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．インターネットの普及につれて，短期間かつ小規模な協調作業をすることが多くなっている．現在でも協調作業を支援するファイル共有システムはあるが，短期間かつ小規模なものに特化したものはなかった．我々はそのような協調作業で使えるシステムを提案する．本システムでは 1 つの仮想ストレージを用意して共有することで，ファイル共有を実現している．本論文ではこのシステムの実現方法について述べる．

A File Synchronization System for Cooperative Work via the Internet

Hiroshi Tsukada,[†] Katsuhiro Suzuki,[†] Hirotake Abe[††]
and Kazuhiko Kato[†,††]

We developed a file synchronization system which supports cooperative work via the Internet. We do short-term and small-size cooperation working more as the Internet spreads. Now there are file sharing systems which support cooperative work but they are not specialized for short-term and smallsize. We suggest the system for such cooperating work. Our sytem provides virtual storage that is shared to share files. This paper presents the implementation of the system.

1. はじめに

近年インターネットの利用が一般企業や家庭にまで広まってきている．それにより，遠隔地での協調作業が今までよりもコストをかけずに，オンライン上で行えるようになった．また比較的短期間，小規模な協調作業も行われるようになった．たとえば，地理的に離れた相手と協調作業をする場合，今までは電話を利用したり実際に会合を行ったりする協調作業を行うことが一般的であった．しかしインターネット普及後は電子メールなどによって移動や通話のコストを抑えることが可能になった．また短期間，小規模な作業の例として，会社内での部署を越えて期間を区切っての作業や，数人程度の個人間の協調作業などがあげられる．

協調作業での主な作業の 1 つに，作業グループのメンバ間でデータを共有し，それを閲覧，更新するという作業がある．たとえば複数人で論文を共著する作業の場合，文書ファイルを共有する．またソフトウェア開発の場合，ソースコードを共有する必要がある．ファイルの共有を行う場合，ファイルストレージの場所と更新のタイミングの 2 つの観点から分類ができる．まずファイルストレージの場所の観点から分類すると，現状では 2 つに分類できる．1 つはサーバを用意してファイルを置き，メンバはサーバにアクセスしてファイルを操作する方法である．ここではこれを集中型と呼ぶ．もう 1 つは，メンバのマシンにそれぞれファイルを置き，必要に応じてファイルをやりとりする方法である．ここではこれを分散型と呼ぶ．

協調作業における，集中型と分散型の特徴を考える．運用に関するコストを考えると，集中型でも分散型でもコストがかかる．集中型の場合，サーバを用意しストレージをそこに置く必要があるため，マシンの設置・管理，ストレージの管理が必要となる．このコストがサーバ管理者に集中する問題がある．たとえば組織をまたがった協調作業の場合，一方の組織がサーバを用意し，設置，管理するコストがかかるのに対し，他方の組織はそのコストがかからない．一方分散型の場合，ストレージは各メンバの持つものを利用するため，管理コストが各メンバに平等に分割される．このため，

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

コストが平等になる分散型のほうが協調作業では導入しやすいと我々は考えた．

次に更新のタイミングの観点から分類すると，現状では 2 種類に分類できる．1 つは check-in/check-out 型である．即時反映型とはファイルを変更し保存すると，その変更が即座に他のメンバが取得できる共有ファイルに反映されるものである．つまり，共有しているファイルと作業で使うファイルが同じであるといえる．それに対し check-in/check-out 方式とは，共有ファイルと普段使うファイル（作業ファイル）は別のものになっており，コマンド入力など明示的な操作により共有ファイルから作業ファイルにコピー，もしくは作業ファイルから共有ファイルにコピーを行うものである．普段作業に使うファイルは共有ファイルとは別になっており，したがって更新してもすぐには共有ファイルには反映されない．作業ファイルを共有ファイルにコピーしたとき初めて更新が反映される．協調作業では，作業中のファイルは他のメンバに使ってほしくないことがよくある．たとえばソフトウェア開発で何かの機能を付けたい場合，その機能がうまく動く状態のコードを他のメンバには使ってもらいたい．作業中のファイルはきちんと動くか分からず，それにより他のメンバの作業を遅らせてしまうことになるかもしれない．コードの作成者，もしくは変更者が完成と思ったものだけを共有できれば，このような問題はなくなる．この場合，即時反映型だとメンバが動作確認をしていないファイルも共有されてしまい，他のメンバがそのファイルを使用してしまう可能性がある．そのため，協調作業目的では即時反映型は適さないと考えられる．しかし，分散型で check-in/check-out の考え方を持つファイル共有システムは，現状で存在しない．

