Pwrakeチュートリアル

田中昌宏
筑波大学 計算科学研究センター

内容

- Rakeの概要
- Rakefileの書き方
- Rakefile記述に必要なRubyの知識
- Pwrakeを動かす準備
- Pwrakeの動かし方

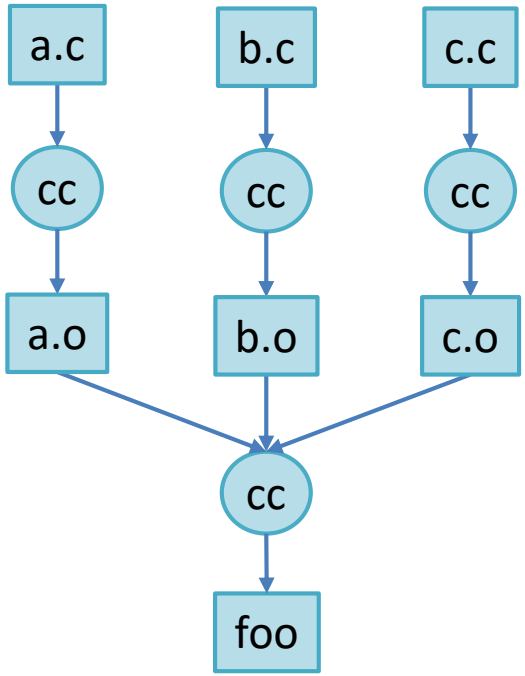
Rakeの概要

Rake

- Ruby版のmake
- タスク定義文法
  - 独自の文法ではなく、Rubyスクリプトとして評価される。
  - このようにホスト言語を利用した言語は、**内部DSL** (Internal Domain-Specific Language)と呼ばれる。
  - Rubyスクリプトとして実行されるため、Rubyの言語仕様をフルに利用できる。
  - ビルドツールとしてだけではなく、ワークフロー定義にも有用

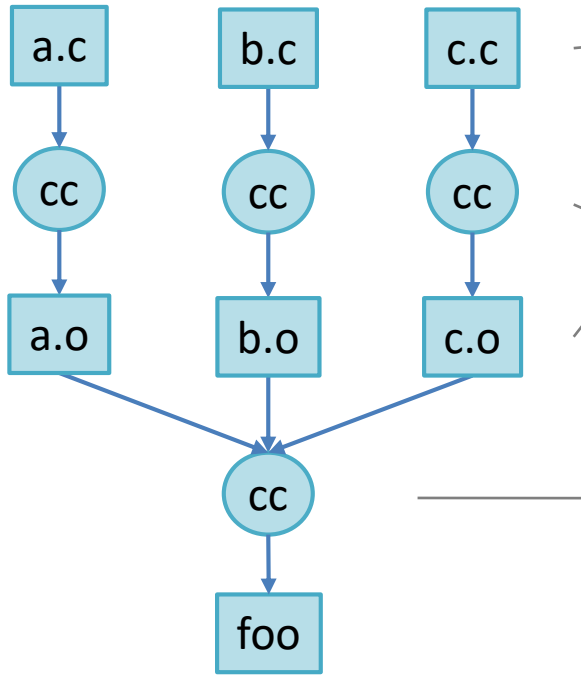
Workflow to Build Program

DAG



Workflow to Build Program

DAG



Makefile (GNU make)

```
SRCS := $(wildcard *.c)
OBJS := $(subst .c,.o,$(SRCS))

all: foo

%.o : %.c
        cc -o $@ -c $<


foo: $(OBJS)
        cc -o $@ $^
```

Workflow to Build Program

Rakefile

```
SRCS = FileList["*.c"]
OBJS = SRCS.ext("o")

task :default => "foo"

rule ".o" => ".c" do |t|
  sh "cc -o #{t} -c #{t.source}"
end

file "foo" => OBJS do |t|
  sh "cc -o #{t} #{OBJS}"
end
```

Makefile (GNU make)

```
SRCS := $(wildcard *.c)
OBJS := $(subst .c,.o,$(SRCS))

all: foo

%.o : %.c
        cc -o $@ -c $<


foo: $(OBJS)
        cc -o $@ $^
```

GNU Makeで書けることは、Rakeでも書ける

Rakefileの書き方

タスク定義

```
task :default => "foo"
```

- task
  - タスクを定義するメソッド。ここでは default というタスク名のタスクを定義。
- :default
  - コロン(:)で始まる文字列は、RubyにおいてSymbolを表すリテラル。
  - Symbol は String と似たもので、Rakeのタスク名にはどちらも使用可能。
  - default タスクは、rake の引数で指定するタスクを省略したときのターゲットを表す。
- :default => “foo”
  - => は、Rubyにおいて、key-value 引数を表す。
  - default タスクが ”foo” タスクに依存することを表す。

ファイルタスク定義

```
file "foo" => OBJS do |t|
  sh "cc -o #{t.name} #{OBJS}"
end
```

- file
  - FileTask を定義するための、Rakeで定義されたメソッド。taskと異なる点は、タスク名を出力ファイル名とみなす点と、ファイルのタイムスタンプによってタスク実行をスキップする点。
- do |t| .. end
  - Rubyコードブロック。中括弧 {} でも記述できる。コードブロックとは無名関数のようなものであるが、変数のスコープはブロックの外側と共有している。
  - コードブロックは、file メソッドに渡され、タスクの依存関係の順に実行される。
  - |t| は、コードブロックへ渡される引数を表す記法。Rakeのタスク定義では、tとして、Rake::Taskクラスのインスタンスが渡される。

外部コマンド起動

```
sh "cc -o #{t.name} -c #{t.source}"
```

- sh
  - 外部コマンドを起動するメソッド。Rakeで定義。
- 2重引用符 ".."
  - 文字列を表すRubyのリテラル。バックスラッシュ記法 と式展開が有効になる。
  - #{..}により、括弧内の式を評価した文字列を埋め込む。
- t.name, t.source
  - t は、Rake::Task クラスのインスタンス。
  - name は、自身のタスク名＝出力ファイル名を参照するメソッド。
  - source は、必要タスク名＝入力ファイル名を参照するメソッド。
  - 複数の必要タスクを配列として得る場合は、t.prerequisites とする。

