Perbandingan Algoritma BFS & DFS dalam Pengiriman Data pada Jaringan Peer To Peer

Gabriel Bentara Raphael 13516119
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516119@std.stei.itb.ac.id

***Abstrak*—Makalah ini membahas perbandingan algoritma breadth first search (BFS) dan depth first search (DFS) dalam pengiriman data pada jaringan peer to peer. Jaringan ini memiliki struktur layaknya seperti graf dimana setiap komputer sebagai node dan hubungan antar node sebagai edge. Jaringan peer to peer memiliki keunggulan dibandingkan jaringan berbasis client-server, salah satunya adalah data yang terdesentralisasi dan up-time yang tinggi. Namun keuntungan tersebut juga disertai beberapa masalah. Pertama, pemindahan data dari sebuah peer ke peer lainnya dapat memakan waktu yang lebih lama karena diperlukannya sebuah algoritma untuk melakukan path finding dari peer awal ke tujuannya, mengingat kemungkinan kedua peer tidak bertetangga. Selain itu, oleh karena data yang ada tersebar maka diperlukan teknik enkripsi untuk mengamankan data-data tersebut. Untuk mengatasi permasalahan bagaimana data tersebut dapat sampai dari peer awal ke tujuan, maka digunakan algoritma BFS & DFS.**

***Keywords*—breadth first search, depth first search, peer to peer, enkripsi.**

I. PENDAHULUAN

Seiring dengan berkembangnya teknologi, kita menjumpai masalah-masalah yang tidak dapat dipisahkan dengan keuntungan dan kemudahan yang didapatkan. Salah satu masalah tersebut adalah persoalan privasi dan kepercayaan. Kita tidak dapat memungkiri bahwa dengan semakin terhubungnya dunia melalui jalur komunikasi seperti sosial media dan *instant messenger* menyebabkan privasi kita terganggu oleh karena data-data yang mungkin mengalami kebocoran (*leak*) atau tersebar tanpa izin.

Permasalahan tersebut kemudian berkembang menjadi masalah kepercayaan (*trust issues*). Orang semakin tidak percaya untuk menaruh datanya pada situs-situs dan kemudian menjadi paranoid hingga titik di mana terdapat sekumpulan orang yang membuat dan menggunakan sistem-sistem tersendiri di mana privasi dapat dijaga dengan sebaik-baiknya.

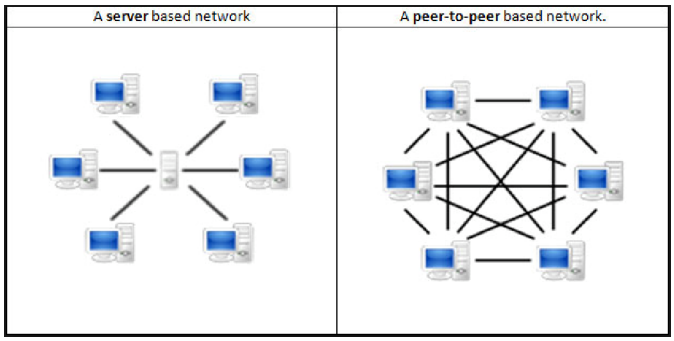
Salah satu teknologi yang diciptakan untuk mengatasi kedua permasalahan di atas adalah teknologi jaringan berbasis *peer to peer*. Dengan model jaringan ini, tidak terdapat penyedia yang terpusat maupun data yang terpusat. Dengan menganut prinsip desentralisasi dan dibantu dengan teknik enkripsi, model jaringan ini dapat mengatasi permasalahan di atas.

Terdapat banyak sekali teknologi yang sekarang memanfaatkan jaringan *peer-to-peer* antara lain teknologi *blockchain* seperti *bitcoin*, *peer-to-peer file sharing* seperti *torrent* dan *usenet*, serta *dark web* pada jaringan *TOR*. Oleh karena itu, penulis tertarik untuk membahas proses perpindahan data dalam jaringan *peer-to-peer* dengan algoritma BFS dan DFS.