モバイル環境下ではつねに無線の電波が届いているとは限らず，ネットワークから切断されることはよくある．このようにネットワークから切断されているときでも，サービスやシステムを使うことを disconnected operation という．モバイル環境が発達している今日では，disconnected operation が重要である．PC を持ち運ぶことで，どこに行っても作業が行えるが，移動中や外出先では，インターネットに接続できないことがある．しかしインターネットに接続できないからといって，作業ができないのは不便である．ローカルの環境に共有ファイルのデータを置いておけば，切断時でもファイルを使った作業が行える．

そこで我々は，遠隔地での協調作業を支援するファイル共有システム，IR（Improvisational Repository）を提案する．本システムは短期間，小規模な作業を主なターゲットとしており，そのため導入コストがかかるサーバが必要ない，分散型を採用した．また，前述のように協調作業に向いている更新方法である check-in/check-out 方式を採用し，メンバがファイルを公開するタイミングや他のメンバの更新を作業ファイルへ適用するタイミングをユーザ側で決定できるようにした．そして共有ファイルは全メンバが全データを持つことで障害に強く，またメンバのうち 1 人でも通信可能であればファイルの同期を行えるようになっている．

本論文は，以下のような構成になっている．2 章で本研究の関連研究を述べる．次に 3 章では提案する IR システムの概要を述べ，4 章では具体的な実装を述べる．5 章では IR システムの評価を述べる．そして 6 章でまとめと今後の研究課題を述べる．

2. 関 連 研 究

Ivy [2] は P2P（Peer-to-Peer）ネットワーク上に構築されたファイルシステムである．DHT [7] の 1 つである Chord [4] を利用し，ファイルや変更差分をネットワーク上のノードに配布する．サーバがなくファイルを各ノードに配置することから，1 章での分散型に分類できる．分散型でファイル共有を目指す点では本提案方式と同じである．しかし Ivy ではファイルの変更はファイルの保存とほぼ同時にネットワーク内に配布される．これは 1 章で述べたように，協調作業には不向きであると考えられる．それに対し IR ではコマンド入力などユーザの明示的な操作がないと，共有ファイルの更新を行わない．

バージョン管理システムには CVS [14] や RCS [15]，Subversion [16] などがある．バージョン管理システムでは，リポジトリを用意しそこにファイルやバージョン情報を置き，ユーザは check-out を行いリポジトリからファイルを取り出す．普段の作業は取り出したファイル（作業ファイル）で行う．作業の終わりには check-in を行い，作業ファイルの更新をリポジトリに反映する．リポジトリを複数人で共有することで，ファイルの共有が行える．このリポジトリを Ivy のファイルシステム上に置くと，分散型かつ check-in/check-out での更新という，IR と同じ目的が達成される．しかし Ivy ではネットワークの分断が起こることを想定していない．ネットワークが分断した場合，到達できないノードができてしまい，そこに配置しているファイルにアクセスできなくなる．よって，check-out ができなくなる可能性がある．また check-in の作業を分断したそれぞれのネットワークで行うと，名前は同

じだが内容の違うファイルがそれぞれのネットワークで作られてしまう．分断が回復したときのファイルのマージの問題もある．本研究の IR はグループ内の全ノードが全データを持つことで，ネットワークの分断のような障害に強くなっている．また分断が回復したあとも特別な処理は必要ない．

DHT を使った P2P 型のファイル共有システムとして，OceanStore [5] や PAST [6] などがある．これらはネットワークに参加している多数のノードの持つストレージを共有し，大規模なストレージ空間を作り出そうというものである．そのため，ファイルの共有というよりはストレージの共有であり，ファイルはその大規模なストレージに追加だけをする．編集や削除などは考えられていない．

FolderShare [12] は Microsoft 社のファイル共有サービスである．同期をさせたいフォルダ（ディレクトリ）を指定しておくと，複数の PC にあるフォルダ内容を自動的に同期する．たとえばファイルをフォルダに追加した場合などは，自動的に他のマシンにも追加が伝わり，他のマシンのフォルダにもそのファイルが追加されている．アカウントを管理するサーバはあるが，データの転送自体は P2P で直接通信をして行う．以上から FolderShare は分散型であり，またデータのコピーを全員が持つ特徴がある．この点では，本提案方式と同じである．しかし自動的に更新が伝わることから，更新方式は即時型である．これは 1 章での理由から，協調作業には適さないと考えられる．

