Penerapan Algoritma Decrease and Conquer untuk Menentukan Bandwidth Minimum Sinkronisasi File sebelum Deadline Menggunakan Binary Search on Answer

Timothy Bernard Soeharto - 13524092
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: berdzhart@gmail.com, 13524092@std.stei.itb.ac.id

***Abstrak*—Sinkronisasi file ke penyimpanan cloud sering dibatasi oleh jendela waktu, misalnya backup malam hari sebelum perangkat dimatikan atau jaringan dipakai kembali pada pagi hari. Masalah praktis yang muncul adalah menentukan bandwidth minimum agar seluruh file selesai disinkronkan sebelum deadline. Makalah ini memodelkan proses sinkronisasi sebagai penjadwalan sejumlah file pada beberapa slot unggah paralel dengan overhead koneksi per file. Predikat kelayakan mengevaluasi apakah sebuah nilai bandwidth cukup untuk menyelesaikan semua file sebelum deadline. Karena predikat tersebut monoton, pencarian bandwidth minimum dapat dilakukan dengan binary search on answer, salah satu penerapan decrease and conquer dengan pengurangan ruang solusi sebesar faktor dua pada setiap iterasi. Eksperimen pada lima dataset sintetis menunjukkan bahwa binary search menghasilkan bandwidth minimum yang sama dengan pencarian linear, tetapi hanya membutuhkan belasan pemanggilan predikat, sedangkan pencarian linear membutuhkan puluhan hingga lebih dari seribu pemanggilan. Hasil ini menunjukkan bahwa pendekatan decrease and conquer cocok untuk masalah optimasi diskret yang memiliki sifat monoton dan batas jawaban yang jelas.**

Kata kunci—binary search on answer, decrease and conquer, sinkronisasi file, bandwidth minimum, penjadwalan.

I. Pendahuluan

Sinkronisasi file sudah menjadi kegiatan rutin pada perangkat pribadi maupun lingkungan akademik. Folder tugas, kode program, dataset praktikum, rekaman video, dan arsip laboratorium sering disalin ke layanan cloud agar dapat diakses dari beberapa perangkat dan tidak hilang ketika perangkat lokal bermasalah. Walaupun proses ini tampak sederhana, pengguna biasanya tidak hanya peduli apakah sinkronisasi selesai, tetapi juga kapan sinkronisasi selesai. Backup malam hari, unggahan dataset sebelum asistensi, atau pemindahan arsip laboratorium sebelum jaringan kampus kembali dipakai merupakan contoh situasi yang memiliki batas waktu alami.

Salah satu pertanyaan yang sering muncul dalam situasi tersebut adalah: berapa bandwidth minimum yang dibutuhkan agar semua file selesai diunggah sebelum deadline? Jika bandwidth terlalu kecil, proses sinkronisasi melewati batas waktu. Jika bandwidth terlalu besar, alokasi jaringan menjadi boros dan dapat mengganggu trafik lain. Layanan transfer modern bahkan menyediakan fitur penjadwalan, filter file, transfer inkremental, dan pembatasan bandwidth agar transfer tidak mengganggu aktivitas utama [6]. Dokumentasi migrasi data cloud juga menunjukkan bahwa ukuran data dan kecepatan jaringan berpengaruh langsung terhadap lama transfer; sebagai contoh, transfer 100 TB pada 1 Gbps masih dapat membutuhkan lebih dari 10 hari [7].

Makalah ini membahas versi sederhana dari masalah tersebut dengan fokus pada strategi algoritma, bukan pada rekayasa jaringan lengkap. Masukan masalah berupa daftar ukuran file, jumlah slot unggah paralel, overhead koneksi per file, dan deadline. Keluaran yang dicari adalah bandwidth minimum dalam Mbps agar simulasi sinkronisasi selesai sebelum deadline. Kontribusi makalah ini adalah formulasi predikat kelayakan yang monoton dan penerapan binary search on answer sebagai algoritma decrease and conquer untuk menemukan jawaban minimum secara efisien.