ルールの記述

```
rule ".o" => ".c" do |t|
  sh "cc -o #{t.name} -c #{t.source}"
end
```

- rule
  - ルールを記述するためのメソッド。
- ".o" => ".c"
  - 拡張子が .o のファイルが、拡張子が .c のファイルに依存することを表す。

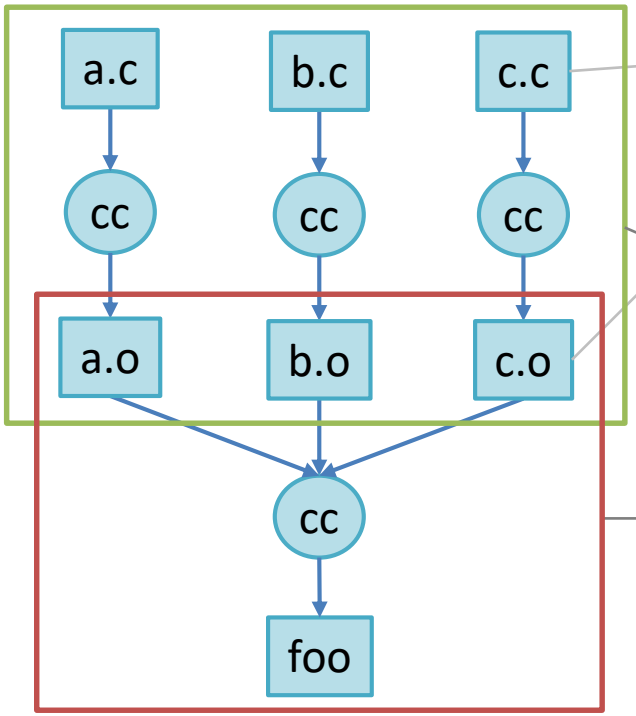
ファイルリスト

```
SRCS = FileList["*.c"]
OBJS = SRCS.ext("o")
```

- FileList["*.c"]
  - “*.c”で展開されるファイルリストを配列で返す。
- SRCS.ext("o")
  - SRCS配列のそれぞれの要素について、拡張子を .o に置き換えた配列を返す。*.o ファイルは最初は存在しないので、FileListは使えない。

DAGの基本パターン

DAG



Rakefile

```
SRCS = FileList["*.c"]
OBJS = SRCS.ext("o")

task :default => "foo"

rule ".o" => ".c" do |x|
  sh "cc -o #{x} -c #{x.source}"
end

file "foo" => OBJS do |x|
  sh "cc -o #{x} #{OBJS}"
end
```

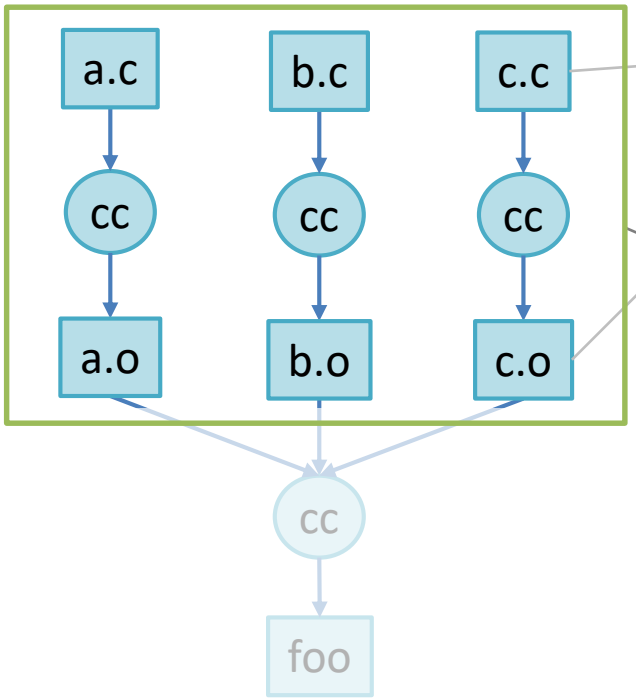
基本パターンができれば応用可能

Rakefileを書くコツ

- 入力ファイルから中間ファイルのリストを作る
  - 初めに中間ファイルは存在しないので、`FileList["*.o"]` ができない。
  - 入力ファイルのリストからパターン変換する
    - `FileList["*.c"].ext("o")`
    - pathmap（後述）
- 出力ファイル名から入力ファイル名へ変換するルールを決める
  - パターンマッチで依存関係が判別できるようなファイル名にする

1対1 パターン

DAG



Rakefile

```
SRCS = FileList["*.c"]
OBJS = SRCS.ext("o")

task :default => "foo"

rule ".o" => ".c" do |x|
  sh "cc -o #{x} -c #{x.source}"
end

file "foo" => OBJS do |x|
  sh "cc -o #{x} #{OBJS}"
end
```

For-Loop

- Montageワークフローの一部

```
INPUT = FileList["r/*.fits"]
OUTPUT = []

for src in INPUT
  OUTPUT << dst = "p/"+File.basename(src)
  file dst => src do |t|
    sh "mProjectPP #{t.prerequisites[0]} #{t.name} region.hdr"
  end
end

task :default => OUTPUT
```

Rule

- 前ページと同じ定義を rule で書き換え

```
INPUT = FileList["r/*.fits"]
OUTPUT = INPUT.pathmap("p/%f")

rule /^p¥/.*¥.fits$/ => "r/%n.fits" do |t|
  sh "mProjectPP #{t.prerequisites[0]} #{t.name} region.hdr"
end

task :default => OUTPUT
```

- Pathmap: パス名を変換するRakeの機能。
  - FileList または String に pathmap メソッドが定義される。
  - rule の出力ファイル名にマッチした文字列に対しても pathmap で変換されて入力ファイル名になる。

