Gfarm Symposium 2015
2015年12月14日東京

Gfarmファイルシステムの概要と最新機能

建部修見
筑波大学

Gfarmファイルシステム

GRID DATA FARM

- 2000年より研究開発
  - 国際会議SC03、SC05、SC06で受賞
- オープンソース広域分散ファイルシステム
  - http://oss-tsukuba.org/software/gfarm/
  - 16,776 downloads since March, 2007
- NPO法人つくばOSS技術支援センターによるサポート
- 特徴
  - 広域で性能・容量がスケールアウト
    - データアクセス局所性、ファイル複製
  - 単一障害点なし
    - 複製数維持機能,、ホットスタンバイMDSサーバ
  - 無停止で拡張、更新可能
  - データ完全性を保証しサイレントデータ損傷も対応可

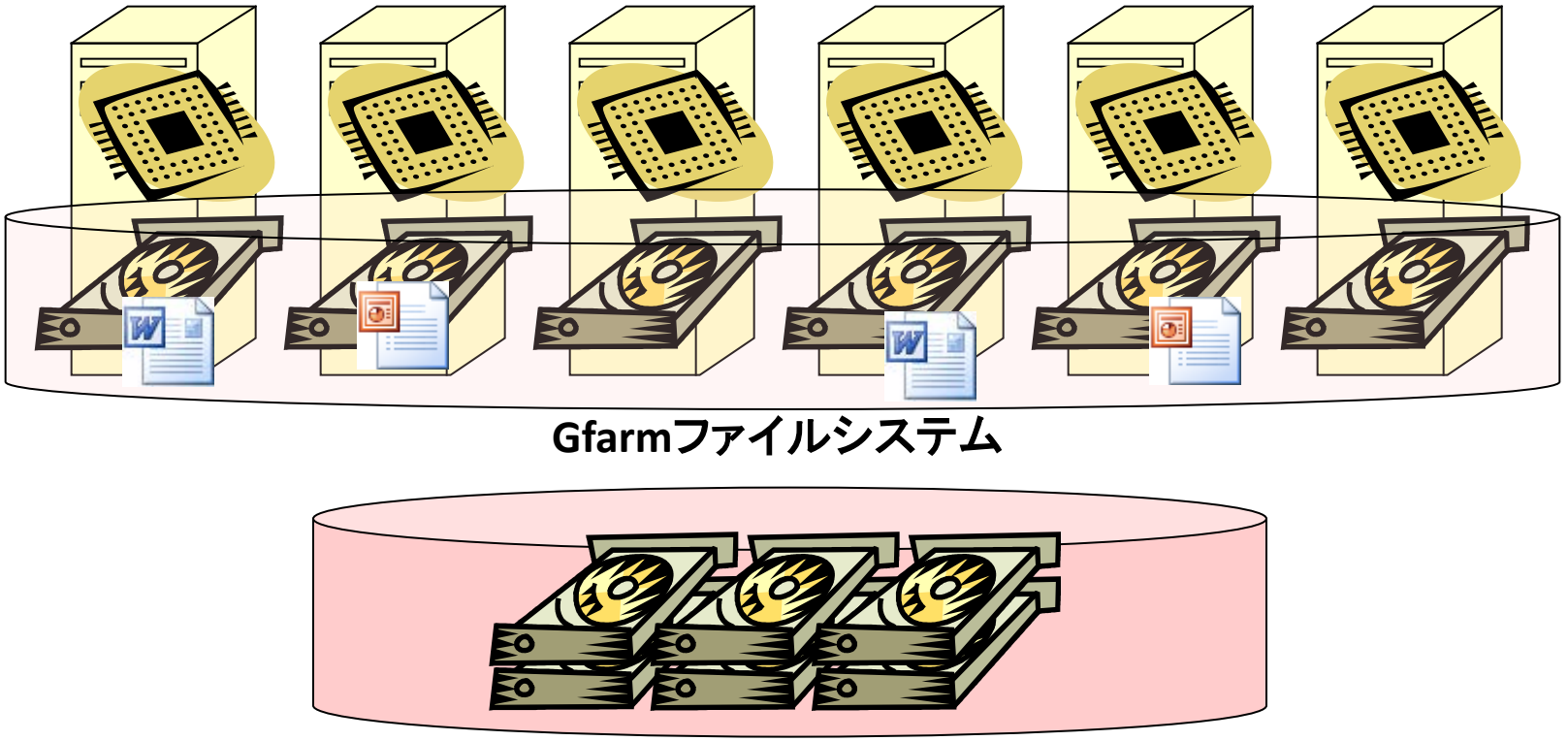
ossTsukuba
oss-tsukuba.org

Gfarmファイルシステム（2）

- JLDG（7PB、8拠点）、HPCI共用ストレージ（22.5PB、3拠点）で実運用
- 計算ノードのローカルディスクを利用するデータ解析
- Pwrakeワークフローシステム、MapReduce、MPI-IO、バッチキューイングシステム
  - データ局所性awareなプロセススケジューリング
  - ディスクキャッシュawareなプロセススケジューリング
  - データ局所性を高めるファイル複製作成

