社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－OS－100 (5)
2005／8／3

インターネットを介した協調作業のためのファイル同期システム

塚田 大[†] 鈴木勝博[†]
阿部洋丈[††] 加藤和彦[†,††]

インターネットを介しての協調作業を支援するためのファイル同期システムを開発した．本システムは協調作業をするメンバーの持つファイルの同期を行う．ファイルを管理するサーバはなく，メンバー間でファイルを転送して同期を行う．転送するファイルは自動的に選ばれるので，ユーザは簡単に同期が行える．本稿ではこのシステムの実現方法について述べ，有効に働くかを検証した．

A File Synchronization System for Cooperating Work via the Internet

HIROSHI TSUKADA,[†] KATSUHIRO SUZUKI,[†] HIROTAKE ABE[††]
and KAZUHIKO KATO[†,††]

We developed a file synchronization system which supports cooperating work via the internet. This system synchronizes files which the members of cooperating work have. There is no server that controls the files, and the members transfer files to each other in order to synchronize. Users can synchronize easily because transferred files are selected automatically. This paper presents the implementation and validation of this system.

1. はじめに

近年のインターネットの普及により，今までは直接会って行っていた協調作業が，オンライン上でできるようになった．例えば，アイディアの議論をするときに，今までは実際に会って会話したりミーティングをする必要があった．現在では，電子メールやインスタントメッセンジャーなどを利用して，インターネットを介してすることができる．SOHOのような小さなオフィスや個人間での場合，離れた相手と協調作業をする必要が出てくるときがある．そのような場合に実際に会うのは移動や時間のコストがかかってしまうが，インターネットを使うことでその分のコストを抑えることができる．

協調作業では，グループのメンバー間でファイルを共有し，そのファイルの閲覧や更新をする作業が必要となることがある．例えば，複数人で論文の執筆を行う場合は文書ファイルを，ソフトウェアの開発を行う場合はソースコードを共有する．このような場合，ファイルの管理方法が現状では2種類ある．1つは共有サーバを用意して全てのファイルをサーバに置いておき，グループのメンバーがサーバにアクセスして作業を行う方法である．ここではこの方法を集中型と呼ぶ(図1)．集中型の利点は，ファイルを一箇所で集中管理しているので，サーバにアクセスできれば確実に最新の作業ファイルが手に入るという点である．しかし，サーバを用意しなければならず，またメンバー分のアカウントを生成しそれらを管理しなければならないなど，コストがかかる．また，協調作業をするグループというのは永続的なものではなく，短期間しか存在しないことが一般的である．例えば論文を共同で執筆する場合，論文を執筆している数ヶ月間しかグループは存在しない．このことを考えると，サーバを用意し管理するのは初期コストがかかりすぎると考えられる．またサーバで集中管理をしているため，サーバの故障などでサーバにアクセスできない場合，ファイルにアクセスできず作業が行えないという欠点もある．

2つめの方法は各メンバーが全く同じファイル構成の共有ファイルをそれぞれ持っていて，それぞれは自分の持っているファイルを編集する．そして他のメンバーが持っているファイルが必要になったら，直接通信してファイルを取得する方法である(図2)．ここではこの方法を分散型と呼ぶ．例えば，メールに添付し

† 筑波大学システム情報工学研究科
University of Tsukuba Graduate School of Systems and Information Engineering
†† 科学技術振興機構 CREST
CREAT, Japan Science and Technology Agency