Pathmap Examples

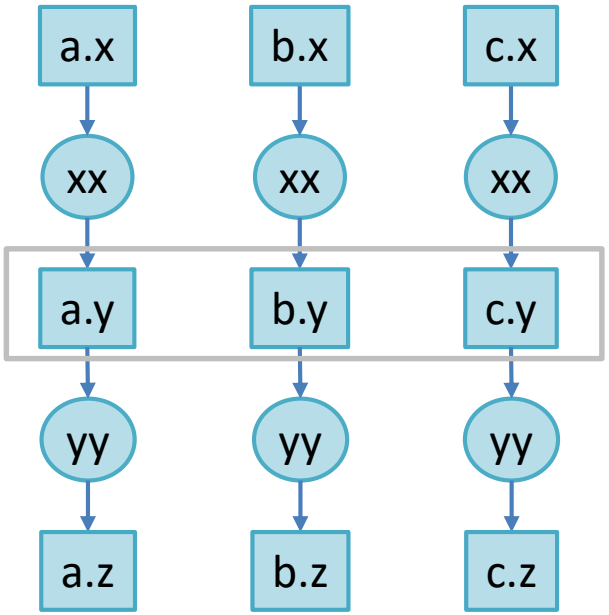
```
p 'a/b/c/file.txt'.pathmap("%p") #=> "a/b/c/file.txt"
p 'a/b/c/file.txt'.pathmap("%f") #=> "file.txt"
p 'a/b/c/file.txt'.pathmap("%n") #=> "file"
p 'a/b/c/file.txt'.pathmap("%x") #=> ".txt"
p 'a/b/c/file.txt'.pathmap("%X") #=> "a/b/c/file"
p 'a/b/c/file.txt'.pathmap("%d") #=> "a/b/c"
p 'a/b/c/file.txt'.pathmap("%2d") #=> "a/b"
p 'a/b/c/file.txt'.pathmap("%-2d") #=> "b/c“

p 'a/b/c/file.txt'.pathmap("%d%s%{file,out}f")
        #=> "a/b/c/out.txt“

p 'a/b/c/file.txt'.pathmap("%X%{.*,*}x"){|ext| ext.upcase}
        #=> "a/b/c/file.TXT“
```

rule のメリット

DAG



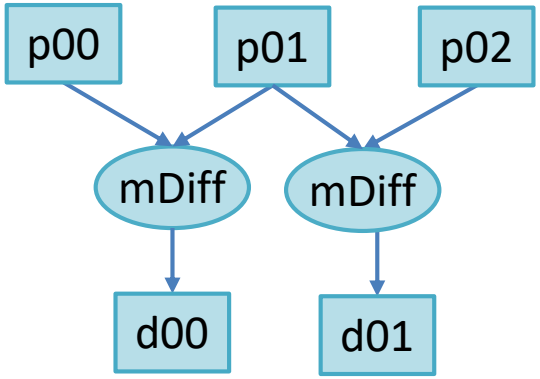

```
XXX = FileList["*.x"]
ZZZ = XXX.ext("z")

rule ".y" => ".x" do |t|
  sh "xx -o #{t} #{t.source}"
end

rule ".z" => ".y" do |t|
  sh "yy -o #{t} #{t.source}"
end
```

- 中間ファイル（*.y ）のリストが不要
- for-loopが不要（ネストが減る）

N対M パターン



- パス名からは依存関係を定義できないケース
- 入出力ファイルの依存関係をマップするデータが必要

```
FILEMAP = {"d00.fits“=>["p00.fits","p01.fits"], ...}

rule /^d.*¥.fits$/ => proc{|x| FILEMAP[x]} do |t|
  p1,p2 = t.prerequisites
  sh "mDiff #{p1} #{p2} #{t.name} region.hdr"
end
```

Parameter Sweep

```
PARAMS = [1,2,5,10]
TASKS = []

for i in PARAMS
  TASKS << tname = "task#{i}"
  file tname => src do |t|
    sh "taskcmd --param #{i} && touch #{t.name}"
  end
end

task :default => TASKS
```

- パラメータ毎にループを回してタスクを定義。
- パラメータを元にタスク名をつける。
  - タスクが実行済みかどうか判定するため、ダミーの出力ファイルを作成するとよい。

Rakefile記述に必要なRubyの知識

Rakefile記述に必要なRubyの知識

- 文字列の処理
- 正規表現
- コレクション： Array, Hash
- ブロックのスコープ

Rubyの基本リテラル

- Numeric（数値）
  - Integer: `123`
  - Float: `12.3  1.2e-3`
- String（文字列）
  - `"abc"`
  - `'abc'`
- Symbol（文字列をラベルとして扱うためのもの）
  - `:abc`
- Regexp（正規表現）
  - `/^abc$/`
- Array
  - `[123, "abc"]`
- Hash
  - `{:a=>1, :b=>2}`
  - `{a:1, b:2}` # 新しい記法

Rubyの変数

- ローカル変数
  - `abc = 123`
  - 変数名が小文字かアンダースコアで始まる
  - ローカルスコープ
- グローバル変数
  - `$abc = 123`
  - 変数名が `$` で始まる
  - グローバルスコープ
- 定数
  - `ABC = 123`
  - 変数名が大文字で始まる
  - グローバルスコープ
- インスタンス変数
  - `@abc = 123`
  - 変数名が `@` で始まる
  - インスタンス内で共有
- クラス変数
  - `@@abc = 123`
  - 変数名が `@@` で始まる
  - クラス内で共有

Rubyのメソッド呼び出し

- 基本形
  - `receiver.method_name(arg0, arg1, arg2)`
  - `receiver.method_name arg0, arg1, arg2`　　# 括弧を省略可
- レシーバがないメソッド
  - `puts rand`
  - 字面からはローカル変数と区別がつかないので注意。
  - メソッドと変数が両方定義されている場合、引数なし、括弧なしのときは、変数が参照される。
  - Rakefileでは、task, file, rule, sh などのメソッドが定義されているので、変数として使わないようにする。
- キーワード引数
  - `foo a, :key => value`
- ブロック引数：コードブロックを
  - `foo do |x| puts x end`
  - `foo {|x| puts x }`