II. DASAR TEORI

A. Jaringan Peer to Peer

Jaringan *peer to peer* atau seringkali disingkat dengan akronim P2P adalah sebuah jaringan di mana node-node yang tergabung di dalamnya tidak hanya bertindak sebagai *server* atau *client* namun keduanya. Hal ini menyebabkan setiap node berkedudukan sejajar, memiliki kemampuan yang sama, dan menghilangkan keperluan akan kontrol terpusat. Oleh karena kemampuannya yang sejajar satu sama lain, node ini disebut *peer*.



Gambar 1 – Jaringan *server based* vs *peer-to-peer (Sumber : https://en.bitcoinwiki.org/wiki/Peer-to-peer)*

Jaringan *peer to peer* ini dibagi menjadi dua berdasarkan bagaimana *resource* diindeks dan dialokasikan, yaitu :

1. *Unstructured Network*

   Jaringan yang tidak terstruktur terdiri atas node-node yang secara acak terkoneksi dengan yang lainnya. Jaringan ini sangatlah aman, redundan, dan *scalable.* Namun oleh karena setiap node terkoneksi secara acak, pencarian suatu node spesifik menjadi sangatlah sulit sehingga untuk melakukan pencarian diperlukan suatu cara yang disebut dengan *flooding*.

Flooding sesuai dengan namanya merupakan suatu cara untuk membanjiri seluruh jaringan dengan data yang sama. Untuk memastikan data tersebut tetap terjaga privasinya, maka data tersebut dienkripsi dan hanya dapat dibuka dengan kunci yang dimiliki oleh target.

2. *Structured Network*

Jaringan terstruktur terdiri atas node-node yang saling terkoneksi satu sama lain dan masing-masing node mempunyai suatu catatan akan keberadaan sebuah *resource* spesifik pada node lainnya dan catatan berisi node-node yang bertetangga dengannya.

Untuk mengimplementasikan kedua catatan tersebut seringkali dipakai DHT (*Distributed Hash Table*). *Hash table* merupakan suatu struktur data berbentuk *array of tuple* di mana setiap tuple terdiri atas (*key, value*). Struktur data ini cocok untuk menyimpan data berupa relasi, sebagaimana digunakan pada jaringan terstruktur ini.

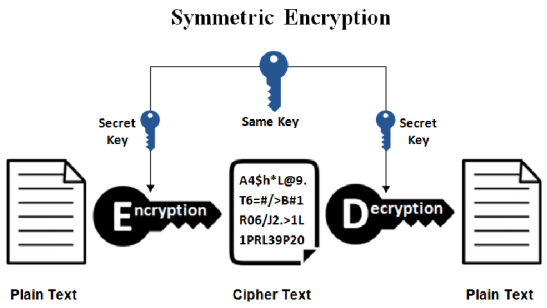
Dengan adanya catatan tersebut, pencarian suatu node spesifik menjadi lebih mudah. Pencarian dapat dilakukan dengan algoritma untuk melakukan traversal suatu pohon/graf. Dalam makalah ini akan dibahas penggunaan algoritma DFS (*depth first search*) dan BFS (*breadth first search*).

B. *Enkripsi*

Seperti yang telah dijelaskan pada poin sebelumnya, diperlukan suatu cara untuk mengamankan data dalam jaringan *peer to peer* karena data akan dioper dari node satu ke lainnya hingga sampai ketujuan. Untuk memastikan keaslian dan kevalidan data maka digunakan enkripsi. Enkripsi adalah suatu proses untuk mengubah suatu pesan atau informasi sedemikian rupa sehingga hanya pihak-pihak yang berwenang sajalah yang dapat mengakses informasi tersebut. Pesan atau informasi yang belum dienkripsi disebut *plain text* sementara informasi yang telah dienkripsi disebut *cipher text*. Untuk mengambil informasi yang telah dienkripsi, dilakukan suatu proses yang disebut dekripsi.

