PROTOTYPE SISTEM *CRAWLING* TERDISTRIBUSI BERBASIS *PEER TO PEER* YANG DIIMPLEMENTASIKAN MENGGUNAKAN *SOCKET PROGRAMMING*

Muhammad Ridho Rizqillah[1], Muhammad Eka Suryana[2], Med Irzal[3]
Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Jakarta, Jakarta Timur, Indonesia
mridhor08@gmail.com[1], eka-suryana@unj.ac.id[2], medirzal@unj.ac.id[3]

***Abstrak*— Mesin pencari memerlukan jumlah data yang besar untuk berjalan dengan optimal. Data yang besar diperlukan proses *crawling* yang masif. Penelitian ini mencoba meningkatkan efisiensi *crawling* dengan mengimplementasikan *crawler* terdistribusi menggunakan socket sebagai media komunikasi antar perangkat. Sistem arsitektur crawler terdistribusi mencakup tracker, manager, dan client. Tujuan utamanya adalah untuk meningkatkan efisiensi dan efektivitas *crawling* dengan cara terdistribusi. Hasil akhir dari improvisasi ini menunjukkan peningkatan data sebanyak 30% secara terdistribusi dengan dua *crawler*, dibanding dengan *crawler* individual. Serta tidak adanya data yang terduplikasi.**

Kata Kunci*— *Search Engine, Distributed Crawler, Socket Programming, Crawling.

I. PENDAHULUAN

Mesin pencari atau *search engine* adalah sebuah teknologi yang digunakan untuk mencari sebuah informasi yang tersedia di internet. Dengan memasukkan *keyword* terkait hal yang ingin diketahui dan akan terlihat berbagai macam situs *web* yang memiliki informasi sesuai. Perkembangan *search engine* melahirkan penelitian yang berkelanjutan, meneliti bagaimana menciptakan algoritma yang efektif untuk menjalankan *search engine* agar menampilkan pencarian yang relevan kepada pengguna.

Penelitian *search engine* yang dilakukan sebelumnya lebih terfokus untuk mengembangkan *crawler* yang dapat berjalan secara *multi-threaded* [11]. Sedangkan pada penelitian selanjutnya mengarah pada *refactoring crawler* dari penelitian sebelumnya dan mencari similarity score dari setiap halaman situs *web* yang di-*crawling* [7]. Dan penelitian lainnya saat melakukan pengumpulan data menggunakan *crawler* yang sudah ada agar dapat dikelola dalam bentuk *user interface* [1].

Kelemahan yang terdapat pada penelitian tersebut adalah *crawling* hanya dapat berjalan pada satu perangkat saja. Hal itu dapat menghambat hasil dari *crawling*, karena membutuhkan waktu lebih lama untuk mendapatkan hasil *crawling* yang masif. Oleh karena itu, diperlukan cara agar crawler dapat berjalan pada berbagai macam perangkat atau terdistribusi satu dengan lainnya. Serta kelemahan yang didapatkan bahwa proses *crawling* dengan *public Internet Protocol (IP) address* dapat menyebabkan *IP public* perangkat terblokir oleh Cloudflare atau layanan penyedia internet, sebab dianggap perilaku mencurigakan.

Diperlukan improvisasi pada algoritma dan menambahkan cara untuk membagi tugas *crawling* supaya menjadi versi terdistribusi. *Crawling* terdistribusi adalah melakukan *crawling* secara terpisah ke setiap perangkat yang tersedia, guna membagi beban kerja dan dapat menghasilkan hasil *crawler* dalam jumlah yang masif secara efisien serta efektif. Sebelum melakukan improvisasi *crawler* yang sekarang menjadi terdistribusi, diperlukan pemahaman terkait sistem arsitekturnya.

Sistem arsitektur crawler terdistribusi memiliki dua komponen penting, yaitu *crawling* system dan *crawling* application. *Crawling System* meliputi *crawl manager*, *downloader*, dan *Domain Name System (DNS) resolvers*. Sedangkan *crawling application* yang melakukan proses *crawling* yang berjalan pada perangkat.

Berdasarkan arsitektur sederhana didapatkan bahwa proses distribusi yang terjadi adalah pada sisi downloader. Setelah *crawl manager* melakukan *crawling* pada suatu *web* yang akan menghasilkan *urls*, kemudian *url* tersebut dilanjutkan kepada masing - masing *downloader*. *Downloader* ini terletak pada server atau perangkat yang berbeda [13].

Socket dapat dianggap sebagai endpoint dalam *two-way communication channel*. Socket dapat melakukan komunikasi antara dua proses, baik dalam satu perangkat atau perangkat yang berbeda. Untuk dapat melakukan komunikasi dua arah, terlebih dahulu dipahami arsitektur *client-server*. *Client* dapat berupa PC, laptop, telepon seluler, dll. Dan setiap *client* memiliki *IP address*-nya masing - masing. Untuk dapat melakukan komunikasi atau pertukaran data diperlukan setiap client untuk terhubung ke sebuah server. Server pun juga memiliki *IP address*. Agar *client* dapat terhubung ke server, maka ia harus mengetahui *IP server*. Dan setiap *client* yang terhubung ke server akan dapat berkomunikasi via server dalam hal ini akan menggunakan socket [6].

Dalam *web search engine* terdiri dari *spidering system*. Kesatuan dari sistem ini berjalan secara paralel, yang mana crawler akan mengunjungi setiap web dan mengumpulkan informasi yang diperlukan. Inti utama dalam *Information Retrieval (IR)* adalah *indexer* dan *query analyzer*. Pada tahapan *query analyzer* melakukan proses paralel terhadap *Query Brokers* dan *Local Searchers*. Dilakukan secara terdistribusi menggunakan teknik *document partitioning*. Strategi untuk membuat paralel *web search engine* harus memenuhi dua hal, *task parallel* dan *data parallel*. Untuk menghindari eksploitasi *processor* secara berlebihan, maka diperlukan *load balancer*, agar setiap *query* dapat berjalan secara efisien dan efektif. Dengan melakukan pembagian *task* secara paralel. Oleh karena setiap query diproses secara terpisah, maka juga memiliki partisi *database*-nya tersendiri. Untuk itu diperlukan kombinasi antara *task* dan *data parallel*, agar hasil akhir dari proses *search engine* lebih relevan [8].

