FUJITSU
shaping tomorrow with you

非Railsアプリのマルチデータベース対応と高速化の取り組み

(株)富士通システムズ・イースト

冨田昌宏

2015-11-12
RubyWorld Conference

自己紹介

FUJITSU

- 富田昌宏
- (株)富士通システムズ・イースト
- 長野勤務
- OSS推進フォーラム アプリケーション部会に参加
- 1998年から個人的にRubyを使用
- 2003年からRubyで製品開発

SYNCDOT

FUJITSU

- メールソリューション製品群
- Webメール / SMTP /POP / IMAPサーバー
- メールフィルター / メールアーカイブ
- 実行時使用OSS
  - Ruby / Apache / Postfix / MySQL
- 開発時使用OSS
  - Git / Redmine / Jenkins / Docker / RSpec / Cucumber

FUJITSU

非Railsアプリのマルチデータベース対応と高速化の取り組み

「非Rails」

FUJITSU

- 2003年から開発開始 ～ Rails以前
- 主な機能はメール処理。ウェブは補助的
- 製品のライフサイクルがRailsとあわない
- Railsのスピードは速すぎる

Railsでなければ何？

FUJITSU

開発当初

- Ruby 1.8.x
- CGI (Apache + mod_ruby)
- 生JavaScript
- 独自 O/R Mapper

ちなみに現在は…

FUJITSU

- Ruby 2.1
- Apache
- Passenger
- Rack
- Padrino(一部)
- jQuery(一部)
- Sequel

FUJITSU

ある日

FUJITSU

「MySQLだけでなくSymfowareにも対応せよ」

Symfoware

FUJITSU

- 富士通 RDBMS製品
- 2013年 PostgreSQL互換エディション追加
- Ruby や Postfix 等、多くの OSS からも使える
- **MySQLに特化した独自O/R Mapper では厳しい**

FUJITSU

Sequel

Sequel

FUJITSU

MySQL/PostgreSQL/その他RDBMSに対応した
O/R Mapper

- SQLをRubyスクリプトで記述

```
conn = Sequel.connect('mysql://user:pass@host/db')
conn[:users].insert(name: 'abc', age: 32)
conn[:users].where(name: 'abc').select(:age).first
  # => { age: 32 }
```

- Hash で条件の指定、値の設定、値の取得
- 複雑なクエリも記述可能

Sequel

FUJITSU

- Active Record パターンにも対応

```
class User < Sequel::Model
  # usersテーブルの構造を動的に取得
end
user = User.new
user.name = 'abc'
user.age = 32
user.save
```

Sequel

FUJITSU

- マイグレーション機能
- コネクションプール
- トランザクション/SAVEPONIT
- サブクエリ
- Join

Sequel対応

FUJITSU

- API, 構文の違いは吸収してくれる
- API 以外は地道にコツコツと…
- MySQLはゆるいがPostgreSQLは厳しい
  - 数値カラムと文字列との比較
  - カラムサイズ以上に文字列格納
  - DATEカラムに不正な日付を登録 等々
- その他: 大文字小文字を区別するかどうか / カラム型の違い / AUTOINCREMENT

