ファイル送信の検知に基づく重複排除を行うサーバの切り替えによるファイル転送時間の増加の抑制

三上 智徳[1]　高橋 風太[1]　串田 高幸[1]

概要：バックアップをする前に重複排除を実行するプリ・プロセス方式では，重複ファイルの数の送信時間分だけバックアップ時間を短縮することができる．しかし，重複排除を実行している際にユーザからファイルの送信があると，I/O が競合して処理を待つ時間が発生する．ここでの課題は，I/O が競合して処理を待つ時間が発生することによるユーザからファイルサーバへのファイル送信時間の増加である．本稿の提案手法では，重複排除の位置をファイルサーバとバックアップサーバで切り替えることで，I/O の競合による処理の待ち時間を抑制する．評価として，提案手法を用いた際と用いていない際のユーザからファイルサーバへのファイル送信時間を比較する．結果として，10GB の動画ファイルにおいて，ファイル送信時間が 29%減少した．

1. はじめに

背景

データのバックアップ操作は, ストレージの障害, 人的ミス, または災害によるデータの損失から保護するために, IT システム管理において重要である [1]．その中で，デジタルデータの量が増加するにつれて，重複排除はストレージとネットワークを節約するために必要になっている [2]. そのため, 重複排除方式は, 冗長データを削除して必要な合計ストレージスペースを削減するために広く使用されている [3].

重複排除は，重複を検出・排除する位置によって方式が分かれている[*1]．その方式の 1 つであるプリ・プロセス方式は，データを転送する前に，サーバ側で重複検出と排除を行う．他の方式であるポスト・プロセス方式は，データを転送したバックアップ先で重複検出と排除を行う．

ファイルサーバとは，クライアントのマシンやシステムの利益のために資料を保存するために存在している [4]．また，ネットワーク上 (例えば LAN) でのファイル管理を主な役割とするサーバである[*2]．利用する目的として，大きく 3 つある．1 つ目は，ファイルやデータを保存するためである．社内外問わずファイルサーバとつながっているパソコンで作成したファイルやデータは，ファイルサーバにも保管することができる．2 つ目は，ファイルを社内や社外に共有するためである．ユーザ間で常に最新のファイルやデータを共有することができる．3 つ目は，ファイルのバックアップをするためである．ファイルの管理を個々で行っていると，操作ミスやパソコンの不具合によってファイルが損失してしまうリスクがある．そのため，ファイルサーバにバックアップをすることで，ファイル損失のリスクを軽減することができる．

課題

ファイルサーバ側で重複排除を行うプリ・プロセス方式では，ユーザからのファイル送信とファイルサーバでの重複排除で I/O が競合して処理を待つ時間が発生する．それにより，ユーザがファイルサーバにファイルを転送する時間が増加することが課題である. 図 1 に課題となるケースを示す．サーバ側で重複排除を行うプリ・プロセス方式で重複排除を行った際に，どれだけファイルを転送する時間が遅くなるかを基礎実験する．

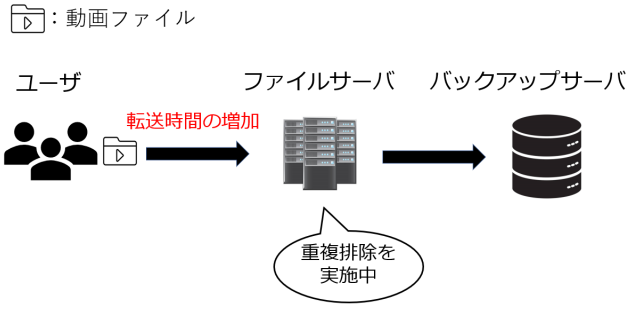

図 1　課題となるケース

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1
[*1] https://insights-jp.arcserve.com/deduplication
[*2] https://info.securesamba.com/media/10570/

基礎実験

サーバ側で重複排除を行っている際と行っていない際でディスクの使用状況とファイルの転送時間を確認する．重複排除には，fdupes コマンドを使用する．このコマンドを実行すると，MD5 でハッシュ値を生成してファイルを比較することで重複排除を行う．以下に基礎実験に用いる実験環境を示す．

