Vol.2010-OS-113 No.11
2010/1/28

yass: yet another simple storage

荒川淳平[†]　笹田耕一[†]　竹内郁雄[†]

yass（yet another simple storage）はユーザに優しいデータ管理の実現を目指して著者が開発している分散ファイルシステムである．yass は非常にシンプルで構築及び導入が容易であり，権限証明書を用いた柔軟なセキュリティを備える他，バージョン管理が可能で，機能追加・修正のコストが低く，システム全体で単一故障点を持たないといった特徴を持つ．本稿では yass の設計及び実装，またそのセキュリティモデルについて述べる．

yass: yet another simple storage

Jumpei Arakawa[†]　Koichi Sasada[†]　and　Ikuo Takeuchi[†]

Yass (yet another simple storage) is a distributed file system for user-friendly data management. Yass is easy to set up and to add or refine features.　In addition, yass has a flexible security model using authorization certificates, a version control function, and no single point of failure of the whole system. In this article, we describe the design and implementation of yass and its security model.

1. はじめに

データ管理において，ファイルシステムが非常に重要な地位を占めていることは言うまでもない．コンピュータを使ってデータを管理する場合，必ず何らかの形でファイルシステムを利用することになる．また，利用者にとってもファイルとディレクトリによる階層化された管理方法とそれを利用するためのアプリケーション（Windows のエクスプローラや MacOS の Finder）は非常に馴染みの深いものであり，コンピュータの基本スキルといって過言ではない．

著者は以前データ管理システム Decas[1]を開発し，ファイルシステムを仮想化することにより，バックアップや暗号化，バージョン管理などのデータ管理の機能を利用者が意識せずに利用することを可能にした．しかし，Decas は単一の計算機上で動作するように，データ管理機能を有するファイルシステム，またそのフレームワークとして，設計されていた．

しかし単一の計算機に閉じたデータ管理には限界が生じる．なぜならば，近年，デスクトップPCに加えてノートPCやネットブックなど２台以上の計算機を利用することは珍しくなく,また特に最近利用が盛んなスマートフォンも含めると,データは様々なデバイスから参照・編集できる必要がある．また，データは個々人の計算機に留まるだけでは真の価値を発揮するとは言えない．つまり，データのライフサイクルを考えた場合，最初は個人によって作成されたデータであったとしても，その後は様々な人々と共有され，互いに参照や編集を繰り返して利用されていくことが多い．実際，この裏付けとして世の中には多くのファイル共有サービスやファイル転送サービスが登場している．

これらのことから，データが複数の機器や複数の人々の間で利用できる形で管理されることが，これからのデータ管理に必要なことだと言うことができる．ここで，その実現に最も適している著者と考えたのが分散ファイルシステムである．分散ファイルシステムは単一のファイルシステムがネットワークを介して複数の計算機ノードに分散して機能するものと定義する．分散ファイルシステムはネットワークを介して利用できるため，複数のデバイスや利用者で共有することが容易である．また，分散ファイルシステムは一般に冗長性を提供するため，データ管理においてもっとも基本的なバックアップの機能を最初から備えている．

2. 要件定義

ここでは上で述べた背景からデータ管理に適した分散ファイルシステムの要件を以下のように定義し，それを満たす新しい分散ファイルシステム yass（yet another

[†] 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

simple storage）を開発した．

- 導入・構築が容易であること
- 機能の追加・修正が容易であること
- 特定のハードウェアや OS に依存しないこと
- システム全体で単一故障点を持たないこと
- 通常利用において十分なパフォーマンスがでること
- バージョン管理機能を備えていること
- セキュリティ管理がシステム的にも管理コスト的にもスケールすること

以下ではそれぞれの要件について詳しく述べる．

2.1 導入・構築が容易であること

データ管理を考える上で，構築されたシステムを利用するだけ純粋な利用者のみをユーザとしてとらえることは正しくない．分散ファイルシステムの導入・構築等を行うシステム管理者も重要なユーザである．純粋な利用者の多くに影響を与えるという意味では，管理者の負荷を軽減することは結果的に多くの利用者の利益になると考える．特に小規模なグループや個人での利用を考えた場合，利用者は管理者を兼ねることも多くあり，導入・構築のコストを低く保つことは重要である．また，導入・構築を簡易化することで，例えば一時的なファイル置き場をアドホックに作成するなど，システムの利用可能範囲を広げることも可能である．