Ruby文字列

- String リテラル

```
"abc¥n"
'abc¥n'
%q!I said, "You said, 'She said it.'"!
%!I said, "You said, 'She said it.'"!
%Q('This is it.'¥n)
"multi line
string"
```

- 式展開

```
"3*2=#{3*2}"   #=> "3*2=6"
'3*2=#{3*2}'   #=> "3*2=#{3*2}"
```

- String へ変換

```
1e5.to_s  #=> "100000.0"
(1..5).to_a.join("-")
  # => "1-2-3-4-5"
```

- String class のよく使うメソッド
  - 連結：+, <<
  - 繰り返し: *
  - 比較：==
  - フォーマット: %
  - 長さ：length, size
  - 空文字列か：empty?
  - 整数化：to_i
  - 部分文字列検索：include?, index
  - 部分文字列の参照・置換：[], []=
  - パターンマッチ：=~, match
  - パターン置換：gsub, sub
  - 区切りで分割：split
  - 1行毎のイテレーション：each_line
  - 改行除去：chomp
  - 空白除去：strip

Pwrakeを動かす準備

計算機クラスタの設定

- メタデータサーバ
  - バックエンドDBのI/O性能が必要になることがある。
    - HDD ではなく SSD を使用するとよい。
  - max open files を大きめ（数万）に設定。
    - /etc/security/limits.conf または /etc/security/limits.d/*.conf で設定
    - ulimit -n で確認
- Gfarmスプール用ストレージ
  - 計算ノードのローカルストレージを利用。
- root権限なしでGfarmをスタンドアロンモードで動かす場合
  - スプール用ディレクトリへの書き込み権限をもらう。
  - FUSEの使用権限をもらう。（fuseグループへの追加）

RubyとPwrakeのインストール

- Ruby のインストール（ver. 2.0 以上 : メンテナンスされているバージョン）
  - yum や apt-get などのパッケージ
  - または、ソースからビルド
- Pwrake のインストール（最新版 ver. 2.1.2）

```
gem install pwrake
```

- グラフ分割スケジューリングを利用する場合
  - METIS (http://www.cs.umn.edu/~metis/) 5.1.0 をインストール
    - CentOS 7 の場合:

```
sudo yum install metis-devel
```

  - RbMetis (https://github.com/masa16/rbmetis) をインストール:

（ 2,3行目は、METIS を自分でビルドした場合に必要）

```
gem install rbmetis ¥
 -- --with-metis-include=/usr/local/include ¥
 --with-metis-lib=/usr/local/lib
```