性能測定

FUJITSU

Jenkins

SMTP

IMAP

POP

⋮

gnuplot

sar開始

サービス停止

サービス起動

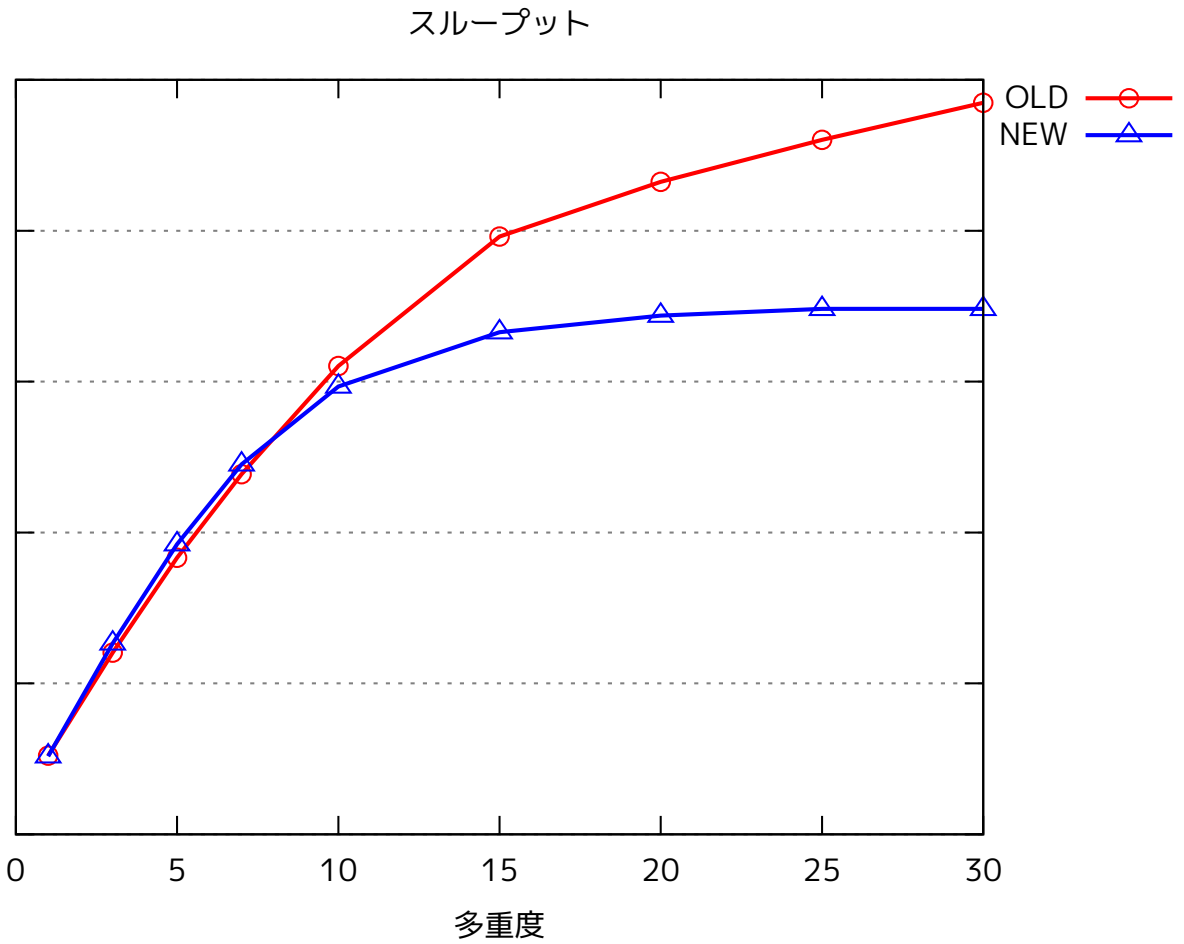
測定

sar停止

結果

FUJITSU

遅い

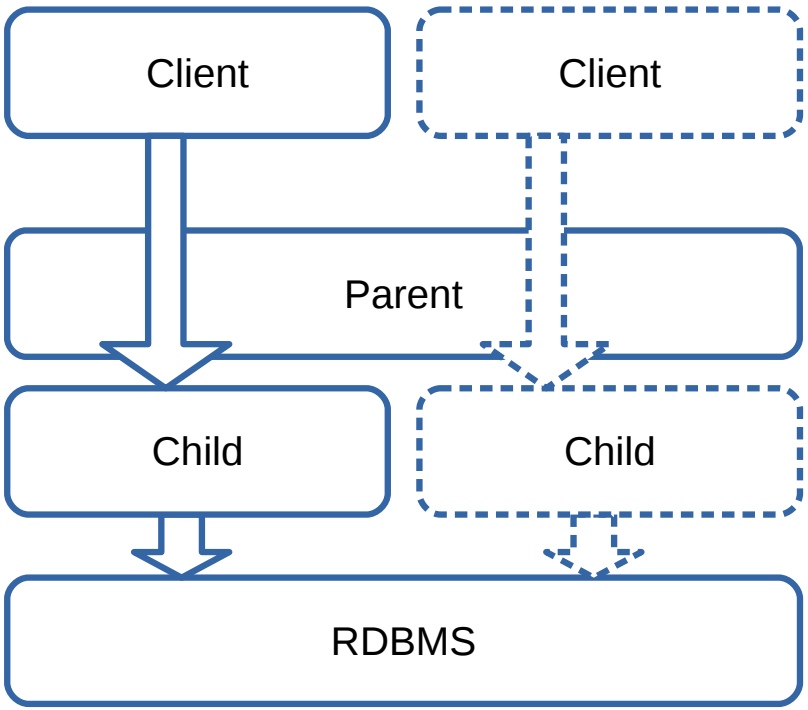


原因

- Sequel層が増えたためCPU使用率増
- Active Record でDB接続時にテーブル構造を取得

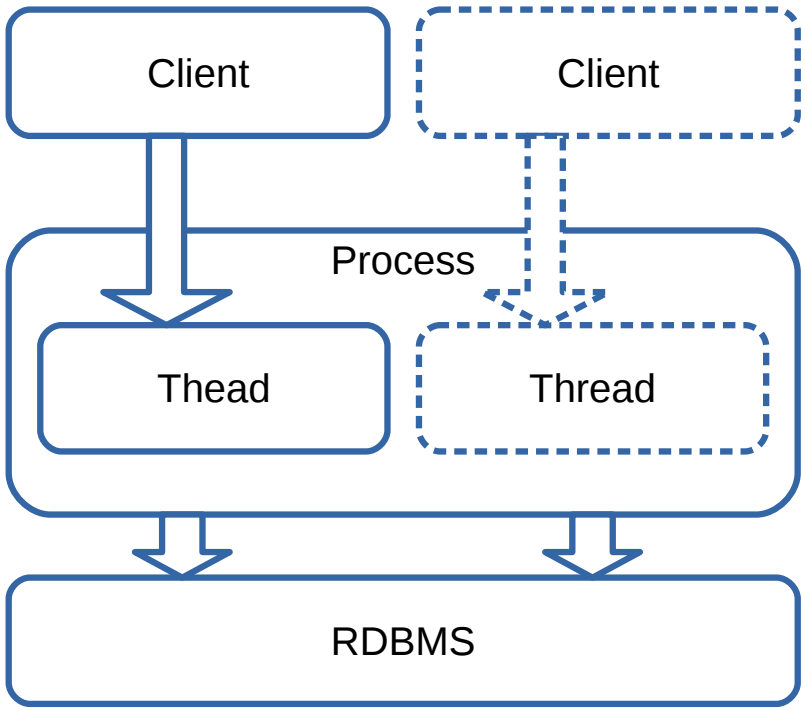
デーモンプロセス

- 古き良きUNIXデーモンモデル
  - クライアントからの接続を受けると子プロセスをfork
  - クライアントから切断されると子プロセスが終了



改善案

- マルチスレッド化