- VM 構成情報 (ユーザ)
  OS：Ubuntu-22.04
  vCPU：1 コア
  RAM：1GB
  HDD：100GB
- VM 構成情報 (ファイルサーバ)
  OS：Ubuntu-22.04
  vCPU：1 コア
  RAM：1GB
  HDD：150GB
- VM 構成情報 (バックアップサーバ)
  OS：Ubuntu-22.04
  vCPU：1 コア
  RAM：1GB
  HDD：100GB

まず，ディスクの使用状況についてである．ディスクの使用状況を確認するために，vmstat と iostat コマンドを使用した．vmstat コマンドでは，カーネル・スレッド，仮想メモリ，ディスク，ハイパーバイザ，トラップおよびプロセッサのアクティビティに関する統計情報を表示することができる[*3]．iostat では，CPU の統計情報，システム全体，アダプタ，TTY デバイス，ディスク，CD-ROM，テープおよびファイルシステムに関する非同期入出力および入出力統計情報を表示することができる[*4]．vmstat の方では，wa の値の変化を確認した．wa の値は，I/O 待ちだった時間の割合を%で示している．図 2 に wa の値の推移を示す．グラフの横軸は，コマンドの実行時間を秒単位で表したものである．グラフの縦軸は，wa の値を%単位で表している．iostat の方では，%iowait の値と%util の値の変化を確認した．%iowait の値は，wa と同じように I/O 待ちだった時間の割合を%で示している．図 3 に%iowait の値の推移を示す．グラフの横軸は，図 2 と同様である．グラフの縦軸は，iowait の値を%単位で表している．%util の値は，デバイスが I/O 処理をしていたことによるビジー状態の時間の割合を%で示している．値が 100 に近いほどディスクを使用していることになる．図 4 に%util の値の推移を示す．グラフの横軸は，図 2 と同様である．グラフの縦軸は，util の値を%単位で表している．すべてのグラフに共通して，青色の線が「重複排除無し」を示しており，オレンジ色の線が「重複排除有り」を示している．