Topik ini dipilih karena cukup dekat dengan kehidupan mahasiswa, mudah dibuat program eksperimennya, dan tetap menunjukkan ide utama strategi algoritma. Masalah ini bukan sekadar studi literatur: makalah menyajikan model, algoritma, implementasi simulasi, dan hasil eksperimen yang membandingkan binary search dengan pencarian linear sebagai baseline.

II. Dasar Teori dan Pemodelan

A. *Decrease and Conquer dan Binary Search on Answer*

Decrease and conquer adalah strategi perancangan algoritma yang mereduksi persoalan menjadi persoalan yang lebih kecil, kemudian hanya memproses satu subpersoalan yang relevan. Dalam bahan kuliah IF2211, metode ini dibedakan dari divide and conquer karena divide and conquer memproses seluruh subpersoalan dan menggabungkan solusinya, sedangkan decrease and conquer hanya melanjutkan ke satu bagian [2]. Salah satu varian yang penting adalah decrease by a constant factor, ketika ukuran ruang pencarian berkurang dengan faktor tetap pada setiap iterasi.

Binary search merupakan contoh klasik decrease by a constant factor. Algoritma ini mempertahankan interval kandidat jawaban dan memilih titik tengah. Berdasarkan hasil perbandingan atau evaluasi predikat, separuh interval dibuang karena tidak mungkin memuat jawaban. CP-Algorithms

menjelaskan bahwa binary search bekerja dengan membelah interval pencarian dan menghasilkan jumlah langkah logaritmik terhadap ukuran interval [4]. Dalam konteks pencarian jawaban, bukan elemen array yang dicari, melainkan nilai minimum atau maksimum yang memenuhi suatu predikat.

Binary search on answer membutuhkan sifat monoton. USACO Guide merumuskan bahwa pencarian pada fungsi boolean hanya valid jika fungsi tersebut monoton, misalnya semua nilai di bawah batas bernilai false dan semua nilai di atas atau sama dengan batas bernilai true [5]. Pada masalah bandwidth, predikat P(B) didefinisikan sebagai 'bandwidth B cukup untuk menyelesaikan sinkronisasi sebelum deadline'. Jika P(B) bernilai true, maka P(B') juga true untuk setiap B' > B karena kecepatan transfer tidak lebih lambat ketika bandwidth diperbesar. Sifat ini membuat jawaban minimum dapat dicari dengan binary search.

B. Sinkronisasi File dan Model Transfer

Masalah sinkronisasi file sendiri telah lama dipelajari. Algoritma rsync, misalnya, berfokus pada pengurangan data yang dikirim melalui koneksi berlatensi tinggi dan bandwidth rendah dengan mengirim bagian file yang berubah saja [8]. Penelitian lain juga membahas sinkronisasi file jarak jauh dengan tujuan meminimalkan komunikasi jaringan [9]. Makalah ini tidak mencoba menggantikan rsync atau protokol sinkronisasi penuh. Fokusnya lebih sempit: setelah ukuran data yang perlu dikirim diketahui, bagaimana menentukan bandwidth minimum agar proses selesai sebelum batas waktu.

Diberikan n file dengan ukuran s1, s2, ..., sn dalam MB. Sistem sinkronisasi memiliki k slot unggah paralel. Setiap slot memproses satu file pada satu waktu. Total bandwidth yang dialokasikan adalah B Mbps dan dibagi rata ke k slot, sehingga satu slot memperoleh B/k Mbps. Untuk setiap file terdapat overhead tetap h detik, misalnya waktu membuka koneksi, membaca metadata, atau melakukan negosiasi awal. Dengan asumsi 1 MB = 8 megabit, waktu proses file i pada satu slot dimodelkan sebagai h + 8k si / B detik.

Agar model tetap mudah diimplementasikan, file dijadwalkan dengan heuristic Largest Processing Time (LPT): file diurutkan dari ukuran terbesar ke terkecil, lalu setiap file ditempatkan pada slot dengan beban sementara paling kecil. Heuristik ini tidak diklaim selalu optimal, tetapi cukup masuk akal untuk simulasi karena file besar ditempatkan lebih awal sehingga beban antar slot cenderung seimbang. Nilai yang diperiksa oleh predikat adalah makespan, yaitu waktu selesai slot paling lambat.