Pada penelitian lain dijelaskan juga mengenai distribusi algoritma, walaupun diterapkan pada perhitungan *page rank* agar lebih efisien memori dalam melakukan perhitungan. Karena untuk perhitungannya diperlukan memori yang besar. Oleh karena itu, diperlukan proses efisiensi memori [9]. Walaupun begitu data perhitungan akan berkurang karena efisiensi memori, memang ada hal yang menjadi timbal baliknya. Nanti hasil dari penelitian tersebut juga akan diterapkan untuk penelitian lebih lanjut setelah penelitian yang penulis lakukan.

Berdasarkan beberapa rujukan sebelumnya, hasil dari penelitian ini adalah sebuah *crawler* yang berjalan secara terdistribusi yang akan mengimplementasikan penggunaan socket dalam mendistribusikan tugas ke setiap perangkat, serta peningkatan jumlah data yang terkumpul.

II. KAJIAN PUSTAKA

A. *Crawler* Individual

Crawler individual adalah *crawler* yang dapat berjalan di satu perangkat. Tidak adanya komunikasi dan koordinasi dengan perangkat lain. Skema *crawler* individual yang dikembangkan (Khatulistiwa 2023), terfokus pada perancangan arsitektur *search engine* yang mana akan diintegrasikan oleh web *crawler*, algoritma *page rank*, dan *document rangking.* Proses *crawling* yang terjadi menggunakan algoritma *Breadth First Search* (BFS), dan berjalan di satu perangkat.

Didapati bahwa proses *crawling* tersebut dapat diimprovisasi dengan menjadikan *crawler* secara terdistribusi. Karena permasalahan utama yang timbul dari *crawler* individual adalah proses *crawling* terlihat kurang masif dan terkoordinir. Dengan dikembangkan menjadi versi terdistribusi, yang mana banyak perangkat dapat berkoordinasi dan menjalankan *crawling* secara serempak tanpa adanya duplikasi data.

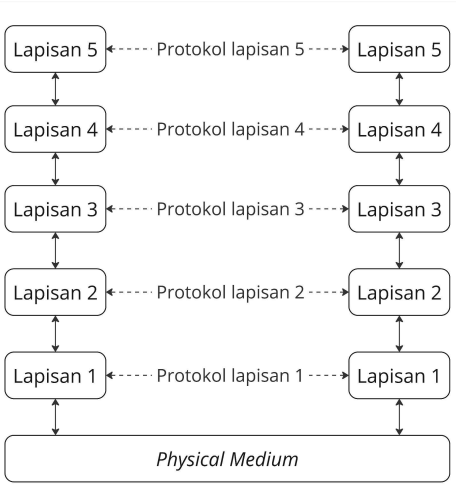
B. Jaringan Komputer

komputer adalah sekumpulan perangkat komputasi yang saling terhubung satu sama lain. Seperti dua komputer yang saling terhubung dan dapat bertukar informasi. Interkoneksi dapat berlangsung pada berbagai macam media transmisi, seperti kawat tembaga, kabel *fiber optic*, dan gelombang radio. Internet adalah contoh yang sering dijumpai sebagai jaringan dari banyak jaringan.

Peran jaringan komputer yang sangat umum adalah untuk mendapatkan akses terhadap suatu informasi. Cara pengaksesan informasi ini pun dapat melalui berbagai cara, menggunakan internet dengan *web browser*, mengakses informasi melalui media sosial. Bahkan dengan menggunakan smartphone sudah dapat mengakses banyak informasi dengan internet yang terhubung. Banyak informasi yang diakses melalui internet menggunakan model *client-server*, klien meminta informasi kepada server melalui jaringan, dan server mengembalikan informasi yang dibutuhkan oleh klien. Model ini sudah banyak dan secara luas diterapkan dalam proses mendapatkan informasi. Model lain yang cukup populer adalah *peer-to-peer*, dalam proses komunikasi *peer-to-peer* dapat langsung berhubungan dengan klien atau server dan tidak pasti siapa yang akan menjadi klien atau server. Jadi, keduanya dapat bertukar informasi secara langsung.

Jaringan juga memiliki protokol yang berarti aturan yang ada pada jaringan. Kegunaannya untuk menjaga agar jaringan yang digunakan dapat berjalan dengan baik. Dengan memperhatikan faktor *Reliability, Resource Allocation, Evolvability,* dan *Security.*

Untuk mengurangi kompleksitas dalam jaringan, dibentuk sebuah *layer* atau lapisan. Masing - masing lapisan terbentuk berdasar lapisan yang ada di bawahnya. Dengan pembuatan lapisan ini menyediakan layanan dan kebutuhan untuk lapisan berikutnya yang lebih tinggi, sementara melindungi detail penting dari setiap lapisan dan implementasinya. Hal ini mirip seperti konsep pada *Object Oriented Programming* (OOP) yang mana terdapat pilar - pilar yang terdiri dari *abstraction* dan *encapsulation*. Berguna untuk hanya menampilkan info yang penting dari setiap lapisan dan menjaga agar setiap lapisan memiliki fungsinya masing - masing.



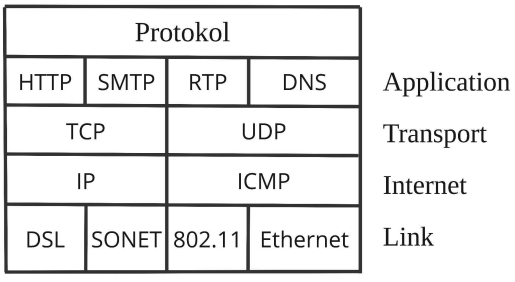
GAMBAR 1. LIMA LAPISAN JARINGAN

Entitas yang terdiri dari setiap lapisan itu disebut dengan peers. Peers dapat berupa proses perangkat lunak, perangkat keras, atau bahkan manusia. Setiap *peers* berkomunikasi menggunakan protokol. Proses transmisi data yang terjadi tidak serta merta langsung antara satu lapisan pada satu perangkat menuju lapisan yang sama pada perangkat yang lain. Melainkan prosesnya terjadi dari lapisan paling atas menuju ke lapisan paling bawah sampai mencapai physical medium, setelah itu baru komunikasi berlanjut pada lapisan itu. Garis putus - putus menunjukkan *virtual communication* melalui protokol dan garis tidak putus - putus menunjukkan *physical communication* antara tiap lapisan.

