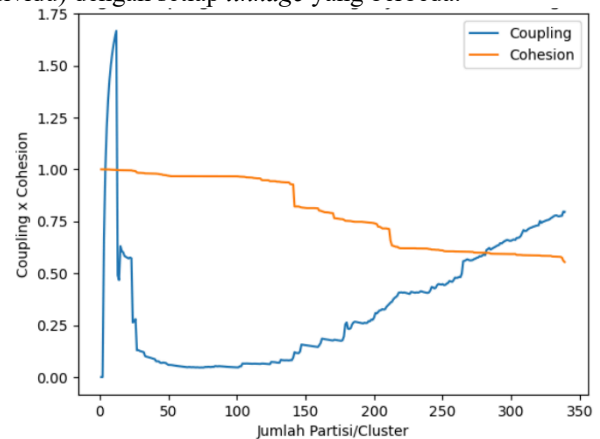
Pemisahan *database* dilakukan setelah dilakukan pemisahan kode karena pada Odoo sudah terdapat ORM yang mengelola *database*.

III. HASIL DAN PEMBAHASAN

Pada bagian ini, dibahas mengenai evaluasi dan pengujian tentang bagaimana hasil dekomposisi aplikasi monolitik menjadi *microservice* dengan menggunakan proses *Hierarchical Clustering*.

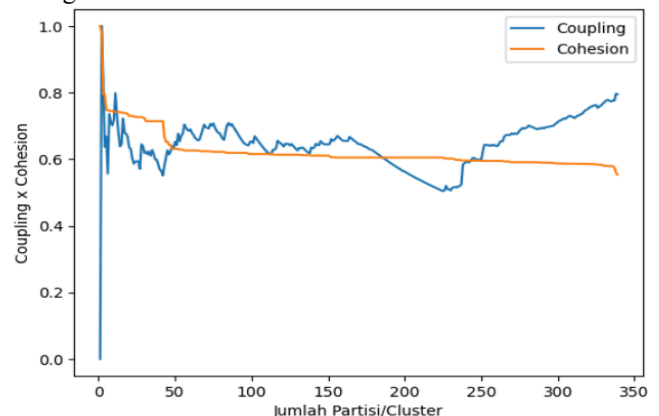
A. Pemilihan Partisi

Hasil Implementasi dapat divisualisasikan dan dianalisis lebih lanjut agar dapat menentukan *linkage* dan jumlah *service* yang ideal. Untuk itu dilakukan perbandingan bagaimana nilai *coupling* dan *cohesion* dari jumlah *service*/partisi yang dimulai dari 1 (monolit) hingga 335 (semua partisi menjadi *service* individu) dengan setiap *linkage* yang berbeda.



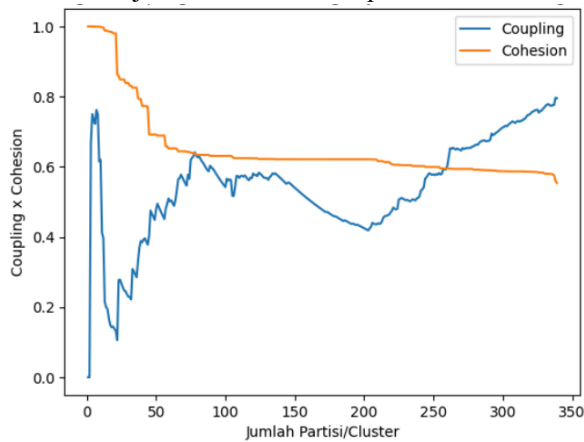
Gambar 4 Perbandingan *Coupling & Cohesion* menurut pendekatan *Single Linkage*

Pada gambar 4, hasil dekomposisi menggunakan *linkage single* memiliki hasil yang buruk untuk *coupling* tetapi masih memiliki *cohesion* yang bagus. Nilai *coupling* menurun drastis ketika jumlah *service* mencapai 25 *service*. Jumlah *service* yang memiliki nilai *coupling* rendah dan *cohesion* yang tinggi berkisar dari 25 *service* ke atas dan dibawah 150 *service*. Ketika jumlah *service* melebihi 150 *service* nilai *coupling* terus meningkat ketika dan nilai *cohesion* terus memburuk.



