# グローバルコンピューティングのスケジューリングのための性能評価システム

竹房 あつ子[†1,†2] 合田 憲人[†2] 松岡 聡[†2,†3]
中田 秀基[†4] 長嶋 雲兵[†5]

グローバルコンピューティングシステムが複数提案される一方，グローバルコンピューティングシステムのスケジューリング手法に対する考察が不十分である．これは大規模かつ再現性のある評価実験が困難であることに起因する．本稿ではグローバルコンピューティングシステムのスケジューリングアルゴリズムとそのフレームワークのための評価基盤を提供する Bricks システムを提案する． Bricks では様々な性能評価環境やスケジュールアルゴリズムおよびスケジューリングに関するモジュールを設定可能である．また，これらのモジュールを既存グローバルコンピューティングシステムモジュールに置き換えることで， Bricks 上での既存システムの機能試験を実施することができる．他システムの Bricks への組み込み例としてリソース情報の予測システムである NWS を用いて本システムの評価実験を行ったところ，Bricks が実環境と同等の挙動を示すことを確認した．さらに， NWS が Bricks 上で正常に動作したことから， Bricks が既存の外部モジュールに対して機能試験環境を提供できることを示した．

## Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms

Atsuko Takefusa,[?] Kento Aida,[†2] Satoshi Matsuoka,[†2,†3]
Hidemoto Nakada[†4] and Umpei Nagashima [†5]

While there have been several proposals of high performance global computing systems, scheduling schemes for the systems have not been well investigated. The reason is difficulties of evaluation by large-scale benchmarks with reproducible results. Our Bricks performance evaluation system would allow analysis and comparison of various scheduling schemes on a typical high-performance global computing setting. Bricks can simulate various behaviors of global computing systems, especially the behavior of networks and resource scheduling algorithms. Moreover, Bricks is componentalized such that not only its constituents could be replaced to simulate various different system algorithms, but also allows incorporation of existing global computing components via its foreign interface. To test the validity of the latter characteristics, we incorporated the NWS system, which monitors and forecasts global computing systems behavior. Experiments were conducted by running NWS under a real environment versus the simulated environment given the observed parameters of the real environment. We observed that Bricks behaved in the same manner as the real environment, and NWS also behaved similarly, making quite comparative forecasts under both environments.

## 1. はじめに

グローバルコンピューティングは広域ネットワーク上に分散した計算 / 情報リソースを活用して大規模計算を実現する計算技術であり，近年これを目的としたシステムが複数提案されている[1)]．各システムではグローバルコンピューティングを効率的に行うために，計算リソースの基本性能，利用状況およびリソース間のネットワークの状況をモニタし，それらの情報をもとに遠隔ユーザの要求するタスクを適切に割り当てる何らかのスケジューリングアルゴリズムおよびそれに基づくスケジューリングフレームワークを実装している．

グローバルコンピューティングシステムにおけるスケジューリング手法の評価では，その特性の調査するために様々な

- ネットワークのトポロジ，バンド幅，混雑度，変動

†1 日本学術振興会特別研究員
Research Fellow of the Japan Society for the Promotion of Science
†2 東京工業大学
Tokyo Institute of Technology
†3 科学技術振興事業団
JST
†4 電子技術総合研究所
Electrotechnical Laboratory
†5 産業技術融合領域研究所
National Institute for Advanced Interdisciplinary Research

- 計算リソースのアーキテクチャ，性能，負荷，変動等を想定した大規模環境での評価を複数回行い，その結果の統計情報により各手法の優劣を判断する必要がある．また，他の研究者により提案・開発された複数のスケジューリングアルゴリズムやそのモジュールを比較する際，各評価実験に対してネットワークや計算リソース等の評価環境の状況が同じように変化しなければ，実験結果がスケジューリングの優劣を示すのか，評価環境の影響によるものなのか，判断し難い．すなわち，公平な比較を行うためには「実験環境の再現性」が求められる．

一方，現在行われている実環境でのスケジューリング手法の評価では評価環境の規模が制約され，実際にその再現性もないことにより，複数スケジューリングアルゴリズムの評価・比較が困難である．また，グローバルコンピューティングリソースのモニタ・予測情報はスケジューリングに大きく影響するにもかかわらず，これらの情報を提供するモジュールの実環境上での機能試験の実施コストが高いという問題がある．

本稿では，グローバルコンピューティングシステムのスケジューリング手法およびそのフレームワークの評価基盤を提供するシステム Bricks[2] を提案する． Bricks はグローバルコンピューティング環境のシミュレータであり，様々なネットワークトポロジ，計算サーバアーキテクチャ，通信モデルおよびスケジューリングフレームワークの各モジュールを設定可能にする． Bricks のユーザは Bricks が提供する Bricks 環境設定スクリプトにより柔軟に環境設定を行い，各スケジューリングアルゴリズムの再現性のある性能評価を行うことができる．また，スケジューリングモジュールへの SPI(Service Provider Interface) を提供しているため，グローバルコンピューティングシステム開発者はこの SPI を実装することで Bricks システム上でスケジューリングフレームワークの既存プログラムモジュールの機能試験を行うことができる．

本稿では，これを実証するためにグローバルコンピューティングのリソース予測システムである NWS(Network Weather Service)[3),4)] を用い，他のスケジューリングフレームワークモジュールの Bricks への組み込み試験を行った．本システムと NWS による評価実験では，通信スループットの実測値を用いた通信モデルにより Bricks が実環境とほぼ同等の挙動を示す評価環境が提供できることを確認した．また， NWS が実環境上での運用と同様に Bricks 上で正常に動作したことから， Bricks 上で既存システムのスケジューリングに関するモジュールの機能試験が可能であることを示した．