Berbagai lapisan menawarkan dua opsi layanan koneksi: *connection-oriented* dan *connectionless*. Pada *connection-oriented,* seperti halnya koneksi telepon. Untuk menggunakan telepon, pemanggil memasukan nomor tujuan dan menelponnya. Pada pihak penerima mendapatkan panggilan dan mengangkat telepon, barulah koneksi terhubung dan terjadi. pemanggil memberikan sebuah informasi dan diterima oleh penerima dan begitupun sebaliknya. Pada *connectionless*, seperti halnya surat fisik pada layanan pos. Pengirim mengirim surat kepada penerima dengan alamatnya. Pesan tersebut akan diterima dalam kurun beberapa hari, bisa sampai dengan tepat waktu atau lebih lambat. Jika penerima juga mengirim surat yang berbeda kepada pengirim pada waktu berikutnya, bisa saja surat dari penerima yang sampai duluan.

C. TCP/IP Model

TCP/IP Model banyak digunakan oleh jaringan komputer secara menyeluruh dari internet di seluruh dunia. Ketika satelit dan jaringan radio hadir, protokol yang ada memiliki masalah dalam beroperasi dengannya, jadi diperlukan model arsitektur yang baru. Oleh karena itu, kemampuan TCP/IP dari awal adalah untuk dapat menghubungkan beberapa jaringan dengan lancar yang merupakan salah satu tujuan utama dari model ini. Arsitektur ini kemudian dikenal sebagai TCP/IP *Reference Model*.



GAMBAR 2. TCP/IP MODEL DAN PROTOKOLNYA

1. *Link Layer*

   Sebagai lapisan paling bawah dalam model TCP/IP lapisan *link* sering dibandingkan dengan kombinasi dari lapisan *data link* dan *physical*. Fungsi dari lapisan ini adalah untuk menyediakan layanan kepada lapisan jaringan atau internet agar dapat mentransfer data dari lapisan jaringan yang berada pada *source machine* menuju *destination machine*. Dan menghubungkan antara *physical* dan *logical network.*

2. *Internet Layer*

   Lapisan ini bertugas untuk mengizinkan host untuk menginjeksi paket ke jaringan dan dikirim ke destinasi tujuan yang juga memungkinkan berada pada jaringan yang berbeda. Pada lapisan ini menjelaskan *official packet format* dan protokol yang disebut dengan *Internet Protocol* (IP) dan protokol pendamping *Internet Control Message Protocol* (ICMP). Perutean paket yang dikirim pada lapisan ini terkadang menjadi kendala dalam sisi manajemen yang menyebabkan kemacetan atau lalu lintas yang padat pada saat pengiriman paket. Hal ini dapat ditangani dengan bantuan lapisan yang lebih tinggi, yakni lapisan *transport*.

3. *Transport Layer*

   Lapisan ini didesain agar *peer* pada *source* dan *destination host* dapat berkomunikasi. Terdapat dua *end-to-end transport protocol* pada lapisan ini: *Transmission Control Protocol (TCP)* yang menggunakan connection-oriented, *User Datagram Protocol (UDP)* yang menggunakan *connectionless*. Tujuan utama dari lapisan ini adalah untuk menyediakan layanan yang *efficient, reliable,* dan *cost-effective* data transmission kepada penggunanya yang nanti akan diproses lebih lanjut pada lapisan *application*. Perangkat keras atau lunak yang melakukan tugas pada lapisan ini disebut dengan *transport* entity atau entitas transportasi.

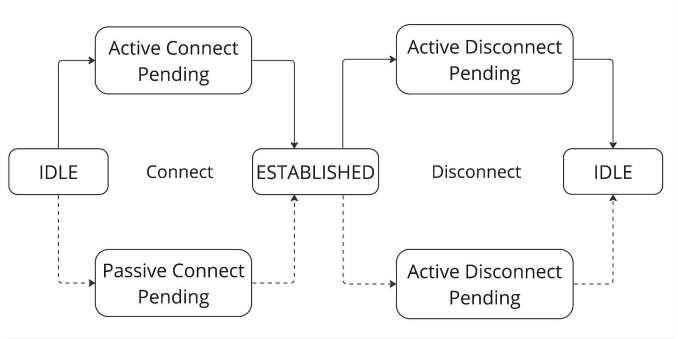
4. *Application Layer*

   Pada lapisan ini mengandung higher-level protocol: *virtual terminal* (TELNET), *file transfer* (FTP), *electronic mail* (SMTP), dan DNS. Hilangnya lapisan *session* dan *presentation* disatukan pada lapisan *application*, jadi tidak semata - mata hilang begitu saja. Karena kedua lapisan tersebut tidak banyak berguna untuk sebagian besar aplikasi, oleh karena itu dihilangkan.

D. *Socket Programming*

Socket pertama kali dirilis sebagai bagian dari Berkeley UNIX 4.2BSD distribusi perangkat lunak pada tahun 1983. Socket dengan cepat menjadi populer. *Primitives* itu sekarang digunakan untuk pemrograman internet pada banyak sistem operasi, terutama sistem berbasis UNIX, dan ada API (Application Programming Interface) bergaya socket untuk Windows disebut ‘winsock. *Primitive* adalah tipe data sederhana yang menjadi dasar dari semua tipe data lain.

Pada Gambar 3 adalah koneksi manajemen yang sederhana dengan model yang memiliki lebih banyak fitur dan fleksibel. Dengan adanya alur untuk menghubungkan koneksi jaringan dan juga memutus koneksi.



GAMBAR 3. SKEMA KONEKSI MANAJEMEN SEDERHANA

Terdapat beberapa *primitives* atau tipe data sederhana pada *Socket,* berdasarkan Tabel 1.