また，分散ファイルシステムは，計算機ノードを追加することで容量や性能が増強できる．利用に応じてノードを後から追加できることは，機器の価格低下の傾向などから鑑みて，コストパフォーマンスの面から考えても優れている．したがって，初期の構築以外にもノードの追加も容易でなければならない．

2.2 機能の追加・修正が容易であること

分散ファイルシステムをデータ管理システムとして利用する場合，純粋な分散ファイルシステムの機能以外にも，継続した機能強化が必要になると考える（実際，バージョン管理機能はファイルシステムとしての機能ではない）．したがって，データ管理システムのフレームワークとしてとらえた場合，新しい機能の追加や既存機能の改良が容易でなければならない．

2.3 特定のハードウェアや OS に依存しないこと

これは 2.1 にも関連するが，管理者にとって重要な要件である．分散ファイルシステムが特定のハードウェアに依存する場合，構築や増強の際にそれらのハードウェアを調達する必要があり，管理者の負担を増やす．特定のハードウェアや OS に依存しないことで，休眠資産や手持ちの計算機（例えばノート PC）などでもシステムの構築・増強が可能になり，システムの適応範囲を広げることが可能である．

2.4 システム全体で単一故障点を持たないこと

どのようなシステムであってもハードウェアの故障を避ける手段はない．一般にファイルシステムでは HDD などの記録装置の故障を意識して RAID 等を利用することがある．しかし，記憶装置以外にも NIC や RAID コントローラなどの故障でも計算機ノードは機能を停止する．データ管理を担う（分散）ファイルシステムにおいて，機能停止は許容できない．したがって，ハードウェア故障やその他の要因で，システムを構成する計算機ノードのどの一つが機能停止に陥ったとしても，システム全体が稼働し続けることが必要になる．ここでシステム全体としたのは，単にデータの参照や保存だけでなく，認証やアクセス制御などを含めたシステムの全ての機能が利用できなければならないためである．

2.5 通常利用において十分なパフォーマンスがでること

分散ファイルシステムは，検索サービスや科学技術計算などの大規模計算用の巨大なファイルを扱うために設計されていることが少なくない．これらのシステムの中には通常利用において十分な性能を発揮できないものも存在する．ここでの通常利用とは，文章や画像などの比較的小さなファイルを数多く扱うようなファイルシステムの利用とする．

2.6 バージョン管理機能を備えていること

バージョン管理機能とは，ファイルを任意の過去の状態に戻せる機能である．また，ファイルだけでなくディレクトリもバージョン管理の対象とする．つまりファイルの追加や削除に対しても適応可能とする．バージョン管理機能は著者がデータ管理の中心的な機能の一つと位置付ける機能である．バージョン管理機能は，ファイルシステムに対するすべての操作を「元に戻す」ことを可能にすることで，ファイルを上書き保存や削除するときに感じる不要なストレスや手動でのバージョン管理のための不要なコピーからユーザを解放できる．

2.7 セキュリティがスケールすること

2.4 とも関連するが，ファイルシステムでは認証やアクセス制御が中央管理的に 1 つの計算機ノードで処理されることが多い．これではユーザ数の増加などに対してシステムとしてスケールしない．それに加えて，ユーザ数の増加によって増えるのはシステムの負荷だけではない．ユーザの登録やアクセス制御の設定などでシステム管理者の負担もユーザ数の増加とともに大きくなる．したがって，安全性を保ちつつも，システムや管理の負荷が分散する仕組みを有している必要がある．