協調作業の支援としてのファイルの共有が本研究の目的であるが，本研究では各自が更新したファイルを共有空間に入れるという方法をとっている．その他のファイル共有のアプローチとして，ファイルを編集するテキストエディタを共有し，編集しているファイルを共有する方法がある．このようなものには，Sobalipse [9]，NTE [10] などがある．これらのシステムでは自分のエディタ上に他人の編集している状況が表示され，これを利用して複数人で同時に 1 つのファイルを編集できる．そしてファイルの変更点はキー入力の単位で即座に同時編集者に伝えられる．これらはソフトウェア開発の協調作業を支援する点で本研究と目的が同じである．しかしこれらのシステムではファイルの変更は即座にグループのメンバに伝わるのに対し，本研究では明示的に操作をしないと更新が伝えられない点が異なる．

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目的は本研究と同じだが，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．

3. 提案方式

3.1 概　　要

ここからは，本論文で提案する IR システムについて述べる．IR システムは分散環境下での協調作業を支援するための，ファイル共有システムである．少人数で短期間な，サーバを用意するにはコストがかかりすぎるような協調作業をターゲットとしている．そのためネットワークの形態は分散型であり，サーバレスである．

ユーザからは 1 つの仮想のストレージが用意されており，そこに（直接は更新できない）共有ファイルが配置されているように見える．ユーザはこの仮想ストレージからファイルをダウンロードし，作業を行う．また仮想ストレージにファイルをアップロードして，共有ファイルの更新を行う．ちょうどファイル同期ツールの rsync [11] と同じように扱うことができる．rsync ではサーバからファイルをダウンロード，あるいはサーバにファイルをアップロードしてローカルのファイルとサーバのファイル内容を同期する．IR システムでは仮想ストレージからファイルをダウンロード，仮想ストレージにファイルをアップロードしてローカルのファイルと仮想ストレージのファイルの内容を同期させる（**図 1**）．

また，IR システムでは共有グループのメンバの増減を認めていない．これは短期間の協調作業をターゲットとしているため，メンバの出入りはないと考えられるからである．新しいメンバをグループに加えたい場合は，グループを作り直す必要がある．

3.2 構　　成

IR システムでは協調作業に参加するノードがそれぞれのローカルストレージを提供し，それらを共有して 1 つの仮想的なストレージができている．各ノードの提供するストレージには全部の共有ファイルのキャッシュが入っている．完全なキャッシュを持つので，ノードがネットワークから切断されていてもファ

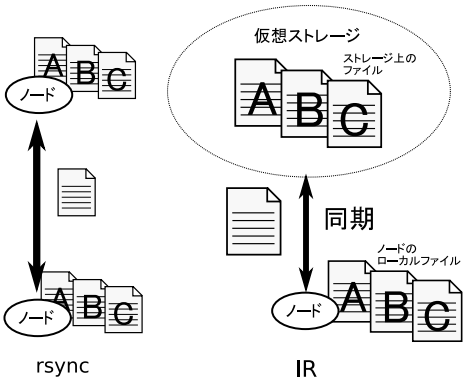


図 1 rsync と IR との比較
Fig. 1 rsync vs. IR comparison.

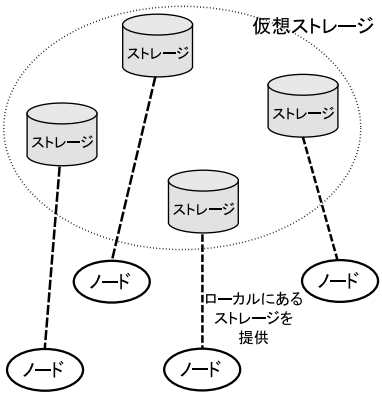


図 2 IR のイメージ
Fig. 2 IR image.

