Perancangan Improvisasi Arsitektur *Web Crawler* Berbasis *Multi-Threading* dan *Multi-Processing* Dengan Menggunakan Bahasa Pemrograman *Rust*

Muhammad Daffa Haryadi Putra, Muhammad Eka Suryana, Med Irzal
Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Jakarta
Jakarta Timur, Indonesia
daffahr15@protonmail.com, eka-suryana@unj.ac.id, medirzal@unj.ac.id

***Abstrak*—Mesin pencari atau *search engine* merupkan *software* yang digunakan untuk melakukan pencarian terhadap informasi tertentu. Untuk menjalankan proses pencarian diperlukan jumlah data yang banyak yang terkumpul dan dapat diakses dengan mudah, proses pengumpulan data ini lah yang disebut *crawling*. Penelitian ini mencoba untuk memperbaiki kekurangan-kekurangan dari *crawler* versi lazuardy dengan penekanan dalam efisiensi peforma dan penggunaan *computing resource*. Penelitian ini menggunakan metode *multi-threading* dan *multi-processing* untuk membagi beban tugas kerja dari *crawler* menjadi dua modul yaitu, *scouter* dan *parser*, selain itu algoritma *breadth-first search* yang digunakan dalam *crawler* dimodifikasi untuk membatasi halaman web apa yang dapat di jelajahi oleh *crawler*. Hasil akhir dari penelitian ini menunjukkan bahwa terdapat improvisasi dengan metode baru ini sebesar 17x dibandingkan dengan *crawler* orisinil, dengan catatan penyeratan halaman terunduh antar *domain* belum berhasil.**

***Kata Kunci*—Search Engine, Web Crawler, Rust Programming Language, Multi-Thread, Multi-Process**

I. Pendahuluan

Search engine merupakan sebuah program yang digunakan untuk menjelajahi dan mencari informasi dari web [1]. Terdapat beberapa komponen yang membangun arsitektur *Search engine* seperti *Web crawler*, *Page rank*, dan *indexer* [2]. Dalam proses pencarian web yang dilakukan *Search engine* tahap pertama yang di lakukan adalah *Web crawler* menjelajahi dan mengekstraksi data-data dari list *url* lalu menyimpan data tersebut dan data lain yang terkait ke dalam database [2]. Data yang disimpan akan di-index, diberikan skor dan di urutkan melalui algoritma *pagerank* [2] *Web Crawler* merupakan komponen penting dalam pembuatan arsitektur *Search engine* secara keseluruhan. Penelitian sebelumnya yang telah dilakukan oleh *Lazuardy Khatulisitwa* telah berhasil mengimplementasikan *Web crawler* kedalam arsitektur *Search engine* yang berjalan [3]. *Web crawler* tersebut mengimplentasikan algoritma *Breadth First Search* dengan modifikasi algoritma *similiarity based* untuk meningkatkan akurasi dari proses *crawling* dan pengambilan data dari suatu halaman [3]. Algoritma *Modified Similarity-Based* yang digunakan oleh *Fathan* untuk memperbaiki akurasi dari *Breadth First Search* memanfaatkan konsep penyimpanan *queue* dalam melakukan proses *crawling* [4]. Dalam proses tersebut *crawler* akan menyimpan 2 jenis *queue* yaitu, *hot queue* untuk menyimpan *url* yang mengandung kata *anchor* sedangkan *url queue* digunakan untuk menyimpan *url* lain [5]. Proses ini dapat membantu *crawler* untuk mengunjungi dan melakukan *crawling* ke dalam *page* yang terdapat di *hot queue* terlebih dahulu bila *page* yang berkaitan dengan kata *anchor* di kunjungi terlebih dahulu maka *child page*-nya kemungkinan besar akan memiliki konten yang berkaitan dengan kata *anchor* tersebut [5].