Gambar 5 Perbandingan *Coupling & Cohesion* menurut pendekatan *Complete Linkage*

Pada gambar 5, hasil dekomposisi menggunakan *linkage complete* memiliki *coupling* yang tinggi sejak dari jumlah *service* 1 ke atas hingga sekitar 175 *service*. Nilai *Coupling* membaik ketika jumlah *service* mencapai ± 200 hingga ± 240 , ketika jumlah *service* melebihi ± 240 nilai menjadi semakin memburuk *coupling*. Untuk nilai *cohesion* turun secara perlahan ketika jumlah *service* mencapai ± 50 .



Gambar 6 Perbandingan *Coupling* & *Cohesion* menurut pendekatan *Average Linkage*

Pada gambar 6, hasil dekomposisi menggunakan *Average linkage* memiliki nilai *coupling* yang cukup tinggi ketika jumlah *service* sebesar ± 25 . Nilai *coupling* meningkat secara tajam ketika jumlah *service* diatas ± 2 dan mulai menurun saat jumlah *service* berkisar ± 100 - ± 210 . Ketika jumlah *service* melebihi ± 210 nilai *coupling* semakin memburuk. Untuk nilai *cohesion*, memiliki nilai yang tinggi saat jumlah *service* berkisar ± 2 - ± 25 , ketika jumlah *service* diatas ± 25 nilai *cohesion* menurun drastis hingga jumlah *service* ± 75 . Nilai *cohesion* tidak lagi meningkat dan turun perlahan ketika jumlah *service* bertambah banyak.

Nilai *coupling* dan *cohesion* yang dihasilkan bisa dilakukan analisis lebih lanjut seperti melihat nilai maksimum, nilai minimum, dan nilai rata-rata pada nilai *coupling* dan *cohesion* untuk setiap *linkage*.

Table I

Nilai maximum, rata-rata, dan minimum untuk setiap *linkage*

| Linkage | Coupling | | | Cohesion | | |
|----------|----------|------|------|----------|------|------|
| | MIN | AVG | MAX | MIN | AVG | MAX |
| Single | 0.04 | 0.33 | 1.6 | 0.61 | 0.83 | 1 |
| Complete | 0.40 | 0.65 | 1.33 | 0.61 | 0.67 | 0.99 |
| Average | 0.14 | 0.51 | 0.83 | 0.61 | 0.7 | 1 |

Dari tabel 1 dapat dilihat *linkage complete* memiliki *coupling* yang tinggi dan *cohesion* yang rendah dibandingkan *linkage* lainnya. Untuk *linkage average* memiliki nilai *coupling* dan nilai *cohesion* yang lebih baik bila dibandingkan dengan *linkage complete* tetapi lebih buruk bila dibandingkan dengan *linkage single*. Nilai minimum dan maksimum dari *cohesion* memiliki nilai yang sama tetapi memiliki rata-rata yang berbeda.

Untuk mengetahui lebih detail bagaimana struktur *service* dikelompokkan, maka untuk setiap *linkage*, jumlah *module* dari

setiap partisi/*service* dicari nilai rentang, minimum, dan maksimum untuk setiap jumlah ukuran partisi tertentu (1, 2, 5, 10, 15, 25, 50, 75, 90, 110, 130, 150, 175, 195, 200, 208, 220, 232, 245, 275, 290, 305, 325, 335). Di mana pewarnaan untuk nilai *coupling* dan *cohesion* ditentukan bila *coupling* lebih tinggi dari *cohesion* maka berwarna merah, bila *coupling* lebih rendah dari *cohesion* tetapi perbedaannya tidak cukup signifikan (0.10) maka berwarna kuning dan bila cukup signifikan berwarna hijau. Sementara pewarnaan nilai rentang ditentukan bila bernilai lebih dari 12 *module* maka berwarna merah dan bila kurang maka berwarna hijau karena sebuah *service* memiliki karakteristik kecil dan berfokus pada satu hal.