## 2. Bricks の概要

Bricks は Java で実装されたグローバルコンピューティングシステムのスケジューリング手法の性能評価シミュレータであり，スケジューリングアルゴリズムの評価とそのフレームワークの運用テストを行うための大規模かつ再現性のある評価実験環境を提供することを目的としている． Bricks の特徴は以下のようにまとめられる．

- Bricks はグローバルコンピューティング環境とスケジューリングユニットから構成されている（図 1）． Bricks ではシミュレーションにおける
  - スケジューリングアルゴリズム
  - スケジューリングに関する各モジュール
  - クライアント，サーバ，ネットワークの構成
  - ネットワーク / サーバでの処理方法（待ち行列）
  - シミュレーションで用いられる乱数分布

  等の設定を Bricks 環境設定スクリプトにより自由に組み立てられることから Bricks と名付けた． Bricks のユーザは Bricks 環境設定スクリプトで記述した環境設定を実行時の入力として与えることにより，様々な環境下でのスケジューリングアルゴリズムの評価実験ができる．
- Bricks によるシミュレーション上でスケジューリングに要するグローバルコンピューティングリソース情報を得るため， Bricks ではスケジューリングユニットを提供する．スケジューリングユニットは， Globus[5)] ， Legion[6)] ， Ninf[7)], AppLeS[8)] 等の既存のスケジューンリグフレームワークをモデルとしており，それらで提供されるモジュールで構成される．スケジューリングユニットの各モジュールはシミュレーション環境下でリソース状況のモニタ，予測，スケジューリング等を行う．モジュール間のインタフェースも同様に既存グローバルコンピューティングシステムでの要求をもとに設計されており，各モジュールは容易に Java で記述されたプログラムモジュールに置き換えることができる．すなわち， Bricks スケジューリングユニット SPI に従って新たなスケジューリングアルゴリズムを実装し， Bricks 上で様々な環境を想定した評価を行うことができる．さらに， SPI とのグルーインタフェースを用意することにより，各プログラムモジュールを既存グローバルコンピューティングシステムの外部スケジューリングモジュールに置き換えることができる． Bricks はモニタや予測等のスケジューリングに関する情報に加え，シミュレーション中の仮想時刻を外部モジュールに提供し，外部モ

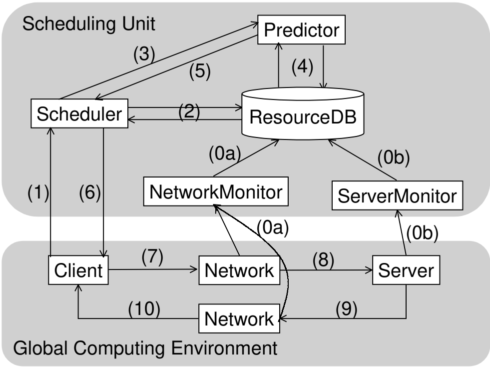


図1 Bricks アーキテクチャ
Fig. 1 The Bricks Architecture.

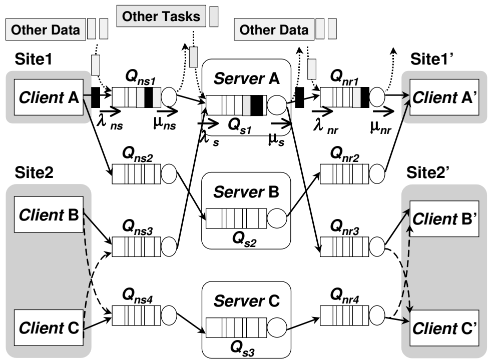


図2 グローバルコンピューティング環境モデル
Fig. 2 An example of the Global Computing Environment Model.

ジュールでの処理結果は Bricks に返される．Bricks システムはまだ試験的段階であるが，既存のグローバルコンピューティングシステム NWS を Bricks に組み込み，その予備的評価実験を行うことができた．

Bricks はグローバルコンピューティング環境部分に待ち行列ネットワークモデルを採用している[9]．図 **2**はシミュレーション環境の一例であり，クライアントからサーバへのネットワーク $Q_{ns}$，サーバからクライアントへのネットワーク $Q_{nr}$，サーバ計算機 $Q_s$ を待ち行列で表している．$Q_{ns}$，$Q_{nr}$，$Q_s$ でのサービス率は各ネットワーク / サーバ計算機の通信スループット / 処理性能を表す．

## 3. Bricks のアーキテクチャ

Bricks はグローバルコンピューティング環境とスケジューリングユニットにより構成される（図 **1**）．

### 3.1 グローバルコンピューティング環境

グローバルコンピューティング環境はグローバルコンピューティングをシミュレートする環境を提供し，次のモジュールにより構成される．

**Client** グローバルコンピューティングにおけるユーザを表し，グローバルコンピューティングのタスクを発行する．

**Network** ユーザの計算機とグローバルコンピューティングの計算リソースをつなぐネットワークを表す．Bricks ではこのネットワークを待ち行列で表現する．実際のネットワークの挙動を表すために，バンド幅，混雑度とその分散を指定するパラメータを用意している．