TABEL 1. *PRIMITIVE* PADA SOCKET

Primitive	**Berarti**
SOCKET	Membuat sebuah komunikasi endpoint baru
BIND	Mengasosiasi alamat lokal dengan socket
LISTEN	Bersedia untuk menerima koneksi dan membuat antrian
ACCEPT	Mengizinkan koneksi masuk dan terbentuk secara pasif
CONNECT	Melakukan percobaan untuk mendirikan koneksi secara aktif
SEND	Mengirimkan beberapa data melalui koneksi
RECEIVE	Menerima beberapa data dari koneksi
CLOSE	Melepaskan koneksi

Empat primitives pertama dalam daftar dieksekusi dalam urutan oleh server. *SOCKET primitive* membuat titik akhir baru dan mengalokasikan ruang tabel untuknya di dalam entitas transportasi. Parameter panggilan menentukan format pengalamatan yang akan digunakan, jenis layanan yang diinginkan (misalnya *reliable byte stream*), dan protokol. Panggilan socket yang berhasil mengembalikan file descriptor biasa untuk digunakan dalam panggilan yang berhasil, sama seperti panggilan *OPEN* pada file.

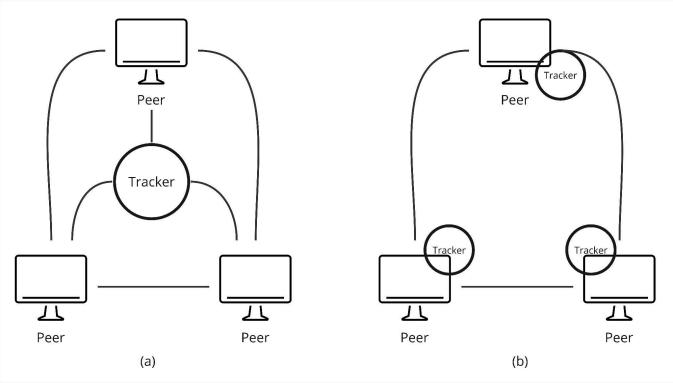
Socket yang baru dibuat tidak memiliki alamat jaringan. Oleh karena itu,menggunakan *BIND*. Setelah server mengikat alamat ke socket, klien jarak jauh dapat terhubung ke sana. Berikutnya adalah panggilan *LISTEN*, yang mengalokasikan ruang untuk mengantri panggilan, jika terdapat beberapa klien yang mencoba untuk terhubung pada waktu yang sama. Ketika segmen yang meminta koneksi tiba, entitas transportasi membuat socket baru dengan properti yang sama dengan yang asli dan mengembalikan *file descriptor* untuk itu. Server kemudian dapat memotong proses atau utas untuk menangani koneksi pada socket baru dan kembali menunggu koneksi berikutnya pada socket asli. *ACCEPT* mengembalikan file descriptor, yang dapat digunakan untuk membaca dan menulis dengan cara standar, sama seperti untuk file.

CONNECT memblokir pemanggil dan memulai proses koneksi. Ketika segmen yang sesuai diterima dari server, proses klien dibuka blokirnya dan koneksi dibuat. Kedua belah pihak sekarang dapat menggunakan *SEND* dan *RECEIVE* untuk mengirim dan menerima data melalui koneksi *full-duplex.* Panggilan sistem *UNIX READ* dan *WRITE* standar juga dapat digunakan jika tidak ada opsi khusus *SEND* dan *RECEIVE* yang diperlukan. Pelepasan koneksi dengan socket bersifat simetris. Ketika kedua belah pihak telah mengeksekusi *CLOSE*, sambungan dilepaskan atau dihentikan.

Socket juga memiliki opsi untuk menggunakan port yang sama baik untuk *BIND* dan juga *CONNECT.* Hal tersebut sangatlah berguna untuk menjaga koneksi agar terfokus untuk satu tugas saja. Baik itu terhubung dengan perangkat pengelola dan perangkat pekerja (yang melakukan *crawling*). Didapatkan bahwa *socket* dapat *listening multiple socket*, opsi ini dinamakan *SO_REUSEPORT,* dengan berjalan di-*thread* yang berbeda untuk pemrosesannya.

E. *Bittorrent*

BitTorrent adalah teknologi atau protokol yang membuat distribusi file, terutama file berukuran besar, lebih mudah dan hemat *bandwidth*. Hal ini dicapai dengan memanfaatkan kapasitas unggah *peers* yang mendownload file. Peningkatan pengunduh yang cukup signifikan hanya akan menghasilkan sedikit peningkatan beban pada server yang meng-*hosting* file tersebut. *Peers* adalah sekumpulan dari klien yang melakukan proses unduh dan unggah. Jadi, peer tidak hanya mendapati hasil unduhan dari *server* saja, tetapi juga dari *peer* lain yang melakukan proses unggah file. Proses unggah yang dilakukan dalam protokol Bittorrent dengan memotong *file* menjadi beberapa kepingan yang disebut pieces. Lalu setiap *peer* yang terhubung akan mendapatkan kepingan yang berbeda. Lalu masing - masing *peer* akan bertukar kepingan untuk melengkapi data dari file tersebut, menjadi satu kesatuan file yang utuh. Pada protokol ini, peer bertindak melakukan unduh dan unggah.



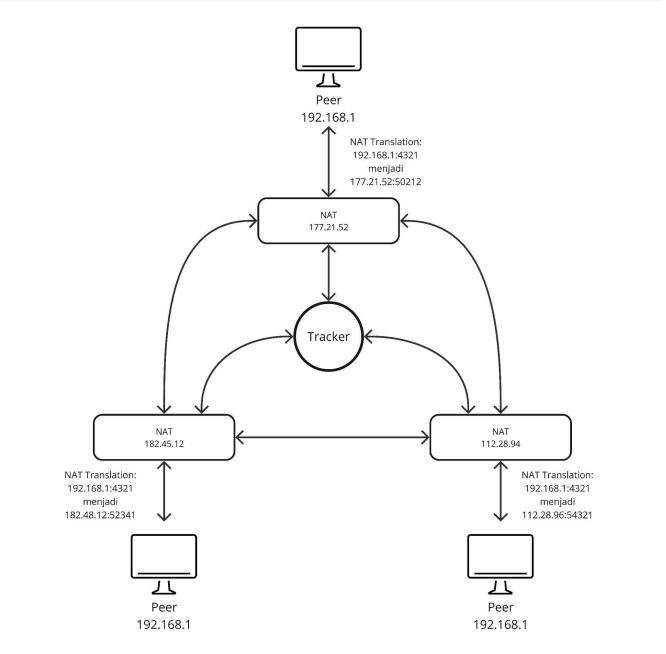
GAMBAR 4. (a) *CENTRALIZED TRACKER*, (b) *DECENTRALIZED TRACKER*