イルを使った作業が行える（**図 2**）．

新しく協調作業に参加するメンバは，共有ファイルのキャッシュデータを他のノードから取得する必要がある．そのとき，各ノードは全キャッシュデータを持っているので，他ノードのうちたとえ 1 つしか起動していなかったとしても，全部のキャッシュを取得できる．全ノードの更新状況を調べたわけではないので，最新版のファイルとは限らないが，全ノードが起動している必要がないというメリットがある．これらの理由から，IR では各ノードは全共有ファイルのキャッシュを持っている．

ファイルの共有方法は check-in/check-out 方式をとり，各ノードは共有用のファイルと作業用のファイルの 2 種類を持つ．仮想ストレージとして提供する空間に共有用のファイルを置く．ユーザがファイルを使った作業を行う場合には作業用のファイルを使用する．本システムでの check-in とは，作業ファイルを共有ファイルとして仮想ストレージにアップロードすることである．本論文では，以降 check-in 時の操作をコミットと呼ぶ．また本システムでの check-out とは，仮想ストレージにある共有ファイルのうち，最も最近に更新されたものを判別してダウンロードし，作業用のファイルとして保存することである．本論文では，以降 check-out 時の操作を同期と呼ぶ．コミットと同期はユーザの明示的な操作が必要となる（**図 3**）．

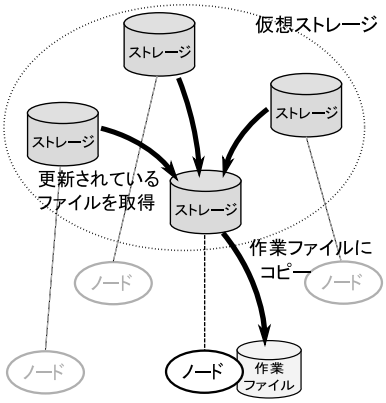


図 3 同期（check-out）
Fig. 3 Synchronization/check-out.

3.3 同期の動作

同期とは，他ノードのストレージから最新のファイルを取得し，作業ファイルとして保存することである．具体的には，仮想ストレージを形成する各ノードのストレージ中のファイルから最新のファイルを判別し，ローカルの作業ファイルと共有ファイルに保存する．ローカルにある共有ファイルも書き換えるのは，disconnected operation を実現するためである．

同期をする際の動きは，次のようになる．まずグループに参加するノードから，それぞれの持つ共有ファイルリストを取得する．次にそれらのリストの情報を比較し，1 つの共有ファイルに対して最も最近に更新しているノードを判別する．オンラインである全ノードからファイルリストを取得するので，オンラインのノードが持つファイルは確実に最新のものが判別できる．そしてそのノードからファイルを取得し，自分の持つ共有ファイルと作業ファイルを更新する．このとき更新の衝突が起こっていた場合は，警告をユーザに出し，同期処理は行わない．衝突の解決は自動的に行うのは困難なため，人間が手作業で行うこととする．IR 自体は衝突検出のみを行い，解決は行わない．これらの処理を全共有ファイル分行う．衝突の検出は同期時にだけ行われることになるが，これは衝突をユーザに確実に認識させるためである．また最も最近に更新されているノードの判別は，ファイルに付与されているバージョン情報をもとに行う．

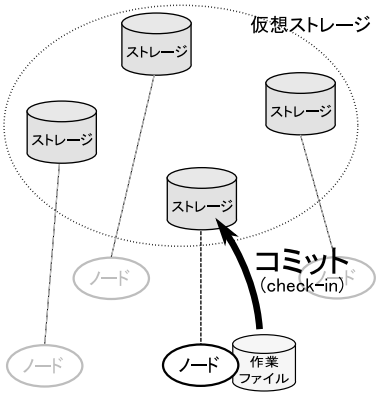


図 4 コミット（check-in）
Fig. 4 Commit/check-in.

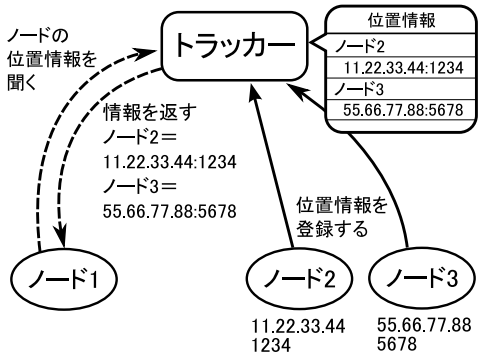


図 5 トラッカ
Fig. 5 Tracker.