**Server** グローバルコンピューティングの計算リソースである計算サーバを表す．ネットワーク同様にサーバも待ち行列で表す．サーバの挙動を表現するパラメータとして，サーバの性能と負荷，およびその分散がある．

次に，Bricks におけるクライアントからのタスクのモデル，通信モデル，サーバのモデルについて説明する．

#### 3.1.1 タスクモデル

シミュレーションにおけるタスクにとって重要なことは，通信時間，計算時間がどの程度かかるかということである．現在の Bricks の実装では，これらを調べるうえで必要な情報

- タスク実行に要する通信量（送信 / 受信）
- タスクの実行時の演算数

をパラメータとしてタスクを表現する．

#### 3.1.2 通信モデル

Bricks ではシミュレーション実行時の設定により，様々な通信モデルが実現できる．現在 Bricks で表現できる代表的な通信モデルは以下の 2 通りある．

1 つめは，ネットワークにはグローバルコンピューティングシステム以外のシステムから流されるデータ（外乱）があることを想定し，通信スループットを外乱のデータの到着率を変化させて表現するモデル[9] である．このモデルでは，いくつかのパラメータ指定のみでシミュレーションを行うことができるが，精度を高めるためにシミュレーションの粒度を小さく設定すると実行コストが非常に高くなってしまう．

2 つめのモデルは実環境で計測された通信スループットをもとに，ネットワークの待ち行列のサービス率を決定するモデルである．すなわち，実環境上で定期的にあるネットワークの通信スループットを測定する．次に，シミュレーションでそのネットワークをシミュレートする待ち行列のサービス率をその測定時刻と測定された通信スループット値を読み取りながら変化させ，実際のネットワークでの通信スループットの変動を表現する．実環境での測定値の代わりに，他の通信モデルとして提案されている自己相似性や確率的モデルにより生成され

た値を採用することもできる．このモデルでは事前に通信スループットの測定を要するが，実際のネットワークと同じ挙動を表現できる，1 つめのモデルより実行コストが低いという利点がある．

これらの通信モデルで適切にパラメータを設定することよりグローバルコンピューティングシステムにおける様々なネットワークの挙動が表現できるが，シミュレーションの精度，実行速度，ユーザの利便性等を向上させるために Bricks システムを拡張していく予定である．

#### 3.1.3 計算サーバモデル

計算サーバは現在 FCFS で処理することを前提としている．サーバもネットワーク同様に待ち行列で表され，外乱のジョブの平均演算数，到着間隔を指定することによりその稼働率を決定する．サーバも負荷の変動を実環境で測定された数値により表現することができる．

### 3.2 スケジューリングユニット

グローバルコンピューティングシステムのスケジューリングをサポートする枠組みとして，Bricks ではスケジューリングユニットを提供している．スケジューリングユニットは既存のグローバルコンピューティングシステムをモデルとしてグローバルコンピューティングのスケジューリングで必要とされる各モジュールにより構成される．

**NetworkMonitor** グローバルコンピューティング環境におけるネットワークの通信スループット，レイテンシ等，ネットワークの状況をモニタする．得られた情報は ResourceDB に格納する．

**ServerMonitor** グローバルコンピューティング環境における計算サーバの性能，負荷，稼働率をモニタし，得られた情報を ResourceDB に格納する．

**ResourceDB** グローバルコンピューティングシステムにおける総合データベースである．各モニタからリソース情報が格納され，Predictor，Scheduler に対してその情報を提供する．

**Predictor** ResourceDB からリソース情報を入手し，そのリソースの可用性を予測する．この予測は，新たに投入されるタスクのスケジューリングに利用される．また，Predictor はネットワークに関する予測を行う NetworkPredictor と計算サーバに関する予測を行う ServerPredictor により構成されており，同時に複数の予測アルゴリズムを提供する Predictor を設定することができる．

**Scheduler** ResourceDB で管理されている情報と Predictor で予測された情報をもとに，ユーザのタスクを利用可能なサーバ計算機の中から最適なサーバに割り当てる．

グローバルコンピューティング環境部分同様スケジューリングユニットの各モジュールは Java で実装されており，各コンポーネントは容易に組み替え可能である．

Bricks ではスケジューリングモジュール間の SPI を提供しており，スケジューリングユニットの各モジュールはユーザの実装したモジュールと置換することができる．すなわち，新たなスケジューリングアルゴリズムを実装したスケジューラや NWS のような既存の Predictor を Bricks に組み込んで，Bricks 上でその評価が行える．

### 3.3 Bricks の実行の流れ

Bricks によるシミュレーションの手順を以下に示す．この手順は図 **1** に対応している．

- **(0a)** NetworkMonitor は定期的に Network にプローブパケットを流し，Network の通信スループット，レイテンシを測定する．測定結果は ResourceDB に格納する．
- **(0b)** ServerMonitor は定期的に Server に問い合わせて，Server の負荷情報を調べる．結果は Network-Monitor 同様，ResourceDB に格納する．
- **(1)** Client でグローバルコンピューティングのタスクが発生すると，Client は Scheduler にタスクを投入すべき Server を問い合わせる．その際，Client は Scheduler にタスクの情報を提供する．
- **(2)** Scheduler は ResourceDB にグローバルコンピューティングシステム上でこの Client のタスクを実行可能な Server を問い合わせる．
- **(3)** Scheduler は (2) で問い合わせた Server とその Server への Network に関するリソース状況の予測値を Predictor に問い合わせる．
- **(4)** Predictor は ResourceDB に Server およびその Server への Network のモニタ情報を問い合わせ，その情報をもとに各リソース状況を予測する．
- **(5)** Predictor はリソース状況の予測が終了すると，Scheduler に予測値を返す．
- **(6)** Scheduler はこの予測値をもとにタスクを発行する Server を決定し，Client に通知する．
- **(7)** Client は決定した Server への Network にタスクを投入する．この際，タスクの送信データは論理パケットサイズに分割する．論理パケットサイズはシミュレーションの精度を左右するパラメータであり，パケットサイズを小さく設定するとより精密なシミュレーションが行えるが，外乱を想定した通信モデルではシミュレーションコストが非常に大きくなってしま