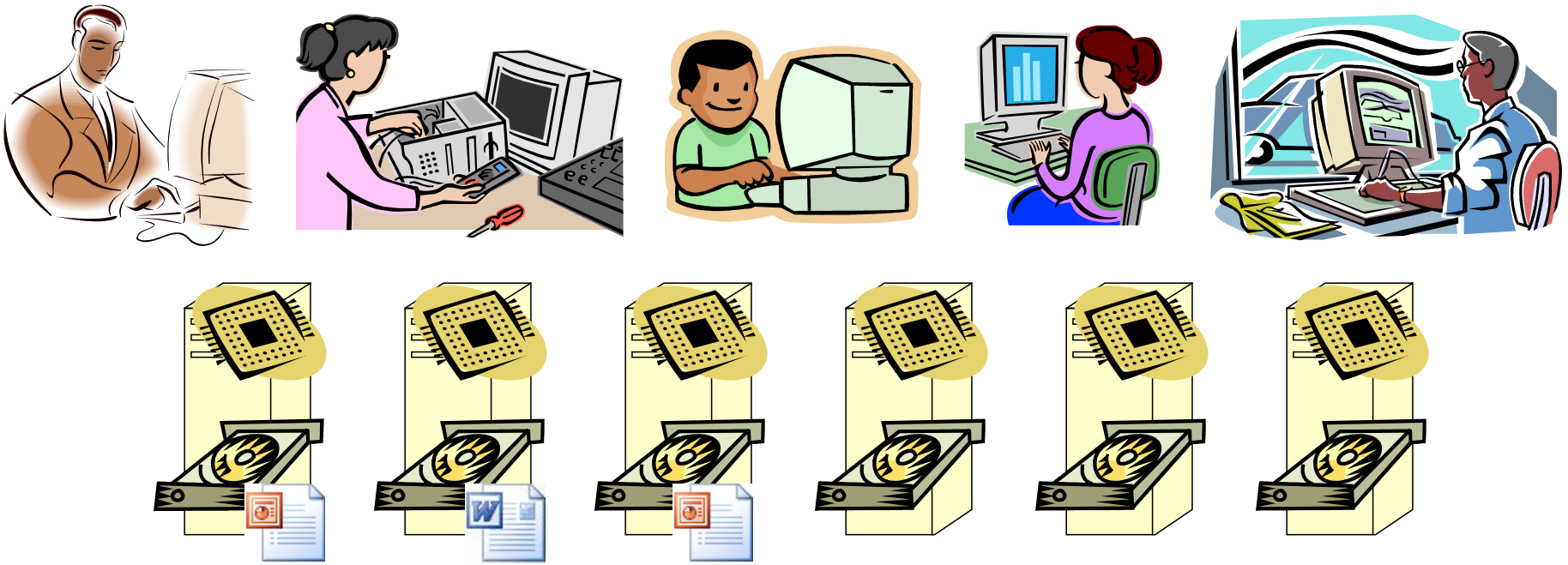
Gfarmファイルシステムの構成

- 一般的なPCのローカルディスクを束ねる
- ユーザには、共有ファイルシステムとしてみえる
- 複数のディスクに分散してデータを保持



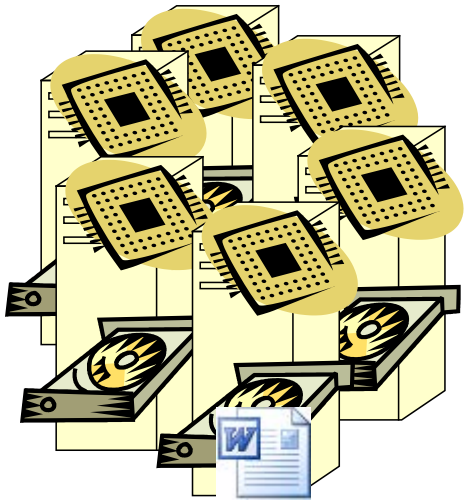
利用例(1):組織内の共有ファイルシステム

- ファイルシステムの容量を運用中に増加
- ファイル複製の数を運用中に増やして、ホットスポットの回避と、信頼性の向上

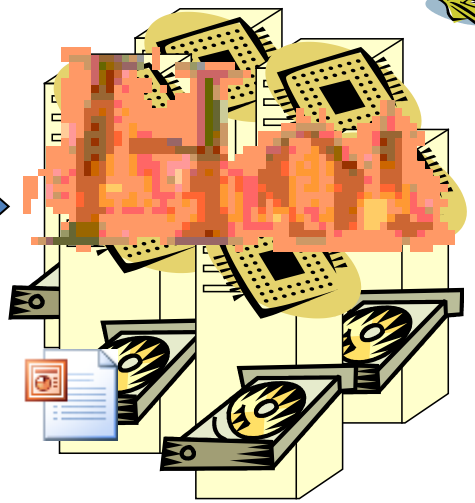


利用例(2):拠点間でのデータ共有

- ミラーリングを行い、それぞれの拠点に保持されたデータをアクセス
  - データが近くにあるため高速なアクセス
- 障害、災害時でも大丈夫



ミラーリング

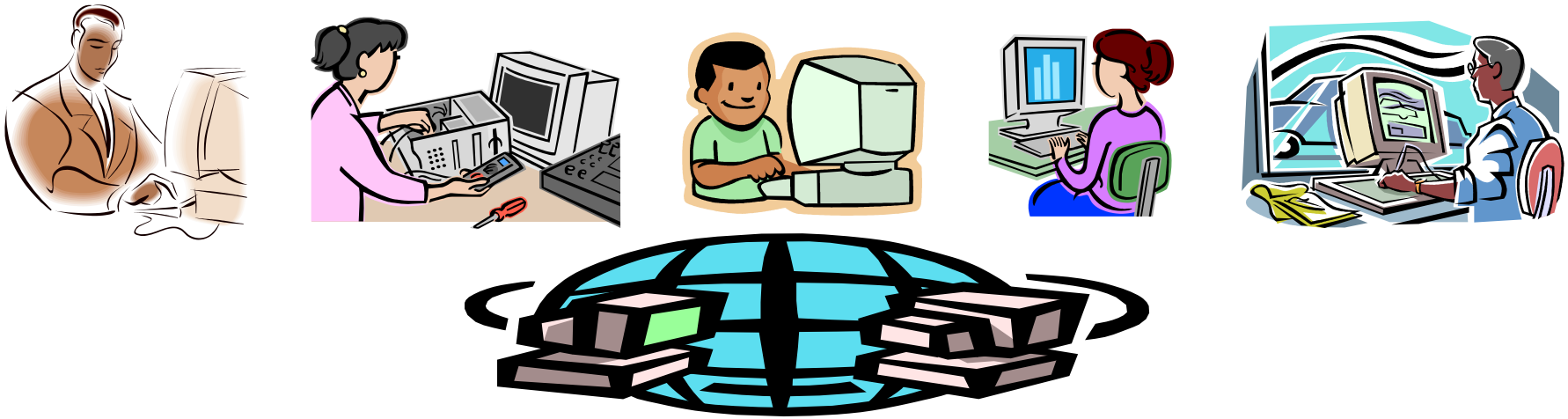


拠点AのPC群

拠点BのPC群

利用例(3):遠隔のファイル格納サービス

- ファイルの複製を地理的に離れた場所に保持することにより、高信頼なサービスを実現

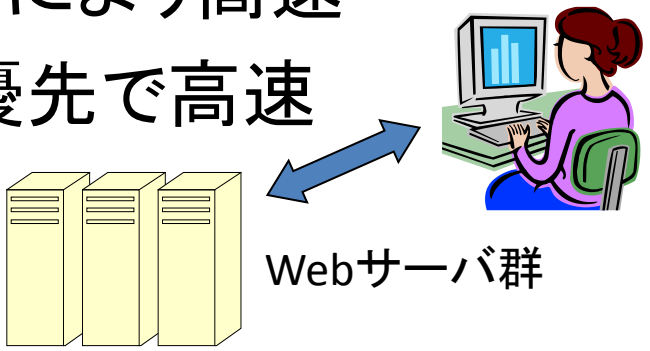


データセンターA

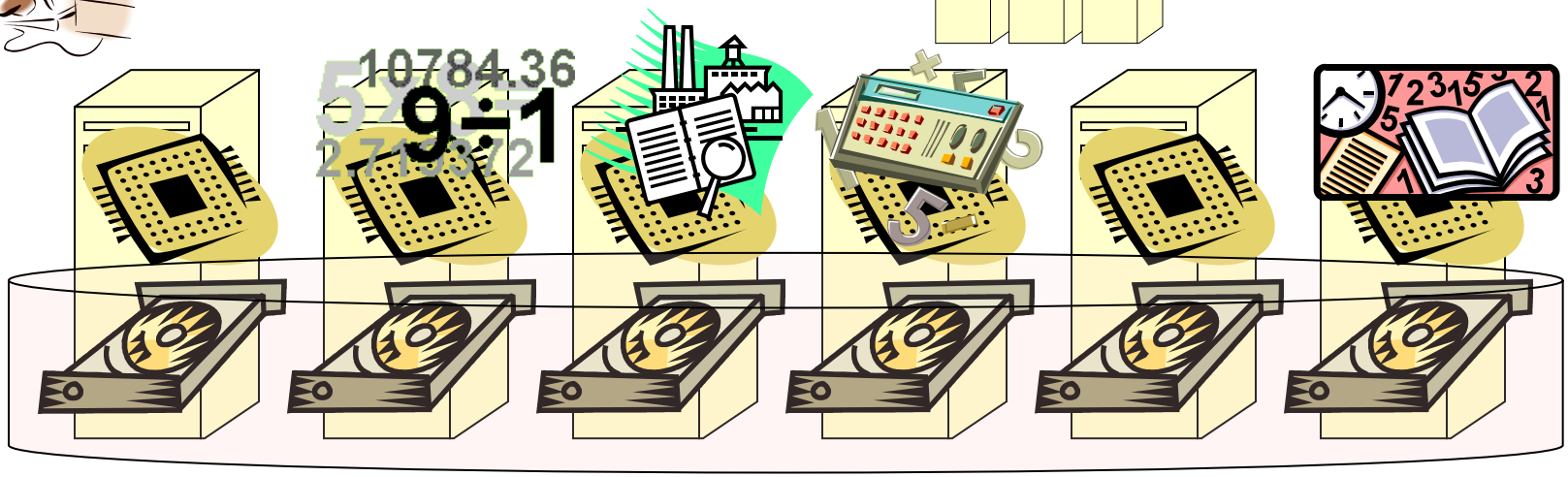
データセンターB

利用例(4):大規模データ処理

- 高性能共有ファイルシステムとして、複数のPCで分散並列処理
  - 読込みはデータの分散保持により高速
  - 書込みはローカルディスク優先で高速



Webサーバ群



最新機能・状況紹介

Gfarm 2.6の主な新機能

- ファイルシステムノードグループによる複製配置指定機能
- クライアント透明なフェイルオーバ
- End-to-endのデータ完全性
- gfs_pio_sendfileによるgfpcopyの高速化
- CentOS 7対応
- Linuxカーネルモジュール

ファイルシステムノードグループによる複製配置指定機能

- ファイルシステムノードグループの指定

  % **gfhostgroup -s** *hostname groupname*

- グループごとの複製数（複製属性）指定

  % **gfncopy -S** *GroupA:2,GroupB:1 directory*

複製数指定との併用

- 複製数指定

  % **gfncopy -s** *NUM directory*

- 複製属性優先
- 複製数の方が複製属性よりも多い場合、任意ノードに指定複製数作成される

余剰複製削除 [Gfarm 2.6.5]

- ファイルシステムノードの長期停止、復帰時などで生じる余剰複製を自動的に削除
- デフォルトで有効

- 自動削除機能の制御
  % **gfrepcheck** remove enable
  % **gfrepcheck** remove disable
- 起動時に自動削除機能を停止
  replica_check_remove disable

クライアント透明なフェイルオーバ

- 運用無停止のGfmdフェイルオーバが可能に
- Gfarm APIは自動的に再接続し実行を継続、クライアントにはエラーは返らない
- ファイル書込中のフェイルオーバに関する競合の解消
  - フェイルオーバによる書込競合で、あとでクローズしたファイルがlost+foundへ移動する場合あり

運用中のソフトウェア更新

- ユーザ、アプリに影響を与えないでソフトウェア更新可能
  1. スレーブgfmdの更新
  2. マスターgfmdを停止、フェイルオーバ
  3. 旧マスターgfmdを更新
  4. gfsdを順次更新
  5. クライアントを順次更新

データ完全性（1）

- サイレントデータ障害の検知
- digestを書込時に計算しメタデータに保存
- 逐次書込（読込）時にgfsdでdigest検査
  - 破損ファイルは読込時にEIO (checksum error)を返し、読込失敗。lost+foundへ移動させ自動修復
- ファイル複製作成時のdigest検査
- Write verifyによるdigest検査 [Coming soon]
- クライアントからのEnd-to-endのデータ完全性

データ完全性（２）

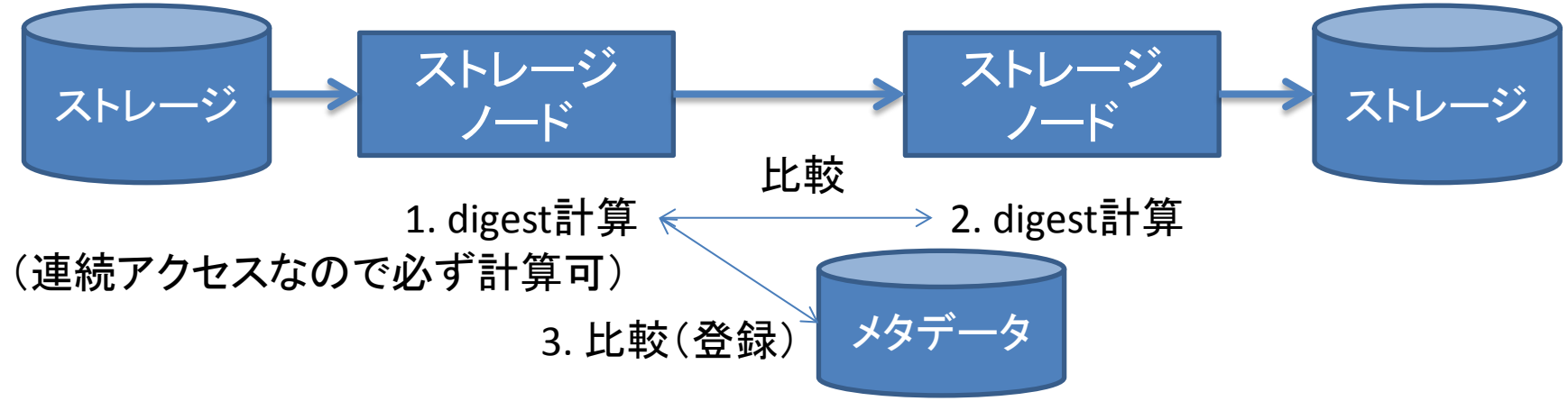
- データ完全性の保証
  digest *digest_type*　　// gfmd.conf
  md5, sha1, sha256, …をサポート
- End-to-endデータ完全性の保証
  client_digest_check enable　　// gfarm2.conf

% **gfcksum** *file*
  – ファイルのdigestを表示
% **gfcksum** *-c [-h host] file*
  – （hostに格納されている）ファイルのdigestを計算し確認