Berdasarkan skemanya, enkripsi dibagi menjadi dua kategori, yaitu :

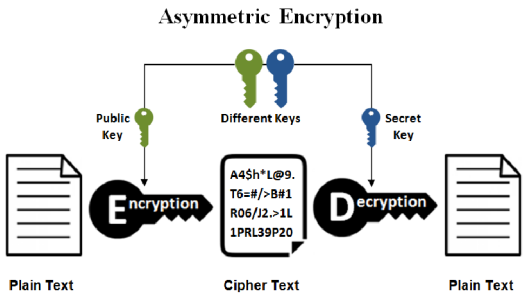
1. *Private key* (Simetris)



Gambar 2 Skema *Symmetric Encryption (Private Key)* (sumber : https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences)

Dalam skema *private key*, *key* yang sama digunakan untuk melakukan enkripsi dan dekripsi. Oleh karena itu, key tersebut disembunyikan dan hanya diketahui oleh pihak-pihak yang berwenang.

2. *Public key* (Asimetris)



Gambar 3 *Asymmetric Encryption* (sumber : https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences)

Dalam skema public key, terdapat 2 *key* yang berbeda untuk melakukan enkripsi dan dekripsi. Sebuah *key* yang bersifat publik (*public key*) digunakan untuk mengenkripsi data sehingga data tersebut hanya bisa di dekripsi oleh pihak yang memegang pasangan *private key* dari *public key* yang digunakan untuk proses enkripsi.

Dengan demikian skema ini dapat digunakan pada banyak implementasi jaringan terdistribusi. Misalkan terdapat 3 buah node yaitu A, B, dan C. B bertetangga dengan A dan C, namun A dan C tidak terhubung secara langsung sehingga data perlu dioper melalui B jika A dan C ingin bertukar data. Maka ketika A ingin mengirimkan suatu data ke C, A akan mengenkripsi data tersebut dengan *public key* milik C sehingga hanya C saja yang dapat membukanya dengan *private key* yang C miliki. Dengan demikian, A dapat dengan aman mentransfer datanya melalui B tanpa khawatir bahwa B dapat melihat data tersebut.

C. *Breadth First Search*

Breadth First Search (BFS) merupakan sebuah algoritma untuk melakukan traversal terhadap sebuah data dengan struktur pohon atau graf. Algoritma ini ditemukan pada tahun 1945 oleh Konrad Zuse dan Michael Burke yang kemudian ditemukan kembali oleh Edward F. More pada tahun 1959. Algoritma ini bekerja dengan tahap sebagai berikut :

1. Ambil simpul awal dan periksa apakah simpul pernah dikunjungi. Jika sudah maka lewati dan lanjutkan ke simpul berikutnya dari *queue*.
2. Periksa apakah simpul merupakan simpul yang ingin dicari. Lakukan proses yang dibutuhkan.
3. Masukkan seluruh simpul tetangga yang belum dikunjungi dalam struktur data *queue*.
4. Kunjungi simpul berikutnya (*head of queue*) dan tandai simpul saat ini sebagai simpul yang telah dikunjungi.

Ulangi tahapan hingga seluruh simpul telah dikunjungi atau simpul telah ditemukan.

Dari proses yang dilakukan oleh algoritma BFS, dapat terlihat bahwa algoritma ini akan mengejar "kelebaran" terlebih dahulu. Dalam menerapkan algoritma ini kita harus memperhatikan banyaknya simpul pada setiap lapisan. Jika dalam suatu lapisan terdapat terlalu banyak simpul, algoritma BFS akan memakan space yang cukup besar.

Kompleksitas dari algoritma ini untuk ruang dan waktunya adalah $O(b^d)$ di mana b adalah *branching factor* dan d adalah kedalaman graf/pohon.