3. 基本となるファイルシステム

本章では yass の基本となるファイルシステムについて述べる．

3.1 通常ファイルの管理

通常ファイルとはバイト列の書き込み・読み込みできるインタフェースを持つ抽象モデルである．yass では複数の計算機ノードに分散してファイルを扱えるようにする

ため，通常ファイルをブロックと呼ぶバイト列に分割して管理する．このブロックはCAS（Content Addressable Storage）方式，ブロックの内容（バイト列）自体がそのブロックを指し示すアドレス方式で管理する．つまり，アドレスが同じブロックということは内容が同じブロックであるということを意味する．これにより，ファイルシステムで一般的なユーザが比較的自由に設定できるパス（文字列）を使ったアドレス方式のように，同一アドレスでの内容の上書きを必要としない．yass では通常ファイルを複数のブロックとそのブロックのアドレスをリスト化したインデックスで表現される．インデックスは通常ファイルを構成するブロックのアドレスを先頭から順に並べたシンプルな構造で，アドレスの算出方法やブロックの分割方法に依存しない（図 1）．

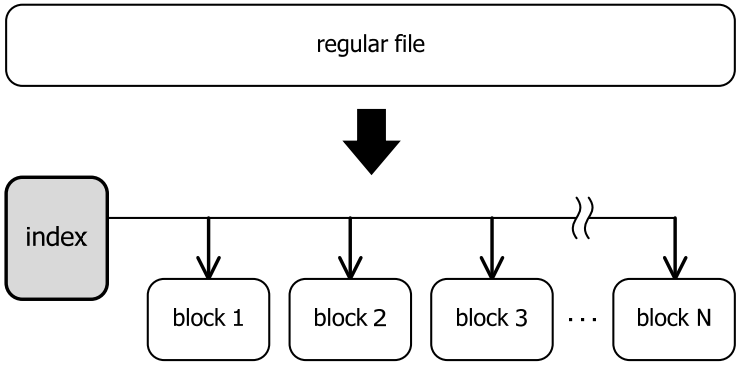

図 1 ブロックとインデックスによる通常ファイルの管理

3.2 ディレクトリの管理

ディレクトリは 0 個以上のエントリ（通常ファイル及びディレクトリの参照情報）を持ち，それらを追加・削除できるインタフェースを持つ抽象モデルである．yass ではディレクトリをエントリホルダと呼ぶデータ構造で実現する．エントリホルダはその名の通りエントリを保持する．また，エントリホルダはシステム全体（単一の計算機ノード内だけでなく，すべての計算機ノード）でユニークな識別子 EHID（Entry Holder IDentifer）を持つ．yass ではエントリホルダに対してエントリをすべて追記で記録する．つまり，ファイルの更新や追加・削除はその発生時刻（エントリの更新時刻）と共に，すべて新しいエントリとしてエントリホルダに保存される．

これにより，ファイルシステムに対するすべての操作に「上書き」という処理が不要になり，分散システムを構築する上で問題になる同期の問題を避けることができる．また，通常ファイル・ディレクトリの状態を全て記録するため，任意の過去のエントリを取得することができる．この性質は，そのままバージョン管理機能の実現を可能にする（図 2）．

name	type	status	update time
hoge.pdf	File	-	2009-12-31 10:10:10
fuga	Directory	-	2009-12-31 11:11:11
hoge.pdf	File	Deleted	2009-12-31 12:12:12
piyo.ppt	File	-	2009-12-31 13:13:13
mohe.xls	Directory	-	2009-12-31 14:14:14
piyo.ppt	File	Locked	2009-12-31 15:15:15
piyo.ppt	File	-	2009-12-31 16:16:16

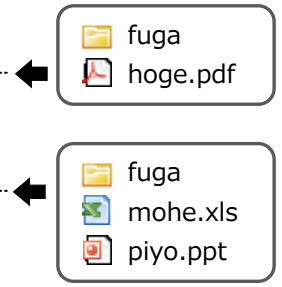

図 2 追記式でのエントリ管理

通常ファイルのエントリには，ブロックのインデックスを保持し，ディレクトリのエントリは EHID を保持する．また，共通してファイル名・更新時刻やそのファイルの状態などのメタ情報も保持する．EHID はディレクトリ作成時に割り当てられ，ディレクトリの移動や名前変更が行われた場合も変化しない．通常ファイルやディレクトリの作成時には親ディレクトリに対応するエントリホルダに新しくエントリが追加される．