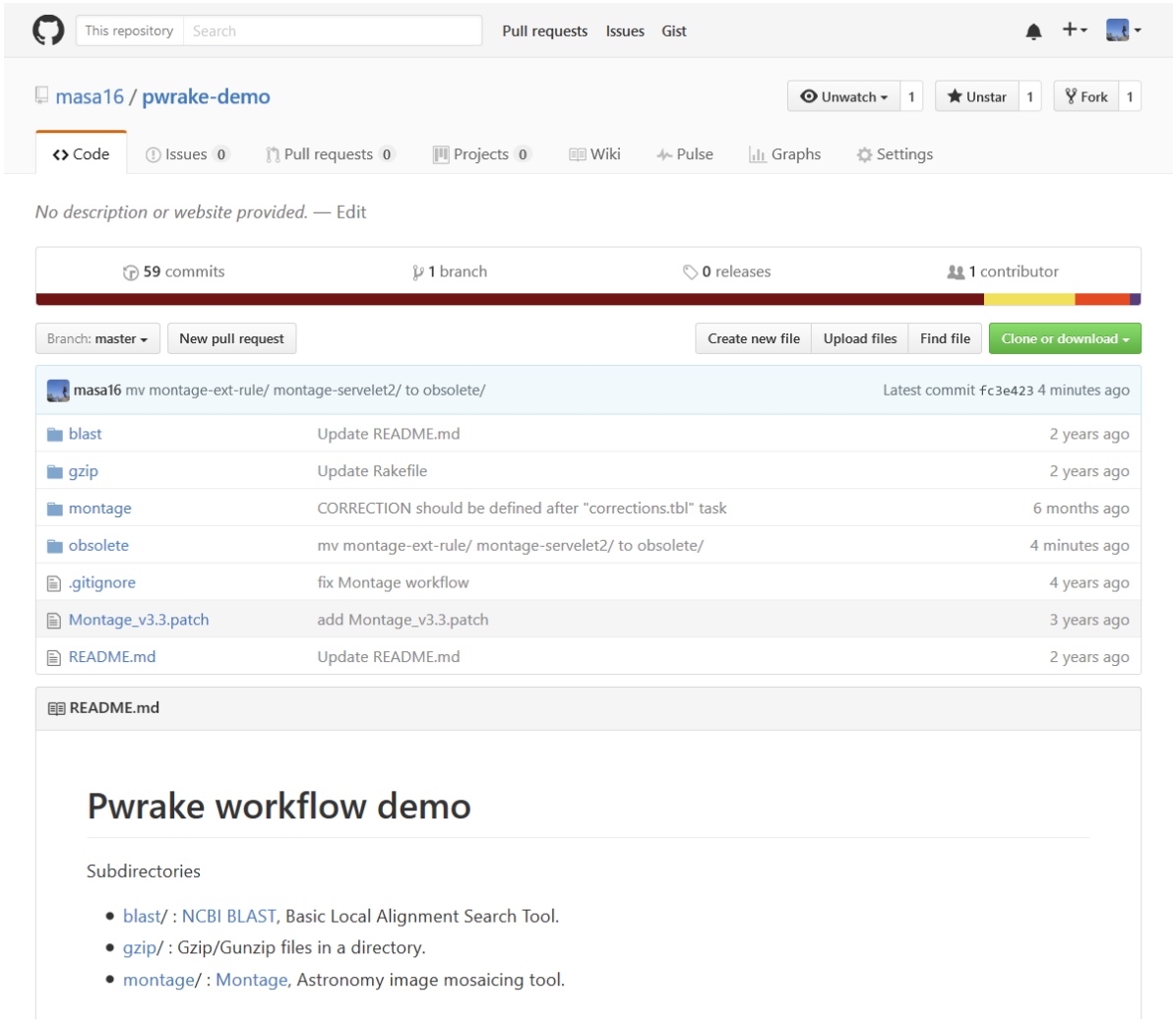
SSHの設定

- 必要な設定:
  - pwrake 実行ノードからワーカーノードへ、パスワードなしでSSH接続が可能であること。
- 注意点:
  - 秘密鍵をクラスターに置くことはできるだけ避ける。
- 望ましい方法:
  - 手元のPCに秘密鍵、クラスタに公開鍵を登録。
  - 手元のPCで ssh-agent を起動しておき、ssh-add でパスワードを登録。
  - 手元のPCからクラスタにSSH接続する際、agent forward を有効にする:
    - ssh に -A オプションをつける、または、
    - .ssh/config に ForwardAgent yes と書く
  - pwrake実行ノードからワーカーノードへの接続も同様に agent forward を有効にすると、手元のPCのagentがforwardされる。
- やむを得ずパスワードなし鍵が必要な場合:
  - Pwrakeをバッチシステムで走らせる場合など
  - 鍵ペアをクラスタのマシンで生成し、クラスタ外部には絶対にコピーしない。

Pwrakeの動かし方

pwrake-demo

https://github.com/masa16/pwrake-demo



Pwrake実行準備

- 必要ファイル
  - Rakefile
  - ホスト名を記述したファイル（HOSTFILE）
  - ワークフローの入力ファイル
  - pwrake_conf.yaml （YAML形式でオプションを記述）
- 手順
  - gfarm2fs で Gfarm FS をマウント。
  - Gfarm FS 内に作業ディレクトリを作成。
  - Rakefileなどのファイルを作業ディレクトリに置く。
  - Rakefileのあるディレクトリで pwrake コマンドを実行。

pwrake コマンドラインオプション

```
$ pwrake --help
pwrake [-f rakefile] {options} targets...

Options are ...
       --backtrace=[OUT]           Enable full backtrace.  OUT can be stderr (default) or stdout.
       --comments                  Show commented tasks only
       --job-stats [LEVEL]         Display job statistics. LEVEL=history displays a complete job list
       --rules                     Trace the rules resolution.
       --suppress-backtrace PATTERN Suppress backtrace lines matching regexp PATTERN. Ignored if --trace is on.
       --all                       Show all tasks, even uncommented ones (in combination with -T or -D)
   -B, --build-all                 Build all prerequisites, including those which are up-to-date.
   -D, --describe [PATTERN]        Describe the tasks (matching optional PATTERN), then exit.
   -e, --execute CODE              Execute some Ruby code and exit.
   -E, --execute-continue CODE     Execute some Ruby code, then continue with normal task processing.
   -f, --rakefile [FILENAME]       Use FILENAME as the rakefile to search for.
   -G, --no-system, --nosystem     Use standard project Rakefile search paths, ignore system wide rakefiles.
   -g, --system                    Using system wide (global) rakefiles (usually '~/.rake/*.rake').
   -I, --libdir LIBDIR             Include LIBDIR in the search path for required modules.
   -m, --multitask                 Treat all tasks as multitasks.
   -n, --dry-run                   Do a dry run without executing actions.
   -N, --no-search, --nosearch     Do not search parent directories for the Rakefile.
   -P, --prereqs                   Display the tasks and dependencies, then exit.
   -p, --execute-print CODE        Execute some Ruby code, print the result, then exit.
   -q, --quiet                     Do not log messages to standard output.
   -r, --require MODULE            Require MODULE before executing rakefile.
   -R, --rakelibdir RAKELIBDIR,    Auto-import any .rake files in RAKELIBDIR. (default is 'rakelib')
       --rakelib
   -s, --silent                    Like --quiet, but also suppresses the 'in directory' announcement.
   -t, --trace=[OUT]               Turn on invoke/execute tracing, enable full backtrace. OUT can be stderr (default) or stdout.
   -T, --tasks [PATTERN]           Display the tasks (matching optional PATTERN) with descriptions, then exit.
   -v, --verbose                   Log message to standard output.
   -V, --version                   Display the program version.
   -W, --where [PATTERN]           Describe the tasks (matching optional PATTERN), then exit.
   -X, --no-deprecation-warnings   Disable the deprecation warnings.
   -F, --hostfile FILE             [Pw] Read hostnames from FILE
   -j, --jobs [N]                  [Pw] Number of threads at localhost (default: # of processors)
   -L, --log, --log-dir [DIRECTORY] [Pw] Write log to DIRECTORY
       --ssh-opt, --ssh-option OPTION
                                   [Pw] Option passed to SSH
       --filesystem FILESYSTEM     [Pw] Specify FILESYSTEM (nfs|gfarm)
       --gfarm                     [Pw] FILESYSTEM=gfarm
   -A, --disable-affinity          [Pw] Turn OFF affinity (AFFINITY=off)
   -S, --disable-steal             [Pw] Turn OFF task steal
   -d, --debug                     [Pw] Output Debug messages
       --pwrake-conf [FILE]        [Pw] Pwrake configuation file in YAML
       --show-conf, --show-config  [Pw] Show Pwrake configuration options
       --report LOGDIR             [Pw] Report workflow statistics from LOGDIR to HTML and exit.
       --clear-gfarm2fs            [Pw] Clear gfarm2fs mountpoints left after failure.
   -h, -H, --help                  Display this help message.
```