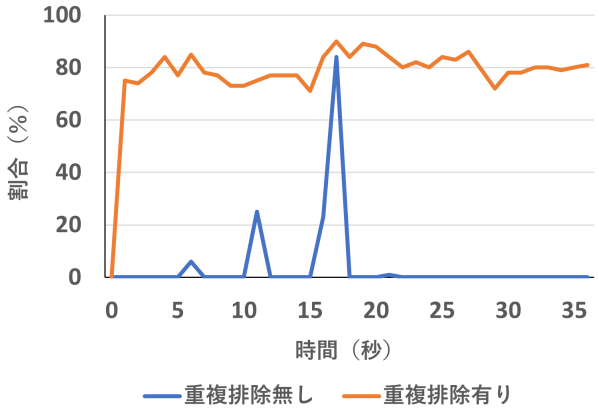

図 2　wa の値の推移

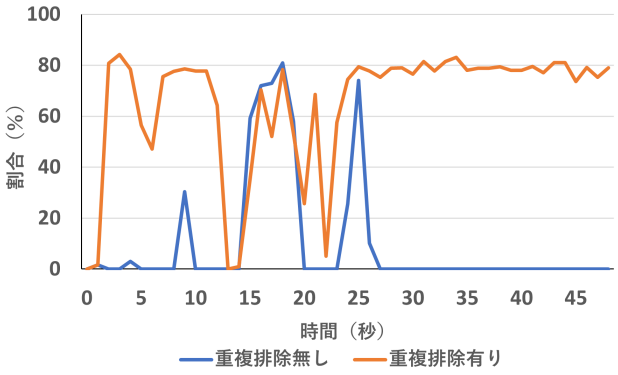

図 3　%iowait の値の推移

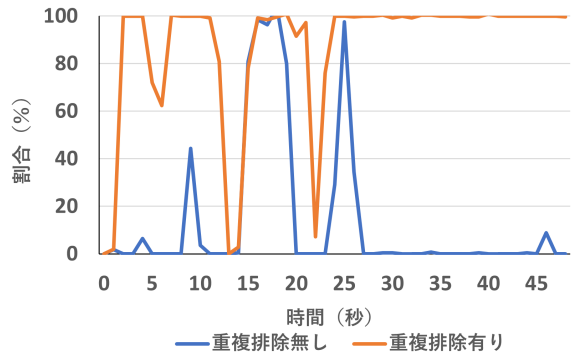

図 4　%util の値の推移

ディスクの使用状況を確認する基礎実験から，重複排除を行うことで，ディスクを使用していることを確認することができた．また，I/O が競合することで処理を待つ時間が発生して，ボトルネックとなっていることを確認することができた．

[*3] https://www.ibm.com/docs/ja/power7?topic=POWER7/p7hcg/vmstat.html
[*4] https://www.ibm.com/docs/ja/aix/7.2?topic=i-iostat-command

次に，ファイルの転送時間についてである．ファイルサーバで重複排除を行っている際と行っていない際の 2 つのファイル送信時間の結果を図 5 に示す．T1 が重複排除を行っていない際で T2 が重複排除を行っている際を示している．回数はそれぞれ 10 回行い，平均を取った．結果として，T1 が 3 分 15 秒であり，T2 が 4 分 21 秒となった．ファイルサーバで重複排除を行っている際にユーザからファイルサーバへファイルを送信すると，ファイル送信時間が増加していることが示された．

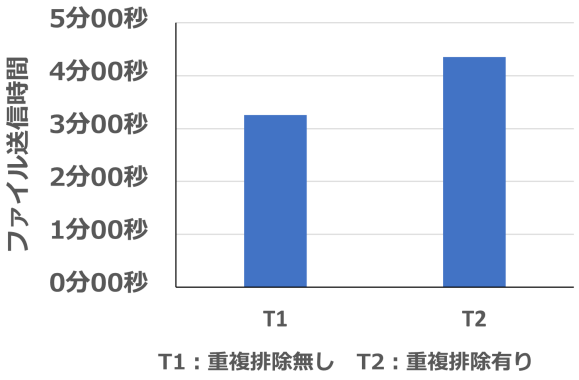

図 5　ファイル送信時間の基礎実験結果

各章の概要

第 2 章では，関連研究について述べる．第 3 章では，本稿の提案方式について述べる．第 4 章では，提案方式を基に開発したソフトウェアの実装と実験方法について述べる．第 5 章では，提案方式の評価と分析について述べる．第 6 章では，提案方式を基に開発したソフトウェアの議論について述べる．第 7 章では，本稿のまとめについて述べる．

2. 関連研究

Yan らは，動的にインラインとオフラインの重複排除を調整する DIODE を提案している [5]．まず，インラインとして重複排除の処理を行い，未処理のデータが直接オフラインの重複排除に渡されている．オフラインの重複排除では，優先度の順序で処理される．優先度の決定は，ファイルの拡張子に基づいて，ファイルを 3 つの異なる Dedup-Types に分類することで決定している．結果として，従来の重複排除よりも優れた I/O 性能と省スペースを同時に達成することができる．しかし，本稿ではファイル形式を動画ファイルに絞っているため課題を解決することができない．

Qianqian らは，評価指標である重複排除率のバランスを取り，重複排除システムの機能を強化するためにボトルネックを回避するための読み取り要求の並び替えおよび再編成戦略を提案している [6]．まず，パイプラインプールに読み取り要求をキャッシュして戦略を利用することで，ブロックを一意に識別するブロック ID に従って I/O 要求をグループに分割して，同一の要求をマージしてワークロードの昇格を回避する．次に，システムをインデックス内のフィンガープリントのシーケンスに基づいてリクエストを並び替えている．結果として，システムは 7.2%のランダムディスクアクセスを削減して最大 100mb/s のランダム読み取りを取得している．また，最大 672mb/s の高速書き込み速度を達成しており，プライマリストレージシステムの要件を完全に満たしている．この研究では，提案方式で閾値を利用して I/O 要求をグループに分割している．本稿では閾値を利用していないため課題を解決することができない．