Gfarmによるデータ完全性保証（1）

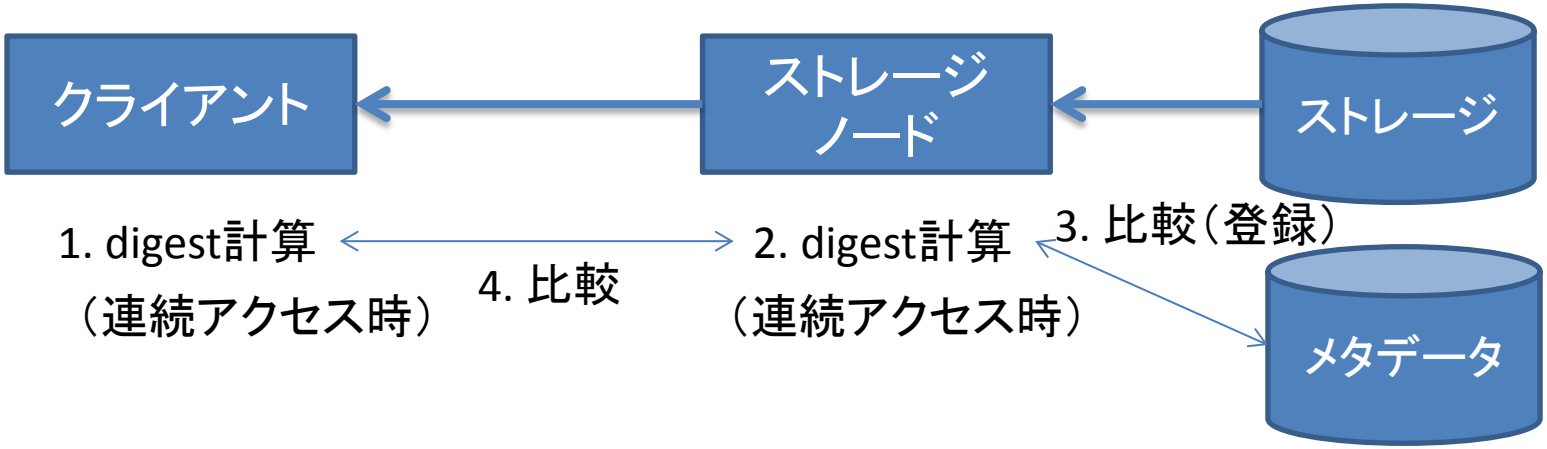
- データ書き込み時

- ファイル複製作成時



Gfarmによるデータ完全性保証（2）

- データ読み込み時



チェックサムが異なる場合、全体ファイル読み込み時にreadがI/Oエラーを返し、lost+foundへ移動

⇨ データ化けファイルの読み込みを防ぐ

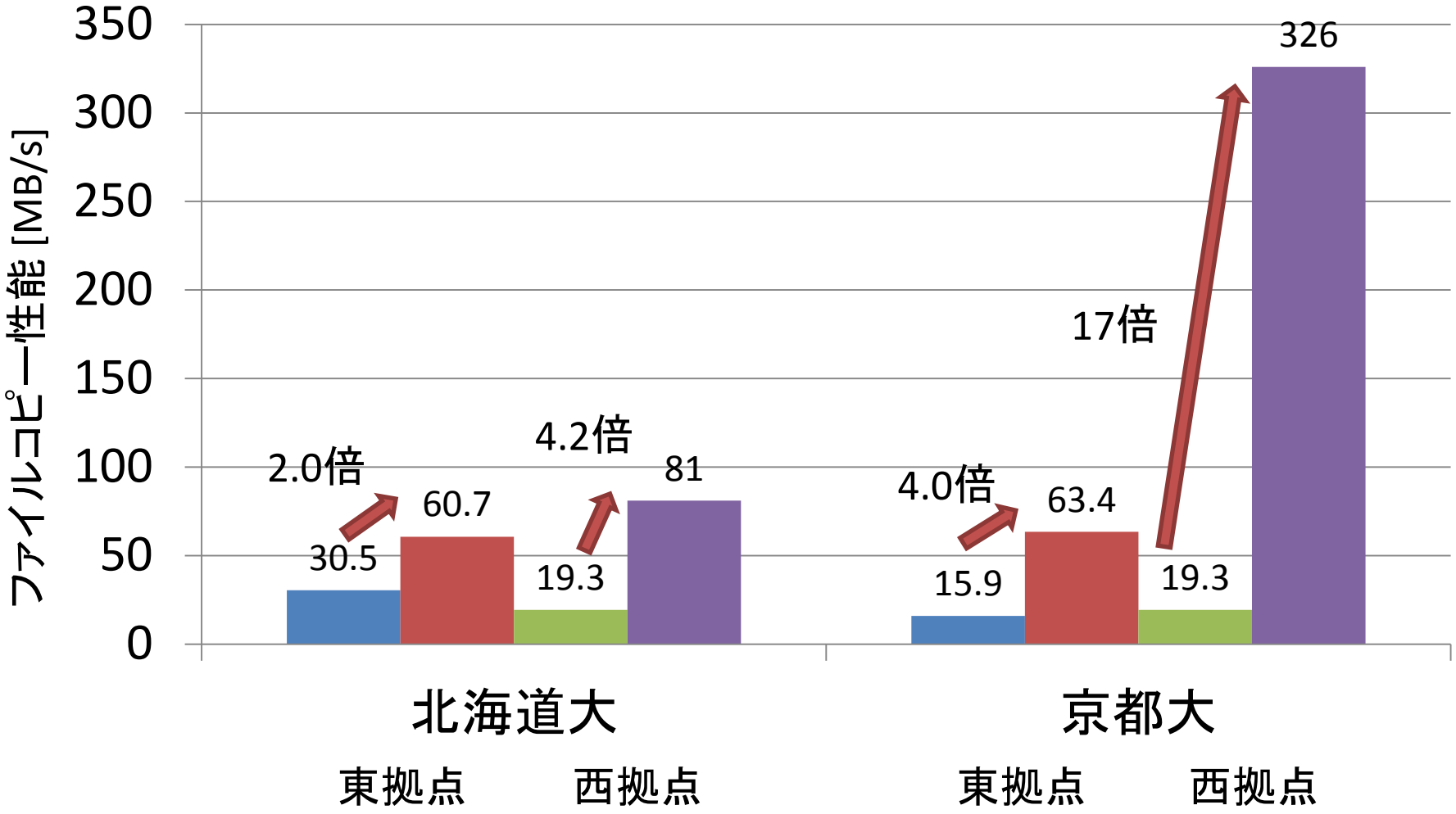
ファイル複製による自動修復

gfpcopyの高速化

- gfs_pio_sendfile/recvfile
  - BULKWRITE/BULKREADプロトコル新設
  - RTT大の環境での性能低下を防ぐ
- 1ストリームあたりの転送速度の大幅向上

HPCI共用ファイルシステムにおける1並列コピー性能

左：2.5.8系，右：2.6系

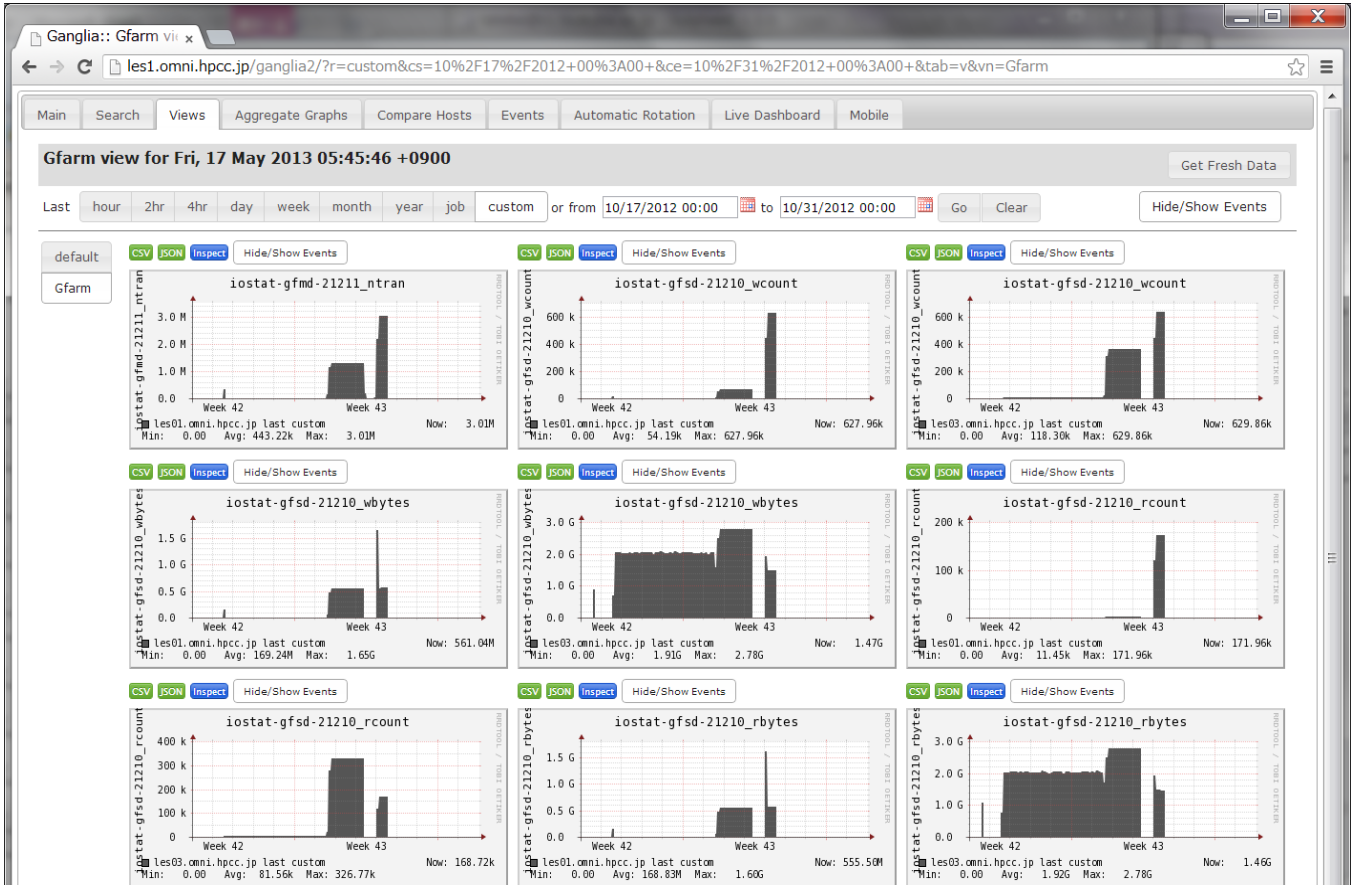


Gfarmファイルシステム運用監視

- Zabbixプラグインにより、各サーバを監視

性能モニタリング [Gfarm 2.5.8]

- Gangliaプラグインによる IOPS、バンド幅のリアルタイム性能モニタリング



Automount [Gfarm2fs 1.2.9.2]

- ディレクトリをアクセスするとGfarmファイルシステムをautomount

```
% ls -a /usr/users

. ..

% ls -a /usr/users/tatebe

. .. .bash_logout .bash_profile .bashrc .emacs
```

- Gfarmのディレクトリを**ホームディレクトリとして**automount可能！

Gfarm-Samba VFS plugin version 1.0.0

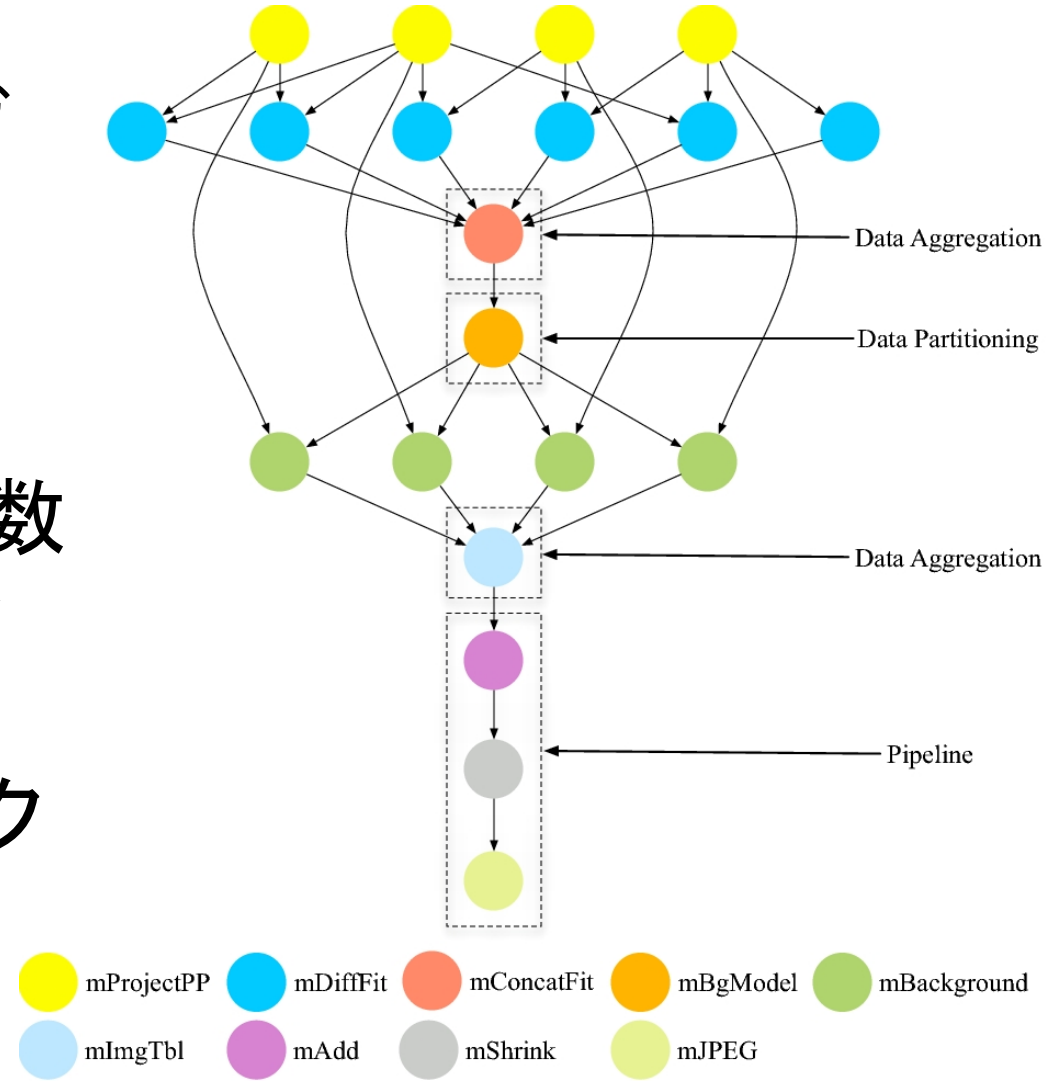
- Gfarm2fsを利用しなくてもSambaからGfarmを利用するためのモジュール
- Samba 3.6系に対応

```
[gfarm]
comment = GfarmFS
path = /
vfs objects = gfarm
writeable = yes
browseable = yes
```