D. Depth First Search

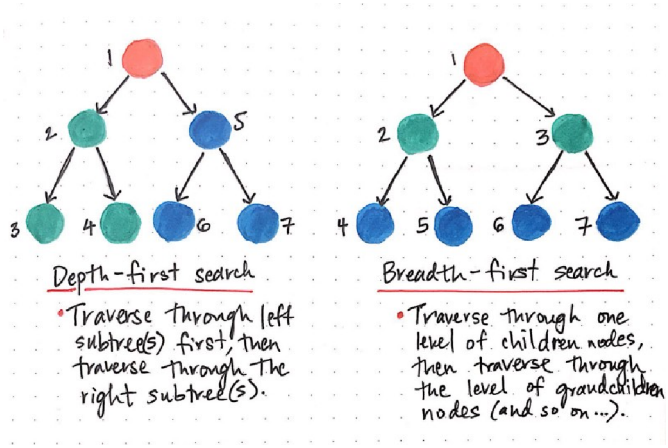
Dalam melakukan traversal dalam suatu graf, terdapat juga algoritma *depth first search* atau seringkali disingkat sebagai DFS. Algoritma ini ditemukan pada abad ke 19 oleh seorang matematikawan Prancis bernama Charles Pierre Tremaux. Prinsip DFS sedikit berbeda dari BFS yaitu algortima ini tidak melakukan traversal secara melebar terlebih dahulu, namun mengejar kedalaman. Struktur data yang mencerminkan urutan kunjungan DFS adalah *stack* (Last In First Out). Untuk lebih jelasnya, berikut tahapan yang dilakukan oleh algoritma DFS.

1. Ambil simpul awal dan periksa apakah simpul sudah pernah dikunjungi, apabila sudah, ambil simpul berikutnya dalam *stack*.
2. Periksa apakah simpul merupakan simpul yang ingin dicari. Lakukan proses yang dibutuhkan.
3. Masukkan seluruh simpul tetangga yang belum dikunjungi dalam *stack*. Simpul-simpul tersebut akan dikunjungi setelahnya mengingat struktur data *stack* bersifat LIFO.
4. Kunjungi simpul berikutnya (*top of stack*) dan tandai simpul saat ini sebagai simpul yang telah dikunjungi. Ulangi langkah ini hingga seluruh simpul telah dikunjungi atau simpul yang dicari telah ditemukan.

Hal yang perlu diperhatikan dalam menggunakan algoritma DFS adalah algoritma ini karena selalu mengarah ke kedalaman terlebih dahulu, biasanya tidak optimal hasilnya apabila dinilai berdasarkan jumlah node/jarak yang ditempuh. Selain itu, pengguna harus memperhatikan ukuran kedalaman graf atau pohon yang ingin dikunjungi. Apabila graf atau pohon yang ingin diolah terlalu dalam, algoritma DFS mungkin tidak akan berhenti.

Kompleksitas waktu dari algoritma ini adalah $O(b^d)$ di mana b adalah *branching factor* dan d adalah kedalaman graf/pohon dan kompleksitas ruangnya adalah O(bd).

E. Perbandingan BFS dan DFS



Gambar 4. BFS vs DFS
(sumber : https://medium.com/basecs/breaking-down-breadth-first-search-cebe696709d9)

Algoritma DFS dan BFS memiliki karakteristik berbeda untuk menangani kasus-kasus spesifik. Penggunaan BFS ataupun DFS yang tidak tepat dapat menyebabkan persoalan tidak selesai. Oleh karena itu, seorang *programmer* harus dapat mengetahui karakteristik dari masing-masing algoritma.

Karakteristik DFS :

1. DFS selalu mengarah ke kedalaman terlebih dahulu (anak dari simpul sebelumnya dikunjungi setelahnya, baru menyamping ke tetangga dari simpul tersebut).
2. Kebutuhan memori (space) cenderung linear terhadap graf yang ingin ditelusuri.
3. Terdapat kemungkinan bahwa pencarian DFS selalu mengarah kebawah dan tidak pernah ke samping oleh karena kedalaman yang tak hingga.
4. Jika terdapat lebih dari 1 solusi, DFS belum tentu akan menghasilkan jalur yang terpendek.

Karakteristik BFS:

1. BFS selalu mengarah ke samping terlebih dahulu (setiap anak dari sebuah simpul dikunjungi terlebih dahulu, baru anak dari anak simpul yang dikunjungi)
2. Kebutuhan memori (space) cukup besar karena faktor branching (percabangan).
3. BFS selalu menghasilkan solusi dengan jarak terpendek (jumlah node yang dikunjungi minimal)

Oleh karena itu, kedua algoritma tersebut cocok apabila digunakan untuk :

Permasalahan yang optimal diselesaikan dengan DFS:

1. Permasalahan yang himpunan solusinya merupakan kombinasi dari setiap simpul/jumlah simpul dalam himpunan solusi pasti sama dengan

jumlah simpul. Contohnya adalah crossword puzzle di mana solusi merupakan kondisi board yang terdiri dari setiap kata yang ada.