Table II

Perbandingan Ukuran *Service* yang dihasilkan oleh *Single Linkage*

| Ukuran Service | Coupling | Cohesion | Statistik Jumlah Module | | |
|----------------|----------|----------|-------------------------|-----|-------|
| | | | MIN | MAX | RANGE |
| 1 | 0.0 | 1 | 335 | 335 | 0 |
| 2 | 0.0 | 1 | 2 | 333 | 331 |
| 5 | 1.2 | 1 | 1 | 326 | 325 |
| 10 | 1.6 | 1 | 1 | 319 | 318 |
| 15 | 0.63 | 1 | 1 | 315 | 314 |
| 25 | 0.48 | 0.99 | 1 | 299 | 298 |
| 50 | 0.09 | 0.97 | 1 | 246 | 245 |
| 75 | 0.11 | 0.97 | 1 | 246 | 245 |
| 90 | 0.11 | 0.97 | 1 | 241 | 240 |
| 110 | 0.06 | 0.96 | 1 | 216 | 215 |
| 130 | 0.07 | 0.94 | 1 | 190 | 189 |
| 150 | 0.1 | 0.91 | 1 | 169 | 168 |
| 175 | 0.18 | 0.85 | 1 | 133 | 132 |
| 195 | 0.23 | 0.81 | 1 | 96 | 95 |
| 200 | 0.25 | 0.79 | 1 | 89 | 88 |
| 208 | 0.28 | 0.76 | 1 | 66 | 65 |
| 220 | 0.31 | 0.74 | 1 | 60 | 59 |
| 232 | 0.32 | 0.74 | 1 | 60 | 59 |
| 245 | 0.38 | 0.74 | 1 | 54 | 53 |
| 275 | 0.64 | 0.66 | 1 | 15 | 14 |
| 290 | 0.71 | 0.64 | 1 | 7 | 6 |
| 305 | 0.75 | 0.64 | 1 | 6 | 5 |
| 325 | 0.79 | 0.63 | 1 | 3 | 2 |
| 335 | 0.83 | 0.62 | 1 | 1 | 0 |

Table III

Perbandingan Ukuran *Service* yang dihasilkan oleh *Complete Linkage*

| Ukuran Service | Coupling | Cohesion | Statistik Jumlah Module | | |
|----------------|----------|----------|-------------------------|-----|-------|
| | | | MIN | MAX | RANGE |
| 1 | 0.0 | 1 | 335 | 335 | 0 |
| 2 | 1 | 0.98 | 11 | 324 | 331 |
| 5 | 0.72 | 0.8 | 7 | 212 | 205 |
| 10 | 0.71 | 0.74 | 7 | 132 | 125 |
| 15 | 0.7 | 0.74 | 2 | 107 | 105 |
| 25 | 0.68 | 0.73 | 2 | 88 | 86 |
| 50 | 0.63 | 0.7 | 1 | 67 | 66 |
| 75 | 0.65 | 0.68 | 1 | 34 | 33 |
| 90 | 0.67 | 0.66 | 1 | 19 | 18 |
| 110 | 0.65 | 0.66 | 1 | 19 | 18 |
| 130 | 0.62 | 0.66 | 1 | 16 | 15 |
| 150 | 0.71 | 0.65 | 1 | 16 | 15 |
| 175 | 0.69 | 0.65 | 1 | 14 | 13 |
| 195 | 0.62 | 0.65 | 1 | 12 | 11 |
| 200 | 0.61 | 0.65 | 1 | 9 | 8 |

| | | | | | |
|-----|------|------|---|---|---|
| 208 | 0.6 | 0.65 | 1 | 9 | 8 |
| 220 | 0.58 | 0.65 | 1 | 5 | 4 |
| 232 | 0.57 | 0.65 | 1 | 5 | 4 |
| 245 | 0.63 | 0.65 | 1 | 4 | 3 |
| 275 | 0.7 | 0.65 | 1 | 3 | 2 |
| 290 | 0.72 | 0.64 | 1 | 3 | 2 |
| 305 | 0.77 | 0.63 | 1 | 3 | 2 |
| 325 | 0.83 | 0.62 | 1 | 2 | 1 |
| 335 | 0.83 | 0.62 | 1 | 1 | 0 |