  - プロセス生成コスト減
  - コネクションプールを使用



スレッドセーフ化

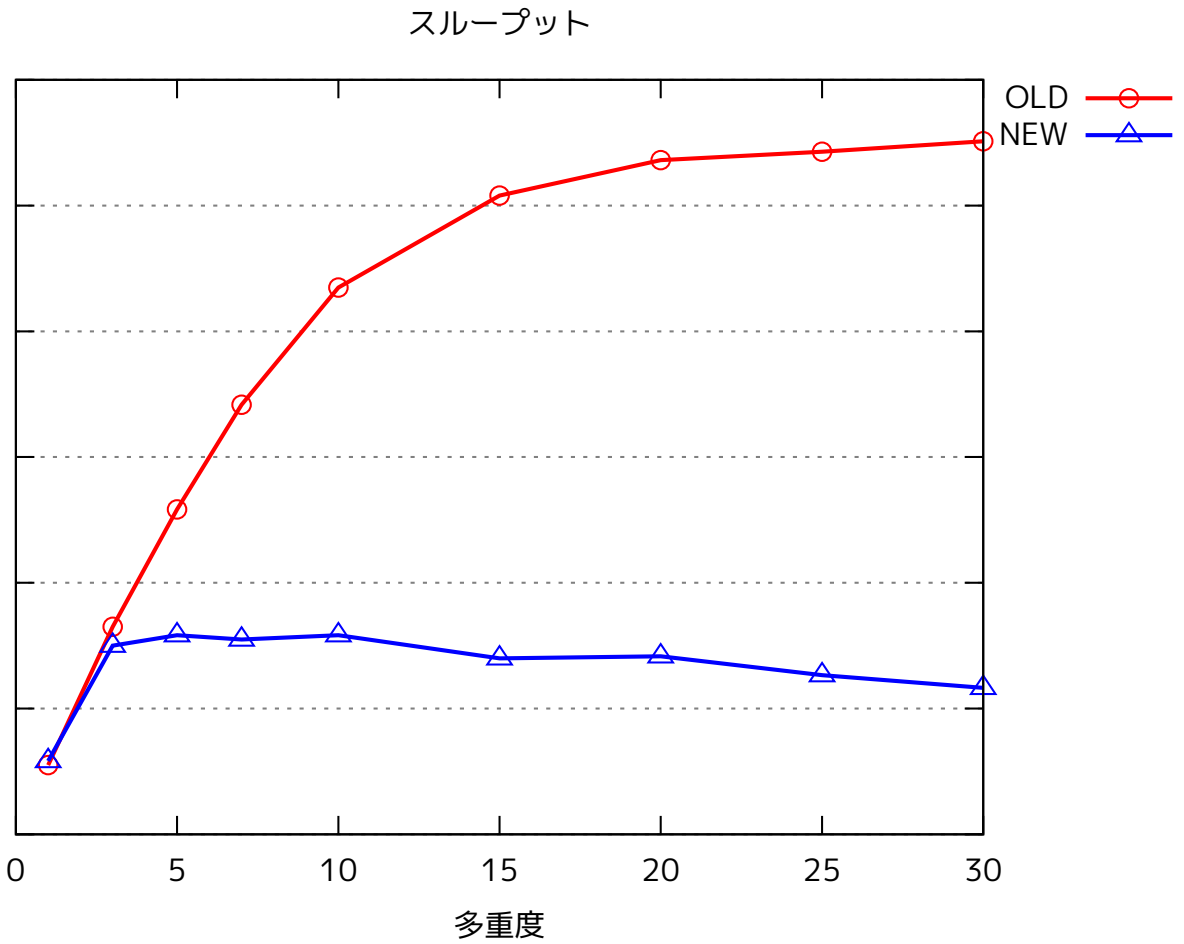
FUJITSU

- 既存コードをマルチスレッド対応
- 地道にコツコツと…
  - クラスメソッド
  - クラス変数
  - グローバル変数
  - シングルトンクラス
  - プロセスIDを一意性のために使用していないか

結果

FUJITSU

遅い



原因

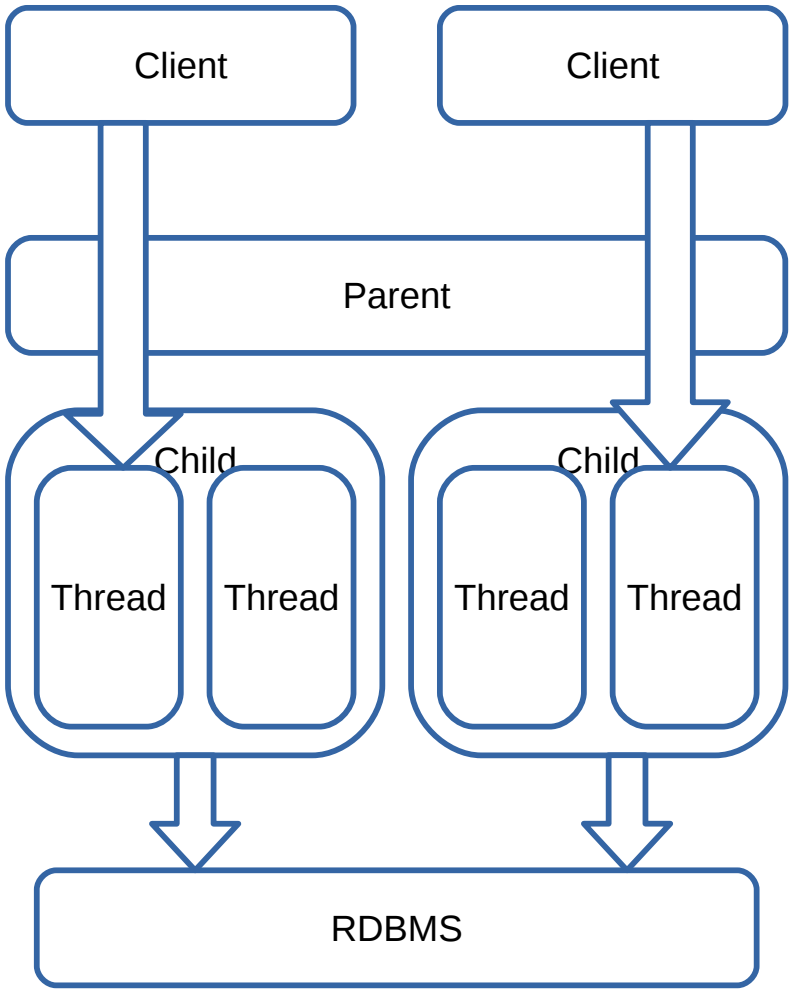
FUJITSU

- Rubyは1プロセスが同時に1CPUしか使えない
- 複数スレッドを作成しても同時に動けるのは1個だけ
- クライアントから大量の接続があると、サーバーのすべての CPU を使いきってないのに頭打ち
- マルチプロセス＆マルチスレッドにしたい