Rakeのオプション

Pwrakeのオプション
[Pw] がついている

pwrake_conf.yaml
Pwrakeオプション記述ファイル

- YAML形式で記述

```
HOSTFILE: hosts
LOG_DIR: true
PASS_ENV :
 - ENV1
 - ENV2
```

- 同じオプションを環境変数で渡すことも可能

```
$ pwrake HOSTFILE=hosts
```

pwrake_conf.yaml オプション一覧

```
HOSTFILE, HOSTS    nil(default, localhost)|filename
LOG_DIR, LOG       nil(default, No log output)|true(dirname="Pwrake%Y%m%d-%H%M%S")|dirname
LOG_FILE           default="pwrake.log"
TASK_CSV_FILE      default="task.csv"
COMMAND_CSV_FILE   default="command.csv"
GC_LOG_FILE        default="gc.log"
WORK_DIR           default=$PWD
FILESYSTEM         default(autodetect)|gfarm
SSH_OPTION         SSH option
SHELL_COMMAND      default=$SHELL
SHELL_RC           Run-Command when shell starts
PASS_ENV           (Array) Environment variables passed to SSH
HEARTBEAT          default=240 - Hearbeat interval in seconds
RETRY              default=1 - The number of retry
FAILED_TARGET      rename(default)|delete|leave - Treatment of failed target files
FAILURE_TERMINATION wait(default)|kill|continue - Behavior of other tasks when a task is failed
QUEUE_PRIORITY          LIHR(default)|FIFO|LIFO|RANK
NOACTION_QUEUE_PRIORITY FIFO(default)|LIFO|RAND
SHELL_START_INTERVAL    default=0.012 (sec)
GRAPH_PARTITION         false(default)|true
DISABLE_AFFINITY    default=false
DISABLE_STEAL       default=false
GFARM_BASEDIR       default="/tmp"
GFARM_PREFIX        default="pwrake_$USER"
GFARM_SUBDIR        default='/'
MAX_GFWHERE_WORKER  default=8
GFARM2FS_OPTION     default=""
GFARM2FS_DEBUG      default=false
GFARM2FS_DEBUG_WAIT default=1
```

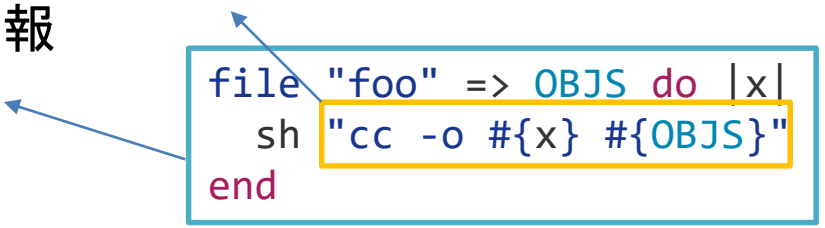
Gfarm関連のオプション

ワーカーノードの指定

- HOSTFILE
  - ワーカーノードのホスト名と使用コア数のリストを記述したファイル
  - 1行に hostname とコア数を記述。
    - コア数を省略すると、マシンの最大コア数を使用
  - 複数ホストをまとめて指定 : host[00..10]
  - hostname は、Locality-aware scheduling のとき、GfarmのFSNの名前と同じにする必要がある。
- HOSTFILE 指定方法
  - コマンドラインの -F または --hostfile
  - pwrake_conf.yaml の HOSTFILE または HOSTS

ログ出力

- ログ出力オプション
  - コマンドラインオプション: -L or --log or --log-dir
  - pwrake_conf.yaml: LOG_DIR
    - ディレクトリ名を指定すると、ディレクトリを作成し、その下にログファイルを生成。
    - ディレクトリ名を省略した場合、Pwrake20161021_1234 のように、日付_時刻でディレクトリを作成。
- 出力するログファイル:
  - pwrake.log : Pwrakeの実行ログ
  - worker-ホスト名-プロセスID.log : ワーカーの実行ログ
  - command.csv : プロセスの実行情報
  - task.csv : タスクの実行情報



統計情報レポート

- レポート出力方法:
  - pwrake --report ログディレクトリ
- 実行ログに基づいて、各種統計情報を report.html に出力
  - プロセス数の推移:全体・コマンド毎・ホスト毎
  - コマンド毎の実行時間の統計
  - ファイルアクセス(ローカル・リモート)の割合
- Gnuplot が必要

Report (1)
プロセス数の推移(全体)

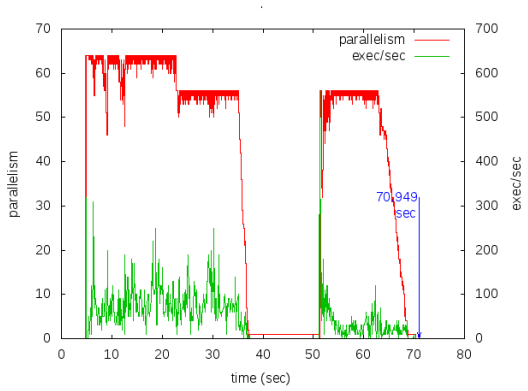
Pwrake Statistics

Workflow

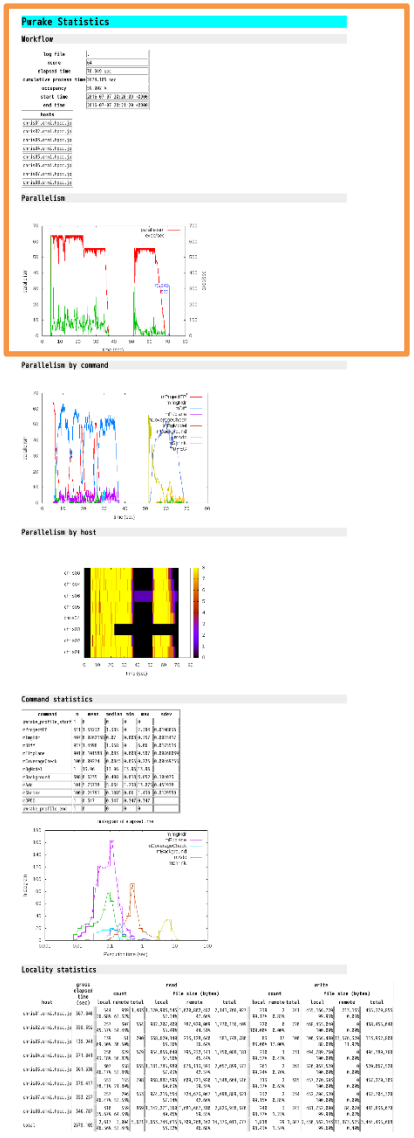
log file	.
ncore	64
elapsed time	70.949 sec
cumulative process time	2679.105 sec
occupancy	59.002 %
start time	2016-07-07 20:20:09 +0900
end time	2016-07-07 20:21:20 +0900

hosts
chris01.omni.hpcc.jp
chris02.omni.hpcc.jp
chris03.omni.hpcc.jp
chris04.omni.hpcc.jp
chris05.omni.hpcc.jp
chris06.omni.hpcc.jp
chris07.omni.hpcc.jp
chris08.omni.hpcc.jp