分散並列処理
（ビッグデータ処理）

ワークフロー

- データインテンシブサイエンスアプリケーションでよく利用される
- 大規模データを複数のアプリケーションで順次処理
- 依存性のあるタスクの集合

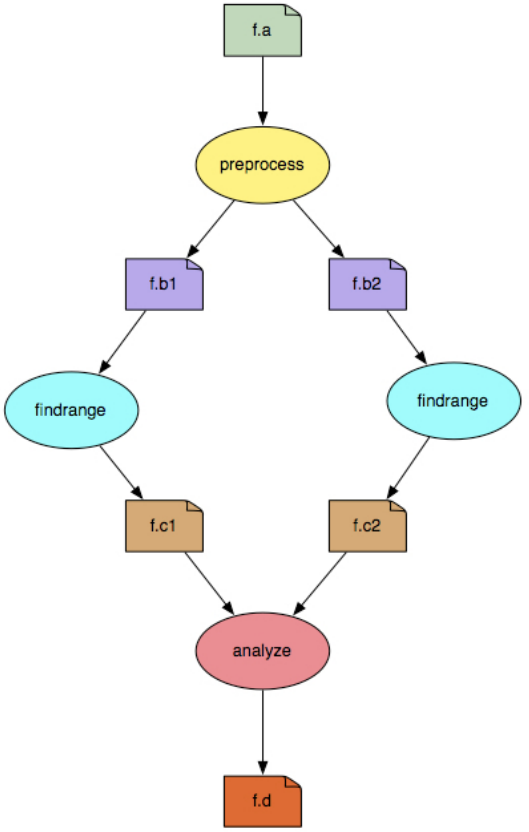


https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

ワークフローの例（ダイアモンド）

Rakefile: (14行)

```
file "f.b1" => "f.a" do
  sh "preprocess -a preprocess -T60 -i f.a -o f.b1 f.b2"
end

file "f.b2" => "f.b1"

file "f.c1" => "f.b1" do
  sh "findrange -a findrange -T60 -i f.b1 -o f.c1"
end

file "f.c2" => "f.b2" do
  sh "findrange -a findrange -T60 -i f.b2 -o f.c2"
end

file "f.d" => ["f.c1","f.c2"] do
  sh "analyze -a analyze -T60 -i f.c1 f.c2 -o f.d"
end

task :default => "f.d“
```



From http://pegasus.isi.edu/wms/docs/3.0/quickstart.php

DAX （41行）

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated: 2010-11-22T22:55:08Z -->
<adag xmlns="http://pegasus.isi.edu/schema/DAX"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://pegasus.isi.edu/schema/DAX http://pegasus.isi.edu/schema/dax-3.2.xsd"
      version="3.2" name="diamond" index="0" count="1">
  <!-- part 2: definition of all jobs (at least one) -->
  <job namespace="diamond" name="preprocess" version="2.0" id="ID000001">
    <argument>-a preprocess -T60 -i <file name="f.a" /> -o <file name="f.b1" /> <file name="f.b2" /></argument>
    <uses name="f.b2" link="output" register="false" transfer="false" />
    <uses name="f.b1" link="output" register="false" transfer="false" />
    <uses name="f.a" link="input" />
  </job>
  <job namespace="diamond" name="findrange" version="2.0" id="ID000002">
    <argument>-a findrange -T60 -i <file name="f.b1" /> -o <file name="f.c1" /></argument>
    <uses name="f.b1" link="input" register="false" transfer="false" />
    <uses name="f.c1" link="output" register="false" transfer="false" />
  </job>
  <job namespace="diamond" name="findrange" version="2.0" id="ID000003">
    <argument>-a findrange -T60 -i <file name="f.b2" /> -o <file name="f.c2" /></argument>
    <uses name="f.c2" link="output" register="false" transfer="false" />
    <uses name="f.b2" link="input" register="false" transfer="false" />
  </job>
  <job namespace="diamond" name="analyze" version="2.0" id="ID000004">
    <argument>-a analyze -T60 -i <file name="f.c1" /> <file name="f.c2" /> -o <file name="f.d" /></argument>
    <uses name="f.c2" link="input" register="false" transfer="false" />
    <uses name="f.d" link="output" register="false" transfer="true" />
    <uses name="f.c1" link="input" register="false" transfer="false" />
  </job>
  <!-- part 3: list of control-flow dependencies -->
  <child ref="ID000002">
    <parent ref="ID000001" />
  </child>
  <child ref="ID000003">
    <parent ref="ID000001" />
  </child>
  <child ref="ID000004">
    <parent ref="ID000002" />
    <parent ref="ID000003" />
  </child>
</adag>
```

DAX generator (Java) 95行

```
import edu.isi.pegasus.planner.dax.*;

public class BlackDiamondDAX {

    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Usage: java ADAG <site_handle> <pegasus_location> <filename.dax>");
            System.exit(1);
        }
        try {
            Diamond(args[0], args[1]).writeToFile(args[2]);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static ADAG Diamond(String site_handle, String pegasus_location) throws Exception {

        java.io.File cwdFile = new java.io.File (".");
        String cwd = cwdFile.getCanonicalPath();

        ADAG dax = new ADAG("blackdiamond");

        File fa = new File("f.a");
        fa.addPhysicalFile("file://" + cwd + "/f.a", "local");
        dax.addFile(fa);

        File fb1 = new File("f.b1");
        File fb2 = new File("f.b2");
        File fc1 = new File("f.c1");
        File fc2 = new File("f.c2");
        File fd = new File("f.d");
        fd.setRegister(true);

        Executable preprocess = new Executable("pegasus", "preprocess", "4.0");
        preprocess.setArchitecture(Executable.ARCH.X86).setOS(Executable.OS.LINUX);
        preprocess.setInstalled(true);
        preprocess.addPhysicalFile("file://" + pegasus_location + "/bin/keg", site_handle);

        Executable findrange = new Executable("pegasus", "findrange", "4.0");
        findrange.setArchitecture(Executable.ARCH.X86).setOS(Executable.OS.LINUX);
        findrange.setInstalled(true);
        findrange.addPhysicalFile("file://" + pegasus_location + "/bin/keg", site_handle);

        Executable analyze = new Executable("pegasus", "analyze", "4.0");
        analyze.setArchitecture(Executable.ARCH.X86).setOS(Executable.OS.LINUX);
        analyze.setInstalled(true);
        analyze.addPhysicalFile("file://" + pegasus_location + "/bin/keg", site_handle);

        dax.addExecutable(preprocess).addExecutable(findrange).addExecutable(analyze);

        // Add a preprocess job
        Job j1 = new Job("j1", "pegasus", "preprocess", "4.0");
        j1.addArgument("-a preprocess -T 60 -i ").addArgument(fa);
        j1.addArgument("-o ").addArgument(fb1);
        j1.addArgument(" ").addArgument(fb2);
        j1.uses(fa, File.LINK.INPUT);
        j1.uses(fb1, File.LINK.OUTPUT);
        j1.uses(fb2, File.LINK.OUTPUT);
        dax.addJob(j1);

        // Add left Findrange job
        Job j2 = new Job("j2", "pegasus", "findrange", "4.0");
        j2.addArgument("-a findrange -T 60 -i ").addArgument(fb1);
        j2.addArgument("-o ").addArgument(fc1);
        j2.uses(fb1, File.LINK.INPUT);
        j2.uses(fc1, File.LINK.OUTPUT);
        dax.addJob(j2);

        // Add right Findrange job
        Job j3 = new Job("j3", "pegasus", "findrange", "4.0");
        j3.addArgument("-a findrange -T 60 -i ").addArgument(fb2);
        j3.addArgument("-o ").addArgument(fc2);
        j3.uses(fb2, File.LINK.INPUT);
        j3.uses(fc2, File.LINK.OUTPUT);
        dax.addJob(j3);

        // Add analyze job
        Job j4 = new Job("j4", "pegasus", "analyze", "4.0");
        j4.addArgument("-a analyze -T 60 -i ").addArgument(fc1);
        j4.addArgument(" ").addArgument(fc2);
        j4.addArgument("-o ").addArgument(fd);
        j4.uses(fc1, File.LINK.INPUT);
        j4.uses(fc2, File.LINK.INPUT);
        j4.uses(fd, File.LINK.OUTPUT);
        dax.addJob(j4);

