Vol.2010-HPC-126 No.30
2010/8/4

広域分散ワークフローのための耐遅延性の高い分散ファイルシステム

柴田剛志[†1]　田浦健次朗[†1]

近年，HPC の分野においても処理したり保存すべきデータの量が大幅に増えてきている．サイエンティフィックワークフローは，そのようなデータインテンシブな情報処理を行う際に有用な手法として注目を浴びており，実際に，すでに様々なシステムが存在する．一方で，そのワークフローの入出力は，中間データを含め，ファイルとして分散ファイルシステム上におかれることも多い．広域分散環境において単一のファイルシステムを構築するさいに，考慮しなければならないは遅延，およびネットワークの帯域である．本稿では，とくにワークフローシステムに向いた，どのようなファイルシステムに対しても，新たな計算ノードに対して，簡単に追加的に拡張することができるような対遅延性のあるファイルシステムを，すでにある SSHFS というアプリケーションを拡張することにより実現する．

A File System which is Tolerant to Latency for Scientific Workflow Systems

Takeshi Shibata[†1] and Kenjiro Taura[†1]

Recently, amount of data to be processed and reserved is increasing exponentially. Scientific workflow is one of the computation models for such data-intensive information processing on distributed environments such as grid and cloud. Many workflow systems have already been developed and applied to scientific workflows. Input and output data is often written in files and transferred between nodes. Those files are often put on a single underlaying distributed file system. To use filesystems efficiently for workflow systems on wide area environments, it is important to considering tolerance of latency between filesystem servers and compute nodes. In this paper, we develops a filesystem which is tolerant to latency for scientific workflow systems by extending an existing filesystem SSHFS.

1. はじめに

近年，複数のクラウドやクラスタを柔軟につなぎあわせて使用することが求められているが，利用者の利便性を損なわないようにするためには，次のようなことが実現できる必要がある．

計算環境とは異なる場所，時には複数箇所にある，大量のデータを，とくに複雑なことをしなくても，簡単に異なる場所にある複数の計算資源を使ってユーザレベルで読み書きし，処理することができる．ここでいう複雑なこととは，データの転送，計算機の指定，使う環境を考慮する，などということである．サイエンティフィックワークフローは，グリッドの分野において，それらの要求を満たす新たな計算資源の使い方として注目されている．サイエンティフィックワークフローとは，計算やデータ処理からなる一連のステップ（以降 job と呼ぶ）からなり，その間のデータの入出力を表す依存関係を記述したもので，特に科学技術計算に特化して設計されたものである．とくに，我々は， job (computational steps) 間の依存関係から並列に計算できるところが判明するため，データの送受信などを記述しなくてもよいという意味で，簡単にスケーラビリティを得ることができることに注目している．なお，依存関係とは，各 job 間における半順序関係のことであり，次のような制約を記述するものである．

- job A が job B に依存しているとは， A を実行する前に B が終了していなければならないということである．

とくに，B から A にデータの受け渡しがある場合は A が B に依存している必要がある．

何らかの言語で記述されたワークフローを実際に分散計算環境で実行するためのワークフローシステムとしては，Dryad[2] や Pegasus[1] などすでに様々なものがすでに存在し使われている．これらのワークフローシステムでは，job 間の入出力をファイルで行うことが一般的である．その際，ユーザーに入出力ファイルを明記して書かせない場合，必然的に全体としてひとつのファイルシステム上に置く必要がある．

一方，現在の計算環境では，単一のクラスタ内では，分散ファイルシステムとして NFS, Luster などが使われているが，これらは広域ネットワークをまたいで使われるようには設計されていない．広域ネットワークでつながれた複数のクラスタ上で使えるものとしては，

†1 東京大学
University of Tokyo

Hadoop のファイルシステムである HDFS や，Gfarm などが知られている．しかし，これらも，現状では高遅延環境を考慮して設計されているとは言いがたい．SSHFS は，SSH で接続したリモートのノードからみえるディレクトリを，FUSE というモジュールを用いて直接ローカルにマウントすることができるソフトウェアである．SSHFS の利点は，どのようなファイルシステムに対しても，新たな計算ノードに対して，簡単に追加的に拡張することができる点である．しかし，SSHFS は，後述するように，そのままでは対遅延性に優れているとは言いがたい．したがって，本研究では，ワークフローシステムに特化した対遅延手法を SSHFS を改良する形で実装する．

2. 関 連 研 究

Gfarm 上で，例えば 1 クラスタにひとつなど，複数のメタデータサーバを立て，それらの間では弱い整合性しか要求しないことによって，遅延を少なくすることができる．メタデータサーバ間では，all to all で通信を行う．