IR は最新のファイルを持つノードを確実に判断するため，同期時にグループに参加するノードのうちオンラインのもの全部と通信する方法をとっている．この方法はスケーラビリティは低い．しかし IR は小規模な協調作業を想定しているため，スケーラビリティがほとんどなくてもよい．

3.4 コミットの動作

コミットとは，作業ファイルを仮想ストレージにアップロードすることである．具体的には，作業ファイルをローカルの共有ファイルにコピーする．ローカルにある共有ファイルのストレージは仮想ストレージの一部として他のノードから参照されるので，ここにコピーすることは仮想ストレージにアップロードすることと同じになる（**図 4**）．

コミット時の動きは，次のようになる．まず，自分の持っている作業ファイルと共有ファイルの内容や最終更新時刻など，違いが分かるもので比較する．その結果作業ファイルに更新されているものがあったら，共有ファイルを作業ファイルの内容に更新する．そして共有ファイルリストにバージョンがあがったことを書き込む．これで共有ファイルへの追加は終了である．

4. 実　　現

4.1 トラッカ

ノードが他のノードと通信するためには，相手のノードの位置情報（IP アドレス，待ち受けポート番号）が必要である．たとえばノート PC や ADSL など，動的に IP アドレスが割り振られる環境では，つなぐたびにアドレスが変わってしまう．変わってしまうと以前の情報では通信できないため，現在のノードの位置情報を管理する必要がある．

以上の管理をするために，トラッカを導入した．トラッカはオンライン上にあるノードの位置情報を管理するもので，ファイル共有ソフトの BitTorrent [3] でも採用されている．ノードは他ノードからの通信の待ち受けを開始する際に，トラッカに自分の待ち受けポートを登録する．またトラッカは送信元情報からノードの IP アドレスを取得し，待ち受けポートとともに保持しておく．そして，共有ファイルの同期時など他のノードへの通信を行うときには，トラッカから位置情報を取得する．

トラッカにはもう 1 つの機能として，プロキシの機能を持たせた．ノードが NAT の中にいると，ルータやファイアウォールの設定により外部からアクセスできないことがある．これではインターネット越しの作業では使用できない．そこでトラッカがそのようなノードへの通信を肩代りすることにした．まずトラッカがノード（A とする）から待ち受けポートの情報を受け取るときに，A に接続できるかテストをする．テストの結果接続できない場合には，A はトラッカへの接続を維持しておくようにする．そしてあるノード（B とする）が通信しようとして A の位置情報をトラッカへ参照する．A は外部から接続できないので，トラッカは B にトラッカの位置情報を返す．B はその情報が A の位置情報だと思い，通信を開始する．実際はその情報はトラッカの情報なので，トラッカに通信がくる．トラッカは接続を確立してある A に B からの通信内容を転送する．このように，B から A へトラッカを介して通信が行える（**図 5**）．

トラッカはノードどうしの通信に必要なため，常時起動している必要がある．このため，トラッカが集中型のサーバのような役割を持つ．しかし集中型のサーバはストレージ管理とユーザアカウント管理が必要であるのに対し，トラッカが管理するのはグループのノードの位置情報だけであり，集中型のサーバに比べ

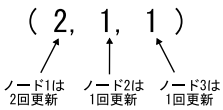


図 6 バージョンベクタの例
Fig. 6 Example of a version vector.

て管理コストを減らしている．短期間の協調作業を考えると，管理コストが少ないほうがよい．

4.2 バージョン情報

共有ファイルの同期を行う際には，ファイルを最も最近に更新したノードを判別する必要がある．判別する基準になるのが，ファイルごとに管理されるバージョン情報である．バージョン情報はファイルごとに付与され，各ノードごとに管理される．コミットを行う際に作業ファイルが更新されていた場合，自分が更新したとの情報をバージョン情報に追加する．同期を行うときには同じファイルに対する各ノードのバージョン情報を比較する．その結果一番新しいファイルを持つノードの判別ができる．また，1 つのファイルを複数人で同時に更新してしまう，更新の衝突も検出できる．

IR システムでは，このバージョン情報にバージョンベクタ[8] を使用した．バージョンベクタは，共有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．共有ファイルを更新する際に，自ノードに対応するバージョン番号を 1 つ上げる．これでファイルを更新したことを表す．また自ノードの管理するバージョンベクタしか変更しないため，各ノードの持つバージョンベクタ間に差が生じる．この差を利用して，更新の有無や衝突の検出をする．

バージョンベクタを利用して最も最近に更新したノードを判別するには，以下のようにする．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となる．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あがっているからである．**図 7** を例にすると，ノード 2 が他のノードより大きな数値を持っているため，ノード 2 の持つファイルが最も新しいファイルとなる．

また更新の衝突の判別は，以下のようにする．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と小さな数値がある場合，そのノード間で衝突を起こしていると判断できる．独自に更新してしまうと，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** を例にとると，1 番目の要素はノード 1 が大きいが，2 番目の要素はノード 2 が大きい．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．

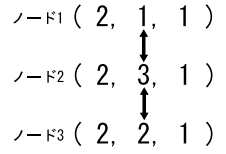


図 7 最新ファイルの検出
Fig. 7 Most recent file detection.