Arsitektur dari *crawler* yang di kembangkan oleh Lazuardi, menggunakan *python* sebagai bahasa pemograman dan *library* pendukung yang digunakan adalah *beautifulsoup4* untuk melakukan *parsing* dari halaman *website*, *request* untuk mengirimkan request kepada halaman *website* yang ingin di ambil data-nya, dan *regex* untuk melakukan pencocokan kata - kata yang telah di dapat dengan *keyword* yang sudah di tentukan [3]. Dari hasil penelitian *lazuardi* terdapat beberapa saran peningkatan yang tercatat, dimana salah satunya terkait dengan meningkatkan kinerja dan peforma dari *web crawler* agar memiliki penggunaan *RAM* yang lebih kecil dan mencapai kinerja yang maksimal [3].

Salah satu metode untuk mempercepat jalannya *search engine* adalah *Multi-threading* [6]. Metode ini sudah pernah digunakan dalam *search engine* sebelumnya, tetapi *search engine* ini mencari data bukan ke *web* tetapi pada kumpulan data teks atau dapat disebut dengan nama *text search* [6]. Dari hasil penelitian tersebut ditemukan metode *multi-threading* yang digunakan berhasil mencapai improvisasi yang sebelumnya membutuhkan waktu 16 menit dalam menjelajahi seluruh data teks menjadi 4 menit, yang berarti berhasil mencapai improvi-

asi waktu eksekusi program sebesar 4x [6]. Dalam penelitian tersebut metode *multi-threading* digunakan untuk memecah proses pengambilan data dari sumber data dan proses parsing dari data teks yang sudah di ambil [6].

Dalam konteks *search engine* untuk pencarian web penelitian yang dilakukan oleh *Pramudita, Y.D et all* telah menunjukkan bahwa mekanisme *multi-threading* dapat diimplementasi dengan benar [7]. Dalam penelitian tersebut tiap-tiap *thread* menjalankan satu *instance* dari *crawler* nya itu sendiri, dan penelitian tersebut berhasil mencapai percepatan waktu *crawling* selama 123 detik [7].

Selanjutnya penelitian hanya akan melakukan improvisasi terhadap komponen *web crawler* saja untuk membatasi area penelitian. Penelitian ini akan berusaha untuk meningkatkan performa, yang dimana merupakan jumlah halaman yang terkumpul pada waktu yang sudah definisikan. Berdasarkan hasil penelitian *Pramudita, Y.D et all*, yang dimana menjalankan keseluruhan proses *crawler* dalam satu thread [7], maka penelitian ini akan berusaha untuk meningkatkan performa dengan memisahkan proses *parsing* dalam *crawler* dalam proses yang berbeda atau yang dapat disebut dengan metode *multi-processing*. Selain itu penelitian ini juga akan berusaha untuk meningkatkan akurasi hasil proses *crawling* dengan menggunakan algoritma *breadht-first search* yang dimodifikasi dengan tujuan agar *crawler* hanya menjelajahi domain yang telah ditentukan saja, sehingga diharapkan hasil proses *crawling* hanya akan berisi halaman web yang diinginkan. Perbaikan lain yang akan dilakukan adalah dengan menggunakan bahasa pemograman dengan waktu eksekusi yang lebih cepat, yaitu *rust* [8]. Keputusan ini didasari dari hasil pengujian bahasa pemograman *rust* dalam proses dengan intensitas tinggi dan konteks *low-level* [8].

II. Kajian Pustaka

A. *Definisi* Search Engine

Mesin Pencari atau *Search Engine* merupakan software yang digunakan untuk pencarian terhadap banyak situs web di internet berdasarkan input kata yang ditanyakan. *Search Engine* memungkinkan pengguna untuk mencari situs web yang berkaitan dengan kata kunci ataupun pertanyaan yang diajukan oleh pengguna [1]. Dalam penggunaannya, search engine hanyalah sebuah halaman situs website yang dapat diakses oleh pengguna yang perannya adalah mengumpulkan dan menampilkan hasil pencarian tersebut kepada user dengan tampilan yang menarik dan informatif [1].