C. Predikat Monoton dan Komplekitas

Predikat kelayakan P(B) bernilai true jika makespan hasil simulasi dengan bandwidth B tidak melebihi deadline D. Secara matematis, jika B meningkat maka komponen 8k si / B tidak meningkat. Overhead h tetap. Karena seluruh durasi file tidak bertambah saat B dinaikkan, hasil penjadwalan LPT dengan daftar file yang sama juga tidak memiliki alasan untuk menjadi lebih lambat. Dengan demikian P(B) bersifat monoton non-decreasing terhadap B: setelah suatu bandwidth cukup, bandwidth yang lebih besar juga cukup.

Masalah optimasi yang dicari dapat ditulis sebagai mencari nilai B minimum dalam bilangan bulat positif sehingga P(B) = true. Ruang jawaban berada pada interval [1, U], dengan U dapat diperoleh melalui doubling: mulai dari 1 Mbps, batas atas dikali dua sampai predikat bernilai true. Setelah batas atas ditemukan, binary search dilakukan pada interval tersebut.

Model ini sengaja memakai satuan Mbps karena satuan tersebut umum dipakai untuk kapasitas jaringan. Ukuran file tetap disimpan dalam MB karena sistem operasi dan layanan penyimpanan biasanya menampilkan ukuran file dengan satuan byte. Konversi 1 MB = 8 megabit membuat hubungan antara ukuran file dan bandwidth menjadi eksplisit. Pada implementasi nyata, perbedaan antara MB desimal dan MiB biner dapat diperhatikan, tetapi dalam makalah ini perbedaan tersebut tidak memengaruhi sifat algoritma.

Deadline diperlakukan sebagai batas keras. Artinya, solusi yang selesai satu detik setelah deadline dianggap tidak layak meskipun secara praktis mungkin masih dapat diterima. Keputusan ini membuat predikat mudah didefinisikan dan menghindari ambiguitas. Jika pengguna ingin memberi toleransi, misalnya lima menit, toleransi tersebut dapat ditambahkan langsung ke nilai D tanpa mengubah algoritma.

Jika daftar file diurutkan setiap kali feasible dipanggil, kompleksitas predikat menjadi O(n log n + n log k). Dalam eksperimen ini pengurutan dilakukan sekali di awal karena urutan LPT tidak bergantung pada B. Dengan demikian satu pemanggilan predikat membutuhkan O(n log k). Karena k adalah jumlah slot unggah yang kecil, bagian ini praktis hampir linear terhadap jumlah file.

Binary search membutuhkan O(log U) pemanggilan predikat setelah batas atas U ditemukan. Proses doubling juga membutuhkan O(log B*) pemanggilan, dengan B* sebagai bandwidth minimum. Total kompleksitas menjadi O(n log k log U). Sebaliknya, pencarian linear dari 1 Mbps sampai B* membutuhkan O(B* n log k). Perbedaan ini besar ketika jawaban berada pada ratusan atau ribuan Mbps.

Dari sisi memori, algoritma hanya menyimpan daftar ukuran file dan heap berisi k beban slot. Memori tambahan di luar daftar input adalah O(k). Hal ini membuat pendekatan mudah dijalankan untuk ribuan sampai puluhan ribu file tanpa struktur data yang rumit.

Apabila ukuran file belum tersedia dan harus dibaca dari sistem berkas, terdapat biaya I/O tambahan untuk mengumpulkan metadata. Biaya tersebut berada di luar algoritma pencarian, tetapi tetap penting dalam implementasi nyata. Setelah daftar ukuran file diperoleh, pencarian bandwidth dapat dilakukan tanpa membaca isi file. Hal ini menjaga eksperimen tetap ringan dan tidak menyentuh data pengguna.

Kompleksitas juga dapat dilihat dari sudut pandang jumlah predikat. Jika bandwidth minimum adalah 1671 Mbps, pencarian linear dari 1 Mbps membutuhkan 1671 evaluasi. Binary search hanya membutuhkan sekitar log2(2048) evaluasi setelah batas atas ditemukan, ditambah evaluasi saat doubling. Perbedaan jumlah evaluasi inilah yang menjadi sumber efisiensi utama.