2. Permasalahan yang memiliki graf/pohon yang terlalu besar untuk ditelusuri dengan BFS.
3. Permasalahan yang mencari jalur terjauh (jumlah node maksimal)

Permasalahan yang optimal diselesaikan dengan BFS:

1. Permasalahan untuk mengoptimasikan jarak (jumlah node minimum). Contoh : mencari lajur minimum dari 2 simpul pada sebuah graf, mencari jarak minimum, dsb.

Sehingga untuk permasalahan penyampaian data dalam jaringan *peer to peer* berdasarkan karakteristik yang ada, Penulis mengasumsikan terlebih dahulu bahwa algoritma BFS lebih cocok karena diperlukan jalur yang pendek untuk menyampaikan pesan/data. Alasan mengapa jalur terpendek penting dalam permasalahan ini adalah sebagai berikut:

1. Dalam sebuah jaringan komputer, terdapat jeda/*latency* dalam pengiriman sebuah data. Pengiriman data juga memerlukan waktu untuk proses *upload* dan *download* antar perangkat. Sehingga, semakin banyak node yang dikunjungi, semakin besar kemungkinan bahwa waktu yang dihabiskan lebih lama.
2. Kebanyakan aplikasi cenderung lebih mengutamakan efisiensi waktu (*responsivity*) dibandingkan memori. Contohnya dalam aplikasi chatting, pengguna mengutamakan kecepatan dalam mengirim dan menerima pesan dibandingkan RAM/CPU yang dipakai, selama masih dalam batas wajar.
3. Apabila jaringan cukup besar, terdapat kemungkinan terputusnya salah satu node menyebabkan perubahan jalur dalam pengiriman data, sehingga apabila menggunakan DFS, jarak dapat menjadi lebih jauh lagi.
4. DFS memiliki resiko untuk membentuk sirkuit jauh lebih besar dibandingkan BFS, sehingga perlu penanganan khusus. Faktor ini biasanya tidak sulit untuk ditangani, namun karena proses traversal pada jaringan *peer-to-peer* tidak dilaksanakan pada satu perangkat saja maka informasi node-node yang telah dikunjungi juga harus disertakan pada data yang dikirim, menyebabkan ukuran data semakin besar dan waktu semakin lama.

III. Penerapan Algoritma dan Uji Coba

A. *Lingkungan Uji Coba*

Dalam mensimulasikan sebuah jaringan *peer-to-peer* sedekat mungkin dengan kenyataan yang terjadi pada implementasi sesungguhnya, Penulis membuat lingkungan uji coba pada jaringan lokal dengan memanfaatkan fitur *socket* dan *multi-threading* pada bahasa Python 3. Aplikasi yang dibuat dalam lingkungan tersebut adalah sebuah aplikasi *chatting peer-to-peer* sederhana di mana setiap *node* dapat bertukar pesan dengan node lainnya. Deskripsi jaringan yang Penulis simulasikan adalah sebagai berikut :

1. Sebuah node disimulasikan sebagai satu instansi program yang berjalan dan mendengarkan pada sebuah port spesifik. Penulis menggunakan port 9000 – 9003 (terdapat 4 buah instansi). Setiap node memiliki id unik berupa *public key* yang dihasilkan dari algoritma RSA.
2. Setiap node hanya mengetahui IP address dari node yang bertetangga dengannya secara langsung. Jika node tidak bertetangga dengan node tujuannya, maka dilakukan message passing secara BFS/DFS melalui node perantara.
3. *Latency* disimulasikan dengan adanya waktu jeda antar proses pengiriman pesan.
4. Setiap *node* memiliki data ketetanggaan setiap *node* lainnya pada jaringan yang di-*broadcast* setiap ada node baru yang bergabung.
5. Data yang dikirim berupa JSON (*Javascript Serial Object Notation*) yang diserialisasi menggunakan *library pickle* yang terdapat pada Python 3. Data pesan dienkripsi dengan algoritma RSA. Setiap data memiliki id unik yang terdiri atas 30 karakter. Apabila sebuah data telah pernah diproses, maka data tersebut diabaikan.
6. Setiap koneksi antar node akan diatur oleh sebuah *thread* sehingga tidak ada waktu yang terbuang oleh karena proses lain.