Liu らは，セマンティックデータ重複排除である SDD を提案している [7]．SDD は，アーカイブファイルの I/O パス内のセマンティック情報であるファイルタイプ，ファイル形式，メタデータを利用して，ファイルをセマンティックチャンクに分割している．その後，各セマンティックチャンクは MD5，SHA1 の暗号化ハッシュ関数を使用して生成されたハッシュ値に従って重複排除されている．結果として，書き込み帯域幅が約 50%〜70%向上している．この研究の提案方式は，セマンティック情報を使用してセマンティックチャンクに分割することを前提としている．本稿ではファイル形式を動画ファイルに絞っており，ファイル形式によってセマンティックチャンクに分割することができないため課題を解決することができない．

Fan らは，k-means，Simhash，分析階層を含む 3 つのテクノロジーを採用したフレームワークである SFPS(Small File Process System) を設計した [8]．SFPS は，重複排除後に類似のファイルを適応ブロックスキップでマージすることにより，大量の小さなファイルと冗長データを削除することができる．また，関連付けられたファイルは次回に読み取られる可能性が高いため，高効率のデータベースである Redis に基づくプリフェッチメカニズムとメタデータ管理モジュールを設計して，高い読み取り率を保証するようにしている．SFPS を使用することで，ハードディスクの I/O とクラスタパフォーマンスを向上させることができる．結果として，比較対象である C&M と比較しても，ハードディスクの I/O が常に小さい値で推移していた．SFPS は，大量の小さいファイルがあることで効果を発揮する．本稿では動画ファイルを扱うため，1 つのファイルの容量が大きい．そのため，課題を解決することができない．

3. 提案方式

提案方式

本提案方式では，ユーザからファイルサーバへのファイル転送を検知して，重複排除の位置を切り替えることによってユーザからファイルサーバへのファイル転送時間の

増加を抑制する．図 6 に本稿の提案方式の流れを示す．

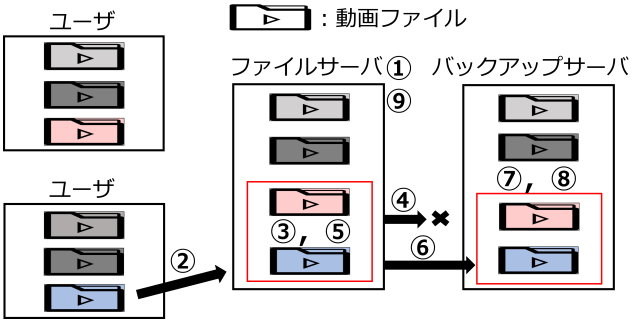

図 6　提案方式の流れ

図 6 の提案方式の流れについて図 6 に記載している数字①から⑨を用いて具体的に説明する．

① ファイルサーバで重複排除を開始：ファイルサーバでその日の深夜 12 時までに送られた動画ファイルに対しての重複排除を開始する．

② 動画ファイルの送信開始：ユーザからファイルサーバに動画ファイルの送信が開始される．

③ ファイルサーバでの重複排除を中止：ファイルサーバがファイルの作成を検知して，ユーザからの動画ファイルの送信開始と判定する．この判定で，ファイルサーバでの重複排除を中止する．

④ バックアップのための動画ファイルの送信は行わない：ユーザからファイルサーバへの動画ファイルの送信とファイルサーバからバックアップサーバへの動画ファイルの送信が同時に行われると，ユーザからファイルサーバへの動画ファイルの送信時間が増加する [9]．そのため，ユーザからファイルサーバへの動画ファイルの送信が終了するまで，ポスト・プロセス方式としての処理は行わない．

⑤ 動画ファイルの送信完了：ファイルサーバがファイルの書き込み完了を検知して，ユーザからの動画ファイルの送信完了と判定する．

⑥ ストレージへの動画ファイルの送信開始：その日の深夜 12 時までに送られた動画ファイル (前の日との差分ファイル) とファイルサーバが重複排除を実行中に送られた動画ファイルをバックアップのためにバックアップサーバへ送信する．