III. METODOLOGI

A. Algoritma Pencarian Bandwidth Minimum

Algoritma terdiri dari dua bagian. Bagian pertama adalah fungsi feasible(B) yang mengembalikan apakah bandwidth B cukup. Bagian kedua adalah binary search untuk mencari nilai B terkecil yang feasible. Struktur ini memisahkan model domain dari strategi pencarian, sehingga predikat dapat diganti tanpa mengubah pola binary search selama sifat monoton tetap dipertahankan.

Pada feasible(B), daftar ukuran file diurutkan menurun. Program menyimpan beban setiap slot dalam min-heap. Untuk setiap file, slot dengan beban terkecil dikeluarkan, durasi file ditambahkan, lalu beban baru dimasukkan kembali ke heap. Setelah semua file ditempatkan, nilai maksimum dari heap adalah makespan. Jika makespan tidak lebih besar dari deadline, bandwidth tersebut diterima.

Pseudocode berikut menunjukkan inti algoritma. Variabel sizes sudah terurut menurun agar file besar diproses lebih dulu. Fungsi heapPopMin mengambil slot dengan beban terkecil, sedangkan heapPush memasukkan kembali beban slot setelah file ditambahkan.

```
Algorithm 1 Binary Search on Answer
FEASIBLE(B): simulate LPT slot loads; return max(loads) <=
                              D
                    lo <- 1; hi <- 1
          while not FEASIBLE(hi) do hi <- 2 * hi
                      while lo < hi do
                 mid <- floor((lo + hi) / 2)
     if FEASIBLE(mid) then hi <- mid else lo <- mid + 1
                        return lo
```

Penggunaan heap membuat simulasi penjadwalan tetap sederhana. Tanpa heap, program harus mencari slot dengan beban terkecil menggunakan scan linear terhadap k slot untuk setiap file. Karena k kecil, scan linear sebenarnya masih cukup cepat. Namun heap menunjukkan bentuk yang lebih umum jika jumlah slot diperbesar, misalnya pada sistem transfer dengan banyak worker atau agent.

Bagian doubling sebelum binary search penting karena dalam banyak kasus pengguna tidak mengetahui batas atas bandwidth. Jika batas atas dipilih terlalu kecil, binary search dapat gagal karena seluruh interval bernilai false. Doubling menyelesaikan masalah ini secara adaptif. Setelah menemukan satu nilai hi yang feasible, jawaban pasti berada di antara 1 dan hi.

B. Contoh Perhitungan Manual

Untuk memperjelas cara kerja predikat, misalkan terdapat empat file berukuran 600 MB, 400 MB, 120 MB, dan 40 MB. Sistem memiliki k = 2 slot unggah paralel, overhead h = 0,2 detik per file, dan deadline D = 260 detik. Jika kandidat bandwidth B = 30 Mbps, maka setiap slot memperoleh 15 Mbps. Durasi file 600 MB menjadi 0,2 + 8*2*600/30 = 320,2 detik. Satu file terbesar saja sudah melewati deadline, sehingga P(30) pasti false.

Jika kandidat dinaikkan menjadi B = 80 Mbps, durasi file 600 MB menjadi 120,2 detik, file 400 MB menjadi 80,2 detik, file 120 MB menjadi 24,2 detik, dan file 40 MB menjadi 8,2 detik. Dengan LPT, file 600 MB masuk slot pertama dan file 400 MB masuk slot kedua. File 120 MB kemudian masuk slot kedua sehingga beban slot kedua menjadi 104,4 detik, lalu file 40 MB juga masuk slot kedua sehingga beban slot kedua menjadi 112,6 detik. Makespan adalah 120,2 detik, sehingga P(80) true.

Dari dua contoh tersebut terlihat pola monoton. Jika 30 Mbps tidak cukup, nilai yang lebih kecil juga tidak cukup karena semua durasi file akan lebih panjang. Jika 80 Mbps cukup, nilai yang lebih besar juga cukup karena semua durasi file akan lebih pendek atau sama. Binary search memanfaatkan fakta ini untuk membuang setengah kandidat pada setiap langkah.