う☆．

（8）　Client から投入されたパケットが Network の待ち行列のサーバ上で

[送信データ量] / [Network のスループット]

時間処理されると，パケットは Server に送られる．

（9）　分割されたタスクのデータがすべて Server に到着すると，Server でタスクが実行される．Server の待ち行列のサーバ上で

[タスクの演算数] / [Server の性能]

時間処理されてタスクの実行が終了すると，Server はその結果をタスクを発行した Client への Network に投入する．ただし，Client からの送信時と同様にタスクのデータは論理パケットサイズに分割される．

（10）　Server から投入されたパケットが

[受信データ量] / [Network のスループット]

時間 Network の待ち行列のサーバ上で処理されると，Client にパケットが送られる．すべてのパケットが Client に送られると，そのタスクの実行が終了する．

## 4. 外部モジュール組み込みインタフェース

**3.2**で述べたように，Bricks のユーザはスケジューリングユニットの各モジュールを既存のグローバルコンピューティングシステム等の外部プログラムモジュールに置き換え，それらの Bricks 上での機能試験を行うことができる．これは，スケジューリングユニットの各モジュールにスケジューリングに関する様々な情報を授受するための SPI を提供することにより実現している．

### 4.1 スケジューリングユニット SPI

Bricks では，スケジューリングユニットのデフォルトのプログラムモジュールを提供しているが，各モジュールは Bricks の提供する SPI を実装した Java で記述されたプログラムモジュールに置換可能である．図 **3**にスケジューリングユニットのモジュール ResourceDB，NetworkPredictor，ServerPredictor，Scheduler の SPI を示す☆☆．Bricks の実装ではスケジューリングユニットの各モジュールに共通するインタフェースの定義とメソッドの実装を `abstract` クラス内で行っているが，図 **3**では便宜上 `interface` 文を用いた疑似コードとして表す．図 **3**の `NetworkInfo` は通信スループット，レイテンシ等のネットワークの情報を表し，`ServerInfo` はサーバ計算機の負荷平均値や CPU 稼働率等を表す．リソース情報に関するパラメータはそれぞれ Java のクラス内で定義しているため，柔軟に他のパラメータを加えることが可能である．

ResourceDB では，NetworkMonitor および ServerMonitor で観測されたリソース情報を ResourceDB に格納するときと，Scheduler，Predictor から ResourceDB に対してリソース情報を要求するときのインタフェースを提供する．NetworkPredictor および ServerPredictor では，Scheduler が予測情報を要求するためのインタフェースを提供する．これらのメソッド内で，ユーザの提案する ResourceDB に格納された情報を利用した予測アルゴリズムを実装することができる．Scheduler ではクライアントがスケジューリング情報を取得するためのインタフェースを提供する．クライアントがこのメソッドを呼び出す際，クライアントのタスクに関する情報を Scheduler に渡すため，Scheduler では ResourceDB に格納されている情報，Predictor による予測情報，クライアントのタスクの情報を利用して，多様なスケジューリングアルゴリズムを実装することができる．また，スケジューリングユニットの各モジュールおよびそのインタフェースは一般的なグローバルコンピューティングをモデルとしているため，既存のグローバルコンピューティングシステムモジュールに対するグルーインタフェースを実装することで，容易に Bricks に組み込むことができる．

例として，Bricks に NWS を組み込む場合の Bricks スケジューリングユニット SPI と NWS の API の関係を図 4に，その際の Bricks 環境設定スクリプトでの指定例を図 **5**に示す．`ResourceDB` 行，`Predictor` 行，`Scheduler` 行でそれぞれ ResourceDB，Network/ServerPredictor，Scheduler の宣言を行う．各行でシミュレーションの際に用いるスケジューリングモジュールの宣言とそのモジュールに必要なパラメータを記述する．また，ユーザが定義したスケジューリングモジュールのための初期化ルーチンは，Java のリフレクション API を用いて Bricks システムから自動的に呼び出される．

### 4.2 NWS システムの組み込み

本稿では，外部モジュールの Bricks による評価例として，UCSD で開発されたグローバルコンピューティングのリソース状況のモニタおよび予測を行うシステム

☆ 限られた計算パワーで大規模評価実験を想定したシミュレーションを行いたい場合は，論理パケットサイズを大きく設定して多少精度を犠牲にするか，2 つめのモデルを用いるという方法で解決する．