B. *Arsitektur* Search Engine

Secara sederhana *Search Engine* bekerja dengan menyimpan dan melakukan pengindeksan informasi-informasi dari situs web dan menyajikannya dalam bentuk yang dapat di mengerti oleh pengguna. Informasi dari halaman situs web didapatkan menggunakan program bernama *Web crawler* yang mengunduh dan menyimpan informasi dari halaman situs web kedalam *Database*. Setelah di simpan, infomasi akan dianalisis dan dipilih oleh program *Indexer* [3].

Proses *crawling* dalam arsitektur *Search engine* milik lazuard memiliki beberapa tahap, Tahap awal adalah *crawler* akan mengakses *origin url* yang disediakan dalam *environment variable*. Untuk melakukan proses *crawling*, perlu untuk menginisiasikan beberapa data yang akan digunakan dalam proses crawling seperti *origin url* yang akan di akses, maksimum *os threads* yang akan digunakan oleh crawler, dan durasi proses *crawling*. Proses pertama yang dilakukan oleh *crawler* setelah di inisiasi adalah dengan melakukan pengecekan ke *database* apakah terdapat page yang sudah di *crawl* atau belum, bila sudah maka crawler akan memulai proses *crawling* dari page terakhir yang sebelumnya telah di *parse*. Bila tidak, maka proses *crawling* akan dimulai dari *origin url*. List dari origin url akan dimasukkan kedalam *queue* yang nantinya akan digunakan oleh proses *Breadth First Search* dalam *crawler*. Sebelum proses parse dilakukan data-data yang berhubungan dengan *page* yang akan di *parse* akan di insert kedalam database, seperti *string url* dan duration *parse*. Proses parsing page dilakukan menggunakan algoritma *Breadth First Search*.

Dalam menjalankan *Breadth First Search*, setiap *instance page scrapper* dijalankan secara paralel didalam *thread process*. *Page scrapper* akan melakukan parsing tiap *page* yang diakses dan mengambil beberapa bagian data dari *page* tersebut. Data yang parse dari *page* merupakan data penting yang berisi inti sari dari *page* tersebut dan data lain yang akan mendukung proses *crawling* dan proses-proses selanjutnya dalam arsitektur *search engine*, beberapa data yang diambil oleh *scrapper* adalah *article body* dari *html page*, *meta description* dari *page*, *meta keyword*, *css page style* dari page, *script* yang di *embedded* dalam page, *list*, *form*, *table*, *image* dalam *page* dan *hyperlink* yang ada di *page* tersebut. Data-data yang telah dikumpulkan tersebut akan di masukkan ke dalam *database* [3]. Dalam proses *crawling* setiap kali *page scrapper* selesai dalam menjelajahi dan melakukan *parsing* dalam satu page, *page scrapper* akan memasukkan url list yang di dapat dari dalam page kedalam *queue*. Agar proses penambahan link kedalam queue tidak terganggu, penambahan *queue* dilakukan secara *syncronous* menggunakan lock. *Url list* yang disimpan di dalam *queue* ini nantinya akan di akses oleh *page scrapper* lain Proses ini akan berlanjut terus secara paralel dan pengaksesan tiap-tiap url dilakukan menggunakan algoritma *breadth first search* [3].