Proses yang dilakukan ketika ada node baru yang masuk :

1. Node baru bertindak sebagai client dan mengirimkan alias (username), list tetangganya, dan *public key* dari node tersebut. Node lama (server) akan mencatat dan mengirimkan data yang sama ke pada client.
2. Kemudian kedua node akan mengirimkan broadcast list tetangganya kepada seluruh jaringan dengan cara :
    a. Mengirimkan data ketetanggaan kepada seluruh tetangga yang node tersebut miliki.
    b. Semua tetangga dari node tersebut akan meneruskan data tersebut ke tetangganya lagi. Ulangi hingga seluruh node tercapai. Tidak akan terjadi infinite loop karena terdapat id unik pada data tersebut. Data dengan id yang sama tidak akan pernah diproses kembali.

B. *Implementasi Algoritma*

1. BFS

Setiap status berisi tuple (node, path)

Listing 1 Algoritma BFS

```
Input
    1.    Array of tetangga (neighbor)
    2.    Array of ketetanggan (node1, node2)
        (rel_neighbor)
    3.    Data yang akan dikirim (data)

Algoritma
//Inisialisasi queue
queue = [(n, []) for n in neighbor]
visited = []
//Loop selama stack tidak kosong
While stack not empty
    Current_node, path = queue.head()
    If current_node in visited then
      Continue
    Else if current_node == data.sender then
      Continue
    path = path + [current_node]
    visited = visited + [current node]

    if current_node == data.target then
        send(path[0],data)
//node path[0] akan melakukan algoritma yang sama
//dan meneruskannya ke node selanjutnya
        break

    for tuple in rel_neighbor
        if tuple[0] == current_node then
              queue.add((tuple[1], path))
        elif tuple[1] == current_node then
              queue.add((tuple[0], path))
```

2. DFS

Setiap status berisi data node

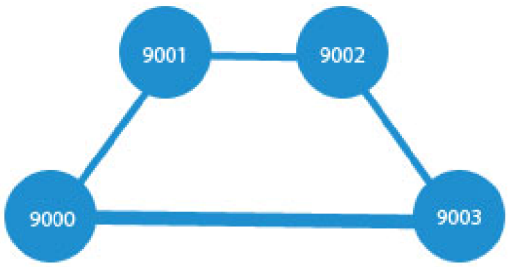
Listing 2 Algoritma DFS

```
Input
    1.    Array of tetangga (neighbor)
    2.    Array of ketetanggan (node1, node2)
        (rel_neighbor)
    3.    Data yang akan dikirim (data)

Algoritma
//Inisialisasi stack
stack = neighbor
visited = []
path = []
//Loop selama stack tidak kosong
While stack not empty
    Current_node = stack.pop()
    If current_node in visited then
      Continue
    Else if current_node == data.sender then
      Continue
    path = path + [current node]
    visited = visited + [current_node]

    if current_node == data.target then
        send(path[0],data)
//node path[0] akan melakukan algoritma yang sama
//dan meneruskannya ke node selanjutnya
        break

    added = False

    for tuple in rel_neighbor
        if tuple[0] == current_node then
              stack.push(tuple[1])
              added = True
        elif tuple[1] == current_node then
              stack.push(tuple[0])
              added = True

    if not added then
        path = path[:-1] //hapus dari path
```

Penulis kemudian akan membandingkan efisiensi kedua algoritma berdasarkan waktu yang diperlukan dalam pengiriman pesan dan jalur yang diambil oleh program tersebut dalam mengirimkan pesan.