ファイルの状態は同じファイル名を持つエントリのうちで最も新しいもの（最新エントリと呼ぶ）が保持している状態で決定する．最新エントリが削除状態であればそのファイルは存在しないものとして扱われ，ロック状態であれば条件を満たさない書き込み（エントリの追加）は拒否される．

4. セキュリティ

この章では yass のセキュリティモデルについて述べる．ファイルシステムのセキュリティは大きく分けて認証とアクセス制御の 2 つがある．認証とは誰がシステムのユーザであるかを決定する機能であり，アクセス制御とはユーザ X がリソース R に対して何ができるを決定する機能である．認証は，一般的にそのユーザしか知り得ない秘密情報（パスワードや公開鍵ペアの秘密鍵）を所持していることをシステムに対して証明することで実現される．アクセス制御のための情報は Access Control Matrix（ACM）として表現することができ，一般的にはリソースごとにユーザ別のアクセス権を記録した Access Control List（ACL）として実現される．これに対して，yass ではユーザごとにリソース別のアクセス権を記録した Capability をベースにした方式を採用する（図 3）．

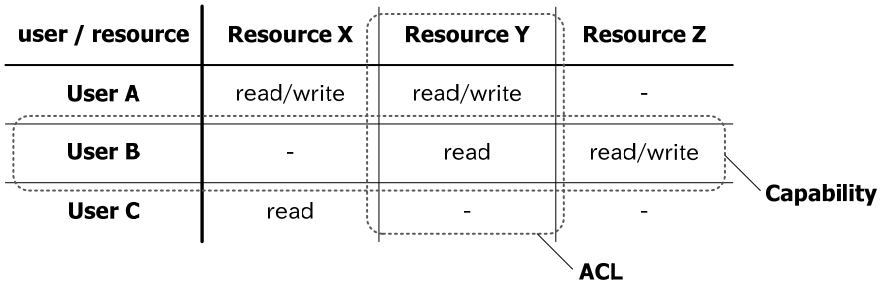

図 3 アクセス制御方式

yass では Capability に正真性（内容が改ざんされていないことを検証できる性質）や否認不可能性（後でその内容を作成していないと偽証できない性質）を付与するためにデジタル署名をした権限証明書を用いてアクセス制御を行う．権限証明書には，所有者のユーザ ID や対象リソースのセット，許可されているアクセスの種類，証明書の有効期間などが含まれる．

ユーザはファイルの操作に先だってシステムに権限証明書を提示する．システムは権限証明書を検証して，妥当なアクセスかどうかを判断する．ここで重要な点は，公開鍵暗号で署名された権限証明書をシステムが保持する必要がないという点である．この性質により，アクセス制御情報を集中的に管理する必要もないため単一故障点にならなず，ユーザ数の増加に対しても記憶コストが増加せず，システム的に良くスケールすると言える．

権限証明書を使ったアクセス制御の利点は，ユーザが自分の有する権限を元に，他のユーザにその権限（のサブセット）を委譲できることである．ファイルの共有は権限の委譲によって実現される．この時，特に管理者などの利用者を介さずに権限の委譲を行えることを Autonomous Delegation と呼ぶ(図 4)．yass では Autonomous Delegation が権限証明書の発行によって実現される．つまりユーザが所持している権限証明書を元に他のユーザに権限証明書を発行することができる．システムは権限証明書の発行時に元の権限証明書を超える権限が設定されていないことや有効期限を超えていないことなどを検証した上で，権限証明書に署名を付けて発行する．

Autonomous Delegation によって，管理者は個々のユーザのアクセス権限を設定する必要はなく，直下のユーザのみに権限を委譲することで自然に権限が適切に設定されていく．これにより，ユーザ数の増加に対して，yass のセキュリティを実現する人的負担は，(特に階層化された組織では）対数的な増加に抑えることができる．