Pada Gambar 4 (a) adalah *centralized tracker* dan Gambar 4 (b) adalah *decentralized tracker*. *Tracker* atau pelacak menyimpan *log peers* yang saat ini mengunduh *file*, dan membantu mereka menemukan satu sama lain. *Tracker* tidak terlibat langsung dalam transfer data dan tidak memiliki salinan file. *Tracker* dan peer yang melakukan proses mengunduh bertukar informasi menggunakan protokol sederhana di atas HTTP. Pertama, pengguna memberikan informasi kepada *tracker* mengenai file mana yang diunduh, port yang digunakan, dll. Kemudian *tracker* merespon mengenai daftar peer lain yang mengunduh file yang sama dan informasi tentang cara menghubungi mereka.

Pada kedua konsep itu, tracker memiliki tugas yang sama. *Decentralized tracker* menjadi sebuah solusi untuk mengatasi kegagalan pada bittorrent dengan central *tracker*. Jika *tracker* tersebut mengalami kegagalan, maka proses distribusi data tidak dapat dilakukan. Dengan adanya *decentralized tracker* menjadi pemecah masalah tersebut. Karena setiap *peer* memiliki *tracker*-nya sendiri atau dapat berperilaku sebagai *tracker*. Solusi ini dikembangkan pada Mei 2005 dengan bittorrent versi 4.1 dan berkonsep pada *Distributed Hash Tables (DHT)*.

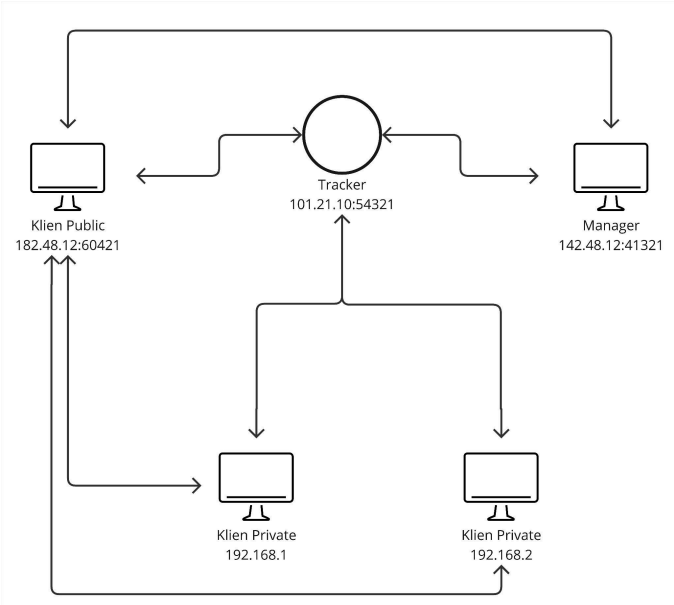
III. METODOLOGI PENGEMBANGAN

Pengembangan arsitektur *crawler* terdistribusi mengalami dua versi. Versi pertama dan kedua mengimplementasi beberapa arsitektur *Bittorrent* yang memiliki *tracker.* Versi pertama ingin mencapai proses komunikasi *peer to peer* langsung antara klien. Tetapi proses ini terkendala karena faktor jaringan *router* IPV4 yang mencegah komunikasi langsung antara *private IP* ke *private* IP. Diperlukan konfigurasi *router*, yang mana tidak semua perangkat memiliki akses untuk *router* tersebut. Dikarenakan terhambat oleh masalah itu, nantinya versi ini juga akan dikembangkan ke versi dua dalam pembahasan setelah ini. Berikut arsitektur versi pertama pada Gambar 5.



GAMBAR 5. ARSITEKTUR *CRAWLER* TERDISTRIBUSI VERSI 1

Akhirnya dibuatlah prototipe arsitektur versi kedua. Arsitektur ini mencoba mengadopsi proses *peer to peer,* walau tidak antar klien *private* IP. Serta pembagian tugas akan lebih jelas dan dapat dengan mudah di-*monitoring*.



GAMBAR 6. ARSITEKTUR *CRAWLER* TERDISTRIBUSI VERSI 2

Arsitektur *crawler* terdistribusi versi kedua dalam penelitian ini terdiri dari lima perangkat lunak dengan konfigurasi tiga perangkat memiliki *public IP address* dan dua perangkat memiliki *private IP address*. Dengan pembagian perannya masing -masing, satu *Tracker*, satu *Manager*, dan

tiga Klien. Pada penerapannya setiap program berjalan dengan bahasa pemrograman Python dan setiap proses dapat berjalan secara paralel dengan menggunakan *threading*. Setelah melakukan percobaan untuk mendapatkan kondisi *peer to peer*, maka dihasilkan arsitektur seperti Gambar 6.

Secara garis besar, pengembangan dan pengujian yang akan dilakukan pada penelitian ini adalah untuk menguji konsistensi data dan efisiensi dengan *crawler* terdahulu. Sumber data yang akan digunakan dalam penelitian ini adalah situs web berita dan data yang diambil itu mencakup konten teks dari halaman *web* serta tautan yang berada di *web* tersebut. Performa *crawler* terdistribusi akan dinilai menggunakan metrik berikut:

1. Proses Komunikasi: Proses komunikasi antara masing - masing perangkat dapat berjalan dengan sistematis.
2. Konsistensi Data: Konsistensi data mengacu pada kemampuan mengirimkan data lengkap tanpa kehilangan atau duplikasi informasi selama proses *crawling*. Konsistensi data penting untuk memastikan keakuratan dan kelengkapan informasi yang dikumpulkan di Internet. Duplikasi atau hilangnya data dapat menyebabkan analisis yang tidak akurat dan berdampak pada keputusan berdasarkan data tersebut. Membandingkan data yang dikumpulkan dengan sumber atau data yang diharapkan untuk memastikan tidak ada duplikasi.
3. Efisiensi dan Optimalisasi: Efisien sumber daya dan optimalisasi jumlah data yang terkumpul dari proses *crawling* terdistribusi dengan *crawling* tidak terdistribusi. Ini mencakup seberapa cepat sistem dapat mengumpulkan data. Karena proses *crawling* yang lambat atau penggunaan sumber daya yang tidak efisien dapat menghambat kinerja *crawler*. Dengan pengujian penyimpanan data, data yang dihasilkan oleh *crawler* terdistribusi dapat memiliki jumlah data yang lebih banyak dalam kurun waktu yang sama, yaitu satu jam.