Table IV

Perbandingan Ukuran Service yang dihasilkan oleh Average Linkage

| Ukuran Service | Coupling | Cohesion | Statistik Jumlah Module | | |
|----------------|----------|----------|-------------------------|-----|-------|
| | | | MIN | MAX | RANGE |
| 1 | 0.0 | 1 | 335 | 335 | 0 |
| 2 | 0.0 | 1.0 | 2 | 333 | 331 |
| 5 | 0.73 | 1.0 | 1 | 329 | 328 |
| 10 | 0.32 | 0.99 | 1 | 305 | 304 |
| 15 | 0.2 | 0.98 | 1 | 293 | 292 |
| 25 | 0.26 | 0.91 | 1 | 198 | 197 |
| 50 | 0.37 | 0.78 | 1 | 74 | 73 |
| 75 | 0.43 | 0.72 | 1 | 29 | 28 |
| 90 | 0.46 | 0.71 | 1 | 29 | 28 |
| 110 | 0.41 | 0.71 | 1 | 29 | 28 |
| 130 | 0.48 | 0.67 | 1 | 16 | 15 |
| 150 | 0.57 | 0.67 | 1 | 13 | 12 |
| 175 | 0.5 | 0.67 | 1 | 13 | 12 |
| 195 | 0.46 | 0.67 | 1 | 13 | 12 |
| 200 | 0.44 | 0.67 | 1 | 13 | 12 |
| 208 | 0.43 | 0.67 | 1 | 13 | 12 |
| 220 | 0.48 | 0.66 | 1 | 10 | 9 |
| 232 | 0.52 | 0.65 | 1 | 6 | 5 |
| 245 | 0.57 | 0.65 | 1 | 5 | 4 |
| 275 | 0.69 | 0.64 | 1 | 3 | 2 |
| 290 | 0.71 | 0.64 | 1 | 3 | 2 |
| 305 | 0.76 | 0.63 | 1 | 3 | 2 |
| 325 | 0.83 | 0.62 | 1 | 2 | 1 |
| 335 | 0.83 | 0.62 | 1 | 0 | 0 |

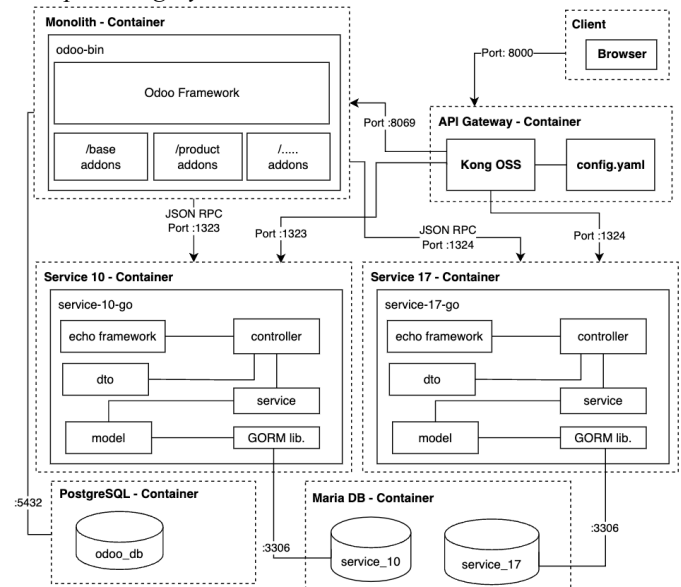
Dari hasil perbandingan ukuran service yang dihasilkan oleh masing-masing linkage pada kasus dekomposisi aplikasi monolitik Odoo, menunjukkan bahwa *complete linkage* tidak cocok untuk melakukan dekomposisi karena service yang dibentuk besar dan memiliki nilai *coupling* tinggi. Sedangkan *single linkage* kurang ideal walaupun memiliki *coupling* yang rendah dan *cohesion* yang tinggi namun service yang dibuat tidak dikelompokkan secara seimbang dan memiliki nilai rentang yang besar, karena ada service yang memiliki jumlah sangat besar. Untuk *Average linkage* dapat membentuk service yang ideal karena ukuran service tidak terlalu besar dilihat dari nilai rentangnya dan memiliki nilai *coupling*-nya rendah serta nilai *cohesion*-nya cukup tinggi.