ParallelServer

FUJITSU

マルチプロセス＆マルチスレッドサーバーライブラリ



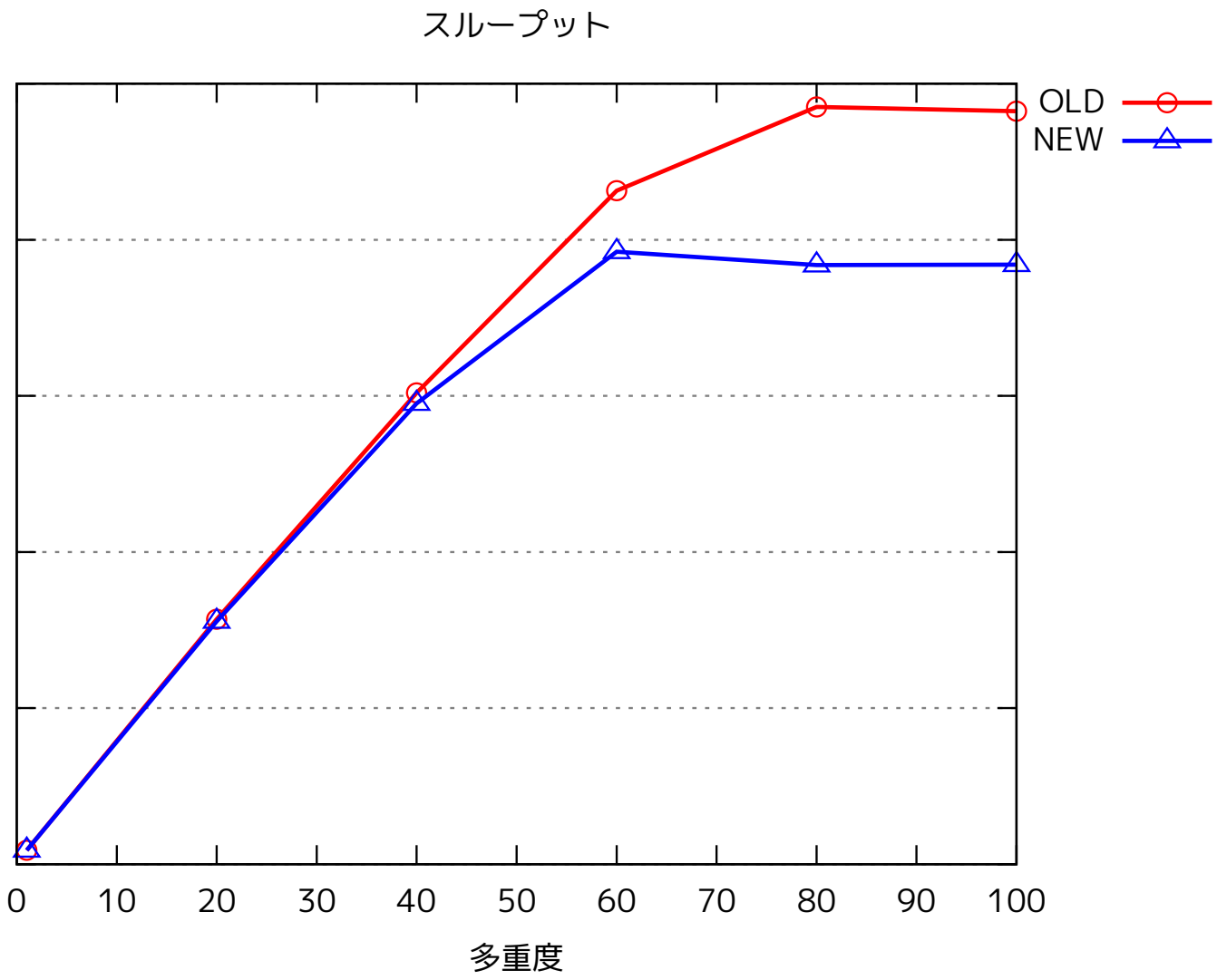
ParallelServer

FUJITSU

- 最小/最大プロセス数、プロセスあたりのスレッド数を指定して、プロセスが増減して動く

```
# TCP/IP 12345 ポートで待ち受ける
ParallelServer::Prefork.new(12345, max_threads: 5, max_processes: 10)
 .start do |sock, addr|
  # ここは子プロセス
  sock.puts 'Who are you?'
  name = sock.gets
  sock.puts "Hello, #{name}"
end
```