Skema uji ini akan membantu dalam membandingkan kinerja antara *crawler* terdistribusi dan terdahulu (tanpa terdistribusi)

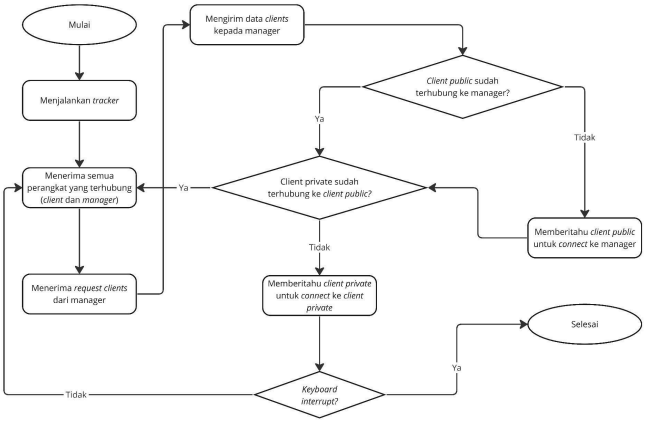
IV. HASIL DAN PEMBAHASAN

Arsitektur *crawler* terdistribusi dalam penelitian ini terdiri dari lima perangkat lunak dengan konfigurasi tiga perangkat memiliki *public IP address* dan dua perangkat memiliki *private IP address*. Dengan pembagian perannya masing -masing, satu *Tracker*, satu *Manager*, dan tiga Klien. Pada penerapannya setiap program berjalan dengan bahasa pemrograman Python dan setiap proses dapat berjalan secara paralel dengan menggunakan *threading*. Setelah melakukan percobaan untuk mendapatkan kondisi *peer to peer*, maka dihasilkan arsitektur seperti Gambar 5.

A. Implementasi

1. *Tracker*

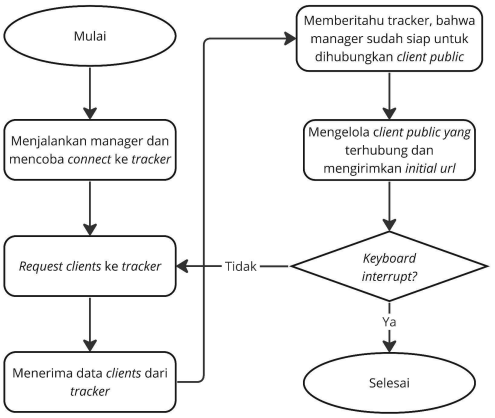
Tracker akan memiliki *public IP address,* dikarenakan tracker memiliki tugas sebagai jalur informasi antara perangkat yang terhubung. Setiap perangkat harus terlebih dahulu untuk terhubung melalui tracker dan juga mengirimkan informasi apakah perangkat tersebut memiliki *public IP address* atau tidak dan tipe dari perangkat tersebut, apakah dia *manager* atau klien. Lalu, *tracker* akan menampung semua perangkat yang terhubung.



GAMBAR 7. *FLOWCHART TRACKER*

2. *Manager*

Manager akan memiliki *public IP address*, dikarenakan manager memiliki tugas untuk mengatur perangkat terhubung yang nantinya akan memiliki *public IP address*. *Manager* juga perlu untuk terhubung ke *tracker*. *Manager* mendapatkan klien yang terhubung dengan meminta data klien yang terhubung kepada *tracker*. Lalu, *tracker* akan mengirimkan data klien yang terhubung. *Manager* akan membuka koneksi untuk klien yang nantinya memiliki *public IP address* untuk terhubung dengannya.

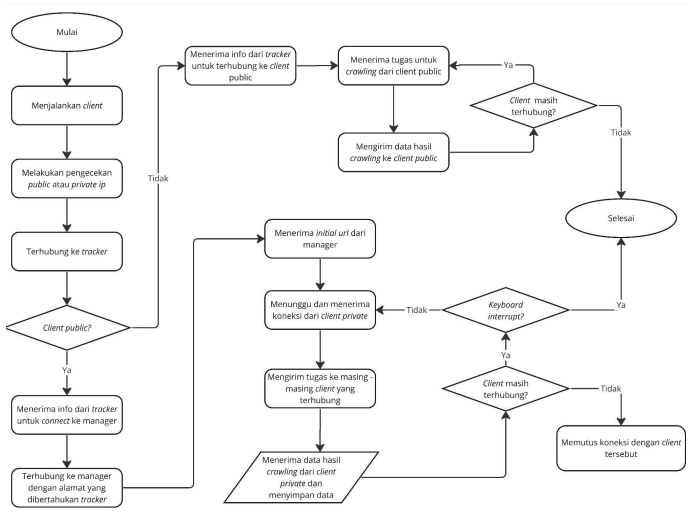


GAMBAR 8. *FLOWCHART MANAGER*

3. Klien

Klien dapat memiliki public dan *private IP address*. Dalam penelitian ini terdapat 1 klien dengan *public IP address* dan 2 klien dengan *private IP address*, yang nantinya klien dengan *private IP address* akan menjadi *worker* yang melakukan *crawling*. Setelah klien *public* terhubung dengan tracker. Dan *tracker* juga sudah mendeteksi adanya *manager* yang sudah terhubung. Maka, *tracker* akan mengirim informasi *manager* kepada klien tersebut. Agar klien tersebut dapat terhubung dengan manager. Untuk klien *private* nantinya akan terhubung dengan klien *public* setelah mendapatkan informasi dari *tracker* bahwa klien *public* sudah siap untuk dihubungkan. Klien *private* akan melakukan *crawling* yang mana pembagian url-nya akan diatur oleh klien public. Data hasil *crawling* akan dikembalikan ke klien *public* untuk disatukan.