Parallelism

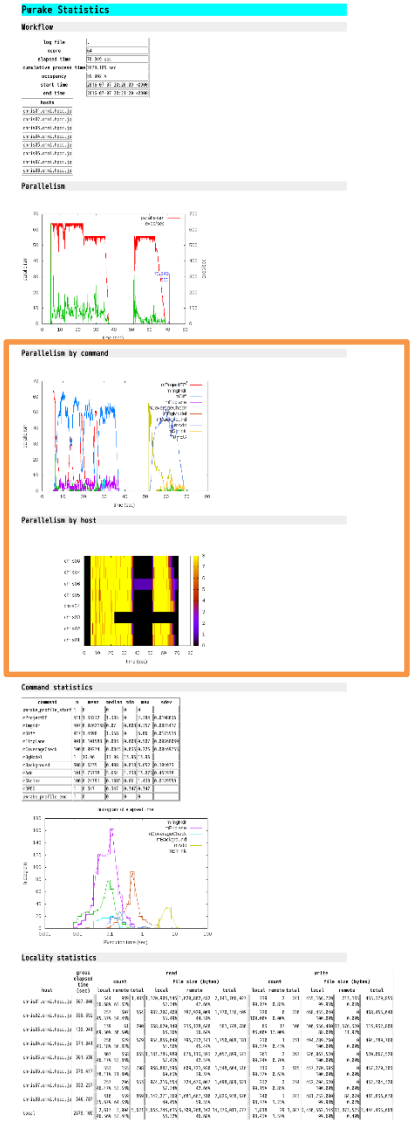


reportページ全体

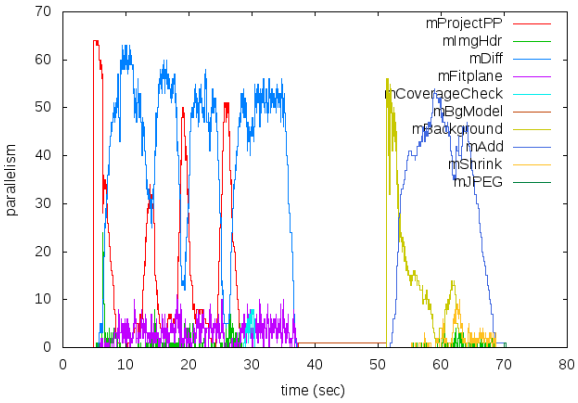


Report (2)
プロセス数の推移（コマンド毎・ホスト毎）

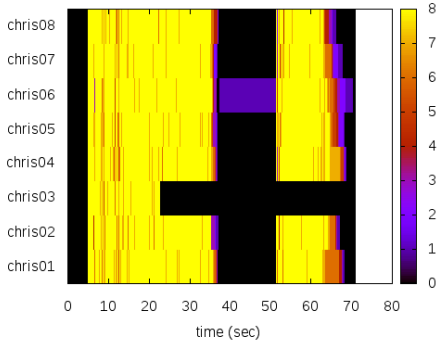
reportページ全体



Parallelism by command

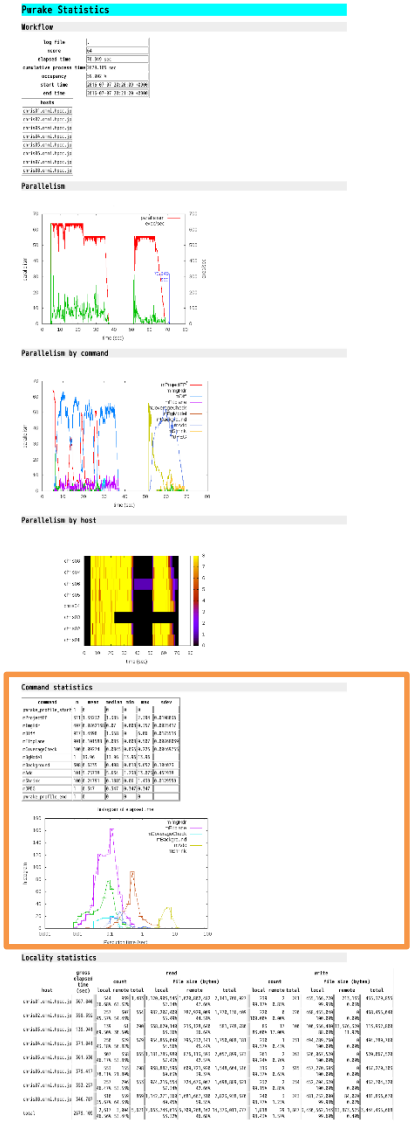


Parallelism by host



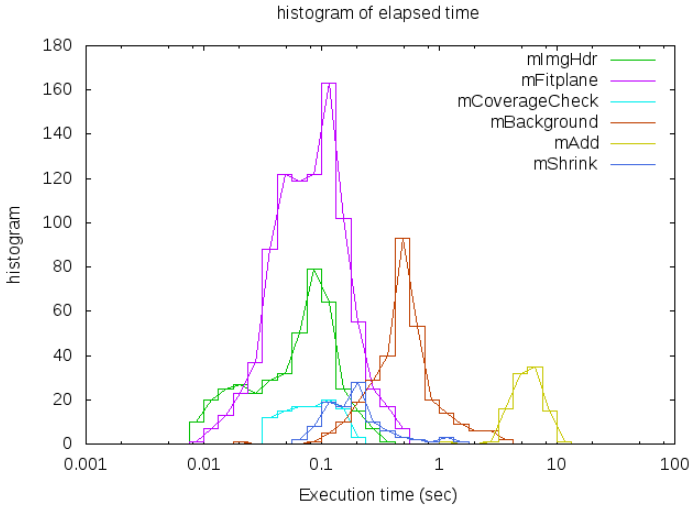
Report (3)
コマンド毎の実行時間の統計

reportページ全体



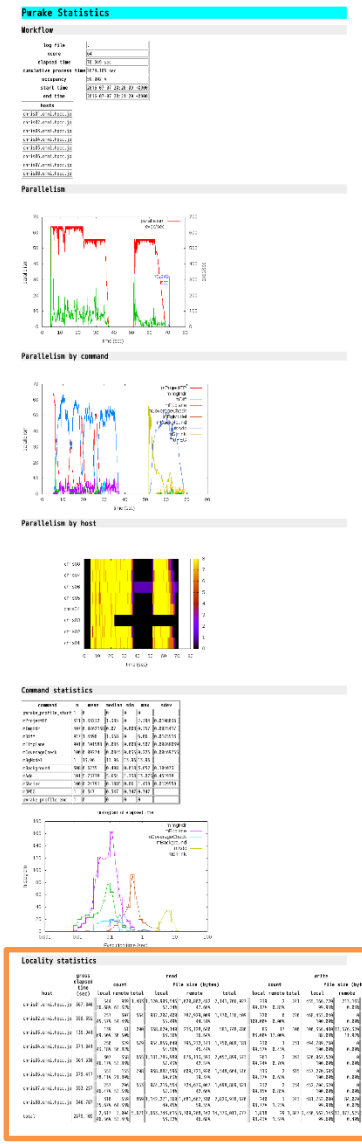
Command statistics

command	n	mean	median	min	max	sdev
pwrake_profile_start	1	0	0	0	0	
mProjectPP	311	1.33232	1.385	0	2.394	0.0198036
mImgHdr	407	0.0762138	0.07	0.008	0.357	0.0025417
mDiff	917	1.4398	1.358	0	6.06	0.0175513
mFitplane	901	0.101588	0.083	0.008	0.507	0.00768039
mCoverageCheck	100	0.09274	0.0845	0.035	0.225	0.00469755
mBgModel	1	13.96	13.96	13.96	13.96	
mBackground	308	0.6235	0.498	0.018	3.652	0.104073
mAdd	101	5.75738	5.634	1.238	13.075	0.451938
mShrink	100	0.24791	0.1885	0.06	1.439	0.0129559
mJPEG	1	0.347	0.347	0.347	0.347	
pwrake_profile_end	1	0	0	0	0	



Report (4) ファイルアクセス（ローカル・リモート）の割合

reportページ全体



Locality statistics

host	gross elapsed time (sec)	read count			read file size (bytes)			write count			write file size (bytes)		
		local	remote	total	local	remote	total	local	remote	total	local	remote	total
chris01.omni.hpcc.jp	367.840	544 36.68%	939 63.32%	1,483	1,120,903,545 52.34%	1,020,802,482 47.66%	2,141,706,027	239 99.17%	2 0.83%	241	455,166,720 99.95%	213,135 0.05%	455,379,855
chris02.omni.hpcc.jp	358.392	257 45.57%	307 54.43%	564	982,207,400 55.49%	787,929,009 44.51%	1,770,136,409	220 100.00%	0 0.00%	220	468,455,040 100.00%	0 0.00%	468,455,040