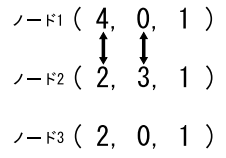


図 8 更新の衝突の検出
Fig. 8 Conflict detection.

4.3 セキュリティ

ファイル共有において，セキュリティは重要である．特に重要なファイルを共有する場合，協調作業のメンバ以外には見られたり書き換えられたりしては困る．IR では，グループのメンバだけファイルにアクセス可能，他のグループのメンバにはノードの位置情報が分からない，というセキュリティモデルを提供する．グループごとにトラッカ用の共有鍵（以下 K_t）とメンバ用の共有鍵（以下 K_m）の 2 つを用意し，あらかじめメールなどでメンバ間で共有しておく．また K_t はトラッカも持っている．トラッカとノードの通信は，まずトラッカが乱数をノードに送る．ノードは K_t で暗号化し，トラッカに返す．トラッカは K_t を用いて復号し，送っていたものと同じならそのノードはグループのメンバであると判断する．こうして，メンバ以外のノードに位置情報が伝えられるのを防ぐ．ノードどうしの通信でも同様に，K_m を用いて認証を行う．トラッカとメンバの鍵が分かれているのは，トラッカにはファイルの内容が伝えられないようにするためである．

5. 実験，評価

本システムに関して，実験を行った．まず本システムの性能を評価した．次に，Ivy 上で動くバージョン管理システムと定性的な比較を行った．使用マシンは**表 1** のとおりである．

5.1 性 能 評 価

IR システムの実行にかかる時間を測定した．実際の作業に近い測定をするために，Web ブラウザの Fire-

表 1 実験マシンの環境
Table 1 Machine environment.

CPU	Intel Pentium4 3 GHz
Memory	2 GB
OS	Gentoo Linux (Kernel 2.6.15)
Java VM	1.5.0_06

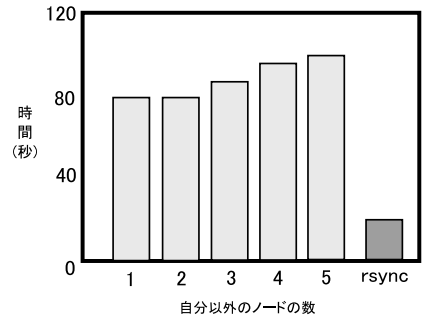


図 9 同期にかかる時間測定
Fig. 9 Synchronization time.

fox[17] の 1.0.6 と 1.0.7 のソースを使用し，1.0.6 から 1.0.7 に更新するときの速度を測定した．ソースのファイル数はどちらも 27045，更新ファイル数はそのうちの 35 であった．

5.1.1 コミット

ローカルにある共有ファイルのキャッシュを 1.0.6，作業ファイルを 1.0.7 のソースコードにし，コミットをしたときに要する時間を測定した．結果，10 秒ほどで終わり，実際の開発にも使用に耐えられると分かった．

5.1.2 同　　期

同期に要する時間を測定した．今回の測定では，自分以外の他ノードはすでに同期が終了している，つまり他ノードはすべて 1.0.7 のデータを持ち，自分は 1.0.6 のデータを持つ状態にしてから行った．また今回の測定は CPU の使用を見るために，すべてのノードを 1 つのマシン内で起動した．

結果は**図 9** のようになった．すでに同期しているノードの数が増えると，多くのノードからファイルを取得するようになり，負荷分散が働き一般的には結果が良くなると考えられる．しかし今回は共有ファイル数が多いので各ノードが持つファイルのチェックに時間がかかってしまい，また更新ファイル数がほとんどないので，ノードが増えるごとに時間がかかってしまったと考えられる．

また，ファイル同期ツールである rsync を使用したところ，22 秒で終わった．rsync は必ず新しいファイルに同期するわけではない（同期の方向が 1 方向なので，同期元のファイルが古くても同期先は古いファイルで上書きされる）など，機能をしぼっている分高速になっている．しかし，複数人がそれぞれ共有ファイルを更新するなど，協調作業においては使いにくい面もある．

表 2 IR と Ivy 上のバージョン管理システムの比較
Table 2 Comparison of IR vs. a version management system using Ivy.