結果

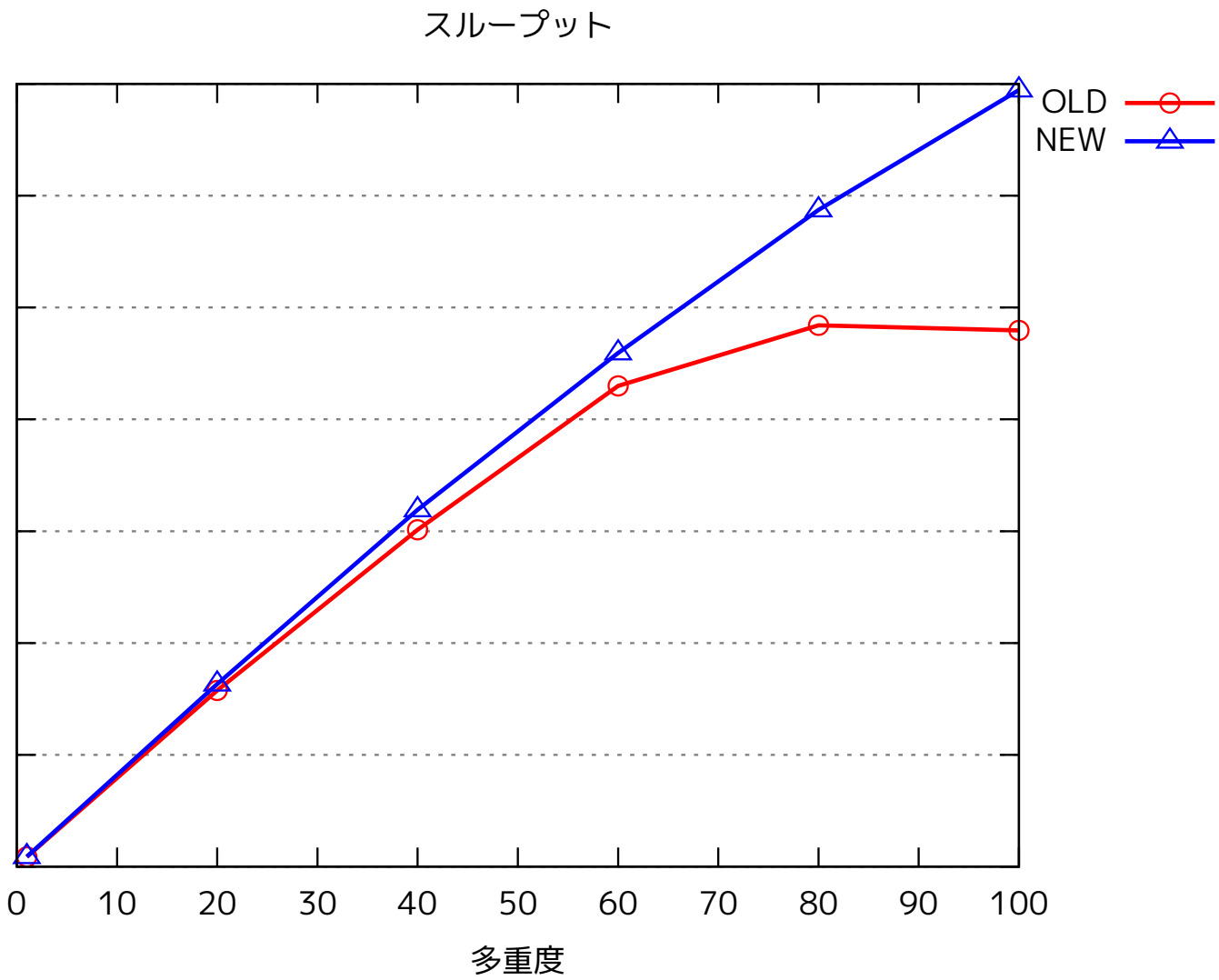


GC

FUJITSU

- メモリ使用量を抑えるためGC.startしてた
- マルチスレッドプロセスだと全スレッドが停止してしまう
- GC.start 廃止

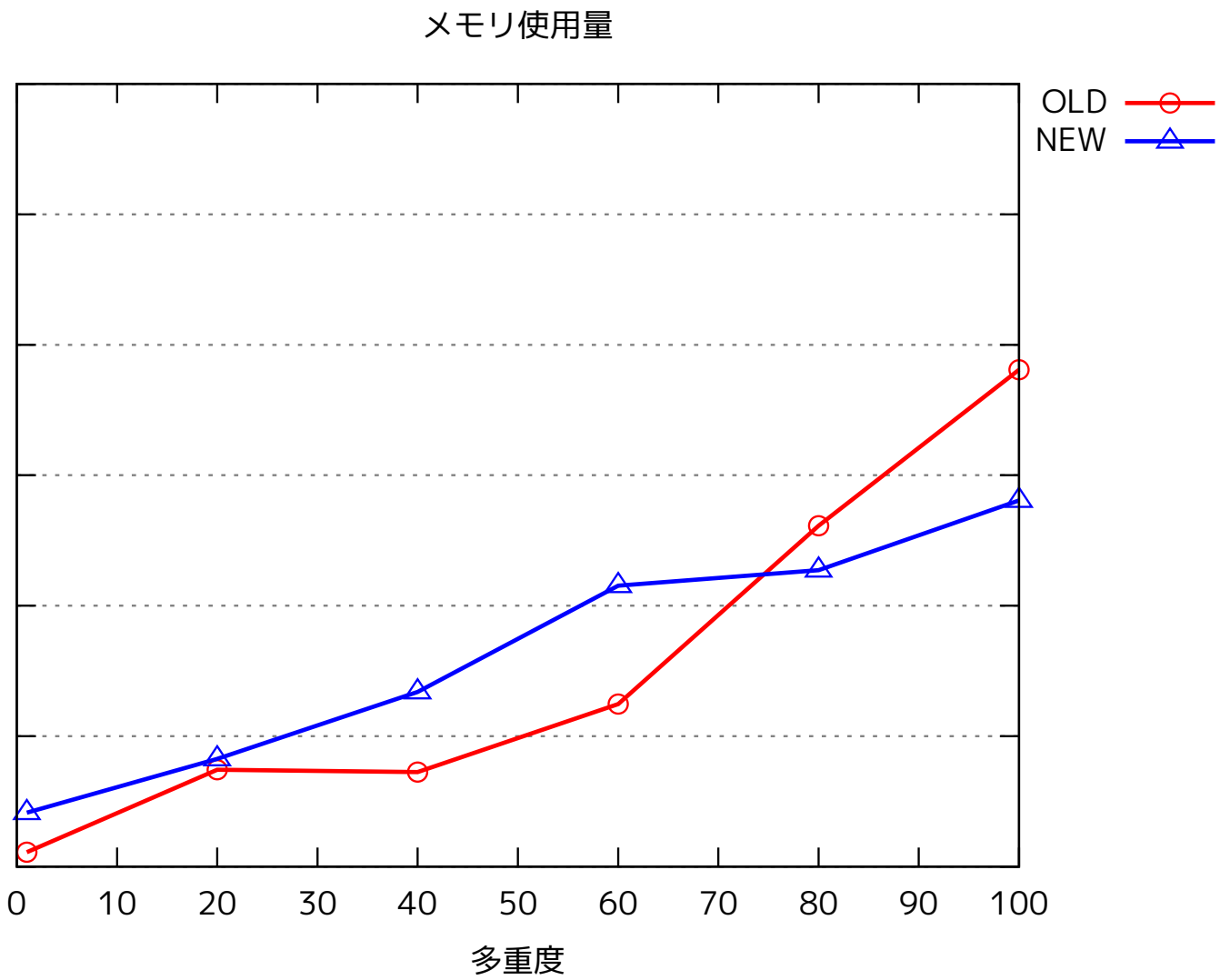
結果



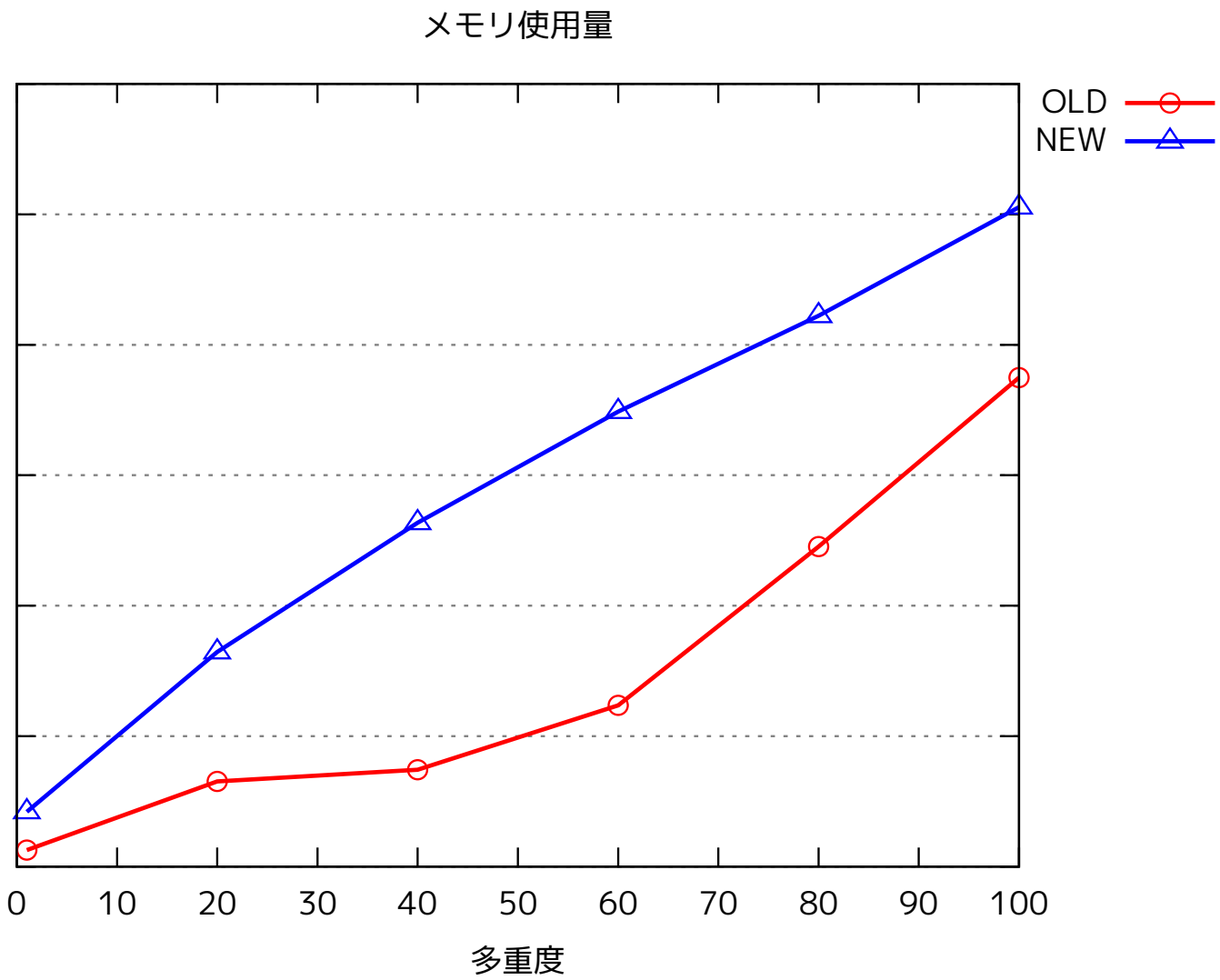
FUJITSU

メモリ使用量

GC.startあり



GC.startなし



メモリ使用量削減

FUJITSU

- Sequel のスキーマキャッシュを使用
- できるだけオブジェクトを作らない
- 無駄な処理を見直し
- Timeout 削減
- jemalloc

Timeout削減

FUJITSU

- データを1行出力する毎にタイムアウト処理
- 大量にスレッド生成＆破棄

```
msgs.each do |msg|
  ～処理～
  Timeout.timeout(999) do
    socket.write data
  end
end
```