Pemilihan jumlah service yang ideal pada *Average linkage*, tidak bisa ditentukan secara pasti karena terdapat *trade-off* untuk setiap kenaikan jumlah service. Oleh karena itu rentang jumlah service yang ideal adalah bisa dari 175, 195, 208, 200, 220, 232, 245. Jumlah service yang tidak ideal adalah dibawah $\pm 150/ \pm 175$ service karena nilai rentang isi module service cukup tinggi yaitu diatas 12 module dan diatas ± 245 service karena nilai *coupling* lebih tinggi dibandingkan nilai *cohesion*-nya.

B. Dekomposisi Monolitik ke Microservice

Pada tugas akhir ini memilih untuk menggunakan hasil *Hierarchical Clustering* dengan *Average Linkage* dengan jumlah service sebesar 200. Hasil evaluasi sebelumnya menunjukkan nilai *coupling* sebesar 0.67, nilai *cohesion* sebesar 0.44 dan nilai rentang jumlah modulnya sebesar 12.

Service yang dipilih untuk dilakukan implementasi dekomposisi yaitu service ke-10 yang memiliki module *addons.product* dan *addons.point of sale*, kemudian service ke-17 yang memiliki modul *calendar*. Model yang dibuat pada tugas akhir ini yaitu model '*calendar event type*', '*product tag*', dan '*pos category*'.



Gambar 7 Architectural Diagram Microservice yang dibangun

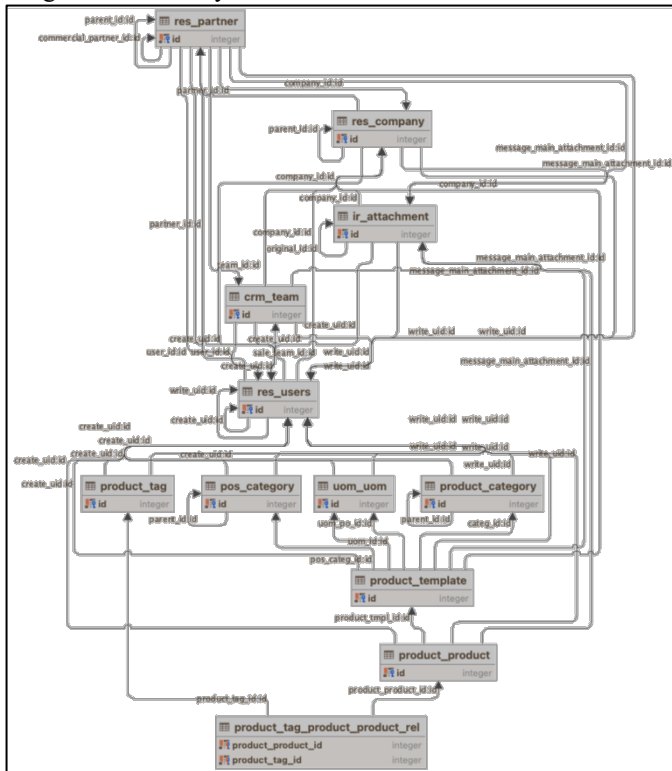
Seperti pada gambar 7, dapat dilihat arsitektur *Microservice* yang dibangun dengan penerapan pola *strangle fig* dan pola *Branch by Abstraction*. Untuk mengakses aplikasi, *client*/pengguna harus menggunakan browser dimana pengguna mengakses *endpoint* menggunakan port :8000. Semua server pada *microservice* ini dijalankan secara *containerization* dengan teknologi Docker. Permintaan dari *client* dikelola oleh Kong sebagai *API Gateway*, Kong yang dikonfigurasi dengan file *config.yaml* dan *DB-less*, sehingga Kong memiliki 3 service yang terdaftar yaitu *odoo-monolitik*, *service-10*, dan *service-17* serta masing-masing service ini memiliki *routes*-nya sendiri. Kong akan mengarahkan *request* dari *client* ke service yang memiliki *endpoint* yang cocok. Pada service-10 dan service-17 terdapat beberapa route yang pasti seperti **/web/dataset/call kw/pos.category/create**. Apabila route tidak ditemukan pada service-10 dan service-17 maka *request* diarahkan ke aplikasi monolitik.