	IR	Ivy
disconnected operation	○	○
分断時の availability	○	△
scalability	×	○
セキュリティ	○	×
NAT 越え	○	×

5.2 Ivy 上のバージョン管理システム

Ivy を共有ストレージとし，そこにバージョン管理システムを利用してファイルを共有すると，IR システムと同じく check-in/check-out 方式の共有が可能になる．ここでは，Ivy 上にバージョン管理システムを載せた場合と IR システムについて比較した（**表 2**）．

disconnected operation に関しては，両方ともネットワーク切断時にファイルを使用可能である．IR では作業ファイルがローカルに置いてあるため，ネットワークから切断されていても作業が続行可能である．作業ファイルを消してしまった場合でも，共有ファイルの全データがローカルに置いてあるため，共有ファイルを作業ファイルにコピーして作業が可能になる．一方 Ivy についても，切断前にローカルの作業ディレクトリに check-out してあれば，ファイルを使った作業が可能である．

ネットワークの分断が発生し，アクセスできないノードが発生したとする．その場合，IR では全データをローカルに置いてあるため，すべてのファイルを入手でき，使用できる．またコミットはローカルにある共有ファイルに作業ファイルをコピーするだけなので，可能である．同期は現在アクセスできるノード間だけで行えるので，これも可能である．分断が解消されたときに全ノード間で同期を行い，アクセスできなかったノードで行われた更新を適用する．一方 Ivy では，ローカルに作業ファイルがある場合は全ファイルが使用可能であるが，新しく check-out する場合にはアクセス不可能なノードに配置されたファイルは使用不可能である．コミットを行った場合，同じ名前で内容の違うのファイルがいくつも作られる可能性がある．また update を行う場合，前述のとおりアクセス不可能なノードにあるファイルは使用できない．

IR はあまりスケーラビリティを重要視していない．これはもともと IR が小規模な協調作業をターゲット

にしており，また全ノードがキャッシュの全データを持つという設計だからである．また，IR で使用しているバージョンベクタにもスケーラビリティがあまりない．したがってバージョン情報をバージョンベクタ以外のもの（たとえば，HashHistory [13] など）に代えると，スケーラビリティが確保できると考えられる．現在の IR の実装ではバージョンベクタしか使っていないが，将来的には他の方法も使用できるように考えている．一方の Ivy の使用している Chord などの DHT はスケーラビリティを考慮して設計されているため，Ivy もスケーラビリティがある．

ファイルを共有するにあたって，セキュリティは大事な要素である．IR では公開鍵，秘密鍵により内容を暗号化し，内容が盗み見られないようにしている．これに対し Ivy はセキュリティに関しては現状ではほとんど考えられていない．

実際に使用する際にあるとよい機能として，NAT 越えがあげられる．NAT 越えとはルータの NAT 機能を利用して 1 つのグローバルアドレスを複数のマシンで共用している環境に，ルータの設定を変えずに外部からルータの内側のマシンにアクセスできることである．IR ではトラッカのプロキシ機能により，擬似的に直接通信が行える．Ivy では全ノードが直接通信が行える前提でデザインされているので，NAT 越えに関しては考えられていない．

6. ま と め

本研究では，分散環境下での協調作業を支援するためのファイル共有システムを提案した．本システムでは，協調作業をするメンバのストレージの集合が，仮想的に 1 つのストレージを形成し，その中に共有ファイルを置く．その仮想ストレージからファイルを取得し，作業用として使う．作業が終わったら作業ファイルを仮想ストレージに入れ，共有ファイルを更新する．実際には各メンバは全共有ファイルのキャッシュデータを持つ．これにより，ネットワークから分散されていても全部の共有ファイルを使用可能になる．また，サーバの管理コストを省くため分散型とした．分散型では通信するノードの位置情報を知っている必要があるが，それを管理するためにトラッカを導入した．トラッカは協調作業グループ内のノードの位置情報を管理し，各ノードは通信をしたい場合トラッカから他ノードの位置情報をする．そして他ノードの更新情報を取得するときに最新ファイルを判別するために，バージョンベクタを採用した．バージョンベクタの比較によって，最新ファイルを持つノードや，更新の衝突が検出できる．