C. *Algoritma* Breadth First Search

Untuk menjelajahi *url list* yang ada di dalam *queue crawler* menggunakan algoritma *breadth first search* algoritma ini pada dasarnya merupakan algoritma untuk menjelajahi *graph* dalam suatu *tree*. Dalam penerapannya di dalam arsitektur *search engine* milik lazuardi *breadth first search* dimanfaatkan

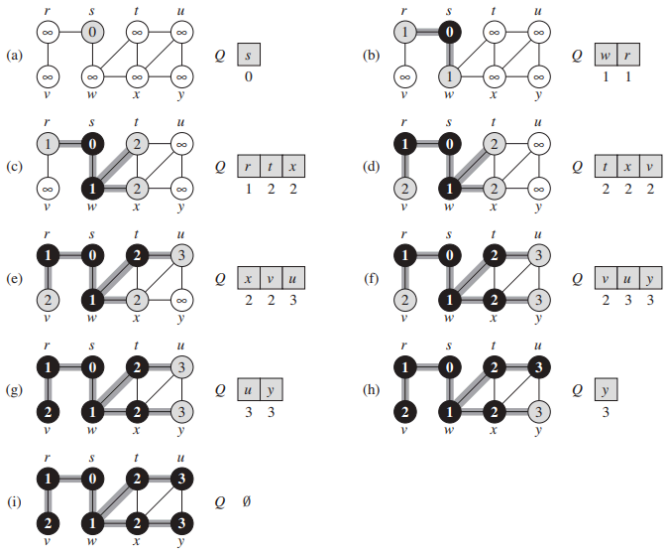

Fig. 1: Diagram alur algoritma *breadth first search* [9]

dalam proses pemilihan url yang akan diakses dalam setiap iterasi proses *page scrapping* [3]. Dalam menjelajahi *tree*, algoritma ini menggunakan struktur data *queue* untuk menyimpan informasi tentang *node* yang akan dijelajahi selanjutnya dan *stack* untuk menyimpan informasi mengenai *node* yang telah di jelajahi. Metode *breadth first search* memungkinkan untuk *crawler* memprioritaskan penjelajahan *url* yang telah dimasukkan ke dalam *queue* terlebih dahulu hal ini menjamin agar setiap tingkatan *node* sudah dijelajahi sebelum lanjut ke tingkat *node* selanjutnya [9].

D. *Definisi* Processes *dalam* Operating Systems

Process pada dasarnya adalah *program* yang sedang dijalankan oleh komputer. *Program* pada sendirinya hanya file pasif yang berisi instruksi - instruksi yang perlu dijalankan oleh komputer. Instruksi tersebut yang menjadi satu *instance* dari *process*. *Process* mengakses data yang diperlukan untuk proses komputasi dari *virtual memory*, data di dalam memory ini bersifat sementara dan disimpan sebagai *cache*. *Memory* yang dapat diakses oleh *process* memiliki susunan tertentu yang terbagi menjadi beberapa bagian.

Dari gambar Fig. 2 dapat dilihat bahwa susunan *virtual memory* yang dapat diakses oleh *process* yang berjalan terbagi menjadi beberapa bagian yang dibagi berdasarkan jenis data yang disimpan dan tingkat alamat dari data tersebut dalam *memory* [10]. Bagian-bagian dalam *virtual memory* adalah,

1) *Text*. Data yang berisi kode yang dijalankan
2) *Data*. Data yang berisi variabel global dalam kode
3) *Heap*. Sejumlah ukuran *memory* yang dialokasikan secara dinamis oleh program saat program sedang berjalan atau *runtime*
4) *Stack*. Penyimpanan data sementara yang disediakan saat pemanggilan fungsi dalam kode.

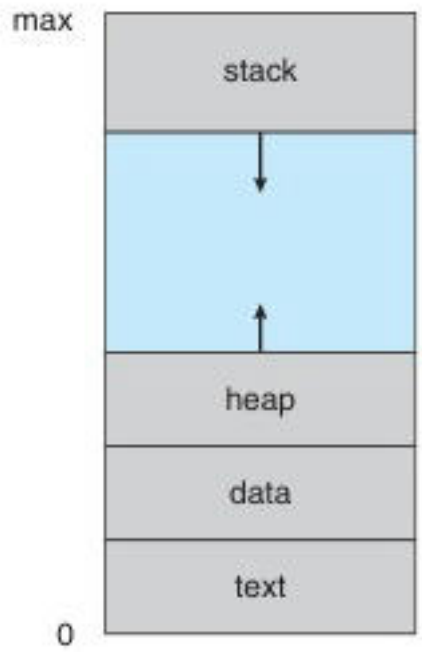

Fig. 2: Susunan bagian dari *virtual memory* [10]

III. Desain Model

A. *Modifikasi Arsitektur* Crawler

Dari arsitektur *crawler* yang sudah ada saat ini terdapat beberapa modifikasi yang perlu dilakukan untuk memperbaiki performa proses *crawling*.