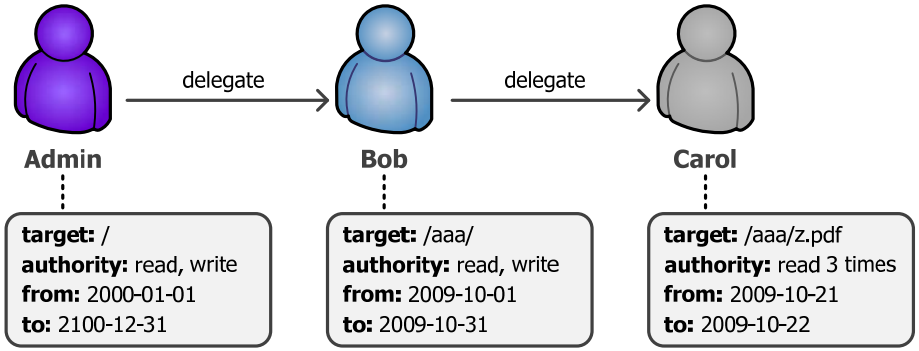

図 4 Autonomous Delegation の仕組み

5. アーキテクチャ・分散方式

本章では yass のアーキテクチャ及び分散方式について述べる．

5.1 アーキテクチャ

yass はサーバとクライアントに分かれる C/S 型のシステムである．ただし，サーバ側は複数の対等な（どれかが特別な役割を持たない）計算機ノードで構成される．サーバ（の計算機ノード）とクライアントは API を通じてやり取り行う．

5.2 分散アルゴリズム

yass は前の 2 つの章で扱ったデータ構造，すなわちブロック，エントリ，権限証明書，CRL を複数のサービスノードに分散させて保持する．基本的には分散ハッシュテーブルのアルゴリズムとして有名な Chord[2]でも用いられる円状のハッシュ空間をベースに，ノード ID とコンテンツ ID によって分散を行う．

つまり，円状ハッシュ空間上にノードをその ID で配置し，コンテンツはその ID が円状ハッシュ空間において時計回りで最も近いノードに保存する．この時コンテンツを保持するノードのことをコンテンツに対する担当ノードと呼ぶ．冗長性の実現や負荷を分散する目的で，設定されたレプリケーション数を n とした場合，担当ノードから n - 1 近傍のノードにもコンテンツを保存する（図 5)．

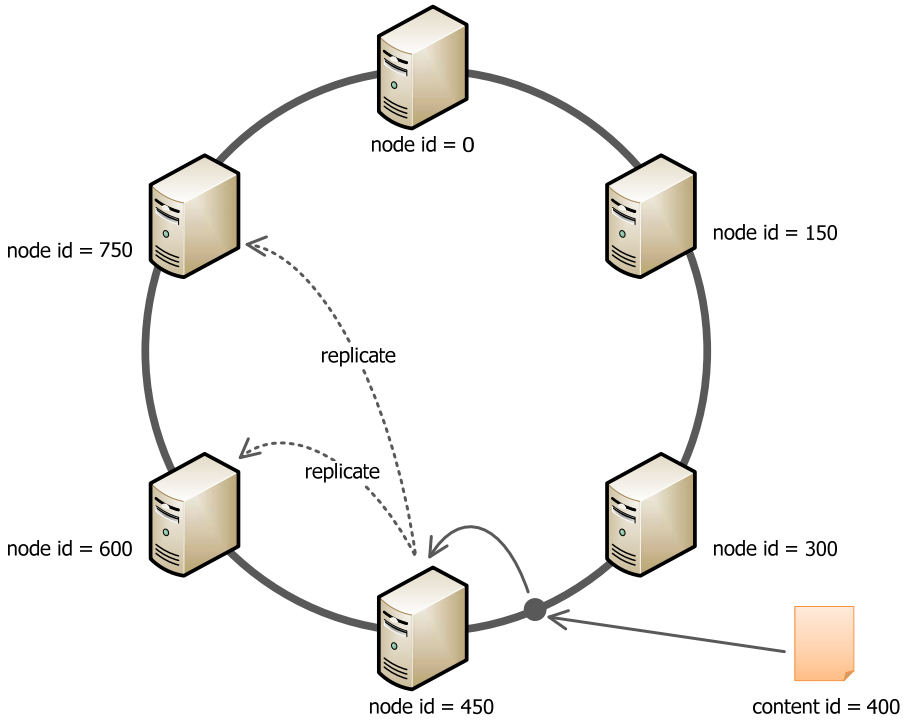

図 5　円状ハッシュ空間上でのコンテンツ分散（レプリケーション数が 3 の場合）

Chord では隣接ノードの管理にフィンガーテーブル等が用いられるが，yass では単純に全ノードがその他の全ノードの情報を保有する．これにより各ノードに必要な記録コストは増加するが, 必ず 1 回の通信で目的のノードを参照することが可能となる．