今後の課題としては，トラッカの分散化があげられる．現在の実装では，トラッカのノードは常時起動している必要がある．このトラッカの機能を各ノードに分散して持たせることにより，常時起動が必要なノードをなくすことを目指す．また，共有ファイルにアクセス制御を入れることがあげられる．現状ではメンバはどのファイルにも自由にアクセスできる．しかし閲覧は許すが書き換えられては困るファイルも出てくると考えられる．そのためメンバごとにファイルへのアクセスを制限できるとよい．

参 考 文 献

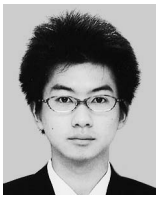
1) 福田信彦，楯岡孝道，中村嘉志，多田好克：ONFS：一時的なネットワーク環境下ですぐに利用できる共有ファイルシステム，情報処理学会論文誌，Vol.44, No.2, pp.353–363 (2003).
2) Muthitacharoen, A., Morris, R., Gil, T.M. and Chen, B.: Ivy: A Read/Write Peer-to-peer File System, *5th Symposium on OSDI* (2002).
3) BitTorrent http://www.bittorrent.com/
4) Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, *Proc. ACM SIGCOMM* (2001).
5) Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B. and Kubiatowicz, J.: Pond: the OceanStore Prototype, *Proc. USENIX FAST* (2003).
6) Druschel, P. and Rowstron, A.: PAST: A large-scale, persistent peer-to-peer storage utility, *Proc. HotOS VIII* (2001).
7) Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R. and Stoica, I.: Looking up Data in P2P Systems, *CACM* (2003).
8) Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H. and Steere, D.C.: Coda: A Highly Available File System for a Distributed Workstation Environment, *IEEE Trans. Comput.* (1990).
9) Sobalipse. http://sobalipse.sourceforge.net/
10) Handley, M. and Crowcroft, J.: Network Text Editor (NTE) — A Scalable Shared Text Editor for the MBone, *Proc. ACM SIGCOMM* (1997).
11) Rsync. http://rsync.samba.org/
12) FolderShare. http://www.foldershare.com/
13) Kang, B.B., Wilensky, R. and Kubiatowicz, J.: The Hash History Approach for Reconciling Mutual Inconsistency, *ICDCS* (2003).
14) CVS. http://www.nongnu.org/cvs/
15) RCS. http://www.gnu.org/software/rcs/

16) Subversion. http://subversion.tigris.org/
17) Firefox. http://www.mozilla.org/firefox/

(平成 18 年 1 月 27 日受付)
(平成 18 年 5 月 23 日採録)

塚田　大（学生会員）

1983 年生．2005 年 3 月筑波大学第三学群情報学類卒業．同年 4 月より同大学大学院システム情報工学研究科コンピュータサイエンス専攻博士前期課程に在学中．システムソフトウェア，分散システム，仮想化技術に興味を持つ．



鈴木　勝博（学生会員）

1983 年生．2005 年 3 月筑波大学第三学群情報学類卒業．同年 4 月より同大学大学院システム情報工学研究科コンピュータサイエンス専攻博士前期課程に在学中．オペレーティングシステム，分散システム，モバイル向けセキュリティに興味を持つ．

阿部　洋丈（正会員）

1999 年 3 月筑波大学第三学群情報学類卒業．2004 年 3 月同大学大学院博士課程工学研究科修了．同年 4 月より科学技術振興機構戦略的創造研究推進事業 CREST 研究員，現在に至る．博士（工学）．システムソフトウェア，特に分散システムとコンピュータセキュリティに興味を持つ．情報処理学会平成 16 年度山下記念研究賞，情報処理学会平成 16 年度論文賞受賞．日本ソフトウェア科学会，IEEE，ACM 各会員．

加藤　和彦（正会員）

1962 年生．1985 年筑波大学第三学群情報学類卒業．1992 年博士（理学）（東京大学大学院理学系研究科）．1989 年東京大学理学部情報科学科助手，1993 年筑波大学電子・情報工学系講師，1996 年同助教授，2004 年筑波大学大学院システム情報工学研究科教授，現在に至る．2003 年より情報処理学会システムソフトウェアとオペレーティングシステム研究会主査．オペレーティングシステム，セキュアコンピューティング，自律連合型分散システムに興味を持つ．