        dax.addDependency("j1", "j2");
        dax.addDependency("j1", "j3");
        dax.addDependency("j2", "j4");
        dax.addDependency("j3", "j4");
        return dax;
    }
}
```

ダイアモンドワークフローの行数比較

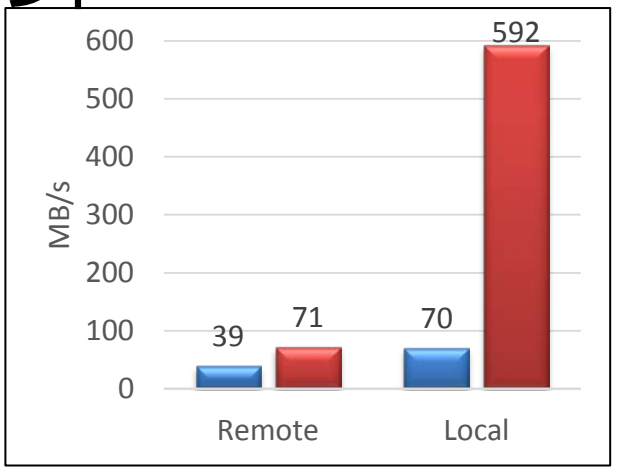
- Rakefile 14
- DAX 41
- Perl 73
- Python 76
- Java 95

Pwrakeワークフローシステム [Tanaka, HPDC 2010]

- **Rake**を拡張（Rakeと共存可能）
  - *Rakefile*で表現されるワークフローをクラスタの複数ファイルで並列実行

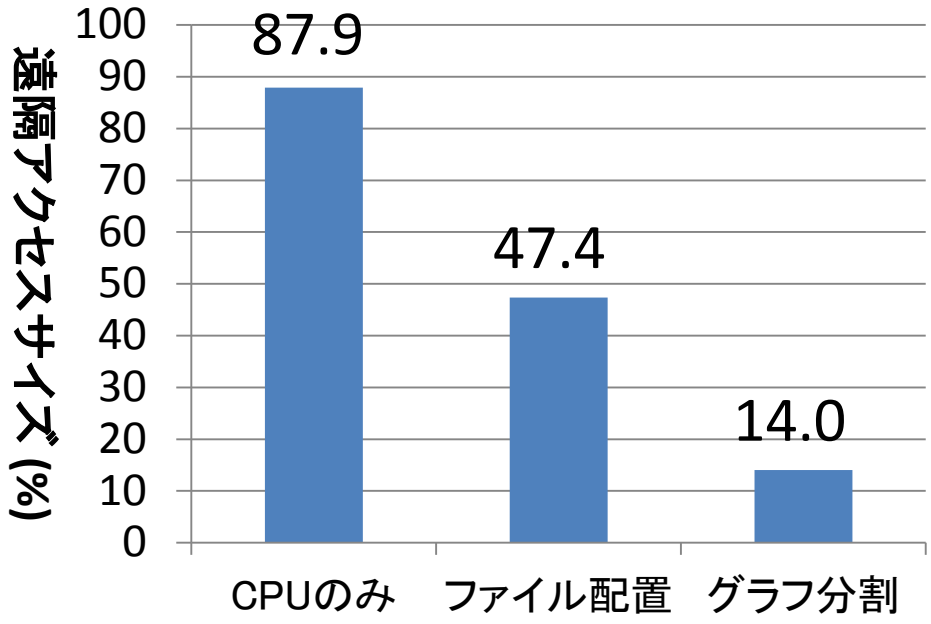
  % **pwrake** --hostfile=hosts
- http://github.com/masa16/pwrake/
- Gfarmファイルシステムのサポート
  - Gfarmファイルシステムを自動マウント
  - データ移動の最小化 [CCGrid 2012]
  - ディスクキャッシュの効率利用 [Cluster 2014]

ローカル
ディスク
キャッシュ

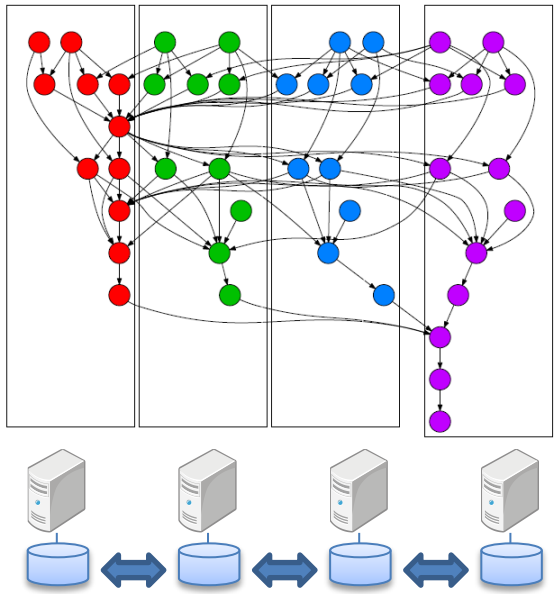


データ移動の最小化

- データ移動を最小化し、スケーラブルなI/O性能を実現



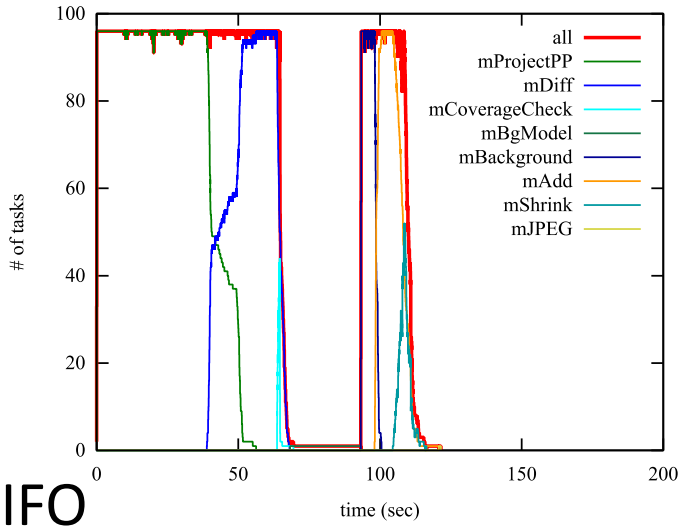
多制約グラフ分割による スケジューリング [CCGrid 2012]



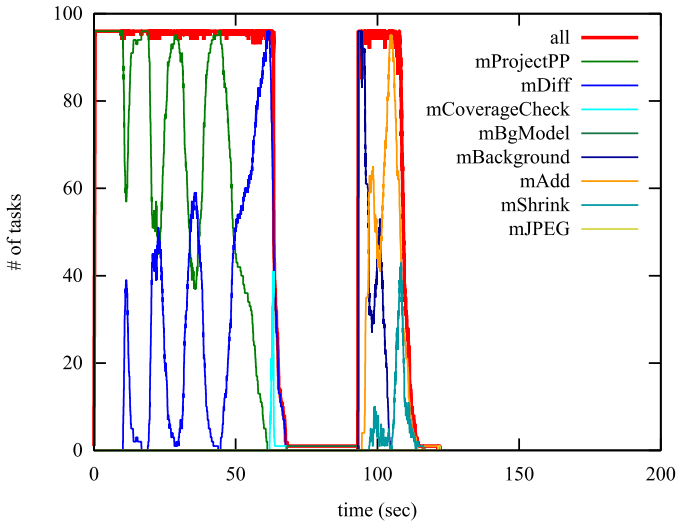
データ移動 14% に削減
実行時間 31% 削減

ディスクキャッシュの効率利用

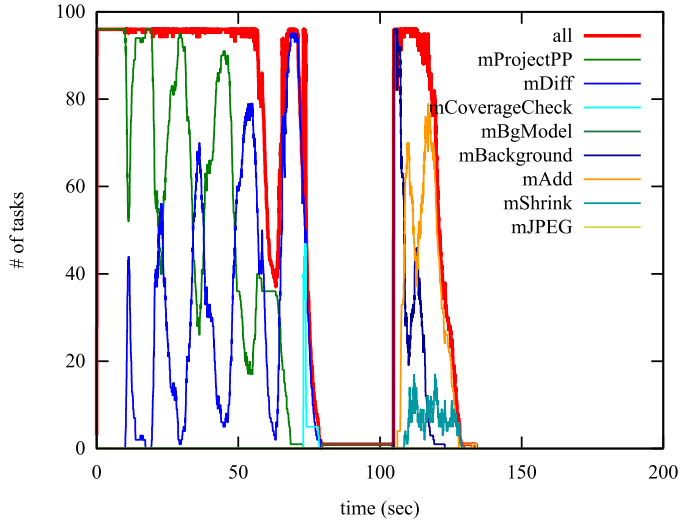
FIFO



LIFO+HRF [Cluster 2014]



LIFO



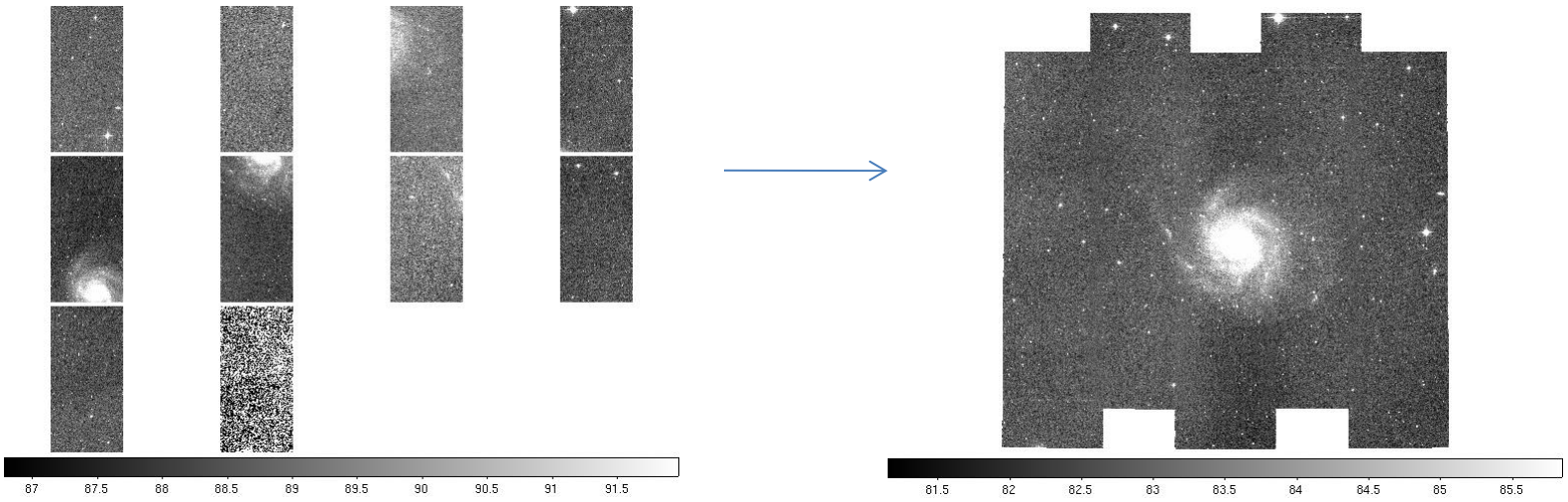
- LIFOはディスクキャッシュを効率的に用いることが可能であるが、末尾タスク問題がある
- 提案手法（LIFO+HRF）はディスクキャッシュを効率的に利用し末尾タスク問題を解決

Pwrake 2.0.0

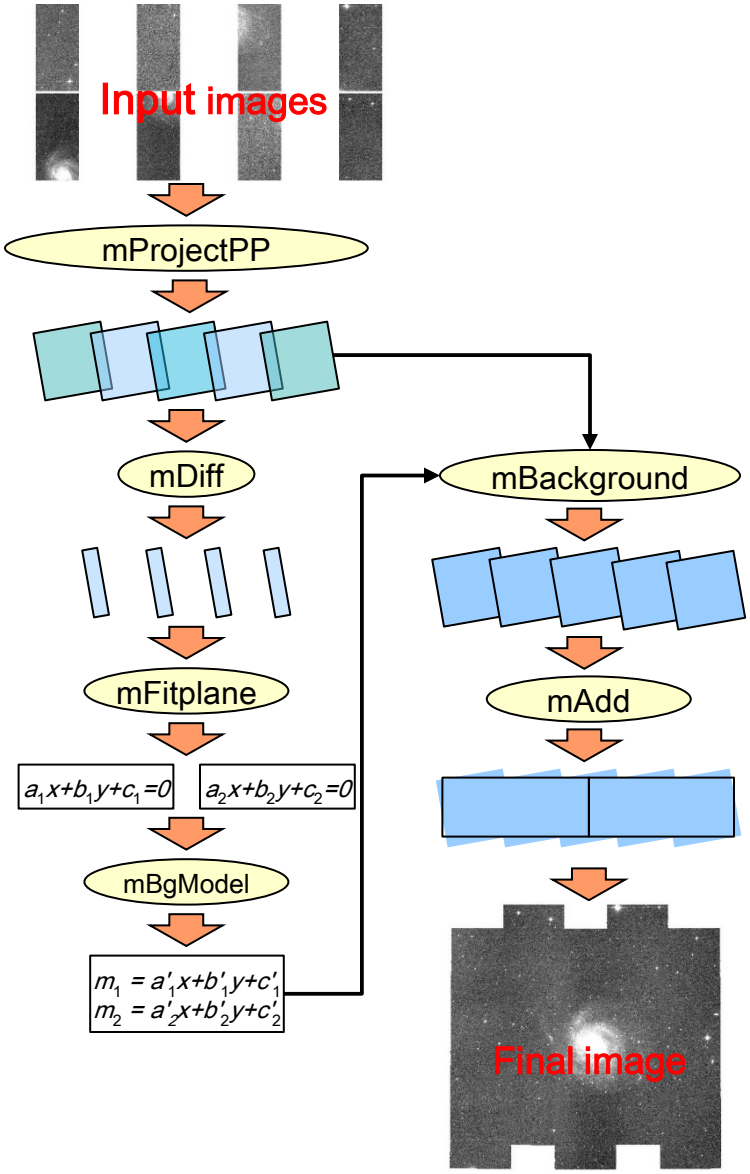
- 2015/12/11にリリース
- 新機能
  - タスクプロパティによる、タスク毎のコア数、ホスト指定、スチール不可などの設定
  - タスク異常終了時の振る舞いの指定
  - プロローグ機能
  - ハートビート間隔の設定
- 高性能化、大規模化対応
  - Fiberによる低オーバヘッド化、省メモリ化
  - コネクション数をワーカ数からノード数に削減

ワークフローの例

- Montage
  - 複数の画像からモザイク画像を生成
  - http://montage.ipac.caltech.edu/

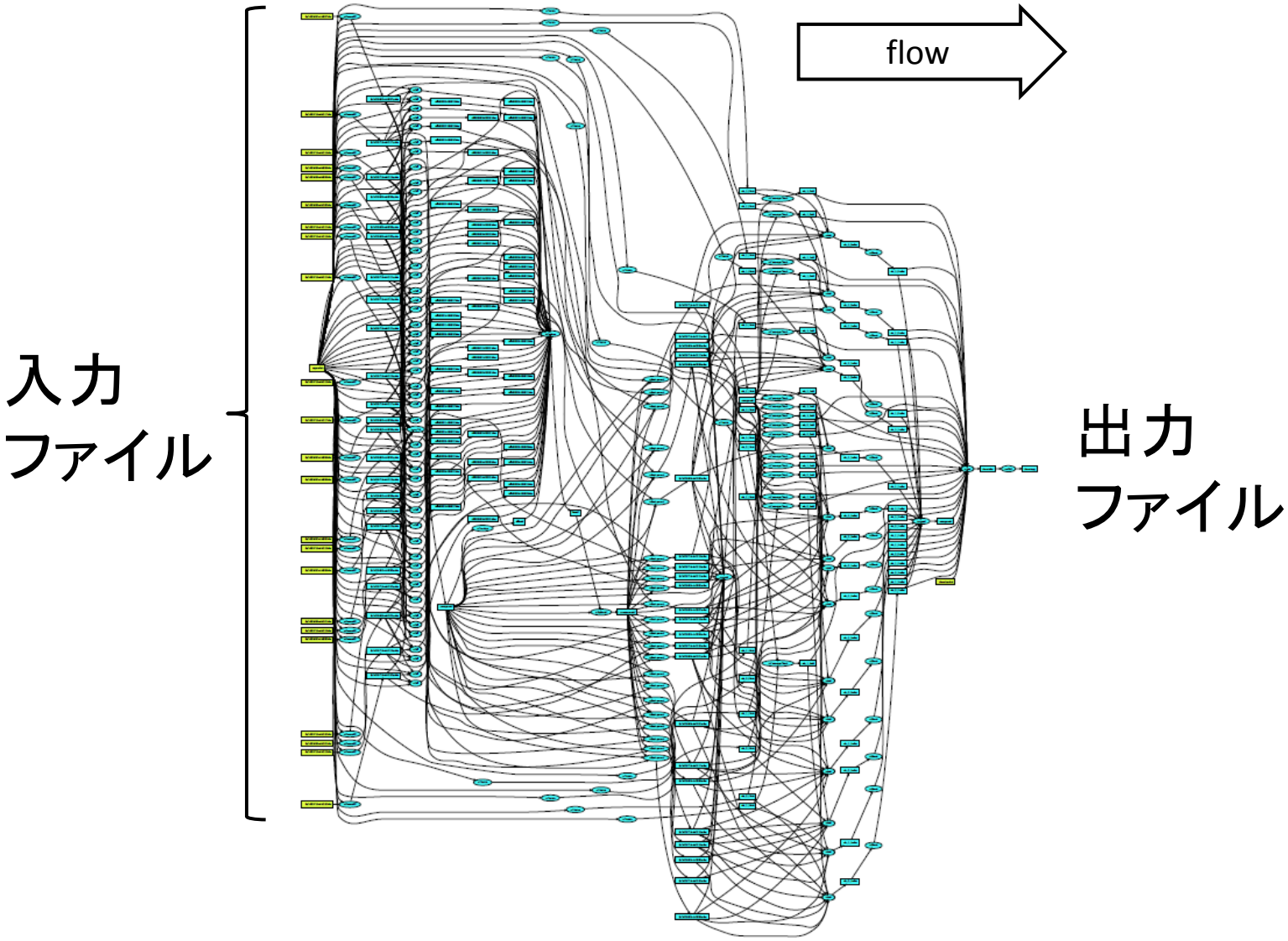


Montageワークフロー（1）



- 処理内容
  - 座標変換
  - 明るさ補正
  - 足し合わせ

Montageワークフロー（2）

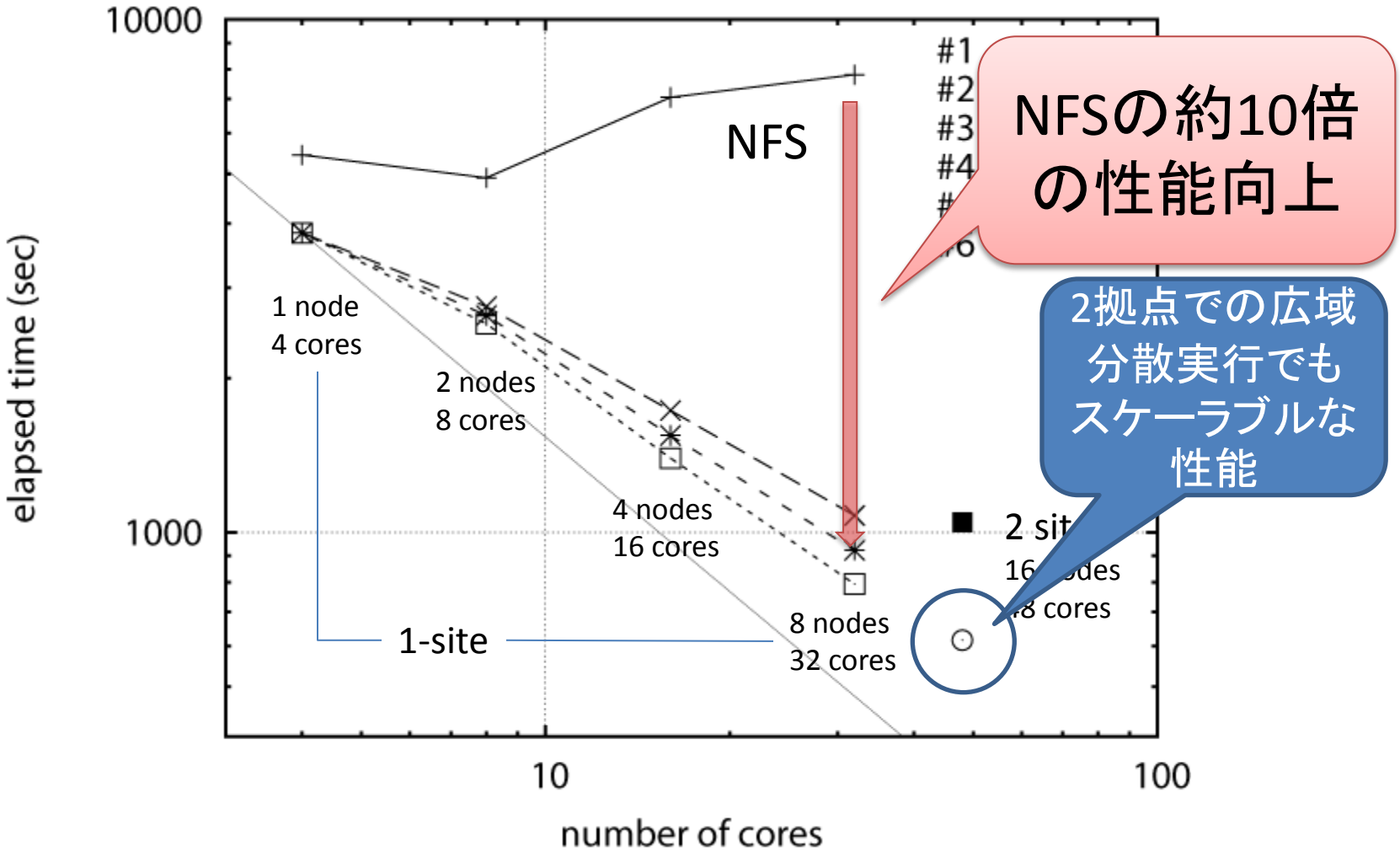


Montageワークフロー(Rakefile)

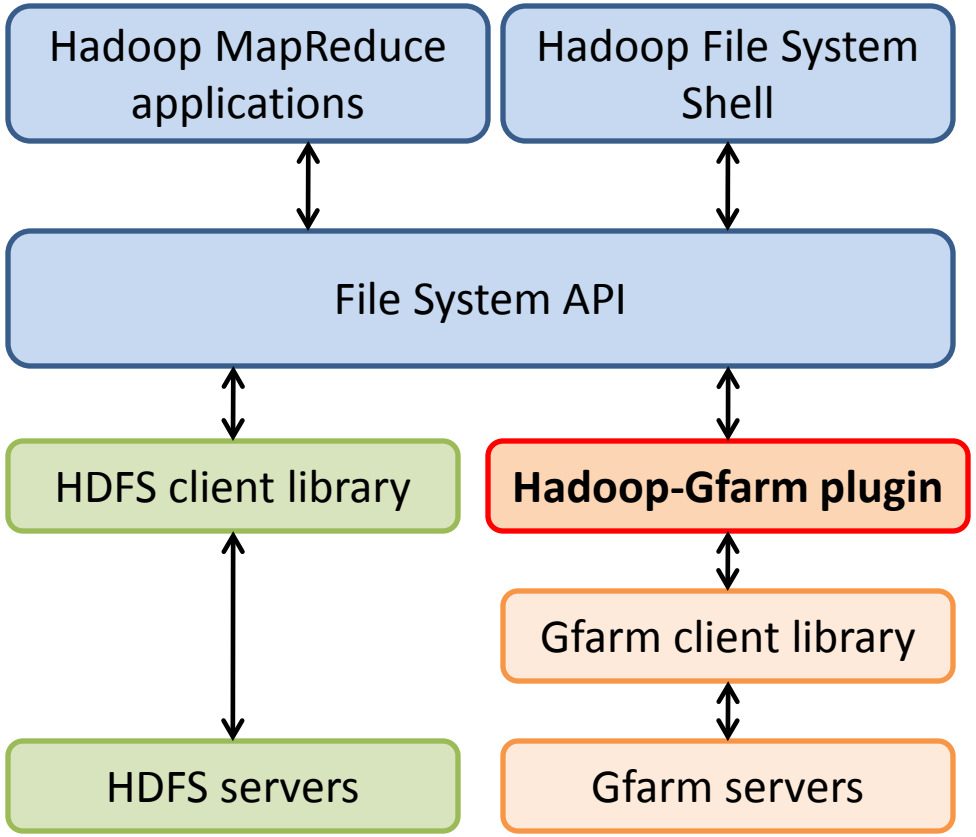
```
require 'montage_tools'
require 'rake/clean'