⑦ 動画ファイルの送信完了：バックアップサーバがファイルの書き込み完了を検知して，ファイルサーバからの動画ファイルの送信完了と判定する．

⑧ 重複排除開始：ポスト・プロセス方式として処理が行われているため，ファイルサーバ側で重複排除を実施していない．そのため，バックアップサーバ側で重複排除を開始する．

⑨ ポスト・プロセス方式からプリ・プロセス方式に戻す：その日のバックアップが完了して，次の日の業務開始時刻になったらポスト・プロセス方式からプリ・プロセス方式に戻して終了する．

ユーザからのファイル送信回数が 2 回目以降の処理

ユーザからの動画ファイルの送信が 1 回行われた時点で，その日のそれ以降の処理はポスト・プロセス方式となる．図 7 にユーザからのファイル送信回数が 2 回目以降の処理の流れを示す．

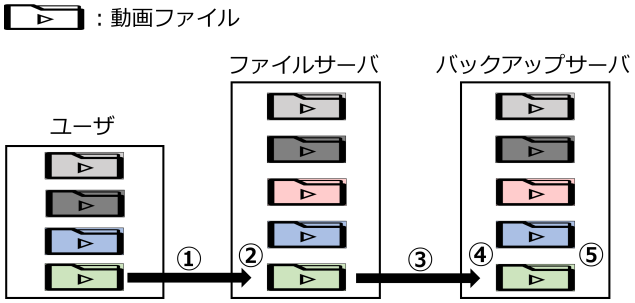

図 7　2 回目以降の提案方式の流れ

図 7 の提案方式の流れについて図 7 に記載している数字①から⑤を用いて具体的に説明する．

① 動画ファイルの送信開始：ユーザからファイルサーバに動画ファイルの送信が開始される．

② 動画ファイルの送信完了：ファイルサーバがファイルの書き込み完了を検知して，ユーザからの動画ファイルの送信完了と判定する．

③ バックアップサーバへの動画ファイルの送信開始：直前にユーザから送られた動画ファイルをバックアップのためにバックアップサーバへ送信する．

④ 動画ファイルの送信完了：バックアップサーバがファイルの書き込み完了を検知して，ファイルサーバからの動画ファイルの送信完了と判定する．

⑤ 重複排除開始：ポスト・プロセス方式として処理が行われているため，ファイルサーバ側で重複排除を実施していない．そのため，バックアップサーバ側で重複排除を開始する．

ユースケース・シナリオ

動画ファイルを扱う映像制作会社の動画ファイルの管理をユースケースとして想定する．ユースケースシナリオを図 8 に示す．動画ファイルの漏洩のリスクを防ぐため，社

内サーバを使用する．ユーザが動画ファイルの収容と共有を行えるようにファイルサーバを使用する．また，動画ファイルを保護するためにファイルサーバからバックアップサーバにバックアップすることを想定している．動画ファイルがファイルサーバ内にある場合，ファイルサーバの故障が原因で動画ファイルを損失してしまうリスクがある．そのため，動画ファイルを保存するためのバックアップサーバを想定している．

ユーザは，動画ファイルを収容・共有するためにファイルサーバに動画ファイルを送信する．動画ファイルのサイズは 10GB 以上としている．ファイルサーバでは，ストレージとネットワークを節約するために重複排除を実行する．また，ファイルサーバが故障した際にも動画ファイルを損失しないために，バックアップとしてバックアップサーバに動画ファイルを送信する．バックアップとしてバックアップサーバに動画ファイルを送信する開始時刻は深夜 12 時とする．ユーザがファイルを送信する際にファイルサーバで重複排除を実行していると，I/O の競合による処理の待ち時間が発生してユーザのファイル送信時間が増加する．そのため，本稿で提案するソフトウェアを使用することで I/O の競合による処理の待ち時間を抑制して，ユーザのファイル送信時間の増加を抑制することができる．