C. *Kasus Uji Coba*

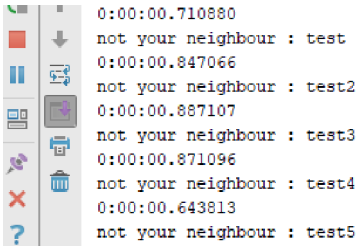
Pada uji coba ini, perangkat disusun seperti pada gambar berikut :



Gambar 5 Jaringan *peer-to-peer* uji coba

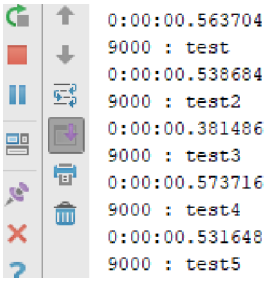
Setelah dilakukan uji coba dengan 5x percobaan, didapatkan hasil sebagai berikut (node yang menerima pesan adalah node 9003 dari node 9000):

1. DFS



Gambar 6 Hasil Uji Coba DFS

2. BFS



Gambar 7 Hasil Uji Coba BFS

Dengan path yang diambil :

1. DFS

   9000 –> 9001 –> 9002 –> 9003

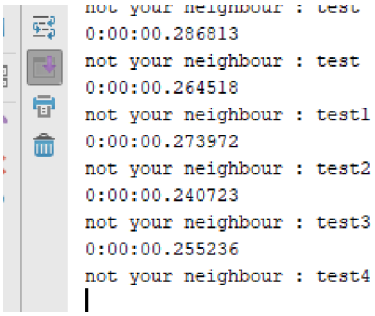
2. BFS

   9000 –> 9003

Penulis kemudian ingin menghilangkan jeda waktu yang digunakan untuk mensimulasikan *latency* antar node sehingga terdapat sedikit sekali perbedaan waktu yang dihasilkan. Namun karena keterbatasan komputer yang penulis miliki, penulis tidak dapat mencatat hasilnya karena CPU dan RAM terpakai 100% disebabkan oleh infinite loop yang tidak memiliki jeda waktu.

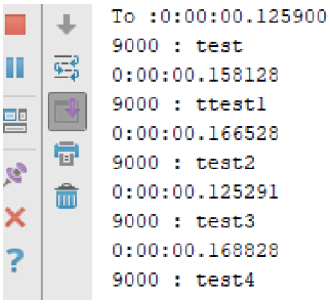
Oleh karena itu, penulis kemudian menguji coba kembali dengan jeda seminimal mungkin. Berikut merupakan hasil uji coba ke dua dengan graf yang sama namun jeda waktu yang berbeda.

1. DFS



Gambar 8 Hasil Uji Coba Kedua DFS

2. BFS



Gambar 9 Hasil Uji Coba Kedua BFS

IV. Analisa

Dari uji coba yang dilakukan, terlihat dengan jelas bahwa jalur yang dihasilkan dengan algoritma BFS lebih singkat dan memakan waktu yang jauh lebih sedikit walau hanya berbeda 2 node.

Hal ini membuktikan bahwa untuk sebuah jaringan *peer-to-peer* di mana setiap node memegang info ketetanggaan node lainnya, algoritma *breadth first search* jauh lebih optimal untuk digunakan.

Tentunya hal ini juga disebabkan oleh karena faktor *latency* yang dibuat sangat tinggi. Dapat dilihat ketika faktor *latency* dikurangi, jeda waktu berkurang secara drastits. Apabila faktor *latency* ditiadakan, misalkan pada jaringan lokal tanpa ada jeda antar proses, maka waktu yang dibutuhkan algoritma DFS dan BFS hampir tidak ada bedanya.

Kedua algoritma ini dapat dilakukan karena masing-masing node mengetahui tetangga dari node lainnya. Namun hal ini juga menjadi kelemahan dari jaringan ini. Terdapat beberapa masalah yang dapat terjadi pada jaringan *peer-to-peer* ini :

1. Jaringan mempunyai skalabilitas yang rendah

   Misalkan jaringan ini memiliki 500 node. Maka setiap node akan memegang data sebanyak 500 dikalikan dengan jumlah rata-rata tetangga pada setiap node. Selain itu, setiap node akan kewalahan untuk memperbaharui data ketetanggaannya karena setiap node harus menyebarkan list ketetanggannya sendiri ke seluruh node pada jaringan.