- Arsitektur ini akan dibagi menjadi 2 *service* yaitu, *crawler* dan *indexer*.
- *Service Crawler* akan dibagi menjadi dua *process*, *Scouter* dan *Parser*. *Scouter* bertugas sebagai pengunduh halaman *website* dan *parser* bertugas sebagai pembangun *language tree* dan melakukan *input* kedalam *database*.
- Jalannya *crawler* dan *indexer* akan secara bersamaan dan otomatis. Untuk mengontrol jalannya *indexer* agar konsistensi data dapat terjaga,

B. *Modifikasi* Breadth-first Search *dengan* Domain Constraint

Untuk menyeragamkan jumlah halaman *web* yang diakses oleh tiap *thread*, algoritma *breadth-first search* yang digunakan untuk mengunjungi tiap-tiap halaman perlu dimodifikasi. Modifikasi yang dilakukan adalah dengan menugaskan jalannya *crawler* di tiap *thread* sebuah domain *url* tertentu, dan membatasi *url* yang dapat diakses oleh *thread* tersebut sesuai dengan *url* yang telah ditugaskan. Setiap *thread* akan mengambil dan menyimpan *url* di dalam *queue* global *Queue* ini merupakan *multi-lock queue* dengan format yang lebih kompleks dari *queue* normal, ini dilakukan agar tidak terjadi *race condition* antar *thread* ketika mengambil ataupun menyimpan data kedalam *queue* tersebut. Gambar Fig. 4 merupakan

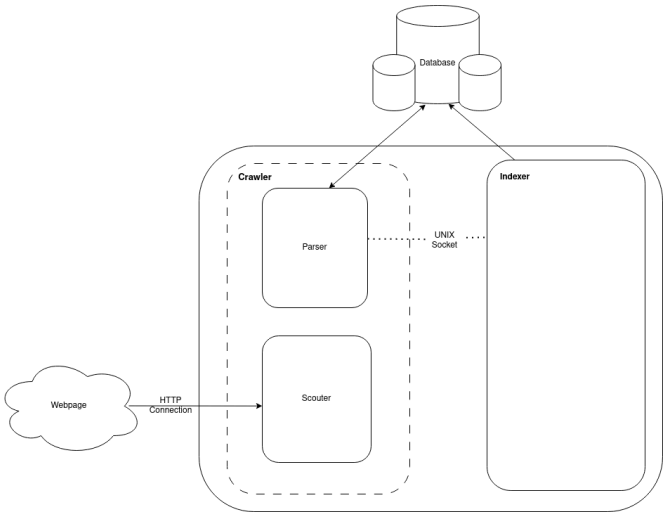

Fig. 3: Diagram Arsitektur Crawler Termodifikasi

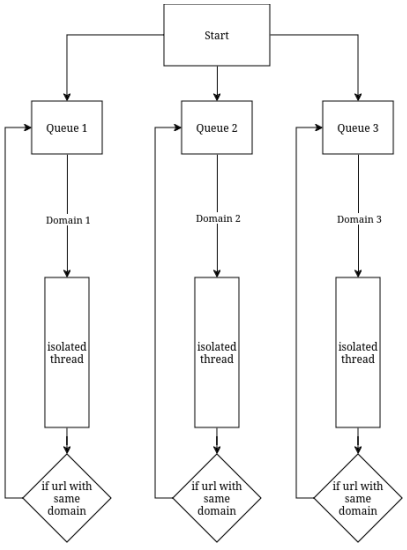

Fig. 4: Diagram cara kerja algoritma *breadth-first search* termodifikasi

ilustrasi dari jalannya algoritma *breadth-first search* termodifikasi.