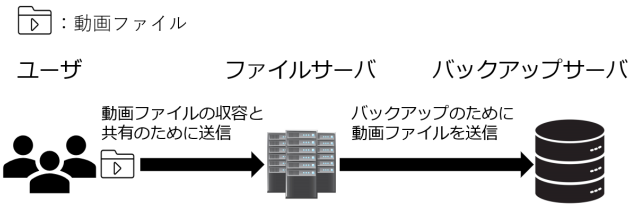

図 8　ユースケースシナリオ

4.　実装

提案手法を基にしたソフトウェアを新たに作成する．実装の全体構成図を図 9 に示す．

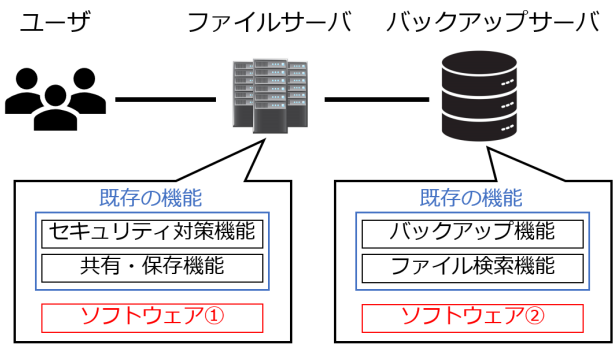

図 9　実装の全体構成図

ソフトウェアは，ファイルサーバ内とバックアップサーバ内に設置する．実装する際に，図 9 のユーザ，ファイルサーバ，バックアップサーバを前提とする．また，ファイルサーバには既存の機能としてセキュリティ対策機能と共有・保存機能がある．バックアップサーバには既存の機能としてバックアップ機能とファイル検索機能がある．

今回ソフトウェアを作成するにあたって，ディレクトリ内のファイルに対する変更を監視するために API である inotify を使用した．inotify とは，ファイルシステムイベントを監視するための機構を提供しており，ファイルやディレクトリを監視するのに使うことができる API である．

まず，ファイルサーバ内に設置するソフトウェア①について説明する．機能を以下に示す．

（1）　ユーザがファイルサーバにファイルを送信すると，inotify によりファイルの作成と書き込み完了を検知する．

（2）　inotify によりファイルの作成を検知した場合，kill コマンドを使用して重複排除に使用している fdupes コマンドを中止する．こうすることで，ファイルサーバで実行されている重複排除を中止することができる．

（3）　inotify によりファイルの書き込み完了を検知した場合，重複排除を実行していた動画ファイルとユーザから送られてきた動画ファイルをバックアップのためにバックアップサーバへ送信する．

図 10 にファイルサーバ内のソフトウェア①に実装する機能のフローチャートを示す．

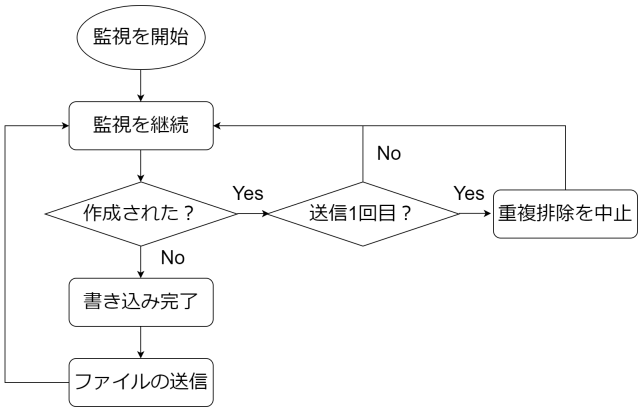

図 10　ファイルサーバ内のソフトウェアの機能のフローチャート

ユーザからのファイル送信をトリガーにして，重複排除を中止する．しかし，本稿の提案方式ではその日に一度ユーザからのファイル送信が行われてプリ・プロセス方式からポスト・プロセス方式に変更した場合，次の日の業務開始時刻になるまでプリ・プロセス方式に戻さない．そのため，ユーザからのファイル送信が 1 回目かどうかもトリガーにしている．また，書き込み完了をトリガーにして，