Service-10 dan Service-17 dibangun dengan bahasa pemrograman Go, menggunakan *echo* sebagai *web framework* dan library *GORM* sebagai *ORM*. Kedua service ini terhubung pada satu *database management system* (DBMS) server yang sama yaitu *mariadb* dimana didalamnya terdapat 2 database yang terpisah untuk setiap service sedangkan untuk aplikasi monolitik menggunakan *PostgreSQL* sebagai *DBMS*. Aplikasi

monolitik dapat berkomunikasi dengan Service-10 dan Service-17 melalui JSON-RPC, pemanggilan tersebut dilakukan oleh adapter yang dibuat di aplikasi monolitik pada fitur yang dipindahkan.

Hasil Dekomposisi yang sudah diimplementasikan dengan 2 sampel *service* yaitu *service* ke-10 dan *service* ke-17. Di dalam *microservice* yang dibangun terdapat 3 aplikasi yaitu aplikasi monolitik yang sudah dimodifikasi, *Service* ke-10(*Product* dan *Point of Sale*) dan *Service* ke-17 (*Calendar*). Untuk mengetahui apakah bagian yang didekomposisi oleh proses *Hierarchical Clustering* relevan dan dapat diimplementasikan maka dapat dilihat melalui struktur tabel yang terbentuk ketika melakukan implementasi.

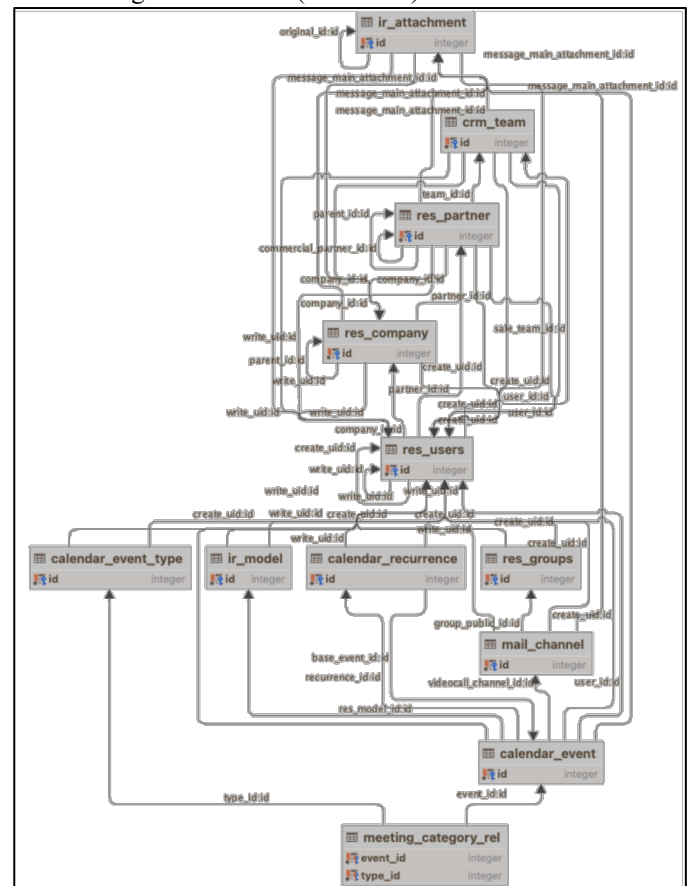
Pada *service* ke-10 terdapat 2 model yang diimplementasikan yaitu *product.tag* dan *pos.category*, Odoo yang memiliki ORM sudah otomatis mengubah nama "dot" menjadi "underscore" sehingga bila dilihat dari nama tabel yang terbentuk akan memiliki nama "product tag" dan "pos_category". Berikut adalah diagram *database* (Gambar 8) yang menunjukkan bagaimana keterhubungan antara model dengan model lainnya.