Misalkan interval awal setelah doubling adalah [1, 128]. Binary search memeriksa 64 Mbps. Jika 64 Mbps feasible, interval dipersempit ke [1, 64]; jika tidak feasible, interval menjadi [65, 128]. Proses ini berlanjut sampai batas kiri dan kanan bertemu pada bandwidth minimum. Contoh kecil ini menunjukkan bahwa algoritma tidak perlu mencoba seluruh nilai 1 sampai 128.

Contoh manual juga menunjukkan mengapa masalah ini lebih menarik daripada hanya membagi total ukuran data dengan deadline. Overhead per file dan pembagian ke beberapa slot membuat waktu selesai bergantung pada penjadwalan. Dua dataset dengan total ukuran yang sama dapat memiliki makespan berbeda jika distribusi ukuran file berbeda. Dataset dengan satu file sangat besar dapat membutuhkan bandwidth lebih tinggi dibanding dataset dengan banyak file sedang yang lebih mudah diseimbangkan ke beberapa slot.

C. Rancangan Eksperimen

Eksperimen menggunakan lima dataset sintetis yang meniru folder backup. Ukuran file dibuat dengan distribusi campuran: sebagian besar file kecil, sebagian sedang, dan sebagian kecil file besar. Pola ini umum pada folder nyata karena satu direktori dapat berisi dokumen kecil, gambar, arsip, dan video. Generator memakai seed tetap agar hasil dapat direproduksi.

Parameter global yang digunakan adalah k = 4 slot unggah paralel dan overhead h = 0,18 detik per file. Skenario S1 berisi 120 file dengan deadline 10 menit. S2 berisi 500 file dengan deadline 15 menit. S3 berisi 1.500 file dengan deadline 20 menit. S4 berisi 5.000 file dengan deadline 45 menit. S5 berisi 10.000 file dengan deadline 30 menit. Untuk setiap skenario, algoritma binary search dibandingkan dengan pencarian linear yang mencoba 1, 2, 3, ... Mbps sampai menemukan bandwidth feasible.

Metrik yang dicatat adalah bandwidth minimum, jumlah pemanggilan predikat, waktu eksekusi, dan makespan pada bandwidth minimum. Kedua metode memakai fungsi feasible yang sama sehingga perbedaan hasil berasal dari strategi pencarian, bukan dari model simulasi. Pencarian linear dipakai sebagai baseline karena mudah dipahami dan pasti menemukan jawaban yang sama untuk ruang jawaban bilangan bulat positif, meskipun tidak efisien.

Dataset sintetis dibuat dengan distribusi campuran log-normal. Sekitar 82% file berada pada kelompok kecil, 15%

pada kelompok sedang, dan sisanya pada kelompok besar. Pendekatan ini meniru karakter folder nyata yang biasanya didominasi file kecil tetapi memiliki beberapa file besar yang memengaruhi waktu transfer. Ukuran file dibatasi agar tidak menghasilkan nilai ekstrem yang tidak realistis.

Semua eksperimen dijalankan pada komputer lokal dengan seed tetap. Karena waktu eksekusi sangat kecil, hasil waktu tidak dimaksudkan sebagai benchmark perangkat keras yang presisi. Angka waktu hanya digunakan untuk menunjukkan arah perbandingan. Metrik yang lebih penting adalah jumlah pemanggilan predikat, karena metrik tersebut tidak terlalu bergantung pada mesin yang menjalankan program.

Program eksperimen menghasilkan berkas CSV sehingga tabel dapat diperiksa ulang. Dengan cara ini, angka pada makalah tidak ditulis manual secara terpisah dari eksperimen. Jika parameter seperti jumlah slot atau deadline diubah, dokumen dapat diregenerasi sehingga tabel, grafik, dan narasi tetap konsisten.

IV. Hasil dan Pembahasan

A. Hasil Eksperimen

Tabel I menampilkan hasil eksperimen. Semua skenario menunjukkan bahwa binary search dan pencarian linear menemukan bandwidth minimum yang sama. Perbedaan terletak pada jumlah evaluasi predikat. Binary search hanya membutuhkan belasan sampai sekitar dua puluh evaluasi, sedangkan pencarian linear membutuhkan sebanyak nilai bandwidth minimum karena memeriksa kandidat dari 1 Mbps secara berurutan.