また，Ioan Raicu ら[3] による Data Diffusion とよばれる手法では，一度読まれたを計算機にキャッシュし，ジョブを投入するときにそのジョブが必要とするファイルをキャッシュしている場所を同時に送る．これはファイルシステムと呼べるものを提供しているわけではないが，遅延を少なくする手法としては本稿における提案手法のメタデータに関するものとほぼ同じである．

しかし，上記のどちらの手法も，簡単に別の既存のファイルシステム，例えば Luster などを追加的に拡張できるようにはなっていない．例えば，クライアントはあくまで Gfarm 専用のサーバが必要であり，本研究のベースとなる SSHFS は，どのようなファイルシステムにたいしても，それをマウントしているノードに SSH がつながり，計算ノードで FUSE が使えるならば，そのファイルシステムを計算ノードに追加的に拡張することが可能である．この特性は他の対遅延性を考慮したファイルシステムと比べても，大きな利点であると考えられる．

3. ワークフローシステムに特化した対遅延手法の必要性

一般的に，ワークフローを記述するときは，job がどのようなデータを入力とし出力とするかについて，全てを正確に記述する必要があるのが普通である．実際に，Pegasus などでは，データの入出力はワークフローの作成者が指定する必要がある．一方，GXP Make や Pwrake[4] などの，分散ファイルシステムの存在を想定したでは，入出力をファイルに限定し，分散ファイルシステムを用いることによって，ユーザに全ての入出力を記述させる必要はなくなっている．これは，事前に計画されたノード間ファイル転送や，ジョブ投入に関するスケジューリングがしにくくなるといった欠点があるものの，ユーザのワークフロー記述における負担を経験するという意味で，十分な利点があると考えられる．

分散ファイルシステムを用いたデータの共有は，一方で，直接明示的に示された入出力データをもとにして必要最小限の転送を行うようなシステムに比べて，効率性にかける部分が出てくる．次の二つが主な原因である．

(1) リードサイズが小さい時や lookup 等のファイルサーバとの遅延によるコスト
(2) ファイルサーバへデータ転送が集中することによるコスト

研究の目的は，上記のコスト，とくに高遅延環境におけるコスト（つまり (1)）を最小限に抑えるような手法の提案である．

広域分散環境におけるワークフローのためのファイルシステムとしては，遅延がおおきくても，スムーズにファイル操作が可能である必要がある．ファイル操作は次の三つに分類できる．

- lookup, open, close などのメタデータアクセス
- read
- write

まず，メタデータアクセスは，遅延の影響がダイレクトに効いてくる．たとえば，　/A/B/C/F というパスのファイルを読むときには，read 以外に，open や 複数の stat，release といったメタデータに関するファイルシステムにおける関数の呼び出しが発生する．これらのたびに毎回リモートにあるファイルサーバにまで問い合わせていては，高遅延の環境下では 1 つの小さなファイルを読むのにも不必要に時間がかかる事になる．

read においても，毎回リクエストをサーバに発行すれば，遅延の影響が大きい．遅延の影響をさけるためには，必要十分なだけ先読みしてバッファにデータをためておく必要がある．必要十分な先読みは，シーケンシャルなリードが行われている場合とくにクリティカルである．

write の場合も，サーバからの write 完了の ack を待つと遅延の影響がある．そのために，ローカルで適度にバッファリングする必要がある．

4. SSHFS における対遅延の仕組み

SSHFS は，SFTP と FUSE を用いた，リモートにあるディレクトリをローカルのディレ

クトリにマウントする仕組みである．ソースは C で書かれており，数千行程度で比較的少ないが，機能的に完成されており，広く使われていて信頼のおけるものである．

SSHFS の仕組みとしては，Fig. 1(a) に示すように，SSH を通してリモートに SFTP サーバを立ち上げ，FUSE 経由できたファイルアクセスのリクエストを SFTP のプロトコルに変換して, 立ち上げたリモートの SFTP サーバとやり取りする，という仕組みである．SFTP サーバは，SSH の暗号化とは独立して，ファイルアクセスのためだけのプロトコルを標準入出力から受け付ける仕組みになっている．

遅延の影響を回避して効率的なファイルアクセスを行うために，SSHFS が行っている仕組みは次のようなものである．

- メタデータアクセス：キャッシング
- read：先行リード
- write：サーバからの ack を待たない

4.1 メタデータのキャッシング

メタデータに対しては次のようなキャッシングが行われている．

(1) 60 秒ごとにキャッシュを全てクリアする．
(2) 前回のクリアから 5 秒以上経過し，かつ 10000 エントリ以上たまったとき，クリアする．
(3) キャッシュの対象となる各要素は，ファイルの情報，ディレクトリの情報，リンクの情報があるが，それぞれ，前回のキャッシュ時刻から cache_{stat,dir,link}_timeout 秒 (default: 20 秒) だけ有効である．
(4) ネガティブキャッシュ(無かったことや消したことを記憶すること) はしない．