Gambar 8 Diagram Database dari Product Tag dan Pos Category

Hasil proses *clustering* sebelumnya mengelompokkan 2 module yaitu *product* dan *point of sale*(POS) menjadi *service* sendiri. Dari diagram *database* menunjukkan bahwa memang benar model *product.tag* dan *pos.category* memiliki keterhubungan satu sama lain yaitu melalui model *product.template* untuk itulah mengapa 2 *modules* ini disatukan menjadi *service* yang sama. Pada *service* ke-17 terdapat 1 model yang diimplementasikan yaitu 'calendar.event.type' dan nama yang terbentuk pada tabel adalah 'calendar event type'.

Service ke-17 ini memiliki isi *module* yang berbeda dengan *service* ke-10, jika dilihat berdasarkan nama modulnya didalamnya. Berikut keterhubungannya yang dapat dilihat melalui diagram *database* (Gambar 9).



Gambar 9 Diagram Database dari Calendar Event Type

Diagram *database* menunjukkan tidak ada hubungan dengan tabel yang terbentuk dari *module product* ataupun *point of sale*(POS), sehingga *service* ke-17 bisa berdiri sendiri. Dari diagram juga terlihat bahwa ke dua *service* ini memiliki hubungan yang sama dengan module lain seperti dengan *ir.attachment*, *res.users*, *res.groups*, *res.company*, dan *ir.model*.

IV. SIMPULAN

Berikut kesimpulan dan rangkuman dari penerapan *Microservice* dengan *Hierarchical Clustering* untuk dekomposisi monolitik pada *Enterprise Resource Planning* yang sudah dilakukan.

Berdasarkan Tabel 1, Tabel 2, dan Tabel 3 menunjukkan *linkage* yang cocok untuk membuat kelompok *service* yang memiliki rentang jumlah *module* yang kecil tetapi masih memiliki *coupling* dan *cohesion* yang baik adalah *Average linkage*.

Berdasarkan gambar perbandingan *coupling* dan *cohesion*. Menunjukkan *Hierarchical Clustering* dapat membuat *Microservice* yang memiliki nilai *coupling* yang kecil dan nilai *cohesion* yang tinggi dengan jumlah partisi tertentu. Jumlah partisi yang ditemukan dapat membuat *Microservice* yang ideal yaitu dimulai dari 175-245 *services* dengan *Average Linkage*.

Berdasarkan Gambar 8 dan Gambar 9 dapat dilihat dari struktur *database* yang terbentuk ketika dilakukan implementasi. Struktur tabel menunjukkan *Hierarchical Clustering* dapat memisahkan *module* yang tidak terhubung dan *service* yang memiliki hubungan kuat dikelompokkan menjadi satu seperti pada *service/partisi* ke-10 pada *Module Product* dan *Module Point of Sale*. Kemudian *module* yang tidak memiliki hubungan dengan *module* lain seperti *Module Calendar* di *partisi* ke-17 tidak dikelompokkan menjadi satu dengan *Module Product* atau *Module Point of Sale*, sehingga hasil dari proses *Hierarchical Clustering* relevan.

Aplikasi Odoo dibangun menggunakan bahasa pemrograman *Python*. Dimana *Python* merupakan bahasa pemrograman yang mendukung *multi-paradigm* seperti *procedural*, *object-oriented*, atau *functional*, sehingga metode dekomposisi menggunakan *Hierarchical Clustering* tidak terpengaruh pada paradigma yang digunakan dalam pemrograman selama proses pembuatan *graph* sebelumnya dilakukan dengan tepat. Hasil implementasi yang dilakukan pada menggunakan 3 model yaitu *PosCategory*, *ProductTag*, dan *MeetingType*. Dimana ukuran *service* yang terbentuk berukuran kecil sehingga memiliki performa lebih baik dibandingkan monolitik.