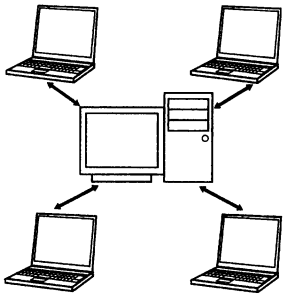

図1　集中型

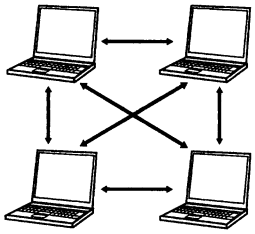

図2　分散型

てファイルをやりとりする方法があげられる．分散型ではサーバを設置，設定する必要が無いので，コストがかからないメリットがある．サーバを用意する手間がなくなるため，一時的なグループ作成にはこちらの方法のほうが有利になると考えられる．また作業に必要なファイルは各メンバーの手元にあるため，たとえネットワークにつながっていなくても作業ができるという利点もある．しかし，各メンバーは自分の持つファイルだけを編集，更新しているため，他のメンバーに更新したファイルを送る必要があるが，受け取るほうは誰の持つファイルが一番最後に更新されたものであるか判別する必要がある．先ほどのメールの例でいえば，複数のメンバーからメールでファイルが送られて来るが，どれが最新のファイルかわからない．さらに，グループのメンバーが直接通信を行えるためには，メンバーの位置 (IP アドレスやポート番号) を知らなければならないが，モバイル機器を利用してさまざまな場所に移動して作業を行う場合，位置情報が変わってしまう．よって，メンバーの位置を追える仕組みが必要となる．

このように，既存の 2 種類の方法では一長一短があり，どちらがよいとはいえない．そこで我々は，集中型と分散型の両方を取り入れた，インターネットを介した協調作業を支援するためのファイル同期システム，Improvisational Repository(IR) システムを提案する．本システムは，グループのメンバーのマシンの

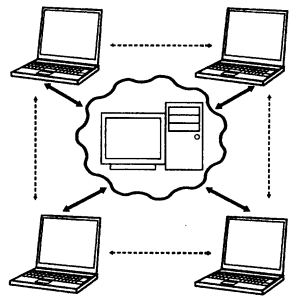

図3　IR のイメージ

あるディレクトリ以下のファイル構造を同期することで，ファイルの共有も行うようになっている．ユーザの動作イメージは rsync[12] のようになっている．rsync は 2 ノード間の同期や，1 つのノードのファイル構成を多数のノードに同期させる同期システムである．rsync はサーバのファイルの内容を，接続してきたクライアントに同期させる集中型の形態をしている．IR システムは rsync と同様なファイル同期を，分散型で行えるようにした．ユーザからはトラッカーと呼ばれるマシンにアクセスしてファイルを取得するように見える (図 3 の実線部分)．しかし実際にはファイルはトラッカーではなくグループの各メンバーが持っており，ファイルを他のメンバーから取得すること (図 3 の点線部分) で，同期を行う．トラッカーはグループのメンバーの位置情報を把握しているだけである．トラッカーの導入により，メンバーがどこに移動しても通信を行えるようにした．このトラッカーはメンバーの位置情報を解決する機能しか持たないため，一般のマシンで十分であり，わざわざサーバ用のマシンを用意する必要がない．そしてグループごとにトラッカーマシンを用意する必要はなく，1 台のマシンが複数のグループのトラッカーになれる．またファイルは各メンバーのコンピュータ内にあるので，ネットワークから切断されていてもファイルにアクセスでき，作業が行える．

本論文では，まず関連研究を示し，次に本システムの概要と実現方法について述べる．そして実装して実験を行った．

2. 関連研究

協調作業を支援するソフトウェアとして，グループウェアがある．グループウェアの持つ機能として，グループのメンバが議論するための電子会議室機能，メンバの予定を共有するためのスケジューラ機能，アイデアを共有するための文書共有機能などがある．グ

ループウェアは主に3形態に分けられる．1つはサーバ・クライアント型である．共有ファイルなどの共有したい情報をすべてサーバに置き，クライアントからそれらを操作する．この方法ではサーバを用意しなければならないので，コストがかかるという問題点がある．2つ目はASP(Application Service Provider)型である．これもサーバ・クライアント型の1種であるが，サーバが同一ネットワーク内ではなく，グループウェア提供会社側にあるところが違う．この方法はサーバを用意する手間がなくなるというメリットがあるが，グループウェア会社に支払う金銭的なコストがかかる．3つ目はP2P(Peer-to-Peer)型である．サーバを用意することなく，各ノードが通信しあって情報を取得，更新する．P2P型のグループウェアでは，Groove Virtual Office[9]，Ariel AirOne[10]が知られている．本研究のシステムはこれらに近い．これらの製品は，実物のノードあるいは仮想のノードがファイルの差分を蓄積している．そしてオフラインからオンラインに復帰したノードはそのノードに接続して，差分を更新している．本研究のシステムは，サーバはノードの場所を返すだけであり，ファイルの更新はファイルを持つノードと接続して行う．