4.2 先行リード

リードについては，次のような工夫が実装されている．

(1) リードがシーケンシャルであると判断した場合，現在のリードサイズと同じだけのサイズを先行的に先読みしておく．

オープンした直後の初期状態は，非シーケンシャルリードである（つまり先読みしない）．リードが呼び出されたとき，今回のリードの先頭のアドレスと，前回のリードの終端のアドレスがとなりあっていれば，シーケンシャルとみなす．ただし，前回と今回の間に，同ファイルに対して書き込みが起こっていた場合は非シーケンシャルとみなされる．

4.3 サーバからの ack をまたないライト

ライトについては，簡単で，ライトが終わったという SFTP サーバからの ack を待たないという実装になっている．これは，ユーザプログラム側から見ると，TCP の送信バッファに書き込まれた時点で write の呼び出しが終了し帰ってくるということである．flush のときや release(close) のときに，それまでに SFTP サーバに送った write リクエストの ack を全て待つ．

5. 提 案 手 法

対遅延性の高い分散ファイルシステムを実現するためには，SSHFS に対して，次に上げるような改良を加えればよいと考えている．

(1) メタデータをまとめてアクセス
(2) より積極的な投機的 read
(3) write のキャッシング

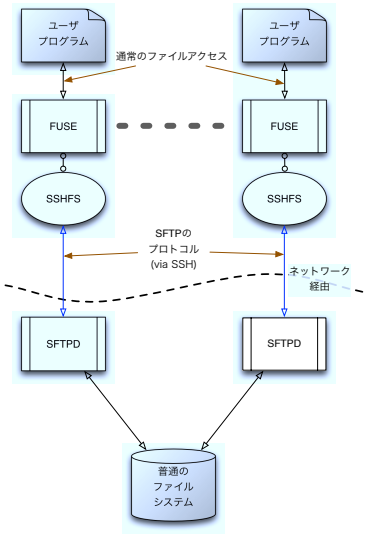
このような改良を実現するためには，SSHFS のデフォルトでの動作のように，各ノードにおいて，SSH のコネクションを張って，SSH を通して立ち上げた SFTP サーバと標準入出力でデータをやりとり方法ではあまりに不便である．幸いにも，SSHFS には，-directport というオプションがあって，これでサーバのアドレスを指定すると，SFTP サーバに SSH を通さず TCP 通信でデータをやりとりすることができるような機能が存在する．これを利用して，SFTP サーバと SSHFS の中間に入るようなサーバを立てて，もともとの SFTP のプロトコルにはないような種類の機能を追加することが簡単にできる（図 1）．

このサーバを仮に SFTPD+と呼ぶことにする．また，改良した SSHFS を SSHFS+とよぶことにする．

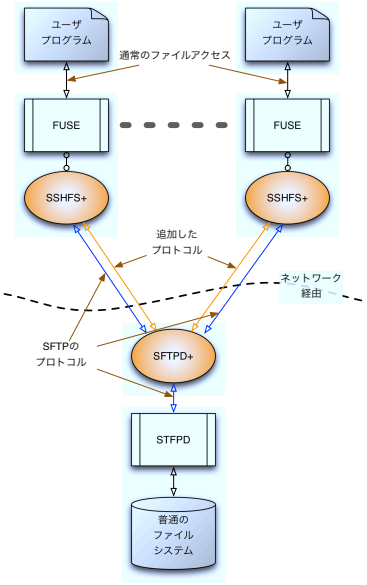
ワークフローの実行において，各ジョブが正しく実行されるために，メタデータに関して満されるべき必要十分な条件は，次の一つのみである．

- ジョブの実行前に，ワークフローでそれの上流 (Fig. 2) に位置するジョブによるメタデータの変更が全て正しく反映されている．

したがって，直接関係のないジョブ，つまり，上流にも下流にもなっていないジョブによるメタデータの変更がどのタイミングでおこっても，それはいっこうに構わないし，そのことでワークフローが意図していたように動かないということがもし起こったとしても，それはワークフローの書き方が間違っていたということになる．makefile でいうと，正しい依存関係がかけていないということを意味する． このようなことがなぜ言えるかは，そもそものワークフローの依存関係がどういうものかを思い出せば，明らかである．ワークフローの依存関係において，ジョブ A の上流にも下流にもなっていないジョブ B があったとする．こ