☆☆ 現在の Bricks の実装では，IP ソケット通信に対するサポートを行っていないため，ユーザは Bricks の提供する NetworkMonitor，ServerMonitor モジュールを用いてシミュレーションを行わなければならない．ただし，Bricks 環境設定スクリプトによりモニタリング間隔，ネットワークのプローブ時のデータサイズなどが指定可能となっている．

```
interface ResourceDB {
  // stores networkInfo
  void putNetworkInfo(
    NetworkInfo networkInfo
  );
  // stores serverInfo
  void putServerInfo(
    ServerInfo serverInfo
  );
  // provides NetworkInfo between
  // sourceNode and destinationNode
  NetworkInfo getNetworkInfo(
    Node sourceNode,
    Node destinationNode
  );
  // provides ServerInfo of serverNode
  ServerInfo getServerInfo(
    ServerNode serverNode
  );
  // implements process when a simulation
  // finishes
  void finish();
}

interface NetworkPredictor {
  // returns Prediction of the Network
  // between sourceNode and destinationNode
  NetworkInfo getNetworkInfo(
    double currentTime,
    Node sourceNode,
    Node destinationNode,
    NetworkInfo networkInfo
  );
}

interface ServerPredictor {
  // returns Prediction of serverNode
  ServerInfo getServerInfo(
    double currentTime,
    ServerNode serverNode,
    ServerInfo serverInfo
  );
}

interface Scheduler {
  // returns serverNodes for the request
  ServerAggregate selectServers(
    double currentTime,
    ClientNode clientNode,
    RequestedData data
  );
}
```

図 3 スケジューリングユニット SPI の概要
Fig. 3 Overview of the Scheduling Unit SPI.

NWS の Bricks への組み込みを行った．NWS は他システムへの C 言語の API を提供しており，AppLeS[8)]，Globus[5)]，Legion[6)]，Ninf[7)] 等のシステムで予測機構として利用する試みがある．NWS の Bricks への組み込みでは，我々が開発した NWS の C 言語 API と同等

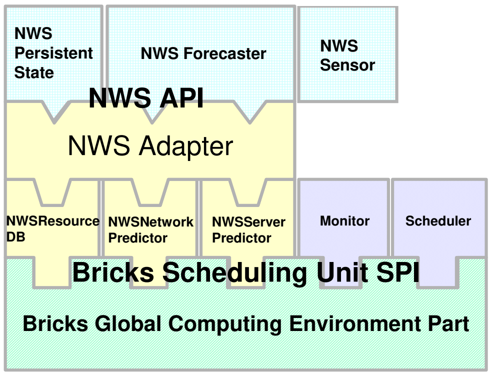


図 4 Bricks SPI と NWS API の相関
Fig. 4 The Interrelationship between the Bricks SPI and the NWS API.

```
# ----- ResourceDB declaration -----
# ResourceDB [name] [selected ResourceDB]
#  (arguments)
ResourceDB db NWSResourceDB(\
 sap4.is.ocha.ac.jp:8050, \
 sap4.is.ocha.ac.jp:8070, true, 1000, 1000)

# ----- Precictor declaration -----
# Predictor [name] ([ResourceDB name],\
#  [selected Predictor](arguments), ..)
Predictor pr (db, \
 NWSNetworkPredictor(sap4.is.ocha.ac.jp:8070),\
 NWSServerPredictor(sap4.is.ocha.ac.jp:8070))

# ----- Scheduler declaration -----
# Scheduler [name] [selected Scheduler]
#  ([Predictor name], arguments))
Scheduler scheduler \
 LoadThroughputScheduler(pr, 100)
```

図 5 Bricks 環境設定スクリプトの例
Fig. 5 Example of the Bricks script.

の機能をもつ NWS の Java API[10)] を用いた．

NWS は次の 4 つのモジュールにより構成される．

**Persistent State** 測定された情報を格納するストレージであり，Bricks における ResourceDB に対応する．

**Name Server** 各モジュールの TCP/IP ポート番号や IP／ドメインアドレス等の参照に用いる．

**Sensor** ネットワーク，計算サーバの情報をモニタするもので，Bricks における NetworkMonitor，ServerMonitor に対応する．

**Forecaster** リソース情報を予測するもので，Bricks の Predictor に対応する．NWS では複数の予測アルゴリズムにより予測を行い，前回の予測エラーが少ないアルゴリズムの予測結果が予測値として採用される．

Bricks の組み込みでは，NWS の Persistent State，Forecaster を用い，Bricks 内で測定された情報を Per-

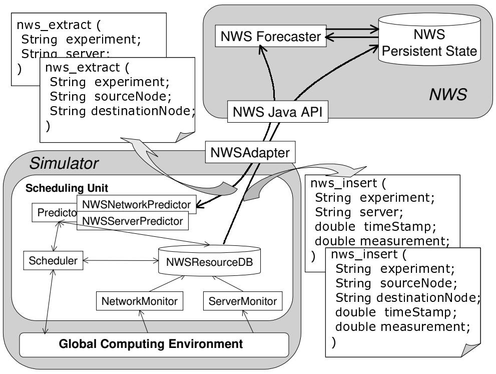


図 6　NWS の Bricks への組み込み
Fig. 6　Incorporating the NWS modules into Bricks.

sistent State に格納して Forecaster で予測されたデータをスケジューリングの際に利用する．これらのモジュールへのグルーインタフェースとして NWSResourceDB，NWSNetworkPredictor，NWSServerPredictor と，NWS Java API と Bricks でのデータタイプの変換を行う NWSAdapter を用意した．

図 4，6に Bricks をシミュレータに組み込んだ様子を示す．Bricks 内で NetworkMonitor / ServerMonitor がモニタした情報を NWSResourceDB に格納すると，NWSResourceDB は NWSAdapter 経由で Persistent State にその情報を格納する．また，Bricks 内で予測情報が必要とされると，NWSNetworkPredictor / NWSServerPredictor が NWSAdapter 経由で NWS Forecaster から予測値を取り出す．