ONFS[1]は一時的なネットワークを構築して，ファイルを共有するファイルシステムである．このファイルシステムはNFSをベースにしているため，サーバ・クライアント方式のネットワーク形態である．主に会議の場での一時ファイル共有を目的としているため，端末はモバイル機器を想定している．一時的にファイルを共有するという点で，本研究と似ているが，サーバを用意するためにサーバ設置のコストがかかる．また同じ場に集まった状態での使用を想定しているため，同一ネットワーク内での使用が前提となっている．そのため，インターネット越しでの作業には向いていない．本研究のシステムはインターネットを介した作業もできるように設計されている．

P2P形式のファイル共有システムは，システムのネットワークに参加しているノード同士が直接通信してデータをやり取りし，ファイルの共有を行う．すべてのノードが対等なため負荷が1つのノードに集中せず分散する特徴がある．このシステムではそれほど強力なマシンを用意することなくファイルの配布が行えるため，大きなファイルの配布に使われている．従来のサーバ・クライアント形式ではサーバに負荷が集中するため，大きなファイルを配布する場合はサーバに強力なマシンを使用しなくてはならなかった．実際の使用例では，BitTorrent[3]がフリーのOSのCDイメージの配布に使われている．またDHT(Distributed Hash Table)[8]を利用したP2Pファイル共有システムもある．DHTを利用すると，利用しないときに比べて効率よくファイルを検索することができ，Scalabilityを確保することができる．DHTを利用したものでは，Tapestry[5]を利用したOceanStore[6]，Pastry[4]を利用したPAST[7]などがあげられる．これらのシステムは，ネットワーク内からファイルを容易に検索，アクセスできるようにすることで，共有を行う．本システムはグループ内の共有ファイルを，全てのノードが実際に持つことで共有を行う．

Coda[2]とInterMezzo[11]はともに分散ファイルシステムである．ともに高いAvailabilityを目指し，ネットワークから切り離されている状態でもファイルを使えるようになっている．ファイルはサーバにあるが，サーバからファイルを取得する際にクライアントにキャッシュを作成する．このキャッシュのおかげで，ネットワークから切断されていてもファイルにアクセスできるようになっている．また，このキャッシュへの操作はログが取られている．Codaではサーバとの切断中にだけログを取り，再接続したときにログを元にサーバ側のファイルをクライアント側と同期する．InterMezzoではサーバがこのログを管理している．ファイルキャッシュを更新したクライアントはログをサーバへ送る．クライアントは定期的にサーバへアクセスしてログファイルを取得し，キャッシュに変更を反映してキャッシュの一貫性を保つ．またサーバといったん切断したあと再接続をした場合は，クライアントが持つログをサーバに送り，サーバの持つログをクライアントに送ってキャッシュを最新にする．この2つのファイルシステムは分散環境下やネットワーク切断時にもファイル操作を行える点で，本研究と同じである．しかし，これらは基本的に集中型でサーバがファイルを管理している．これに対し，本研究では集中型と分散型の両方を取り入れ，ファイルを各ノードが管理するので，サーバ的なノードが故障しても他のノードが起動していればファイルの同期を行える．

3. 提案手法

3.1 概　　要

ここからは，本研究で提案するIRシステムについて説明する．IRシステムは，少人数である決まったユーザ間でのファイルの共有，同期を目的としている．ユーザから見ると，動作は集中型のように見える．しかしサーバはファイルを持たず，ファイルを実際に持つのは各ノードである．更新したいノードは実際には

各ノードからファイルを取得して同期する．ユーザはノード間通信を意識することなく，コマンド 1 つでファイルの同期を行える．共有ファイルは相手が欲しいときに自動的に送られるため，いちいちメールなどでファイルを送る作業が必要なくなる．また，共有ファイルは各ユーザが自由に更新でき，更新されたファイルも自動的に他のユーザに拡散するので，更新ファイルを送る必要がなくなる．