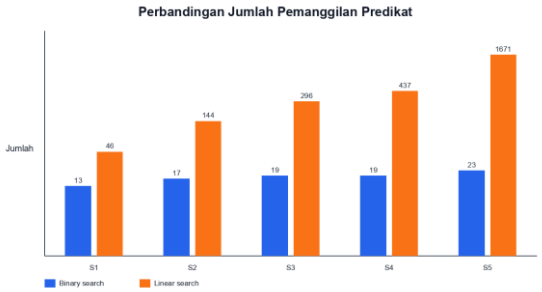
Pada eksperimen terbesar, binary search memakai 23 pemanggilan predikat untuk menemukan 1671 Mbps, sedangkan pencarian linear memakai 1671 pemanggilan.

Gambar 1 memperlihatkan jumlah pemanggilan predikat pada skala logaritmik. Selisih antara kedua metode semakin terasa ketika bandwidth minimum membesar. Hal ini sesuai dengan analisis kompleksitas: binary search tumbuh mengikuti logaritma batas jawaban, sedangkan pencarian linear tumbuh mengikuti nilai jawabannya. Pada masalah nyata, evaluasi predikat bisa lebih mahal karena melibatkan model jaringan, simulasi retry, pembacaan metadata, atau estimasi kompresi. Dalam kondisi tersebut, pengurangan jumlah evaluasi menjadi lebih penting.

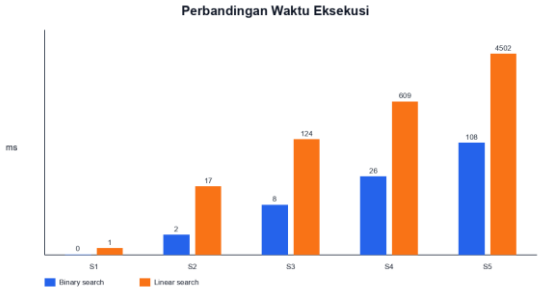
Gambar 2 memperlihatkan waktu eksekusi. Waktu absolut dalam eksperimen ini kecil karena model sengaja dibuat sederhana dan berjalan lokal. Namun pola relatifnya tetap terlihat: binary search lebih stabil, sedangkan pencarian linear membesar seiring bandwidth minimum. Perbedaan waktu pada skenario kecil tidak terlalu penting, tetapi pada skenario besar binary search memberikan ruang lebih besar untuk memperkaya predikat tanpa membuat eksperimen lambat.

Tabel I. Hasil eksperimen pencarian bandwidth minimum.

Sk.	n	Data	B min	Pred. BS/linear
S1	120	3.1 GB	46 Mbps	13/46
S2	500	15.4 GB	144 Mbps	17/144
S3	1500	40.8 GB	296 Mbps	19/296
S4	5000	131.8 GB	437 Mbps	19/437
S5	10000	275.2 GB	1671 Mbps	23/1671



Gambar 1. Perbandingan jumlah pemanggilan predikat.



Gambar 2. Perbandingan waktu eksekusi.

B. Analisis Hasil

Hasil ini juga menunjukkan karakter penting binary search on answer: algoritma tidak memperkirakan jawaban dengan rumus langsung, melainkan menggunakan predikat keputusan. Pendekatan ini berguna ketika rumus tertutup sulit dibuat karena ada overhead, slot paralel, kebijakan pembagian bandwidth, atau aturan penjadwalan. Selama predikat tetap monoton, binary search dapat digunakan untuk mencari batas minimum.

Pada S1, bandwidth minimum hanya 46 Mbps sehingga pencarian linear masih cukup ringan. Namun binary search tetap membutuhkan lebih sedikit evaluasi. Pada S2 dan S3, jawaban meningkat menjadi 144 Mbps dan 296 Mbps. Di titik ini, linear search mulai terlihat boros karena setiap kenaikan 1 Mbps memerlukan simulasi ulang terhadap seluruh daftar file.