このトレードオフの妥当性を以下に示す．データ管理の基盤としての分散ファイルシステムは，分散ハッシュテーブルの実用例として多いファイル共有ソフト（Winny や BitTorrent）のようなアプリケーションと比べて，安定したネットワーク，つまりノードの頻繁な追加や離脱が発生しない環境を想定している．また，ノード数自体も 1 桁〜多くとも 3 桁のオーダーであり，利用時の応答性に対して，ノード情報を管理するための通信コストや記憶コストが十分に低いと言える．

また Chord では，分散対象のコンテンツのハッシュ値をキーとして円状ハッシュ空間にマッピングを行うが, yass ではコンテンツの種類に応じて表 1 のように異なる ID を採用している．またコンテンツ保存時にキーが重複した場合の挙動も異なる．

表 1　コンテンツ別の分散方法

コンテンツ	分散に用いるキー	キー重複時の保存方法
ブロック	ブロックの内容からの算出値 （一般的にはハッシュ値）	（重複なし）
エントリ	親ディレクトリに対応する エントリホルダの EHID	追記
権限証明書 （システムで保存する場合）	権限証明書の ID	上書き

特にエントリの分散において，キーに対象エントリ自体の値ではなく親ディレクトリの EHID を用いるのは，ファイルシステムで頻繁に行われるディレクトリ内のエントリ一覧の取得などの操作が複数のノードに分散することで，応答性が著しく低下することを回避するためである．

ブロックの分散については，キーが同一の場合，3 章で述べた通り同一内容のブロックを意味するため，新たに送信・保存する必要がなく，時間のかかる分散化を効率良く実現できる．

5.3 ノードの追加

最初の 1 台は，ノード ID とその他のノード情報（記憶容量の最大値など）を指定して新しいノード（新規ノード）を初期化する．

2 台目以降のノード追加は，ノード ID とその他のノード情報に加えて，既に存在するノード（既存ノード）の URI を指定して新規ノードを初期化する．初期化では，新規ノードはまず既存ノードからシステムを構成する全ノードの情報を取得する．これにより，新規ノードは現在の円状ハッシュ空間（旧ハッシュ空間）を計算することができる．次に新規ノードを加えた新しい円状ハッシュ空間（新ハッシュ空間）を計算し，その空間で新規ノードが担当するべきコンテンツを旧担当ノード（旧ハッシュ空間でコンテンツが割り当てられているノード）から取得する．旧担当ノードから担当すべきコンテンツの取得が完了すると，新ハッシュ空間の構成情報を他の既存ノードに配布する. そして最後に旧担当ノードに不要になったコンテンツの削除を要請する．

yass では基本的にノードが複数同時に追加されることは想定していないが，実際には後述するノードの修復により，複数ノードが同時に追加されても安定的にシステムは稼働できる．

5.4 ノードの離脱

明示的に離脱するノード（離脱ノード）は，まず既存の円状ハッシュ空間から離脱ノードを取り除いた新しい円状ハッシュ空間（新ハッシュ空間）を計算する．次に離脱ノードが保持しているコンテンツの新しい円状ハッシュ空間上で担当すべきノード（新担当ノード)に配布する. 新担当に保持していたコンテンツの配布が完了すると，

新ハッシュ空間の構成情報を既存の他のノードに配布する．そして最後に自ノードの不要になったコンテンツを削除する．

5.5 ノードの修復

ノードの修復は，予期せぬノードの離脱した場合，例えばネットワーク障害などにより一時的にノードが落ちていて普及した場合や記憶装置の故障などによりノードの情報が完全に失われ, ゼロからノードを復元しなければならない場合などに対処する．