task( :default => "mosaic.jpg" )

dir=Dir.glob(Dir.pwd+"/Montage_*/bin")
ENV['PATH'] = "#{dir.last}:"+ENV['PATH']

INPUT_DIR = ENV["INPUT_DIR"] || "m101/rawdir"
REGION_HDR = "m101/template.hdr"

### Projection
SRC_FITS = FileList["#{INPUT_DIR}/*.fits"]

P_IMGTBL = []
PRJ_FITS=[]
SRC_FITS.each do |src|
  desc prj = src.sub( %r|^(.*?)([^/]+).fits|, 'p/¥2.p.fits' )
  file( prj => [src,REGION_HDR] ) do |t|
    sh "mProjectPP #{src} #{prj} #{REGION_HDR}" do |*x| end
    Montage.collect_imgtbl( t, P_IMGTBL )
  end
  PRJ_FITS << prj
end

file( "pimages.tbl" => PRJ_FITS ) do
  Montage.put_imgtbl( P_IMGTBL, "p", "pimages.tbl" )
end

### dif & fit
file( "diffs.tbl" => "pimages.tbl" ) do
  sh "mOverlaps pimages.tbl diffs.tbl"
end

file( "fitfits.tbl" => "diffs.tbl" ) do
  DIFF_FITS=[]
  FIT_TXT=[]
  FIT_TBL=[]
  diffs = Montage.read_overlap_tbl("diffs.tbl")
  diffs.each do |c|
    p1 = "p/"+c[2]
    p2 = "p/"+c[3]
    DIFF_FITS << dif_fit = "d/"+c[4]
    file( dif_fit => [c[2],c[3],REGION_HDR,"pimages.tbl"] ) do |t|
      x1,x2,rh = t.prerequisites
      sh "mDiff #{x1} #{x2} #{t.name} #{REGION_HDR}"
      r = `mFitplane #{t.name}`
      puts "sh 'mFitplane #{t.name}' => #{r}"
      FIT_TBL << [c[0..1],r]
    end
  end

  task( :dif_fit_exec => DIFF_FITS ) do
    Montage.write_fitfits_tbl(FIT_TBL, "fitfits.tbl")
  end.invoke
end

### background-model
file( "corrections.tbl" => ["fitfits.tbl", "pimages.tbl"] ) do
  sh "mBgModel pimages.tbl fitfits.tbl corrections.tbl"
end