本システムは 3 つのコンポーネントから構成される．1 つは各ユーザのノードにアクセスして最新版のファイルを取ってくるプログラム (irsync) である．1 つは各ユーザのノードに常駐して irsync の接続を待つデーモンプログラム (ird) である．そして最後にグループに参加している全ユーザの位置情報 (IP アドレス，待ち受けポート) を管理するトラッカープログラムである．トラッカーは IR ネットワークにおけるサーバとなる．

irsync は他ノードとの通信時にトラッカーを利用して，他ノードの位置情報を取得する．そして，トラッカーにアクセスできなかったときのために，前回通信時の他ノードの位置情報をキャッシュしている．キャッシュの情報は古い可能性があるため，なるべくトラッカーは常時起動している必要がある．また，参加する各ノードは GUID(Global Unique ID) を持っている．この ID でトラッカーはノードを識別，管理している．

また，通信を行いたいノードが両方とも NAT の中にいる場合，通常はルータの設定を変更してポートを開放しなければならない．しかし，会社など組織内のノードの場合は，設定の変更ができないこともある．そこでトラッカーがそれらのノードの仲介役となって，トラッカーを介してノード同士が通信を行う．

3.2 ネットワーク構成

IR のネットワーク構成は，図 4 のようになっている．ファイルを共有するノードとトラッカーでネットワークが構成されている．各ノードには 1 から順番に番号 (グループ内 ID) が割当てられている．それぞれのノードには ird が起動している．またトラッカーは各ネットワークに必ず 1 つ存在している．

各ノードは，共有するファイルのリストを持っている．そのファイルリストには，自分のグループ内 ID，共有ファイルのファイル ID，共有ファイルの名前，共有ファイルの更新日時，共有ファイルのバージョン情報が入っている (図 5)．

3.3 動作の概略

本システムの動作の概略を以下に示す．

まずトラッカーをどこかのノードで起動する．このノードは前述のとおり，常時起動できるノードが望ましい．そしてトラッカーにグループを登録する．これで，グループ内のノードが通信できるようになる．

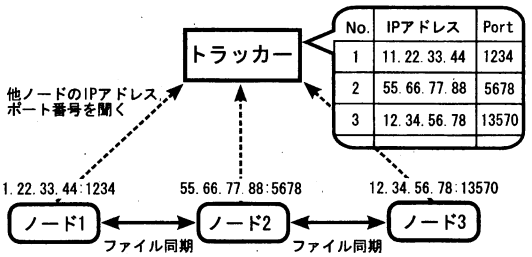

図 4　構成

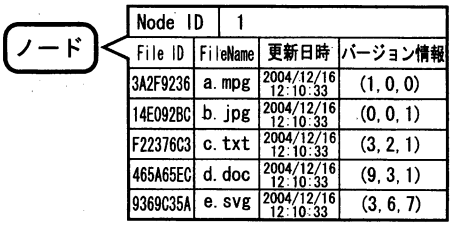

図 5　リストファイルの例

各ノードは ird を起動しておく．ird は起動すると，共有したいファイルのリストを作成する．そして他ノードの irsync からの接続を待つ．

作業開始時など，自ノードのところにある作業ファイルを最新のものにしたい場合に，irsync を起動する．irsync は起動するたびに，共有ファイルの更新をチェックする．前回チェック時から更新されていたら，リストファイル内のバージョン情報を更新する．次に，トラッカーもしくはキャッシュから他ノードの位置を取得する．そして，グループ内の全他ノードにリストを要求する．要求先のノードが ird を起動していれば ird が要求を受け取り，リストファイルを返す．起動していなかった場合はリストファイルを受け取れないので何もしない．そして，手に入れられるすべてのファイルリストを手に入れた後リストファイルを比較し，自ノードより新しいファイルを持つノードがあればそのノードにファイルを要求する．こうして自ノードの共有ファイルを最新に保つ．

以下では，各コンポーネントに関して詳しく述べる．

3.4 トラッカー

トラッカーは BitTorrent に採用されている仕組みで，グループに参加しているノードの位置情報 (IP アドレスと待ち受けポート番号) を記録する．また直接通信ができないノード同士の通信の仲介役も果たす．

トラッカーを起動した後は，共有グループをトラッカーに登録する．グループはグループ名，最大人数，管理者の名前，管理者の ID，メンバーの名前，メンバーの ID を登録する．1 つのトラッカーには複数のグループを登録できる．各ノードの位置情報は，各ノードがそれぞれ ird を起動したときに自分の位置情報をトラッカーに伝えることで，登録する．