## 5. Bricks の評価実験

評価実験では，（1）Bricks のグローバルコンピューティング環境がスケジューリングアルゴリズムを評価するための再現性のある試験環境を提供し，実際のネットワークの挙動が表現可能な性能評価ツールであることを示す．また，Bricks の提供するスケジューリングユニットのモジュールの代わりにスケジューリングに関する既存外部モジュールである NWS を用い，NWS の Bricks 上での正常な動作を確認することで（2）Bricks のユーザが既存のスケジューリングモジュールまたはユーザが提案・実装したモジュールの機能試験が Bricks 上で実施可能であることを示す．

本稿の評価実験は，Bricks システムの最終的な目的であるスケジューリングアルゴリズムの比較・評価を行うものではない．しかし，第 1 段階として Bricks システムがグローバルコンピューティングのスケジューリングアルゴリズムおよびスケジューリングに関するモジュールの評価システムとして妥当なものであることを示す必要があるため，（1），（2）にあげたような評価システムの基本性能を調べる実験を行った．

### 5.1 評価実験方法

評価では実環境においてグローバルコンピューティングシステムの資源予測フレームワークを提供する NWS を実行して実際のネットワークの変動の測定と予測を行い，それを Bricks 上で再現する．実環境と Bricks 上で測定されたスループット値の比較により（1）を，双方の環境下での NWS Forecaster の予測結果の比較により（2）を実証する．

まず，東京工業大学（東京都）と電子技術総合研究所（つくば市）の 2 つの計算機に NWS Sensor を設定し，2 つのサイト間の通信スループット，レイテンシ，サーバの稼働率を測定する．NWS Forecaster には，測定されたネットワークの情報をもとに各タイムステップにおける予測を行わせる．ただし，サーバをモニタする間隔は 10[sec]，ネットワーク状況をモニタする間隔は 60[sec] とし，通信スループットを測定するためにネットワークに送信するプローブパケットのサイズを 300[KB] とした．

次に，実環境で NWS により測定された通信スループット値を用いた通信モデルを設定し，Bricks によるシミュレーションを行う．ここで，時刻と通信スループットの離散データからすべての時刻に対する通信スループットを算出するために，3 点の時刻と通信スループットのみで補間値を計算できる 3 次スプライン補間を用いた．また，シミュレーションにおける論理パケットサイズは 10[KB] に設定する．シミュレーションでは実際に NWS の Persistent State と Forecaster を実行し，実測と同じモニタ間隔を設定して実環境で測定された通信スループットが Bricks で得られるか，Bricks 上の NWS Forecaster が実環境同様に予測を行うかどうか調べる．

### 5.2 評価実験結果

図 7に実環境で NWS の Sensor が測定した通信スループットと Bricks 内で NetworkMonitor が算出した通信スループットを示す．横軸は実時間およびシミュレーション時間を示しており，実測は 1999 年 2 月 1 日月曜日深夜 0 時から 24 時間分の測定結果となっている．縦軸には通信スループットを [KB/sec] で示している．

図 7より，全体的に実測と Bricks のシミュレーション結果はほぼ同様の通信スループットを示していることが分かる．図 8は 2 時間分の実測値と Bricks で算出された通信スループットを比較したものであり，この図からも実測値と Bricks での通信スループットの算出値の一致が確認できる．実際のネットワークにおける TCP/IP 通信のモデル化の試みは複数行われているが，その挙動

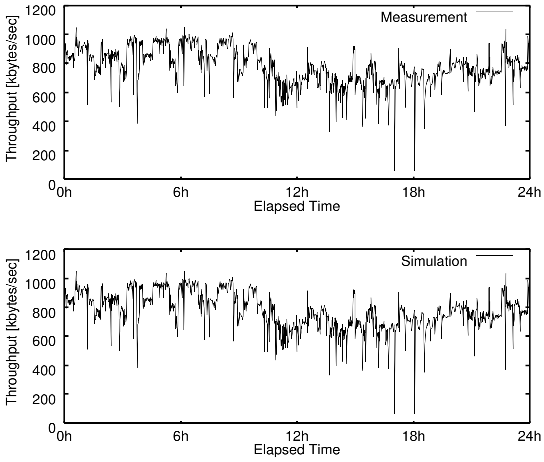


図 **7** 実環境（上）と Bricks シミュレーション（下）における TITECH-ETL 間の通信スループットの比較（24 時間）

Fig. 7 One day's worth of bandwidth measured between TITECH and ETL under the real environment in the figure above versus Bricks in the figure below.

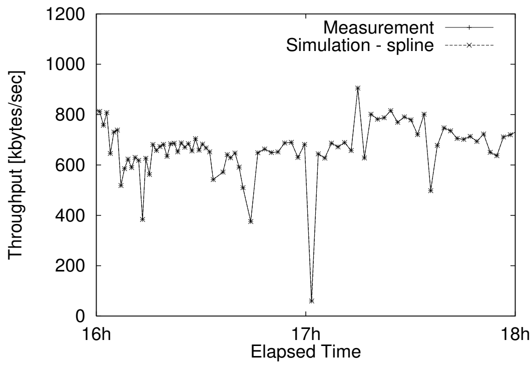


図 **8** 実測と Bricks での通信スループットの比較（2 時間）

Fig. 8 The comparison of bandwidth measured under the real environment versus Bricks for two hours.