S4 dan S5 memperlihatkan dampak yang lebih jelas. S4 memproses 5.000 file dengan total 131,8 GB dan deadline 45 menit. Jawaban minimumnya 437 Mbps. S5 memproses 10.000 file dengan total 275,2 GB dan deadline 30 menit, sehingga bandwidth minimum naik menjadi 1671 Mbps. Pada S5, binary search membutuhkan 23 evaluasi predikat, sedangkan linear search membutuhkan 1671 evaluasi. Artinya, jumlah evaluasi linear sekitar 72 kali lebih banyak.

Perbedaan tersebut tidak hanya soal kecepatan program eksperimen. Dalam sistem nyata, predikat dapat melibatkan simulasi yang lebih mahal, misalnya memperhitungkan perubahan throughput per jam, prioritas direktori, batas bandwidth per host, atau file yang dapat dikompresi. Jika satu evaluasi predikat mahal, strategi pencarian yang mengurangi jumlah evaluasi akan jauh lebih bernilai.

Hasil bandwidth minimum juga masuk akal terhadap ukuran data dan deadline. S5 memiliki total data sekitar 275 GB dan

deadline hanya 30 menit, sehingga kebutuhan bandwidth berada di atas 1 Gbps. Nilai ini sejalan dengan intuisi bahwa transfer ratusan GB dalam waktu singkat membutuhkan kapasitas jaringan besar. Dengan demikian, meskipun dataset sintetis, hasil tidak berada di luar skala teknis yang wajar.

C. *Batasan Model*

Model dalam makalah ini menyederhanakan jaringan. Bandwidth diasumsikan stabil, tidak ada packet loss, tidak ada retry, dan setiap slot memperoleh porsi bandwidth yang sama. Dalam jaringan nyata, throughput dapat berubah karena kemacetan, limitasi server, TLS, disk I/O, atau kebijakan quality of service. Oleh karena itu, angka bandwidth minimum dalam eksperimen tidak dimaksudkan sebagai rekomendasi operasional untuk layanan cloud tertentu.

Dataset yang digunakan juga sintetis. Keuntungannya adalah eksperimen dapat direproduksi, tetapi distribusi ukuran file mungkin tidak sama dengan folder nyata milik semua pengguna. Jika makalah ini dikembangkan lebih lanjut, dataset dapat diambil dari statistik folder lokal tanpa menyertakan isi file, misalnya hanya nama anonim dan ukuran file. Dengan cara itu, eksperimen tetap menjaga privasi tetapi lebih dekat dengan kondisi riil.

Heuristik LPT yang digunakan pada predikat bukan satu-satunya cara menjadwalkan file. Strategi lain seperti FIFO, Shortest Processing Time, atau penjadwalan berbasis prioritas dapat menghasilkan makespan berbeda. Namun fokus makalah adalah strategi pencarian bandwidth minimum. Selama predikat yang dipilih konsisten dan monoton, binary search tetap berlaku.

Ancaman validitas lain adalah asumsi pembagian bandwidth rata ke semua slot. Beberapa aplikasi sinkronisasi dapat mengalokasikan bandwidth secara dinamis: ketika hanya satu file aktif, file tersebut dapat memakai hampir seluruh bandwidth. Model dinamis seperti itu dapat menghasilkan makespan lebih pendek daripada model konservatif yang digunakan di sini. Namun model konservatif dipilih karena lebih mudah dianalisis dan tidak membutuhkan simulasi event yang kompleks.

Selain itu, eksperimen mengukur bandwidth dalam bilangan bulat Mbps. Pada layanan nyata, pengaturan bandwidth mungkin menerima nilai pecahan atau memakai satuan lain seperti MB/s. Jika satuan pecahan dipakai, binary search tetap dapat digunakan dengan batas toleransi tertentu, misalnya berhenti ketika interval lebih kecil dari 0,1 Mbps. Makalah ini memilih bilangan bulat agar baseline linear dan tabel hasil mudah dipahami.