トラッカーには通信を行いたいノードが他ノードの位置を問い合わせてくる．トラッカーは問い合わせたノードが所属しているグループとノードの ID を取得する．そしてトラッカー内に登録してあるグループにそのノードの情報があった場合，グループ内の全他ノードの位置情報を返す．

irsync が通信時に他ノードの位置情報の参照にトラッカーを使うため，トラッカーは常時起動しているのが望ましい．しかし実際は irsync は他ノードの位置をキャッシュしているため，トラッカーが起動していないからといって完全に通信不能に陥るわけではない．

各ノードが ird を起動して自分の位置情報を伝えてきたときに，トラッカーは実際にその情報で通信ができるかテストをする．ルータの設定などでポートが閉じていて通信ができなかった場合には，そのノードにトラッカーとの通信を維持しておくように伝える．他のノードがそのノードの位置情報をたずねてきたときは，正しいノードの位置情報の代わりに，自分の位置情報を返す．そして直接通信できないノードへの通信がきた場合，トラッカーは先ほど維持していた通信を利用してそのノードへ通信内容をそのまま転送する．そのノードから他のノードへの通信もトラッカーが転送をする．このようにトラッカーがプロキシの役割を果たすことで，直接通信ができなくても通信ができるようにする．

3.5 ird

ird はトラッカーノード以外のノードで，他ノードからの通信を待ち受けているデーモンプログラムである．

起動するとまずトラッカーに自分の所属するグループ名，自ノードの ID，そして IP アドレスと待ち受けポート番号を伝える．トラッカーはそれを受け取り，グループ内のノードであると判断したら，トラッカーのもつノードの位置情報を，受け取ったものに更新する．

次に，自分の持っている共有ファイルのリストを作成する．すでに作成してあった場合は，新たに共有ファイルが追加されていないか，また更新されていないか調べ，あった場合には追加，更新する．ファイルの更新はリストにあるファイルの最終更新日時と，実際のファイルの最終更新日時を比べることで判断する．

そして待ち受けを開始し，irsync からのリクエストに応じてリストファイルや共有ファイルを送信する．

3.6 irsync

irsync は実際にファイルの同期を行うプログラムである．rsync のように同期元と同期先を指定する必要がある．rsync はサーバからファイルを受信して同期を行うが，irsync の場合はグループ内の他ノードに接続しファイルを受信して同期を行う．

irsync は起動するとまず自分の持っている共有ファイルのリストの作成，もしくは更新を行う．この作業は ird のときと同じである．そして，トラッカーに自分の所属するグループ名，自ノードの ID を送り，グループ内の他ノードの位置情報を取得する．

トラッカーが起動していないなどでアクセスできなかったときのために，irsync は前回通信したときの他ノードの位置情報をキャッシュとして残している．キャッシュの内容は古いかもしれないので，通常はトラッカーから情報を取得し，トラッカーと通信できないときにキャッシュから他ノードの位置情報を取得する．

取得したらそれらのノードから，それぞれのノードの持つ共有ファイルリストを取得する．この共有ファイルリストの取得はマルチスレッド化されており，1 つ取得要求を出すたびに 1 つスレッドが作られる．そして通信はそのスレッド内で行われる．マルチスレッド化して 1 つのファイル取得の終了を待つことなく，次々と通信を行うことで，リストファイル取得の高速化を図っている．また，オフラインになっているノードからは取得できないので，そのときは何もしない．

入手可能な全リストファイルを取得し終わると，それらのリストのバージョン情報と自ノードの持つリストのバージョン情報を比較し，更新を調べる．この比較方法は後述する．そして調べたファイルが更新されていた場合，そのファイルを持つノードからファイルを取得する．このファイルの取得もマルチスレッド化されている．よってファイル要求リクエストを送った後，ファイルの取得が完了するのを待つことなく次のファイルのバージョン情報の比較ができる．これは共有ファイル数が多くなった場合に高速化が期待できる．また，ファイルの取得と同時に，自ノードの持つ共有ファイルリストの中のバージョン情報を，取得先のノードのものに更新する．最新版を持つノードと同じバージョン情報を持つことで，自分も最新版を持っていることを他ノードに伝えられるのである．また，複数のノードが最新版のファイルを持っていた場合，このファイル更新作業中にもっとも要求を出していないノードにファイルを要求する．こうして 1 つのノー