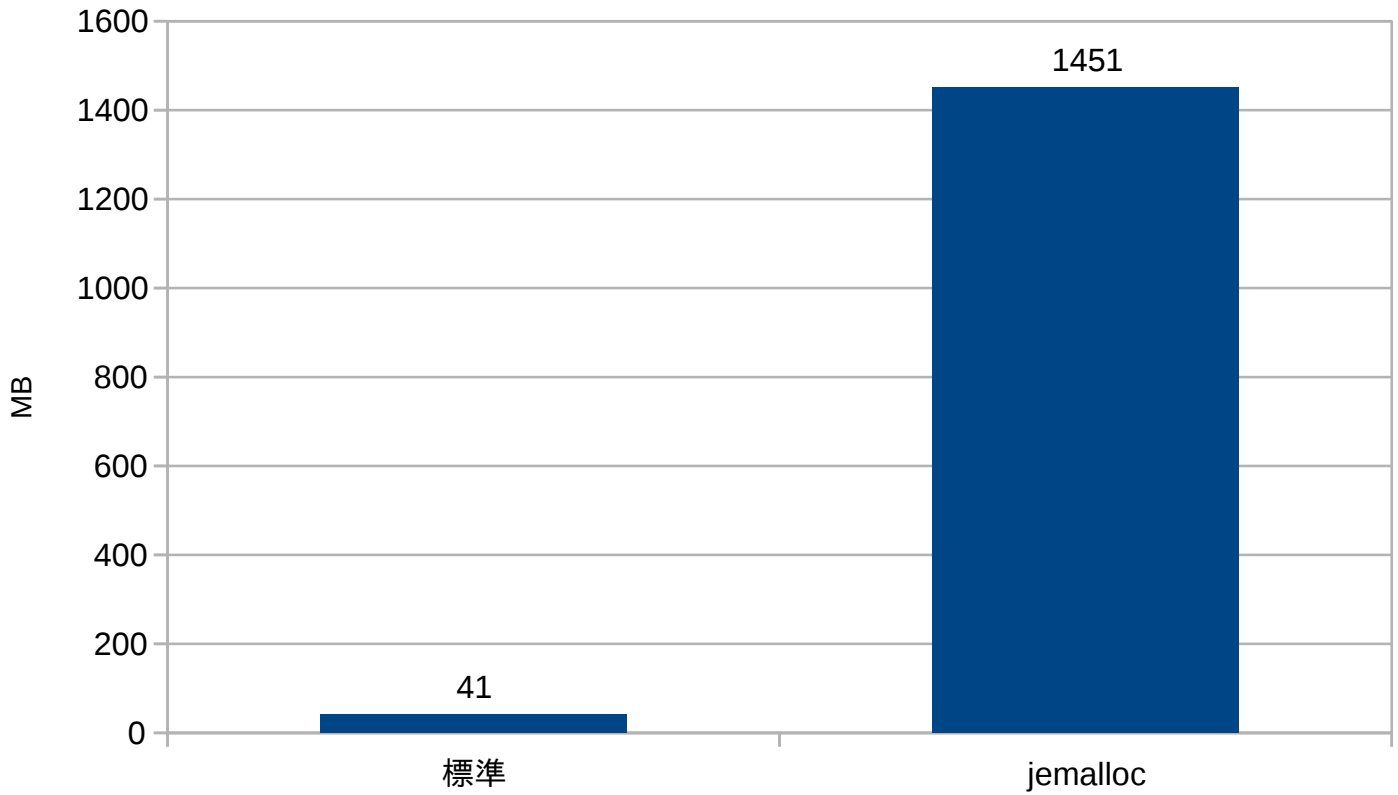
- 仕組みを変更して性能向上＆メモリ使用量削減

jemalloc

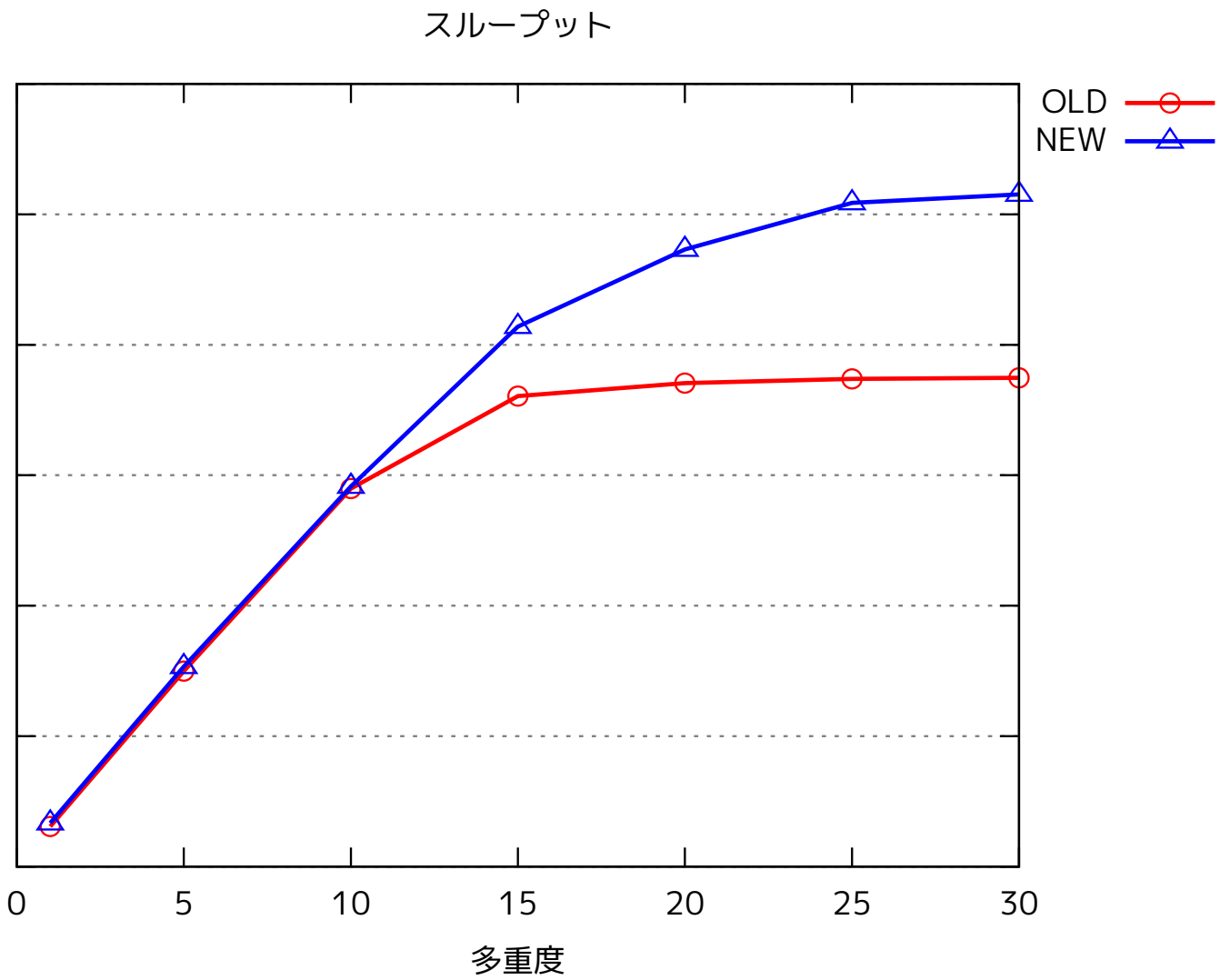
FUJITSU

- 逆効果

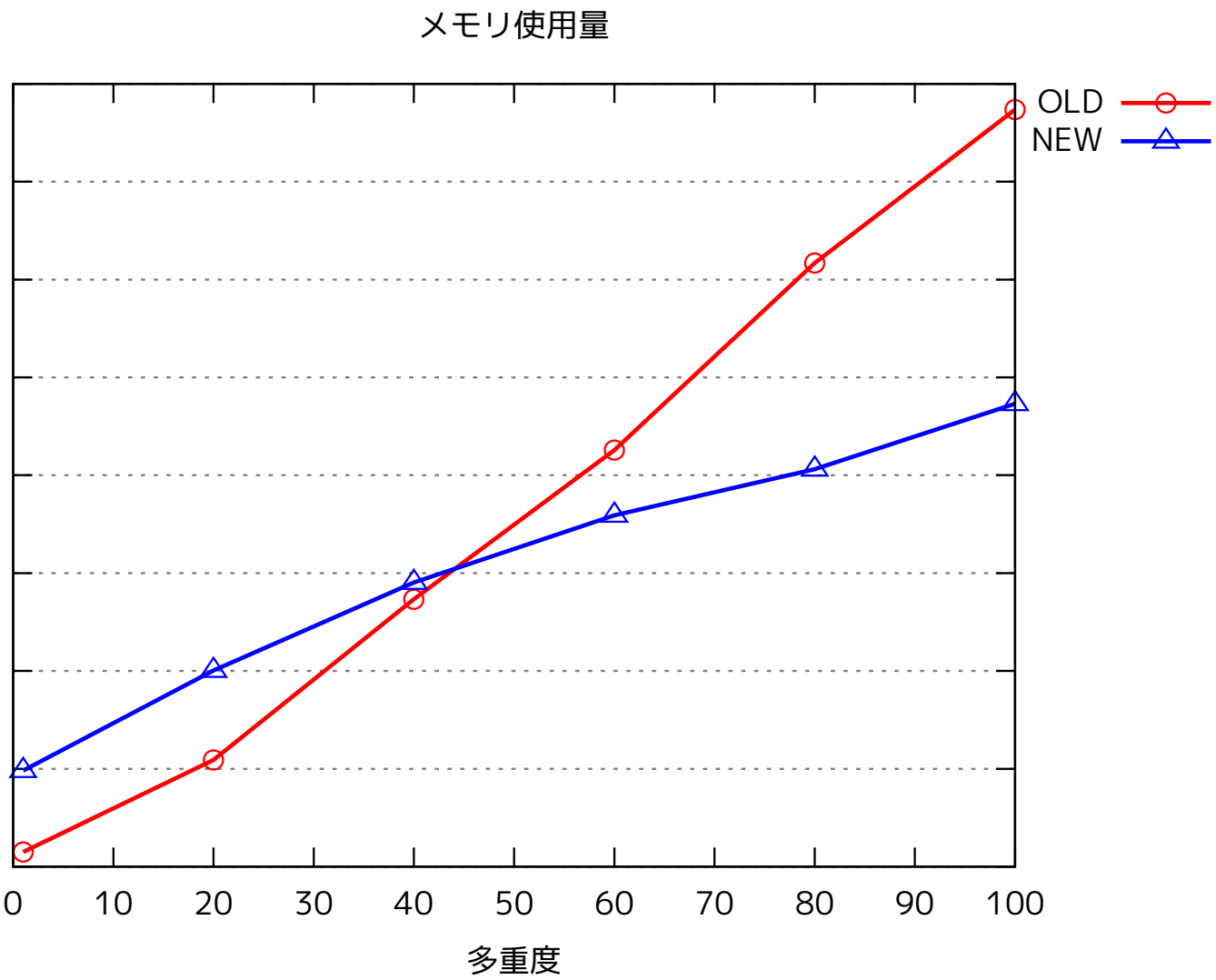
Rubyで1000スレッド作成したときのメモリ使用量の増分(MB)



最新状況（スループット）