Proses *crawling* harus dilakukan oleh klien private dikarenakan jika dilakukan oleh klien public akan berdampak pada *Domain Name System (DNS)* klien tersebut yang dapat diblokir. Karena situs *web* memiliki syarat dan kebijakan yang memiliki batasan terhadap *crawling* otomatis, dan dengan penggunaan klien *private* dapat meminimalkan risiko. Klien *private* juga membantu untuk menghindari batasan tingkat permintaan *(rate limiting)*, karena kalau dengan klien *public* peraturannya akan lebih ketat karena terikat dari penyedia layanan internet atau server layanan .



GAMBAR 9. *FLOWCHART CLIENT*

B. Pengujian

Pada penelitian ini akan terfokus untuk menguji proses komunikasi yang terjadi, konsistensi data, efisiensi sumber daya dan optimalisasi data . Dijalankan proses *crawling* selama enam jam. Pada *crawler* individual dan *crawler* terdistribusi. Pengujian dilakukan dengan *initial url*: "https://fnn.co.id/". Dengan *crawler max thread*-nya 4.

TABEL 2. HASIL CRAWLING SELAMA SATU JAM DENGAN DUA JENIS CRAWLER YANG BERBEDA

Jenis *Crawler*	*Total Rows*	*Unique Rows*
Crawler Individual	43612	43612
2 *Crawler* Terdistribusi	34876	34876

1. Proses Komunikasi

Berikut adalah proses komunikasi pada setiap perangkat:

```
(env) root@vmi1165273:~/p2p/new_crawler# python tracker.py
Server listening on 0.0.0.0:55555
New connection from ('185.227.134.123', 35676)
Received from ('185.227.134.123', 35676): {'type': 'client', 'ip': '185.227.134.123', 'is_private': False}
New connection from ('103.166.156.127', 46120)
Received from ('103.166.156.127', 46120): {'type': 'manager', 'ip': '103.166.156.127', 'is_private': False}
clients: [{'address': ('185.227.134.123', 35676), 'type': 'client', 'is_private': False}, {'address': ('103.
166.156.127', 46120), 'type': 'manager', 'is_private': False}]
manager ready? True
notify client
received data: {'msg': 'CLIENT_CONNECTED_TO_THE_MANAGER'}
clients: [{'address': ('185.227.134.123', 35676), 'type': 'client', 'is_private': False}, {'address': ('103.
166.156.127', 46120), 'type': 'manager', 'is_private': False}]
New connection from ('180.244.164.42', 6403)
Received from ('180.244.164.42', 6403): {'type': 'client', 'ip': '192.168.1.4', 'is_private': True}
clients: [{'address': ('185.227.134.123', 35676), 'type': 'client', 'is_private': False}, {'address': ('180.
244.164.42', 6403), 'type': 'client', 'is_private': True}, {'address': ('103.166.156.127', 46120), 'type': '
manager', 'is_private': False}]
```

GAMBAR 10. PROSES KOMUNIKASI PADA *TRACKER*

```
(env) [root@jft p2p]# python manager.py
My IP address: 103.166.156.127
Information sent to the server.
TCP Manager listening on 103.166.156.127:46120
public: ('185.227.134.123', 35676)
notify tracker
Received client list: [{'address': ['185.227.134.123', 35676], 'type': 'client', 'is_private': Fals
}, {'address': ['103.166.156.127', 46120], 'type': 'manager', 'is_private': False}]
Clients: {<__main__.Client object at 0x7f68d64f68d0>, <__main__.Client object at 0x7f68d64f6860>}
Accepted connection from ('185.227.134.123', 37298)
Received data: Hi from client with public ip
```

GAMBAR 11. PROSES KOMUNIKASI PADA *MANAGER*

```
(env) root@vmi1165273:~/p2p/new_crawler/client# python client.py
My IP address: 185.227.134.123
Connected to the tracker
Private: False
manager address: ('103.166.156.127', 46120)
start_urls: ['https://www.detik.com/']
table page_information already exists
Client public listening for incoming connection..
0 Slave(s) Connected
0 Slave(s) Connected
0 Slave(s) Connected
0 Slave(s) Connected
Accepted connection from ('180.244.164.42', 7720)
Client with identifier <socket.socket fd=5, family=2, type=1, proto=0, laddr=('192.168.1.4', 50148),
 raddr=('185.227.134.123', 35676)> is a new slave.
init_url https://www.detik.com/
len queue: 1
health: 100
1 Slave(s) Connected
len queue: 1
health: 100
```

GAMBAR 12. PROSES KOMUNIKASI PADA KLIEN *PUBLIC*

```
❯ python client.py
My IP address: 192.168.1.18
Connected to the tracker
Private: True
connecting to client public
table crawling already exists
table page_information already exists
table page_linking already exists
table page_images already exists
table page_tables already exists
table page_styles already exists
table page_scripts already exists
table page_list already exists
table page_forms already exists
table tfidf already exists
table tfidf_word already exists
table pagerank already exists
table pagerank_changes already exists
Starting the crawler from the start urls...
Running breadth first search crawler...
https://news.detik.com/berita/d-7136374/husen-
s-20-tahun-bui | BFS | 12/01/2024 10:03:56
```

GAMBAR 12. PROSES KOMUNIKASI PADA KLIEN *PRIVATE*

2. Konsistensi Data

Berdasarkan Tabel 2, didapatkan hasil data url yang di-*crawling* itu unik dan tidak ada duplikasi. Berdasarkan hasil perhitungan sebagai berikut:

$$\textit{Jumlah baris yang terduplikasi}$$

$$= \textit{Total Rows} - \textit{Unique Rows} = 34876 - 34876 = \textit{0}$$

3. Efisien Sumber Daya dan Optimalisasi Data

Berdasarkan Tabel 2, didapatkan hasil penurunan jumlah data dengan waktu yang relatif sama, yaitu enam jam. Pada *crawler* individual didapatkan jumlah baris data dalam tabel database sebanyak 43612 dan pada *crawler* terdistribusi didapatkan jumlah baris data dalam tabel database sebanyak 34876. Terdapat penurunan jumlah data yang didapatkan oleh *crawler* terdistribusi. Hal ini memungkinkan dikarenakan oleh proses pembagian tugas kepada klien memakan waktu lebih banyak. Diperlukan peningkatan efisiensi algoritma untuk mengatasi hal ini.