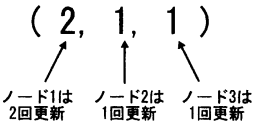

図 6　バージョンベクタの例

ドに集中して要求を出さないことで，負荷分散を図っている．

3.7 バージョンベクタ

共有ファイルのバージョン情報は，バージョンベクタという形で書かれている．バージョンベクタは分散ファイルシステムの Coda に採用されている技術である．ファイルを複数人で更新する際の最新版の判別や，2 人以上が同時に更新してしまう更新の衝突の検出ができる．本システムのバージョンベクタは Coda のものとは少し異なっている．

バージョンベクタは，あるファイルに対する各ノードの更新回数のリストである (図 6)．そしてノードごとに独自に管理されている．irsync 起動時などファイルの更新を調べた際に更新が確認された場合，自ノードに対応するバージョン番号を 1 つ上げる．こうしてファイルが更新されたことを表す．

ファイルが更新されたときにバージョン番号を上げるバージョンベクタは，自ノードが管理しているものだけである．他ノードが持っている同じファイルに対するバージョンベクタは更新されない．つまりバージョンベクタ間にずれが生じる．これを利用して最後に更新したノードを判別することができる．

最新のファイルを持つノードを探す方法であるが，同じファイルに対する各ノードの管理するバージョンベクタの要素を比較し，他のノードよりも大きな数値があるものが最新のファイルとなる．これは，最後に更新したノードのバージョンベクタは，他のノードのバージョンベクタに比べて数値が上がっているからである．図 7 を例にして説明する．それぞれのノードが同じファイルに対して持つバージョンベクタが図のようであった場合，ノード 2 が他のノードよりも大きな数値を持っている．この場合，ノード 2 の持つファイルがもっとも新しいファイルとなる．

また，更新の衝突もバージョンベクタを比較することで検出できる．前述のようにバージョンベクタを比較し，他のノードよりも大きな数値と小さな数値がある場合，更新の衝突が起こっていると判断できる．これは各ノードが独自に更新してしまったために，それぞれのノードの管理するバージョンベクタの違う要素の数値が上げられてしまうからである．図 8 を例にして説明する．1 番目の要素はノード 1 が 4 でもっとも大きい．しかし 2 番目の要素はノード 2 が 3 でもっとも大きくなっている．この場合，ノード 1 とノード 2 の間で更新の衝突が起こっている．

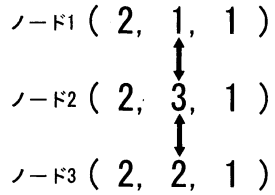

図 7　バージョンベクタの比較

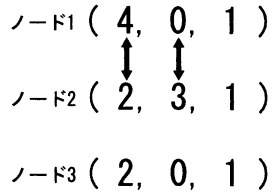

図 8　更新の衝突

4. 実装と実験

4.1 実　　装

本システムの実装を，C++で行った．コンポーネントをスレッド部分，通信部分，トラッカー部分，irsync 部分，ird 部分に分けて開発した．プログラムは 9700 行あまりとなった．

共有ファイルの ID は，最初にファイルをチェックしたときのファイルの内容の MD5 ハッシュ値とした．複数人が同じ内容のファイルを同時に共有ディレクトリに追加しても，同じファイルとみなせるようにした．

現在の実装状況では，トラッカーの機能の 1 つである直接通信できないノードの通信の仲介は，まだ行っていない．また irsync の機能の 1 つである，前回通信時の位置情報のキャッシュも，まだ実装していない．

4.2 実　　験

本システムの評価を行った．評価は実際に使用が想定される環境に近くして行うほうがよいため，インターネットを介した分散環境で行った．使用マシンは以下の通りである．

CPU	Intel PentiumIII 600MHz
Memory	512MB
OS	Debian GNU/Linux 3.1 (Kernel 2.4.27-2)