バックアップのためにファイルサーバからバックアップサーバに動画ファイルを送信する．

次に，バックアップサーバ内に設置するソフトウェア②について説明する．機能を以下に示す．

(1) ファイルサーバがバックアップサーバにファイルを送信すると，inotify によりファイルの書き込み完了を検知する．

(2) inotify によりファイルの書き込み完了を検知した場合，fdupes コマンドで重複排除を開始する．

図 11 にバックアップサーバ内のソフトウェア②に実装する機能のフローチャートを示す．

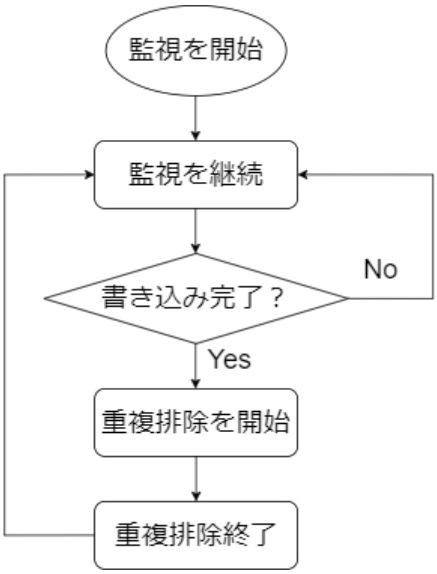

図 11 バックアップサーバ内のソフトウェアの機能のフローチャート

ファイルサーバからの動画ファイルの送信をトリガーにして，重複排除を実行する．ファイルサーバからの動画ファイルの送信がない際は，監視を継続する．

5. 評価と分析

課題となる際のファイルの送信，作成したソフトウェアを実行した際のファイルの送信の 2 つの送信時間を比較して評価を行う．

実験環境

実験環境は，基礎実験と同様に仮想マシン (VM) を用いる．ユーザ，ファイルサーバ，バックアップサーバでそれぞれ異なる VM を作成する．その中で，ファイルサーバとバックアップサーバの VM に作成したソフトウェアを設置する．ユーザがファイルサーバに送信する動画ファイルは，10GB の容量の動画ファイルを用いる．ファイルサーバでの重複排除には，10GB の動画ファイル，11GB の動画ファイル，12GB の動画ファイル，13GB の動画ファイル，10GB の動画ファイルを複製したファイルの計 5 つを用いる．重複排除には，基礎実験と同様に fdupes コマンドを使用する．このコマンドを実行すると，MD5 でハッシュ値を生成してファイルを比較することで重複排除を行う．図 12 に今回の実験環境を示す．

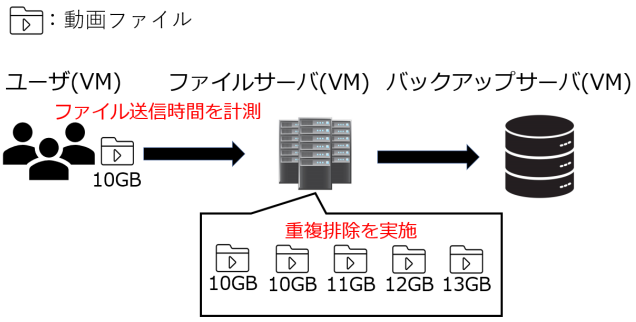

図 12 実験環境

実験結果と分析

開発したソフトウェアを使用した際のファイル送信時間と課題となるファイル送信時間の 2 つを比較した結果を図 13 に示す．T1 が開発したソフトウェアを使用した際のファイル送信時間，T2 がユーザからファイルサーバへのファイル送信中にファイルサーバで重複排除を行っていることで発生する課題のケースでのファイル送信時間を示している．回数はそれぞれ 10 回行い，平均を取った．実験の結果，T1 が 2 分 43 秒，T2 が 3 分 50 秒であった．

図 13 より，T1 は T2 と比較してファイル送信時間が 1 分 7 秒減少している．これは，提案手法を用いることで，ファイルサーバでの重複排除によるファイル送信時間の増加を抑制できていることが示されている．