表 **1** 実環境と Bricks において NWS Forecaster が採用した予測アルゴリズムの比較

Table 1 The comparison of forecasting algorithms which the NWS Forecasters applied under real and simulated environment.

| 総予測回数 | 一致した回数 | 一致した割合 |
|---|---|---|
| 1197 回 | 1058 回 | 88.4% |

は複雑で，特定のパケットに対するモデル化を行っているのが現状である．Bricks ではそれらのモデルを採用可能であることはもちろん，実環境における通信スループットの実測値を利用することで，実環境に則したグローバルコンピューティングシステム上での通信を再現できることが分かる．

図 **9**に実環境と Bricks において NWS Forecaster が予測した通信スループットを示す．また，図 **10**は実測と Bricks での通信スループットの予測値の 2 時間分のデータの比較結果である．図 **9**，**10**より，測定された

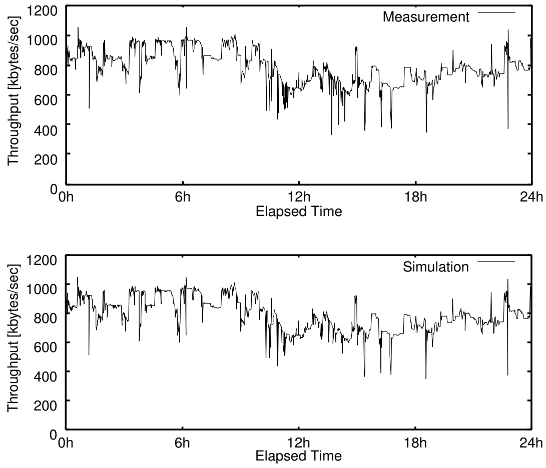


図 **9** 実環境（上）と Bricks シミュレーション（下）における TITECH-ETL 間の通信スループットの予測値の比較（24 時間）

Fig. 9 One day's worth of bandwidth predicted by the NWS Forecaster under the real environment in the graph above versus Bricks in the graph below.

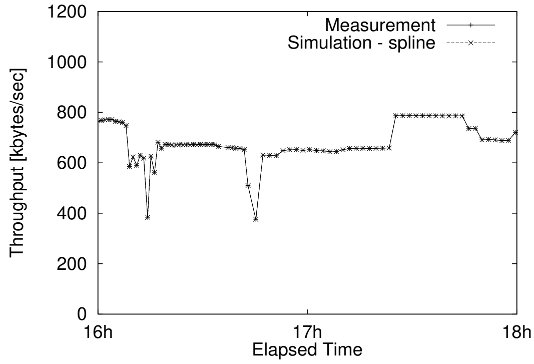


図 **10** 実環境と Bricks での通信スループットの予測値の比較（2 時間）

Fig. 10 The comparison of the behavior of the NWS Forecaster under the real environment versus Bricks for two hours.

通信スループット値と同様，実環境と Bricks での予測値もほぼ一致していることが分かる．

表 **1**に実環境と Bricks で NWS Forecaster が採用した予測アルゴリズムの比較結果を示す．NWS Forecaster は同時に複数☆の予測アルゴリズムを用いて予測エラーの少ないアルゴリズムの予測値を採用しているにもかかわらず，表 **1**から実環境と Bricks では 88.4% の確率で同じ予測アルゴリズムを採用していたことが分かる．NWS をはじめとするスケジューリングモジュールは，グローバルコンピューティング環境の状況，すなわち各スケジューリングモジュールに対する入力値

☆ 評価実験で用いたバージョン 1.1.1.0 の NWS パッケージでは 8 つの予測アルゴリズムを用いている．

によりその挙動が変化する．よって，実環境とシミュレーション環境で同様の予測結果を示したことは，グローバルコンピューティング環境が実環境と同様の通信スループット値をスケジューリングモジュールに対して与えることができたことを意味する．一方，NWS Forecaster が 11.6% は異なるアルゴリズムを選択した原因は，ネットワークのプローブのタイミングのずれにあると考えられる．Bricsk シミュレーション中に NWS Forecaster に渡される通信スループットの測定値は，NWS Sensor が実環境でプローブを終了した時刻に Bricks の NetworkMonitor がプローブパケットを流しはじめて得た値であり，実環境とは異なっていたためである．

よって，NWS Forecaster が Bricks 上で正常に動作したことから他のスケジューリングに関する外部モジュールの機能試験が Bricks 上で実施可能であることが示された．

## 6. 関連研究

分散システムにおけるスケジューリング手法の研究は以前から多数行われているが，未実装であったり，実装されている場合でも他のスケジューリングアルゴリズムとの比較が非常に難しい．これらの問題に着目したシミュレーションによる評価の試みが，以下のプロジェクトでなされている．

分散システムにおけるスケジューリングアルゴリズムの評価技術としては，フロリダ大で行われている Osculant[11] の Osculant Simulator があげられる．Osculant は bidding ポリシーに従って計算サーバの性能を表す bids を計算し，bids が一番高いものを選択するボトムアップリソーススケジューラである．Osculant Simulator では Bricks 同様にネットワークのトポロジや計算ノードの設定が柔軟に行える．Osculant Simulator は Osculant スケジューリングアルゴリズムの評価を目的としており，グローバルコンピューティングシステムのスケジューリングフレームワークを考慮して設計されていない．