chris03.omni.hpcc.jp	139.941	139 69.50%	61 30.50%	200	368,620,140 63.36%	213,128,640 36.64%	581,748,780	83 83.00%	17 17.00%	100	100,356,480 88.08%	13,576,320 11.92%	113,932,800
chris04.omni.hpcc.jp	371.841	250 43.18%	329 56.82%	579	954,856,040 54.56%	795,232,141 45.44%	1,750,088,181	230 99.57%	1 0.43%	231	494,789,760 100.00%	0 0.00%	494,789,760
chris05.omni.hpcc.jp	361.630	307 46.17%	358 53.83%	665	1,181,785,980 57.43%	876,113,593 42.57%	2,057,899,573	261 99.24%	2 0.76%	263	520,067,520 100.00%	0 0.00%	520,067,520
chris06.omni.hpcc.jp	379.417	553 78.11%	155 21.89%	708	938,882,596 60.63%	609,721,920 39.37%	1,548,604,516	313 99.37%	2 0.63%	315	457,270,303 100.00%	0 0.00%	457,270,303
chris07.omni.hpcc.jp	353.257	257 46.47%	296 53.53%	553	974,216,554 57.34%	724,673,067 42.66%	1,698,889,621	232 99.15%	2 0.85%	234	452,704,320 100.00%	0 0.00%	452,704,320
chris08.omni.hpcc.jp	346.787	310 35.67%	559 64.33%	869	1,142,271,360 40.45%	1,681,667,310 59.55%	2,823,938,670	240 98.77%	3 1.23%	243	481,752,000 99.98%	84,070 0.02%	481,836,070
total	2679.105	2,617 46.56%	3,004 53.44%	5,621	7,663,743,615 53.32%	6,709,268,162 46.68%	14,373,011,777	1,818 98.43%	29 1.57%	1,847	3,430,562,143 99.60%	13,873,525 0.40%	3,444,435,668

タスク毎のオプション

- Rakeタスクにつけ注釈の機能を流用
- Rakefile に記述したタスクの直前に書く

```
desc "ncore=4 allow=ourhost*"
rule ".o" => ".c" do
  sh "..."
end
```

```
(1..n).each do |i|
  desc "ncore=2 steal=no"
  file "task#{i}" do
    sh "..."
  end
end
```

```
ncore=integer       - タスクが使用するコア数
exclusive=no|yes    - ノード内で排他的にタスクを実行
allow=hostname      - 実行を許可するホスト名 （ワイルドカード使用可）
deny=hostname       - 実行を拒否するホスト名 （ワイルドカード使用可）
order=deny,allow|allow,deny - 評価の順序
steal=yes|no        - タスクスチール（入力ファイルが存在するホスト以外での実行）を許可
```