Dalam penelitian lanjutan, Pengelompokan dapat mempertimbangkan tipe *coupling* yang berbeda dalam menentukan suatu hubungan antara objek, menggunakan beberapa metode *linkage* lainnya yang dapat digunakan ketika melakukan proses *Hierarchical Clustering*. Hasil *clustering* yang tidak relevan dapat diperbaiki dengan meningkatkan akurasi *graph* yang dibuat, dan ketika rancangan modul di aplikasi yang tidak jelas dalam batasan antar komponennya dapat menyebabkan hasil *clustering* tidak akurat dan penentuan *scope* objek sebelum dilakukan proses *Hierarchical Clustering* lebih sulit selain itu diperlukan kasus uji nyata lebih lanjut yang dapat melihat dampak performa seperti CPU dan memori dari pembentukan *service* oleh *Hierarchical Clustering*.

DAFTAR REFERENSI

- [1] Amini, Mohammad and Abukari, Arnold. (2020). "ERP Systems Architecture For The Modern Age: A Review of The State of The Art Technologies." *Journal of Applied Intelligent Systems and Information Sciences*. Volume 1(2), pp.70-90. Available: <https://doi.org/10.22034/jaisis.2020.232506.1009>.
- [2] Bender, B.; Bertheau, C. and Gronau, N. (2021). "Future ERP Systems: A Research Agenda." In *Proceedings of the 23rd International Conference on Enterprise Information Systems*. 2, pp.776-783. Available: <http://dx.doi.org/10.5220/0010477307760783>.
- [3] Slamaa, A.A., El-Ghareeb, H.A. , Saleh, A.A. (2021). "A Roadmap for Migration System-Architecture Decision by Neutrosophic-ANP and Benchmark for Enterprise Resource Planning Systems." *IEEE Access* . 9,pp.48583-48604.Available: <https://doi.org/10.1109/ACCESS.2021.3068837>.
- [4] Sam Newman, *Monolith to Microservices*, Sebastopol, CA: O'Reilly Media, Inc., 2020, pp. 12-15
- [5] Khaled Sellami, Mohamed Aymen Saied, and Ali Ouni. (2022). "A Hierarchical DBSCAN Method for Extracting Microservices from Monolithic Applications" In *The International Conference on Evaluation and Assessment in Software Engineering 2022 (EASE 2022)*. ACM, New York, NY, USA, 11. Available: <https://doi.org/10.1145/3530019.3530040>.
- [6] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony, "From monolithic architecture style to Microservice one based on a semi-automatic approach," 2020 IEEE International Conference on Software Architecture (ICSA), 2020.
- [7] Jain, A.K. and Dubes, R.C. (1988) *Algorithms for clustering data*. Englewood Cliffs, NJ: Prentice Hall.
- [8] Chaitanya K. Rudrabhatla. (2020). "Impacts of Decomposition Techniques on Performance and Latency of Microservices." *International Journal of Advanced Computer Science and Applications(IJACSA)*. 11(8). Available: <http://dx.doi.org/10.14569/IJACSA.2020.0110803>.
- [9] A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee, "Mono2micro: A practical and effective tool for decomposing monolithic Java applications to microservices," *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021.
- [10] Richardson, C. (2018) *Microservice patterns: With examples in Java*. Shelter Island, NY: Manning Publications.
- [11] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, and J. Sander, "Decomposing object-oriented class modules using an agglomerative clustering technique," 2009 IEEE International Conference on Software Maintenance, 2009.

Albertus Septian Angkuw, menerima gelar Sarjana Komputer dari Institut Teknologi Harapan Bangsa di Bandung tahun 2023.

Hans Christian Kurniawan, menerima gelar Sarjana Teknik Informatika dari Institut Teknologi Harapan Bangsa di Bandung pada tahun 2016 dan Magister Informatika dari Institut Teknologi Bandung pada tahun 2019. Saat ini aktif sebagai pengajar di ITHB serta sebagai Engineering Manager di Bukalapak.