$$\textit{Persentase data}$$

$$= ((34876 - 43612) \div 43612) \times 100 \approx -20.0\%$$

C. Analisis Hasil

Berdasarkan proses perbandingan *crawling* yang berjalan selama satu jam, analisis hasil pengujian adalah sebagai berikut:

1. Proses komunikasi antar perangkat berjalan dengan sistematis.
2. Data yang dihasilkan dari *crawler* terdistribusi lebih sedikit daripada *crawler* individual. Didapatkan sekitar 20% lebih sedikit.
3. Duplikasi data dapat terhindarkan saat mengelola lebih dari satu *crawler* dengan melakukan *crawling* berdasarkan url yang diterima dari pengelola (klien *public*) dan mengembalikan temuan *url* baru kepada pengelola.
4. Efisiensi sumber daya dan optimalisasi belum tercapai. Penurunan tersebut juga dipengaruhi oleh latensi jaringan, *overhead* koordinasi, dan potensi *bottleneck*.
5. Proses penyatuan data menjadi centralized pada satu perangkat, yang mana merupakan penggabungan dari setiap *crawler* yang melakukan pekerjaan.

Hasil perbandingan *crawler* individual dengan *crawler* terdistribusi menunjukkan hasil yang baik dalam sisi koordinasi, pencegahan duplikasi, dan penggabungan basis data. Peningkatan ini dapat diraih dengan menerapkan metode terdistribusi. Ada hal yang harus dibayar dalam meningkatkan efisiensi, walaupun perbedaan yang terlihat menurun sebanyak 20%. Diperlukan perbaikan algoritma agar menjadi lebih efisien.

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil implementasi dan pengujian terhadap *crawler* terdistribusi. Maka diperoleh kesimpulan seperti berikut

1. Pembuatan *crawler* terdistribusi dengan menggunakan mekanisme socket programming memerlukan pemahaman yang baik terkait jaringan komputer. Karena sering berfokus pada proses komunikasi antar perangkat.
2. Pembuatan *crawler* terdistribusi memerlukan setidaknya tiga perangkat dengan public ip address sebagai pondasi utama komunikasi dapat berjalan dengan sistematis.
3. Penyimpanan data pada *crawler* terdistribusi menggunakan sqlite3 sebagai *database*-nya. Dan datanya dapat ter-centralized pada pengelola *crawler* (klien public).
4. Pengembangan *crawler* terdistribusi menjadikan proses *crawling* yang dilakukan pada peta *web* dapat lebih cepat untuk mengumpulkan data dalam jumlah yang banyak.
5. Perbandingan *crawler* individual dengan *crawler* terdistribusi memiliki rasio pengumpulan data lebih besar sekitar 30% dengan menggunakan *crawler* terdistribusi.

B. Saran

Adapun saran untuk penelitian selanjutnya adalah:

1. Melanjutkan penelitian dan pengembangan proses *crawling* agar semakin efektif dan masif dalam proses pengumpulan data dari internet. Dapat dengan menggunakan *crawler* dalam jumlah yang masif pula.
2. Melanjutkan penelitian menggunakan bahasa lain yang lebih cepat dan efisien dalam melakukan proses komunikasi dan *crawling* seperti bahasa C.
3. Melanjutkan penelitian dengan menerapkan database NoSQL seperti MongoDB atau mendesain database yang lebih efisien dalam mengelola penyimpanan data.

DAFTAR PUSTAKA

[1] Asmara, A. (2024). “Perancangan User Interface Search Engine dan Admin Console Untuk Manajemen dan Visualisasi Data Hasil Pengindeksan”. In.

[2] Babatunde, O. and O. Al-Debagy (2014). “A comparative review of internet protocol version 4 (ipv4) and internet protocol version 6 (ipv6)”. In: arXiv preprint arXiv:1407.2717.

[3] Boldi, P. dkk., (2018). “BUbiNG: Massive crawling for the masses. ACM Transactions on the Web (TWEB)”. In: pp. 1–26.
[4] Cambazoglu dkk., (2009). “Quantifying performance and quality gains in distributed web search engines”. In: International Journal of Information Technology & Decision Making, pp. 411–418.
[5] Ford, B., P. Srisuresh, and D. Kegel (2005). “Peer-to-Peer Communication Across Network Address Translators”. In: USENIX Annual Technical Conference, pp. 179–192.
[6] IBMCorporation (2021). What is a socket? URL: https://www.ibm.com/docs/en/zos/2.1.0?topic=services-what-is-socket.
[7] Khatulistiwa, L. (2023). “Perancangan Arsitektur Search Engine dengan Mengintegrasikan Web Crawler, Algoritma Page Ranking, dan Document Ranking”. In.
[8] Orlando dkk., (2002). “Design of a parallel and distributed web search engine. In Parallel Computing: Advances and Current Issues”. In: pp. 197–204.
[9] Pradana, F. H. (2023). “Perbandingan Implementasi Algoritma-Algoritma Pagerank pada Satu Mesin Komputer”. In.
[10] Qiaoqiao, L. (2021). What Is Network Address Translation (NAT)? URL: https://info.support.huawei.com/info-finder/encyclopedia/en/NAT.html.
[11] Qoriiba, M. F. (2021). “Perancangan Crawler sebagai Pendukung pada Search Engine”. In.
[12] Rosenberg, J. dkk., (2003). “STUN-simple traversal of user datagram protocol (UDP) through network address translators (NATs)”. In: rfc3489.
[13] Shkapenyuk, V. and T. Suel (2002). “Design and implementation of a high-performance distributed web crawler”. In: Proceedings 18th International Conference on Data Engineering, pp. 357–368.
[14] Tanenbaum, A. S., N. Feamster, and D. J. Wetherall (Sept. 2021). Computer Network (Sixth Edition). Encyclopedia of Database Systems. Springer Verlag.