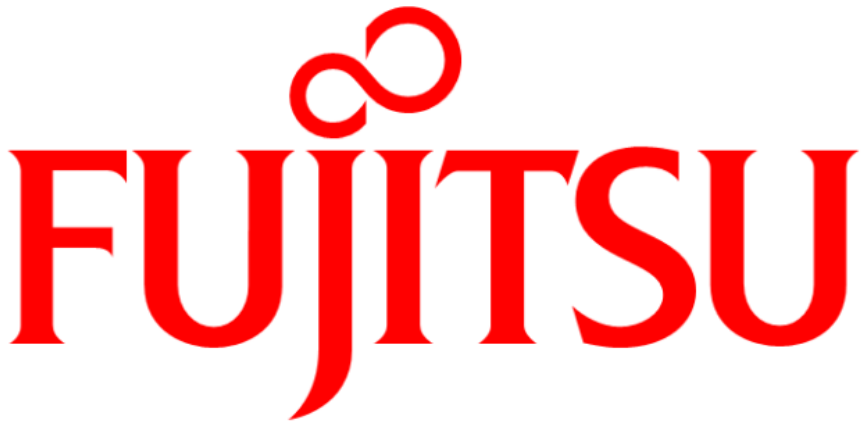
最新状況（メモリ）



まとめ

FUJITSU

- Sequel
  - Rubyの構文でSQLクエリを組み立てるのがかなり読みやすい
  - マルチデータベース対応でなくてもおすすめ
- マルチプロセス＆マルチスレッド
- 「推測するな、計測せよ」
- 計測～グラフ生成まで自動しておくと捗る



shaping tomorrow with you