基本的には，ノードの追加と同様の流れだが，旧ハッシュ空間と新ハッシュ空間は同一になる．つまり，担当すべきコンテンツ（のレプリカ）を持つノードからコンテンツを取得することでノードを修復する．ただし，すでに自ノードが保持しているコンテンツは当然取得する必要はない．修復完了後には，修復が完了したことを既存の他のノードに通知する．

5.6 ノードの相互監視

ノードの一時的な停止やそれによるコンテンツの所有漏れを検出する目的で，定期的にノード同士が互いに所有しているコンテンツの一覧等を交換する．コンテンツの所有漏れや過剰所持が判明した場合は，ノードの修復を実行することで，システムを正しい状態に戻すことができる．

このノードの相互監視の頻度を高くすることでシステムの状態をより安定的に保つことが可能だが，通信コストや処理コストを多く払うことになる．加えて，一時的なノード停止やコンテンツの所有漏れがあったとしても，十分なレプリケーション数を設定しておくことで，システム全体としては正しく動作することが可能であり，過剰に相互監視の頻度を高くする必要はない．

6. 実装

この章では yass の実装について述べる．

サーバプログラムは HTTP サーバ上で動作する単一の PHP スクリプト（yass.php）として実装した．yass ではファイルシステムをウェブの公開用ディレクトリが設置されているファイルシステムの上に構築する．ファイルシステムは PHP から利用可能な POSIX に準拠したファイルシステムであれば利用可能である．これにより，特定のハードウェアや OS（やベースとなるファイルシステム）に依存せずにシステムの実現が可能となっている．

ブロック分割には実装の容易な固定サイズでの分割を，ブロックアドレスはブロックのハッシュ値（SHA1）とサイズを結合した文字列を，認証にはシンプルなパスワード認証をそれぞれ採用して実装を行った．

また，yass.php をウェブディレクトリに設置するだけ導入が可能となっている．初期化はウェブブラウザでアクセスして必要な情報（ノードの URI は実際には HTTP の URL になる）を入力するだけで，必要なディレクトリやファイルを生成する．

yass.php は 3000 行程度の非常にコンパクトな実装となっている．また，API 単位で機能が追加･削除可能な形で実装がされており, 機能追加や修正が容易になっている．

PHP では実行インスタンスはウェブサーバに対するリクエストの単位でしか存在できない．このため，yass.php ではノードの相互監視などの定期的な処理を実行するために，スリープと自身へのリクエスト発行を用いて定期タスク実行機構を実装した．

また，yass.php はブラウザ経由で利用可能なウェブクライアント（yassui）も内包している．yassui は Flash を用いて実装し，Windows のエクスプローラのようなインタラクティブな操作を可能としている．yassui では複数ファイルの選択や右クリックによるメニュー表示にも対応している（HTML の input タグでは単一ファイルしか選択することができない）．yassui によって，クライアントプログラムを別途インストールせずにファイルシステムを利用することが可能となっている．

yassui に加えて，ローカルのファイルシステムに統合して yass を利用するためのデスクトップクライアント（yassfs）を実装した．yassfs は Decas フレームワークを利用いて構築したため，ドライブやディレクトリにマウントしてシェルやアプリケーションから直接利用が可能である．また，ローカルにキャッシュを持つことでオフライン時でも利用可能となっている．これは単にファイルの読み込みが可能なだけでなく，ローカルに一時的にサーバ側のノードと同様のファイルシステムを構築し，オンラインになった後に差分（ブロックやエントリ）をサーバ側に送信することで，オフライン時に書き込みも可能としている（図 6）．