IV. Hasil dan Pembahasan

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae.

V. Kesimpulan dan Saran

A. Kesimpulan

Berdasarkan hasil implementasi dan pengujian fitur sistem informasi yang telah dirancang, maka diperoleh kesimpulan sebagai berikut:

1) *Crawler* berhasil mengumpulkan data dari halaman web yang *domain*-nya telah di definiskan dalam *origin url*.
2) Dari perbandingan hasil proses *crawling*, bahasa pemograman berabstraksi rendah lebih cocok untuk digunakan dalam *high-intensity application* seperti *web crawler* ini.
3) Migrasi *database* dari berbasis *SQL* menuju berbasis *MonggoDB* berhasil dan data yang tersimpan konsisten.
4) *Crawler* yang dirancang menggunakan metode *multi-threading* berhasil mengumpulkan jumlah halaman web lebih banyak daripada *crawler* sebelumnya.
5) Algoritma *breadth-first search* termodifikasi dalam skripsi ini belum cukup untuk meningkatkan akurasi proses *crawling* terhadap halaman web yang didefiniskan oleh *top-level domain*.
6) Penggunaan *resource* lebih banyak berada di *scouter service* bila dibandingkan dengan *parser service*.

B. Saran

Adapun saran untuk penelitian selanjutnya adalah:

1) Melanjutkan penelitian dalam eksplorasi algoritma *crawler* lain untuk meningkatkan akurasi pengumpulan halaman web yang telah didefinisikan oleh *top-level domain*.
2) Melanjutkan penelitian dalam eksplorasi algoritma *information retrieval* yang mengakomodasi lebih banyak jenis *website*.
3) Eksplorasi penggunaan *filesystem* sebagai platform untuk menyimpan data hasil *crawling* untuk mengakomodasi struktur halaman web yang berbeda-beda.
4) Eksplorasi implementasi *distributed crawler* dengan menggunakan skema *multi-threading* dengan bahasa pemograman berabstraksi rendah seperti *Rust*, *C/C++*, atau *Zig*.

References

[1] T. Seymour, D. Frantsvog, and S. Kumar, "History of search engines," *International Journal of Management & Information Systems (IJMIS)*, vol. 15, no. 4, pp. 47–58, 2011.

[2] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," 1998.

[3] L. Khatulistiwa, "PERANCANGAN ARSITEKTUR SEARCH ENGINE DENGAN MENGINTEGRASIKAN WEB CRAWLER, ALGORITMA PAGE RANKING, DAN DOCUMENT RANKING," 2023.

[4] M. F. Qorriba, "PERANCANGAN CRAWLER SEBAGAI PENDUKUNG PADA SEARCH ENGINE," 2021.

[5] J. Cho, H. Garcia-Molina, and L. Page, "Efficient crawling through URL ordering," 1998.

[6] G. Sun, H. Xiang, and S. Li, "On Multi-Thread Crawler Optimization for Scalable Text Searching," 2019.

[7] Y. D. Pramudita, D. R. Anamisa, S. S. Putro, and M. A. Rahmawanto, "Extraction System Web Content Sports New Based On Web Crawler Multi Thread," *Journal of Physics: Conference Series*, vol. 1569, no. 2, p. 22077–22078, Jul. 2020.

[8] Y. Lin, S. M. Blackburn, A. L. Hosking, and M. Norrish, "Rust as a language for high performance GC implementation," *SIGPLAN Not.*, vol. 51, no. 11, pp. 89–98, Jun. 2016, doi: 10.1145/3241624.2926707.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[10] G. G. Abraham Silberchschatz Peter Baer Galvin, *Operating System Concepts*. Willey, 2018.