### background correction
C_IMGTBL=[]

file( "cimages.tbl" => ["corrections.tbl","pimages.tbl"] ) do
  pfiles = FileList["p/*.p.fits"]
  cfiles = pfiles.map do |s|
    src = s.sub(%r{p/(.*)¥.p¥.fits}, '¥1.p.fits')
    desc dst = src.sub(%r{(.*)¥.p¥.fits}, 'c/¥1.c.fits')
    file( dst => ["p/#{src}","corrections.tbl","pimages.tbl"] ) do |t|
      sh "(cd p; mBackground -t
            #{src} ../#{dst} ../pimages.tbl ../corrections.tbl)"
      Montage.collect_imgtbl( t, C_IMGTBL )
    end
    dst
  end

  task( :cimages_tbl_exec => cfiles ) do
    Montage.put_imgtbl( C_IMGTBL, "c", "cimages.tbl" )
  end.invoke
end

file( "mosaic.fits" => ["cimages.tbl", REGION_HDR] ) {|t|
  sh "mAdd -p c #{t.prerequisites.join(' ')} #{t.name}"
}

file( "mosaic.jpg" => "mosaic.fits" ) {|t|
  sh "mJPEG -ct 0 -gray #{t.prerequisites[0]} -1.5s 60s gaussian -out
            #{t.name}"
}

mkdir_p "p"
mkdir_p "d"
mkdir_p "c"