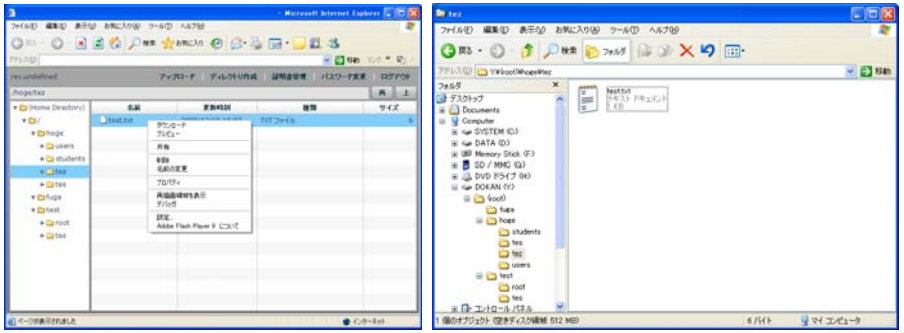

図 6　yassui 及び yassfs による yass の利用

7. 既存システム・関連研究との比較

本章では yass を既存システム及び先行研究と比較して述べる．

Decas はバックアップ機能やバージョン管理機能を備えたファイルシステムとして

実装されたデータ管理システムである．しかし，単一のマシンでしか動作せず，冗長性を実現するためには別途計算機を用意する必要がある．また Decas はセキュリティシステムを持たず，Decas が構築されているオペレーティングシステムのセキュリティ機能を利用するためスケールするとは言えない．

Gfarm[3]は C 言語で実装されているオープンソースの高機能な分散ファイルシステムであるが，動作環境が Linux に限定されており，バージョン管理機能も有していない．

GFS（Google File System）[4]は Google で利用されている分散ファイルシステムである．バージョン管理機能も備えた高機能なファイルシステムであるが，非公開であり開発に利用することができない．

HDFS（Hadoop Distributed File System）[5]は GFS の言わばオープンソース版であり，Java で開発されているため OS に依存しない．GFS と同じく高機能でバージョン管理機能も有している（ただし HDFS が提供するバージョン管理機能は yass でいうブロックレベルであり，ファイルやディレクトリに対しては適応できない）．しかし，GFS も HDFS も巨大ファイルを対象とした MapReduce 用のファイルシステムであり，一般ユーザが利用するような通常の用途ではパフォーマンスが低下し十分な応答性が得られないことが知られている．また，HDFS はメタデータを管理するノードが単一故障点となる．

cagra[6]は C++で開発されているオープンソースの分散ファイルシステムである．上述べた 3 つのファイルシステムと比較して，cagra はプログラムをインストールし，起動するだけで必要なコンフィグレーションを行ってくれるため導入が非常に容易である．ただし cagra は分散型の key-value ストレージでありファイルシステムではなく，当然バージョン管理機能やセキュリティ機能なども備えてない．

MogileFS[7]は Perl で開発されているオープンソースの分散ファイルシステムである．Perl で記述されているため OS に依存せず，機能追加や修正も上で述べた他のファイルシステムに比べると容易である．しかし，バージョン管理機能を備えておらず，MySQL 等の外部プログラムを利用しているため導入と構築に手間がかかる．

8. まとめ

本稿では，理想的なデータ管理の実現のための要件を定義し，それらを満たす新しく開発した分散ファイルシステム yass について述べた．yass はウェブサーバ上に PHP を用いて実現することで，導入・構築や機能の追加・修正が容易で特定のハードウェアや OS に依存しない．エントリの保存方式や分散方式を工夫することで，システム全体で単一故障点をもたず，バージョン管理機能も実現した．また，権限証明書を用いることでユーザ増加に対してスケールするセキュリティを実現した．通常利用において十分なパフォーマンスが出ることは応答性としては確認できたが，今後詳細なベンチマークを実施して，定量的な評価を実施する予定である．

参考文献

1) 荒川淳平，淺川浩紀．意識しないで自然に使えるデータ管理システム Decas，第 49 回プログラミング・シンポジウム報告集, pp.65‐72, (2008).
2) Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM 2001, pp.149-160, (2001).
3) 建部修見，曽田哲之: 広域分散ファイルシステム Gfarm v2 の実装と評価，情報処理学会研究報告, 2007-HPC-113, pp.7-12 (2007).
4) Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung: The Google File System, 19th ACM Symposium on Operating Systems Principles, (2003).
5) エヌ・ティ・ティレゾナント株式会社，株式会社 Preferred Infrastructure: Hadoop 調査報告書, (2008).
6) 古橋貞之，上野康平: P2P 分散ストレージ「cagra」 http://d.hatena.ne.jp/viver/20080429/p1 (2008).
7) MogileFS http://www.danga.com/mogilefs/ (2009).