V. KESIMPULAN DAN SARAN

Makalah ini memformulasikan masalah penentuan bandwidth minimum sinkronisasi file sebelum deadline sebagai masalah pencarian jawaban pada predikat monoton. Dengan memodelkan sinkronisasi sebagai penjadwalan file pada beberapa slot unggah paralel, setiap kandidat bandwidth dapat diuji melalui simulasi makespan. Karena bandwidth yang lebih besar tidak membuat proses lebih lambat, predikat kelayakan bersifat monoton dan cocok untuk binary search on answer.

Eksperimen pada lima dataset sintetis menunjukkan bahwa binary search menghasilkan jawaban yang sama dengan pencarian linear, tetapi dengan pemanggilan predikat jauh lebih sedikit. Pendekatan ini memperlihatkan penerapan decrease and conquer yang sederhana, mudah diprogram, dan relevan dengan masalah sehari-hari. Pengembangan berikutnya dapat memasukkan throughput dinamis, prioritas file, kompresi, retry, serta dataset ukuran file nyata agar model semakin mendekati sistem sinkronisasi modern.

Secara pedagogis, studi kasus ini menunjukkan bahwa decrease and conquer tidak terbatas pada pencarian elemen dalam array terurut. Selama sebuah persoalan optimasi dapat diubah menjadi predikat keputusan yang monoton, binary search dapat menjadi alat yang praktis untuk menemukan batas minimum atau maksimum. Inilah alasan binary search on answer sering muncul pada masalah optimasi diskret.

LAMPIRAN

A. *Reproduksibilitas Eksperimen*

Eksperimen pada makalah ini dapat direproduksi tanpa mengakses isi file pengguna. Data yang dibutuhkan hanya jumlah file, ukuran file, deadline, jumlah slot unggah, dan overhead per file. Untuk menjaga privasi, isi file, nama file asli, dan struktur direktori tidak perlu disimpan. Dalam eksperimen sintetis, ukuran file dihasilkan dari seed tetap 2401 sampai 2405 untuk skenario S1 sampai S5.

Langkah reproduksi adalah sebagai berikut. Pertama, buat daftar ukuran file dalam MB. Kedua, urutkan daftar tersebut secara menurun. Ketiga, jalankan fungsi feasible(B) untuk sebuah kandidat bandwidth B. Keempat, gunakan doubling untuk mencari batas atas feasible. Kelima, jalankan binary search pada interval yang diperoleh sampai batas kiri sama dengan batas kanan.

Parameter default yang digunakan adalah k = 4 slot unggah paralel dan h = 0,18 detik. Deadline tiap skenario adalah 10 menit, 15 menit, 20 menit, 45 menit, dan 30 menit. Perubahan nilai k atau h tidak mengubah struktur algoritma, tetapi dapat mengubah bandwidth minimum. Jika k diperbesar, overhead dapat tersebar ke lebih banyak slot, tetapi pembagian bandwidth per slot juga berubah sesuai model.

Untuk dataset nyata, daftar ukuran file dapat dikumpulkan dengan skrip sederhana yang membaca metadata sistem berkas. Setelah ukuran terkumpul, algoritma tidak perlu membuka isi file. Hal ini membuat eksperimen relatif aman untuk folder pribadi selama daftar nama file tidak dipublikasikan.

B. *Link Github*

https://github.com/tmthyberd/bandwidth-minimum-makalah/tree/main

REFERENSI

[1] R. Munir, "Strategi Algoritma 2025-2026," STEI ITB. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/ [Accessed: Jun. 11, 2026].

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[3] e-maxx contributors, "Binary search," Algorithms for Competitive Programming, last updated Aug. 19, 2025. [Online]. Available: https://cp-algorithms.com/num_methods/binary_search.html

[4] USACO Guide, "Binary Search," [Online]. Available: https://usaco.guide/silver/binary-search

[5] A. Tridgell and P. Mackerras, "The rsync algorithm," Australian National University, Tech. Rep., 1996. [Online]. Available: https://rsync.samba.org/tech_report/

[6] H. Yan, U. Irmak, and T. Suel, "Algorithms for Low-Latency Remote File Synchronization," in Proc. IEEE INFOCOM, 2008.

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, 19 Juni 2026

Timothy Bernard Soeharto
13524092