2. Jaringan rawan terhadap serangan MITM

   Jika jaringan disusupi oleh sebuah node yang bertindak sebagai *man in the middle* dan melakukan serangan dengan salah satu cara seperti :

   1. Tidak meneruskan *broadcast* data ketetanggaan sebuah node
   2. Mengaku-ngaku sebagai tetangga dari sebuah node padahal bukan.

   Maka jaringan ini dengan mudahnya akan hancur. Data dapat dicegah sebelum sampai ke node tujuan walaupun penyerang tidak dapat mengubah atau melihat isi pesan karena data terenkripsi. Oleh karena itu diperlukan suatu cara untuk membuat data ketetanggaan menjadi tidak perlu di*broadcast* ke seluruh jaringan. Terdapat beberapa solusi, yaitu :

   1. Membuat jaringan semi-desentralisasi

      Terdapat beberapa server yang bertindak sebagai penyimpan data ketetanggaan seluruh node pada jaringan.

   2. Membuat data ketetanggaan sebagai sebuah fungsi hash yang dapat dihitung jaraknya secara *lexicographic.*

      Jika list tersebut dapat digantikan dengan sebuah fungsi yang dapat dihitung jaraknya maka tidak diperlukan lagi sistem penyebaran data, cukup dengan menggunakan fungsi yang sama pada setiap node. Algoritma yang digunakan untuk menelusuri jaringan juga tidak lagi memakai BFS dan DFS biasa, namun dengan menggunakan algoritma seperti *backtrack* atau *random walker*. Penulis tidak dapat membuat implementasinya karena kesulitan dalam pembuatan jaringannya.

Jaringan seperti ini digunakan pada beberapa *service* berskala dunia seperti *spotnet/usenet*.

V. KESIMPULAN

Dari analisa yang telah dilakukan, penulis menarik kesimpulan bahwa algoritma BFS dalam penyampaian data pada jaringan *peer-to-peer* dimana setiap *node/peer* dengan *latency* atau jeda waktu yang signifikan lebih baik daripada algoritma DFS. Jalur yang dihasilkan algoritma BFS selalu lebih baik atau sama dengan jalur yang dihasilkan oleh algoritma DFS.

Namun dari segi pemakaian memori dan skalabilitas, penggunaan algoritma DFS dengan variasinya seperti *backtracking* dapat menjadi opsi yang dapat dipertimbangkan oleh karena efisiensinya dalam pemakaian memori.

UCAPAN TERIMA KASIH

Pertama, penulis mengucapkan syukur kepada Tuhan Yang Maha Esa, oleh karena kasih karunia-Nyalah penulis dapat menyelesaikan makalah berjudul "Perbandingan Algoritma BFS dan DFS dalam Pengiriman Data pada Jaringan Peer to Peer" tepat pada waktunya. Penulis juga ingin mengucapkan terima kasih kepada orang tua penulis yang telah mendukung penulis selama proses pembuatan makalah ini. Penulis tidak lupa untuk mengucapkan terima kasih juga kepada M.T., Dr. Nur Ulfa Maulidevi, S.T., M.Sc., Dr. Ir. Rinaldi Munir, dan M.T., Dr. Masayu Leylia Khodra, S.T. selaku dosen mata kuliah IF2211 Strategi Algoritma yang telah membimbing dan memberikan penulis materi selama proses pembelajaran pada mata kuliah Strategi Algoritma.

REFERENCES

[1] Munir, Rinaldi. Diktat Kuliah Strategi Algoritma. unpublished

[2] Lawrence L. Larmore, DFS and BFS Algorithms using Stacks and Queues (http://www.egr.unlv.edu/~larmore/Courses/CSC477/bfsDfs.pdf)

[3] Rüdiger Schollmeier, *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (2002)

[4] Bellare, Mihir. "Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements." Springer Berlin Heidelberg, 2000. Page 1.

[5] Delfs, Hans & Knebl, Helmut (2007). "Symmetric-key encryption". *Introduction to cryptography: principles and applications*. Springer

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 13 Mei 2018

Gabriel Bentara Raphael
13516119