同じ構成のマシンを 3 台用意した．各ノードはイン

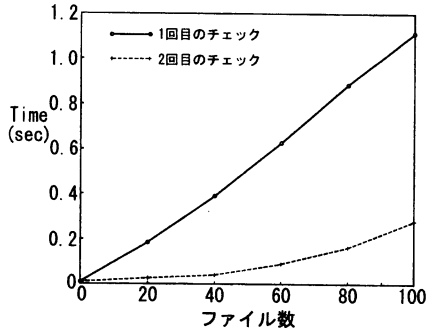

図 9　ファイル更新チェック時の実験結果

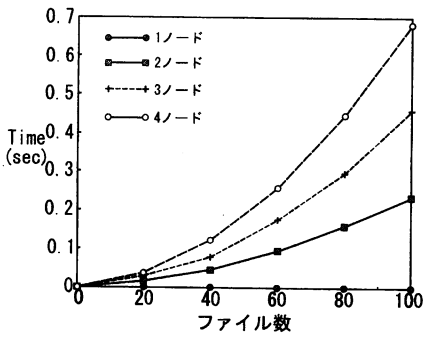

図 10　バージョンベクタ比較の実験結果

ターネットを通してつながっている．

4.2.1　ファイルの更新チェック

ird や irsync 起動時に行われるファイルのチェックにかかる時間を測定した．まず 200KB のファイル 0 個から 100 個を新規に追加した場合にかかる時間を測定した (1 回目チェック)．次にそれらのファイルをすべて更新した場合にかかる時間を測定した (2 回目チェック)．

結果は，図 9 のようになった．新規に追加したときはファイルの ID を決めるために，ファイルの MD5 ハッシュ値を調べるため，その分の時間がかかる．この時間はファイル数に大体比例している．また一度追加してしまうと，あとはファイルの更新時間だけを調べるので，それほど時間はかからない．この時間は更新時間を調べるシステムコールにかかる時間がほとんどを占めていた．

4.2.2　バージョンベクタの比較

取得するファイルを決めるために行われる，バージョンベクタの比較にかかる時間を測定した．ファイル数を 0 から 100 個まで，グループ内のノード数を 1 から 4 まで変えて irsync を実行した．なお，この実験では 1 つのマシンでトラッカー，ird，irsync を動かして測定した．irsync が比較を行っているときはトラッカーと ird は接続待ちの sleep 状態となっているため，測定に影響はないと考えられるからである．

結果は，図 10 のようになった．バージョンベクタの比較回数はノード数-1 とファイル数に比例する．同じファイル数の場合はノード数-1 に比例して時間がかかっている．また同じノード数の場合は，ファイル数の増加につれて線形よりも多く時間がかかっている．比較のアルゴリズムのさらなる効率化が必要である．

4.2.3　同期の速度

実際に運用が想定される状況や環境下でファイル同期にかかる時間を測定した．同様のファイル同期システムである，rsync と比較した．

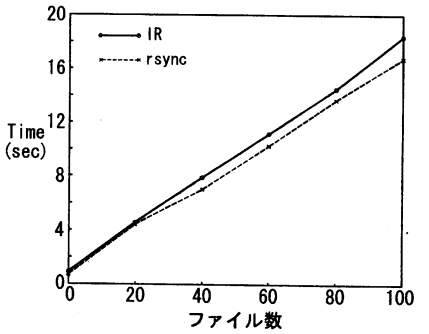

図 11　同期の実験結果

実験は，以下のような状況を想定した．まず 2 人のグループを作成し，200KB のファイル 100 個を同期させる．1 人が 100 個のうち 0 個から 100 個のファイルを更新する．そしてもう 1 人がファイルの同期を行う際にかかる時間を測定した．この実験では，ローカルファイルの更新チェック，バージョンベクタ取得と比較，ファイルの取得のトータル時間が計測される．

結果は図 11 のようになった．ファイル数にほぼ比例した時間がかかっていた．また，本システムは rsync とほぼ同等の性能を出せていた．

4.2.4　負荷分散

同じファイルを複数のノードが持っている場合，一番要求を出していないノードに要求を出すのは，前述したとおりである．その負荷分散の効果を調べた．200KB のファイル 0 個から 100 個を 1 つのノードから同期する場合と，2 つのノードから同期する場合にかかる時間を測定した．なお，同期元の 2 ノードはすでにファイルが同期している状態となっている．

結果は図 12 のようになった．1 つのノードからのファイル同期に対し，2 つのノードからのファイル同期は半分くらいの時間で完了している．よって，負荷

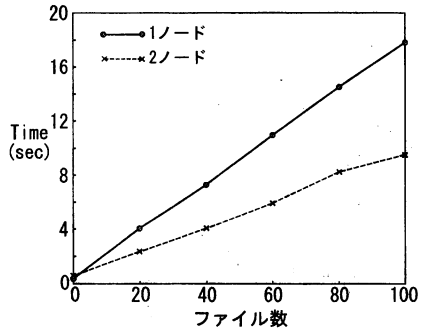

図 12　負荷分散の実験結果

分散の効果は十分出ていると考えられる.