(a) SSHFS の仕組み



(b) SSHFS+の仕組み

図 1 目的を達成するための改良

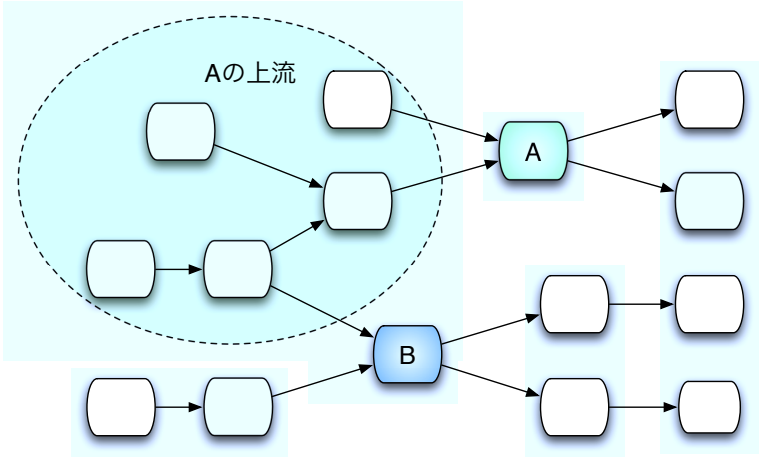


図 2 必要十分なメタデータの更新情報は上流によるもの

のとき，A と B のどちらを先に（あるいは同時に）実行してもよいという約束になっていた．A と B のどちらを先に実行してもいいということから，A にとって B によるメタデータの変更がいつ行われてもよいということが言える．

5.1 システムの動作

おおよそ次のような動作となるように実装している．

(1) SFTPD+の初期化時に，対象となるディレクトリに含まれるディレクトリとファイルのメタデータを取得し，メモリ上に持っておく．

(2) SSHFS+を各計算ノードで SFTPD+のアドレスを指定して起動する．SSHFS+は，ローカルから接続してメタデータのアップデートのリクエスト等を受け付けることができるようになっている．

(3) ワークフローシステムは，計算ノード上で，ジョブを実行する直前に，SSHFS+に対し，メタデータのアップデートをリクエストする．

(4) メタデータのアップデートの終了後，ジョブが実行される．

(5) ワークフローシステムは，ジョブが終了したら，メタデータの変更が全て SFTPD+にコミットされるまで待つ．

(6) 全てのメタデータが SFTPD+を通して反映されたら，ワークフローの次のステージ

のジョブに進む．

このようにすることで，lookup, open, close など，メタデータアクセスの毎に SSHFS+ サーバとの通信が発生するということはなくなる．

ここで，メタデータのアップデートであるが，全てのメタデータを送ると，並列度が増加したとき，大量のファイルがあるせいでメタデータ全体のサイズが無視できないほど大きくなり，通信量が増えてしまったりメモリが不足するようなことがおきる可能性があることである．それがボトルネックとなって，かえってワークフローのメイクスパンが長くなってしまう可能性もある．そのための対策としては，

- メタデータのアップデート時に，上流のジョブによる変更の差分のみを送受信する．
- 必要なファイルを予想してそのメタデータのみを送る．

といった方法が考えられる．前者の方法は，メタデータの変更の履歴をなんらかの形で持つ必要があり，SFTPD+の負荷が増える．あまりにも必要な履歴が多くなると，メモリに乗り切らなくなってしまう可能性もある．したがって，本稿では，ジョブ投入に必要なメタデータは分かっているものとして，後者の方法を用いる．

6. 結論と今後の展望

本稿では，SSHFS を拡張し，SSHFS では不完全であった対遅延性を持たせるための手法を提案した．メタデータのキャッシュの更新を，ワークフローのジョブの投入と同期させることによって，必要最低限に抑えることができる．また，SSHFS の拡張であるため，既存のファイルシステムに対して，簡単に追加的に計算ノードを参加させることができる．一方，現在，ジョブが必要とするファイルを既知としてメタデータを計算ノードでアップデートしているが，このようなことが常に可能であるとはいえない．ではない．そのために，テスト実行によるログの取得よる，必要なファイルの予測が考えられる．また予想の方法としては，コマンドラインの文字列の利用も考えられる．予想が外れたものに関しては，SFTPD+まで問い合せる．

謝辞 本研究は特別推進研究「高度言語理解のための意味・知識処理の基盤技術に関する研究」の助成を受けて行われた．

参 考 文 献

1) Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G.Bruce Berriman, John Good, Anastasia Laity, JosephC. Jacob, and DanielS. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13(3):219–237, 2005.
2) Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2007 Eurosys Conference*, 2007.
3) Ioan Raicu, Ian T. Foster, Yong Zhao, Philip Little, Christopher M. Moretti, Amitabh Chaudhary, and Douglas Thain. The quest for scalable support of data-intensive workloads in distributed systems. In *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 207–216, New York, NY, USA, 2009. ACM.
4) Masahiro Tanaka and Osamu Tatebe. A parallel and distributed flexible workflow management tool for wide-area data intensive computing. In *Proceeding of HPDC2010*, 2010.