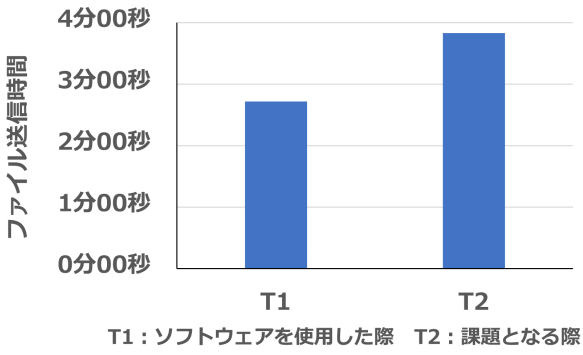

図 13 ファイル送信時間の実験結果

6. 議論

本稿の提案方式では，ファイルサーバで重複排除を実行している際にユーザからファイルの送信があった場合，重複排除が残り僅かな状態でも中止されるようになっている．

例として，5 個のファイル中 4 個のファイルまで重複排除が完了していたとしても中止されるようになっている．これは，重複排除が完了したファイル数を 1 個ずつ増やしながら実験を複数回行い，それぞれのバックアップ時間の平均を算出して最もバックアップ時間の平均が短い重複排除が完了したファイル数で中断するようにすることで改善が可能である．ファイル数を 1 個ずつ増やしていく理由として，今回使用した fdupes コマンドでの最小単位だからである．

7. おわりに

ユーザからファイルサーバへのファイルの送信とファイルサーバでの重複排除が同時に行われると，ファイルサーバ側で I/O が競合して処理を待つ時間が発生する．ここでの課題は，I/O が競合して処理を待つ時間が発生することによるユーザからファイルサーバへのファイル送信時間の増加である．提案では，重複排除の位置をファイルサーバとバックアップサーバで切り替えることで，I/O の競合による処理の待ち時間を抑制した．評価では，提案手法を用いた際と用いてない際のユーザからファイルサーバへのファイル送信時間を比較した．結果として，10GB の動画ファイルにおけるファイル送信時間は，29%減少した．

参考文献

[1] Yin, X., Alonso, J., Machida, F., Andrade, E. C. and Trivedi, K. S.: Availability modeling and analysis for data backup and restore operations, *2012 IEEE 31st Symposium on Reliable Distributed Systems*, IEEE, pp. 141–150 (2012).

[2] Zhengda, Z. and Jingli, Z.: A novel data redundancy scheme for de-duplication storage system, *2010 Third International Symposium on Knowledge Acquisition and Modeling*, pp. 293–296 (online), DOI: 10.1109/KAM.2010.5646152 (2010).

[3] Lu, G., Jin, Y. and Du, D. H.: Frequency based chunking for data de-duplication, *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, IEEE, pp. 287–296 (2010).

[4] Birrell, A. D. and Needham, R. M.: A universal file server, *IEEE transactions on software engineering*, No. 5, pp. 450–453 (1980).

[5] Tang, Y., Yin, J., Deng, S. and Li, Y.: DIODE: Dynamic inline-offline de duplication providing efficient space-saving and read/write performance for primary storage systems, *2016 IEEE 24th international symposium on modeling, analysis and simulation of computer and telecommunication systems (MASCOTS)*, IEEE, pp. 481–486 (2016).

[6] Xing, Q., Li, F. and Li, J.: RRSR: A Read Request Sorting and Reorganization Strategy to Improve Data Read Performance for Deduplication Systems Based on Sparse Index and Pipeline Parallelism, *2012 Fourth International Conference on Computational and Information Sciences*, pp. 1021–1024 (online), DOI: 10.1109/ICCIS.2012.270 (2012).

[7] Liu, C., Ju, D., Gu, Y., Zhang, Y., Wang, D. and Du, D. H.: Semantic Data De-duplication for archival storage systems, *2008 13th Asia-Pacific Computer Systems Architecture Conference*, pp. 1–9 (online), DOI: 10.1109/APCSAC.2008.4625441 (2008).

[8] Fan, Y., Wang, Y. and Ye, M.: An Improved Small File Storage Strategy in Ceph File System, *2018 14th International Conference on Computational Intelligence and Security (CIS)*, pp. 488–491 (online), DOI: 10.1109/CIS2018.2018.00116 (2018).

[9] 高橋風太，串田高幸：”ファイル受信の検知を用いたバックアップの中断によるファイル送信時間の増加の抑制”，クラウド・分散システム研究室，Vol. CDSL-TR-087 (2022).