4.2.5 結　　論

本システムは同期システムである rsync と同等以上の機能 (複数対 1 の同期が行える) を備えていながら, 速度はほとんど rsync と変わらないことを示した. また複数ノードから取得する場合には, 負荷分散の機能が有効に働いていることを示した.

5. 議　　論

各メンバーがそれぞれ独自にファイルを更新してしまった場合には, 更新の衝突が起こってしまう. 本システムではそれを検出はできるが, 防ぐことはできない. しかし分担して作業を行う場合には, 自分の担当するファイルの範囲が決まっていて, それ以外のファイルを見ることはあっても修正することはあまり無いと考えられる. このような理由により, 更新の衝突はそれほど起きないと考えられるので, 防ぐことができなくてもあまり問題にならないと考えられる.

6. ま と め

本研究では, インターネットを介した協調作業を支援するためのファイル同期システムを提案した. 本システムを利用することで, 分散した環境下でも複数人でのファイル同期を簡単に実現できる. 本システムでは, 協調作業をするグループのユーザの指定したディレクトリ以下のファイル構造を同期する. また, 同期を利用してファイル共有も行える.

ファイルはグループに参加するメンバーが各自管理し, 同期時に他のメンバーからファイルを取得して同期を行う. メンバーのノードの位置情報を管理するトラッカーを導入した. これにより, ノードが移動したり IP アドレスが変わっても通信を行えるようにした.

今後の課題として, トラッカーを分散させることがあげられる. 現状ではトラッカーが起動していなくても, 短期間ならキャッシュによりノードの通信は行える. しかし長期に渡って起動していないと, ノードの IP アドレスが変わるなどしてキャッシュの内容と実際が異なり, 通信できないノードが出てくる. そして最終的に全てのキャッシュが古くなると, まったく同期ができなくなってしまう. そこでトラッカーの機能を分散してグループ内のノードに持たせることで, 他ノードのどれかが起動していれば通信が行えるようにする. また, 共有ファイルにアクセス制御を行うことがあげられる. 現状ではグループのメンバーはどのファイルも自由に更新できる. 分担作業をしていると他のメンバーに読まれてもよいが更新されては困るファイルも出てくる. そのため, メンバーごとにファイルへのアクセスを制御できると望ましい.

参 考 文 献

1) 福田信彦, 楯岡孝道, 中村嘉志, 多田好克:ONFS: 一時的なネットワーク環境下ですぐに利用できる共有ファイルシステム, 情報処理学会論文誌, Vol.44, No.2, pp353-363(2003)
2) J. J. Kistler, M. Satyanarayanan, "Disconnected Operation in the Coda File System", ACM Transactions on Computer Systems, 1992.
3) BitTorrent http://www.bittorrent.com/
4) A. Rowstron, P. Drunschel, "Pastry: Scalable Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems", Lecture Notes in Computer Science, 2001
5) Ben Zhao, Ling Huang, Jeremy Stribling, Sean Rhea, Anthony Joseph, John Kubiatowicz, "Tapestry: A Global-scale Overlay for Service Deployment", IEEE Journal on Selected Arean in Communications, 2003
6) Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, John Kubiatowicz, "Pond: the OceanStore Prototype", in Proc of USENIX FAST, 2003
7) P. Druschel, A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", in Proc HotOS VIII, 2001
8) Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, "Looking up Data in P2P Systems", CACM, 2003
9) Groove Networks http://www.groove.net/
10) アリエル・ネットワーク　http://www.ariel-networks.com/
11) Peter J. Braam, "InterMezzo: File Synchronization with InterSync". http://www.intermezzo.org/docs/intersync.pdf
12) rsync http://rsync.samba.org/