CLEAN.include %w[ p d c  ]
CLEAN.include %w[ mosaic.fits mosaic_area.fits mosaic.jpg ]
CLEAN.include %w[ fittxt.tbl fitfits.tbl ]
CLEAN.include %w[ rimages_all.tbl rimages.tbl ]
CLEAN.include %w[ pimages.tbl cimages.tbl simages.tbl ]
CLEAN.include %w[ diffs.tbl corrections.tbl ]
```

81行

Montageワークフローの性能評価

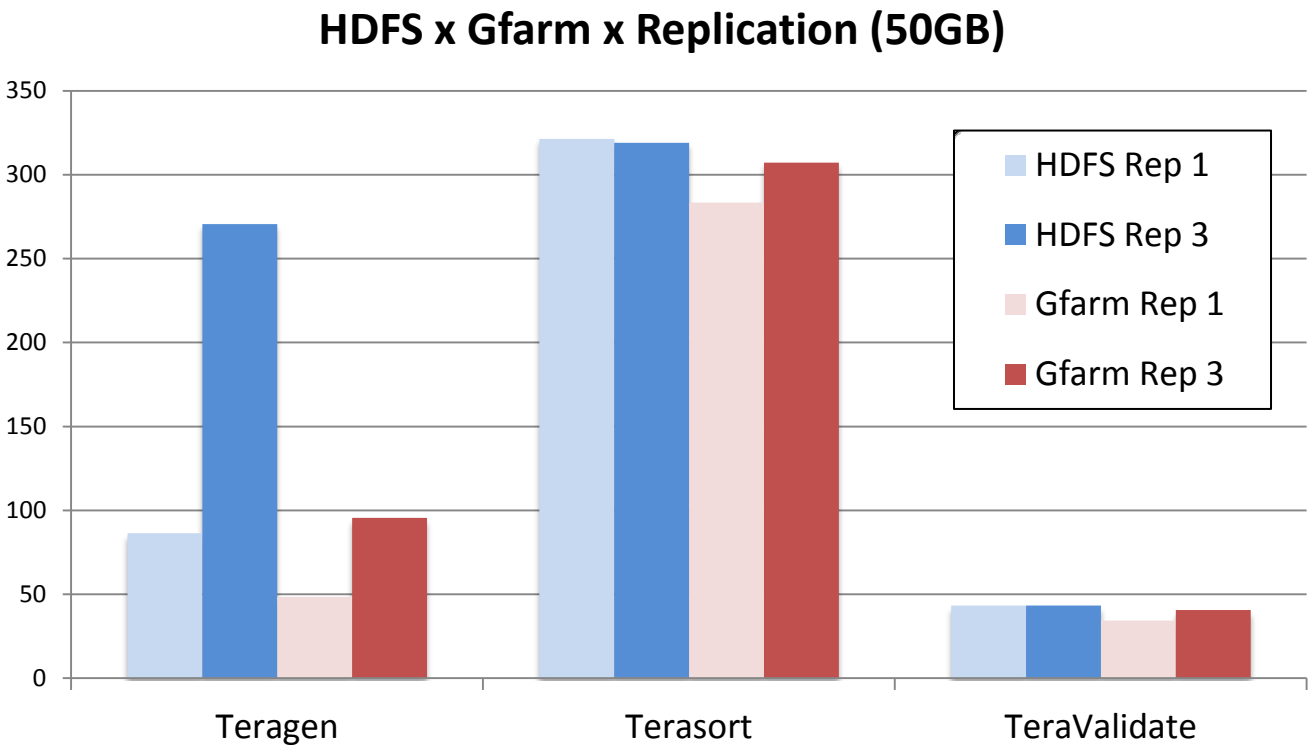


Gfarm Hadoopプラグイン [Mikami, Grid 2011]



- Hadoop から**Gfarm URL**でGfarmへアクセスするためのプラグイン
- http://sf.net/projects/gfarm/
- JNIによりHadoopからGfarmのクライアントライブラリを呼んでいる
- Hadoopアプリケーションは**ファイルの格納位置を考慮してスケジューリング**

HDFSとの性能比較[Marilia, SWoPP13]

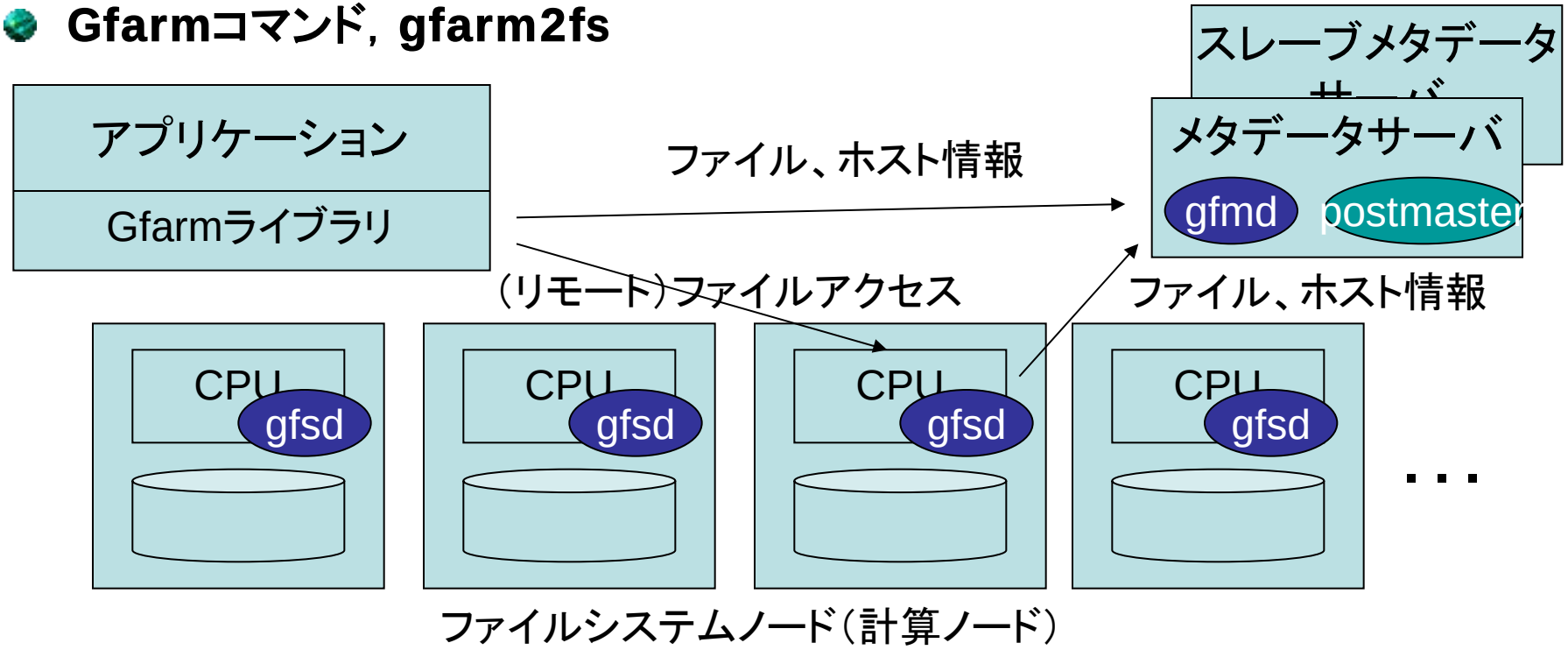


Gfarmは複製数を3としても性能への影響がほとんどない

Gfarmの実装

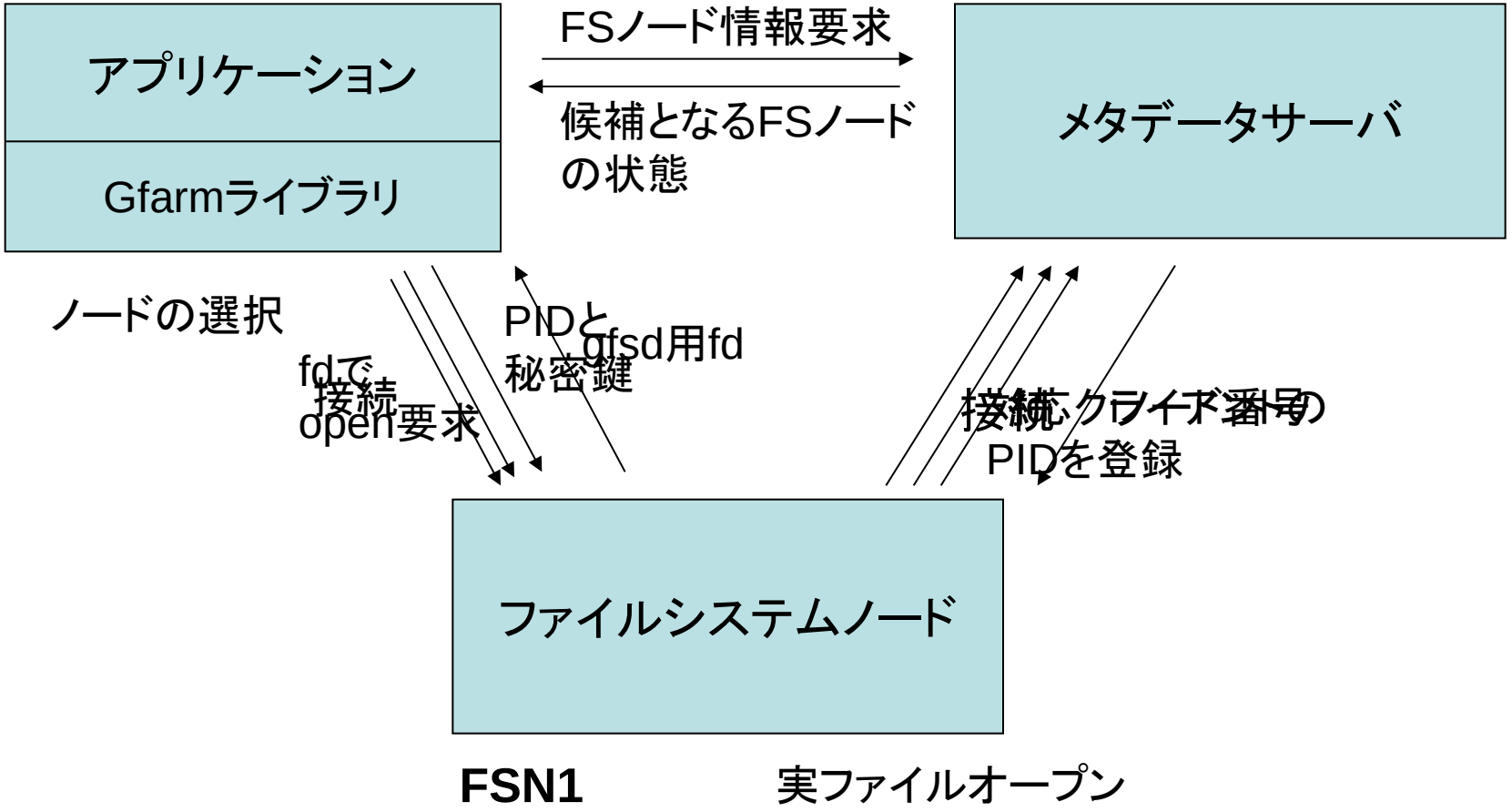
Gfarmの実装の概要

- **gfmd – メタデータサーバ（MDS）**
  - ディレクトリ情報、複製カタログ、ホスト情報、プロセス情報
- **gfsd – I/Oサーバ**
  - （リモート）ファイルアクセス
- **libgfarm – Gfarmライブラリ**
  - Gfarm API
- **Gfarmコマンド，gfarm2fs**



ファイルのアクセス時のgfsdとの接続

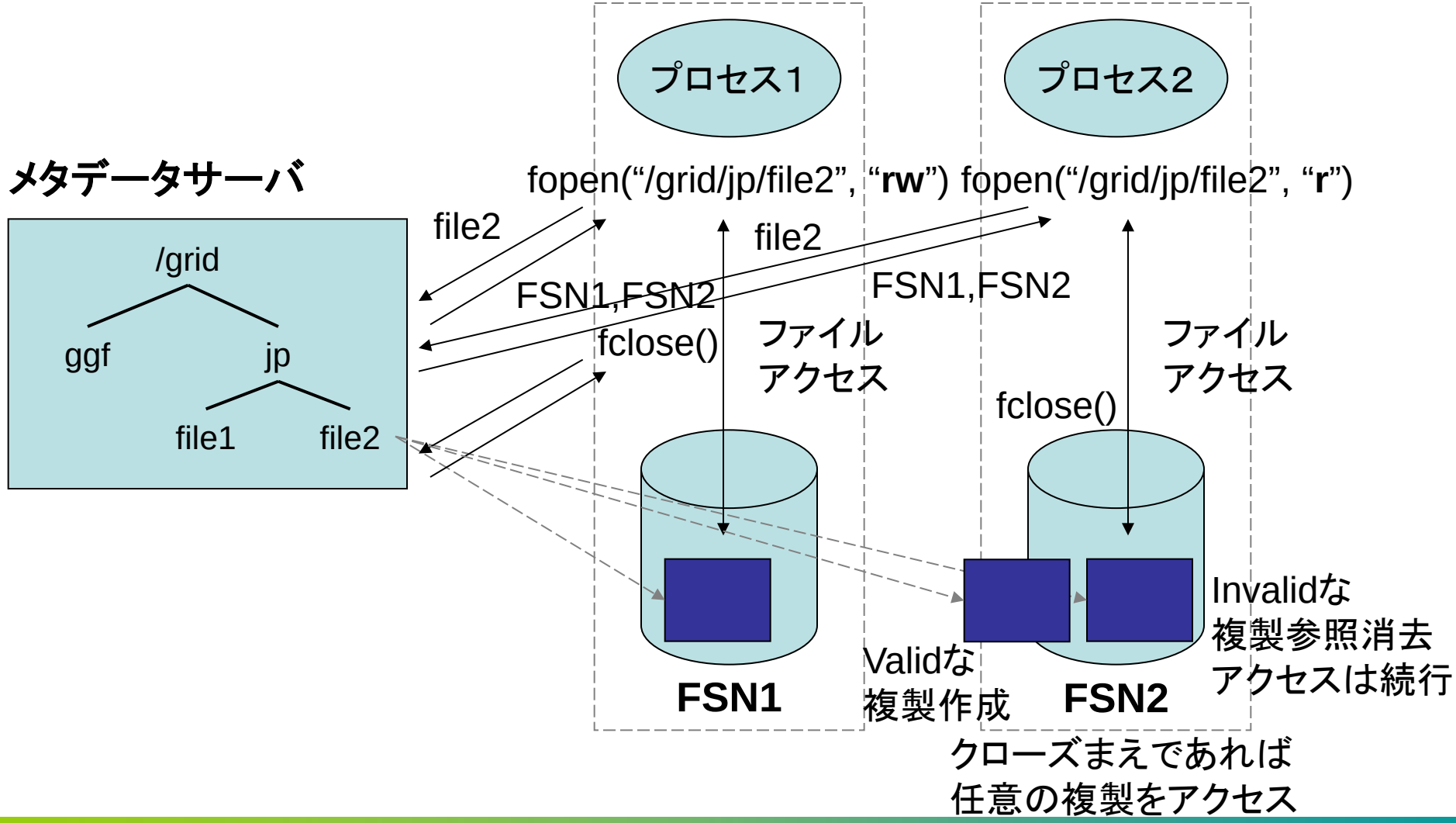
- **候補となるFSノード情報をもらい、ネットワーク距離によりFSノードを選択**



close-to-open一貫性の保持と複製数維持

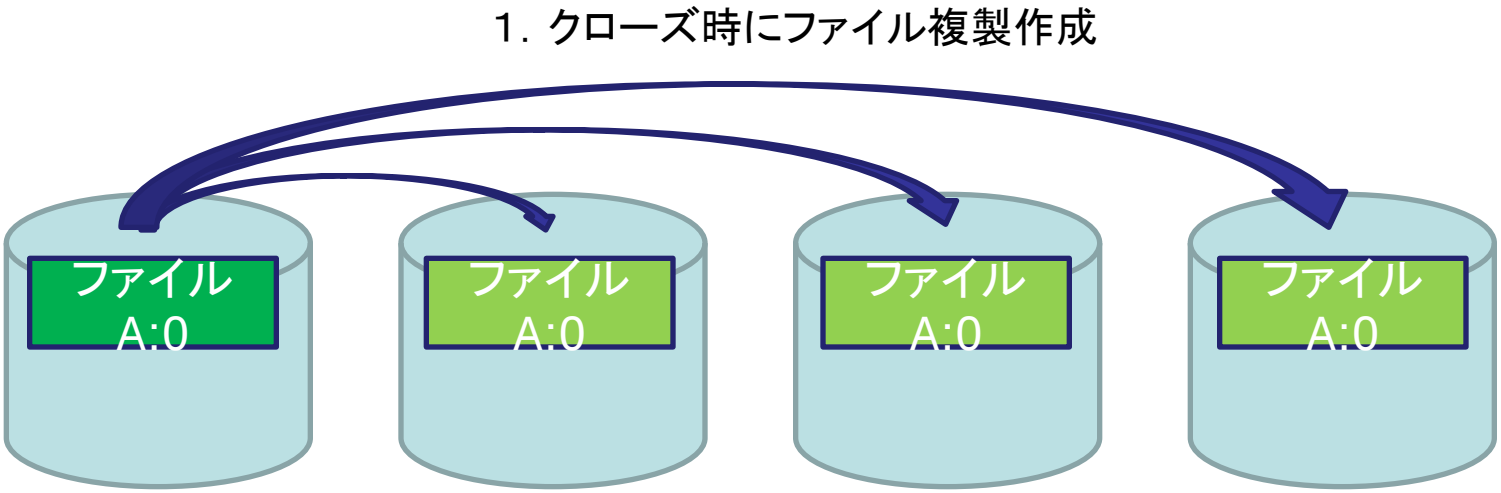
GRID DATA FARM

- クローズ時に他のファイル複製を消去



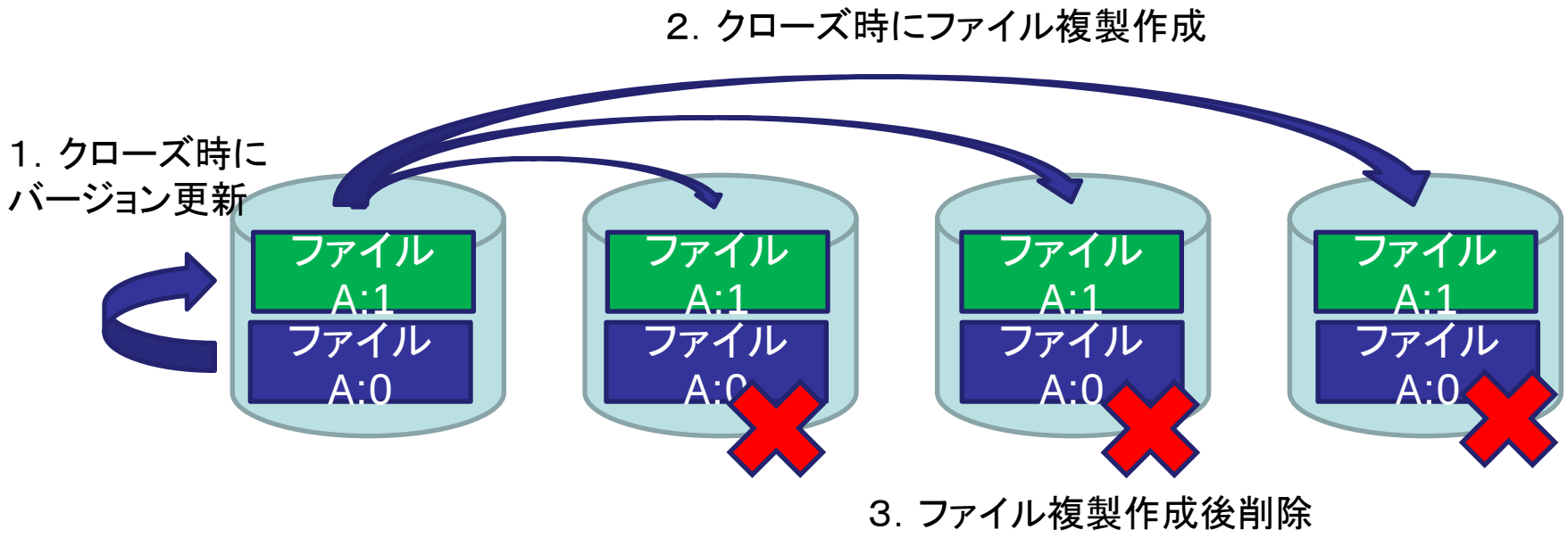
ファイル複製の自動作成（1）

- **ファイルクローズ時に指定された数の複製を作成**
  - 拡張属性（gfncopy）でディレクトリ単位に指定



ファイル複製の自動作成（2）

- **ファイル複製数を減らさない仕組み**
  - 無効化型一貫性制御では更新時複製数が1となる
  - 更新型一貫性制御を安全に行うため、クローズ時にバージョニングと複製作成



複製数維持

- **複製数変更の可能性があるとき、gfmdの専用スレッドで全ファイルの複製数のチェック**
  - 足りなければ作成、多すぎれば削除
- **チェックのタイミング**
  - gfmd、gfsd起動時
  - gfsdが停止して指定時間後
  - 複製作成数変更時
  - ファイル、ディレクトリの他ディレクトリへの移動時
  - 複製作成失敗時
- **チェックのon/offを動的に指定可能**

  % **gfrepcheck** enable | disable

ネットワーク切断時の処理

GRID DATA FARM

- **クライアント-gfmd間**
  - 再接続し続行
  - ファイルオープンのフェイルオーバ処理 [2.6から]
    - オープン中のファイルの再オープン
- **クライアント-gfsd間**
  - 読込アクセスは別複製アクセスへ自動的に切替
  - 書込アクセスはエラーを返す
- **gfsd-gfsd間**
  - ファイル複製作成をスケジューリングからやり直し
- **gfmd-gfmd間**
  - 再接続、ジャーナル転送

→アプリケーション透明にgfmdのフェイルオーバ、停止
gfsdの停止、起動が可能

障害対応と一貫性チェック

GRID DATA FARM

- gfmd起動時のメタデータの参照数チェック
- gfsd起動時のメタデータと実ファイルとの一貫性チェック（存在、サイズ）
- アクセス時、複製作成時のチェックサムによるファイル損傷チェック（サイレントデータ損傷対応）
- 複製数維持機能（障害発生時、複製数変更時）
- I/Oエラー発生時のgfsd自動停止
- ディスクフル、障害によるROFSへの移行時に空き容量0とみなす
- Zabbix pluginによる動作監視と自動gfmdフェイルオーバ

まとめ

- Gfarmファイルシステム
  - NPO法人つくばOSS技術支援センターによるサポート
- 大容量、高速なデータ共有
- 耐障害性に優れる
- データ完全性、サイレントデータ損傷対応
- HPCI共用ストレージ、JLDGなど実運用実績
- ビッグデータ処理