WARMstones は未実装であるが，Bricks 同様スケジューリングアルゴリズムの評価基盤としてシラキュース大で提案されているシステムである．シミュレーションにおけるタスクやシステムの表現形式，および複数スケジューリングアルゴリズムを柔軟に表現するための MIL（MESSIAHS Interface Language）やライブラリ等，その基礎技術は MESSIAHS[12] に基づいている．一方，Bricks システムはオブジェクト指向フレームワークで Java 言語用のスケジューリングユニット SPI を提供し，その SPI を実装することにより様々なスケジューリングアルゴリズムや外部プログラムモジュールの Bricks 上での評価を可能にさせる．WARMstones ではいくつかの技術的提案がなされているが，実装された際にシステムの拡張が容易であるか，また既存グローバルコンピューティングシステムのプログラムモジュールの組み込みとその評価が可能かどうか不明である．

## 7. まとめ

本稿では，グローバルコンピューティングシステムのスケジューリング手法およびそのフレームワークのシミュレーションによる評価基盤を提供するシステム Bricks を提案した．Bricks の提供する Bricks 環境設定スクリプトにより，ユーザは柔軟にネットワークトポロジ，計算サーバアーキテクチャ，通信モデルおよびスケジューリングフレームワークの各モジュール等を設定し，再現性のある評価実験を行うことができる．また，グローバルコンピューティングのスケジューリングを行ううえで不可欠なリソース状況のモニタ・予測情報の提供をサポートする既存システムモジュールの機能試験環境を提供する．

Bricks とグローバルコンピューティングのリソース情報の予測システムである NWS を用いた評価実験では，実測通信スループットを用いた通信モデルを採用することで Bricks が実際の環境に則したグローバルコンピューティング上での通信を表現できることを確認した．また，Bricks 上で NWS の Persistent State，Forecaster が実環境上と同様に動作したことから，Bricks が既存のスケジューリングに関する各モジュールの機能試験環境を提供できることを実証した．

今後は Bricks システムを以下のように拡張する．

- 様々なグローバルコンピューティングシステム設定を表現可能にするために現在のタスク / 通信 / サーバモデルを洗練していく．すなわち，並列アプリケーションタスクが表現可能なタスクモデルや，タイムシェアリング等 FCFS 以外のタスクの処理方式および SMP，MPP 等マルチプロセッサアーキテクチャに対応したサーバモデルを検討する．
- シミュレーションモデルを洗練した後，様々な性質を持つ実アプリケーションを想定したスケジューリング手法の評価を行っていく．評価ではアプリケーションの性質やそれらの計算要求の頻度の違いによるスケジューリングアルゴリズムの性能の比較等を行っていく．
- 評価環境の設定言語として Bricks 環境設定スクリプトの代わりに XML を用いて Bricks システムの汎

用性を高めていく．また，1999 年 5 月に発足された Grid Forum[13] では，グローバルコンピューティング技術に関するデータ表現やインタフェースの規格化を目指して現在さかんに議論されており，今後 Grid Forum で定められた規格に対応したプログラムモジュール間の授受データの表現形式等を採用することで，他のグローバルコンピューティングシステムを容易に組み込み可能となるようにする．

- 本稿の評価では，あらかじめ NWS を用いて実環境での観測情報を収集し，その情報をもとに Bricks による評価実験を行った．しかしながら，Bricks のユーザがこのような実環境での評価実験を行う負担は大きく，各ユーザによって収集したデータセットが異なるために，他のユーザとの Bricks での評価結果の共有も難しい．また，様々なネットワークのトポロジ，計算リソース情報等の環境設定を Bricks 環境設定スクリプトにより記述した初期設定ファイルを作成するコストも大きい．よって，Bricks ではベンチマークセットとして実環境での観測情報を含めたデータ情報および初期設定ファイルを提供することによりこれらの問題を回避する．
- Bricks によりスケジューリングアルゴリズムの評価を行うためには，様々な環境設定パラメータを設定したシミュレーションを多数回行い，その結果の統計をとる必要がある．すなわち，Bricks は EP プログラムとして並列計算機または PC/WS クラスタ上で実行することができる．本プログラムを Ninflet[14] システムを用いて並列化し，33 ノードの PC クラスタ上でグローバルコンピューティングシステムにおける適切なスケジューリングアルゴリズムの調査，考案を行う．

謝辞　NWS の Bricks への組み込みに際し多大な助言をいただいたテネシー大の Rich Wolski 氏と UCSD の Jim Hayes 氏，また日頃よりご討論いただく Ninf プロジェクトの皆様，特に電子技術総合研究所関口智嗣氏，新情報開発機構佐藤三久氏，ならびにお茶の水女子大学細矢治夫教授に深く感謝いたします．

本研究の一部は JST PRESTO プログラムとソフトウェア工学事業団の支援によるものである．

## 参　考　文　献

1) Foster, I. and Kesselman, C.(eds.): *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann (1999).
2) Bricks. `http://ninf.is.titech.ac.jp/bricks/`.
3) Wolski, R., Spring, N. and Peterson, C.: Implementing a Performance Forecasting System for Metacomputing: The Network Weather service, *Proceedings of the 1997 ACM/IEEE Supercomputing Conference* (1997).
4) Wolski, R., Spring, N. T. and Hayes, J.: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, Technical Report TR-CS98-599, UCSD (1998).
5) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, Vol. 11(2), pp. 115–128 (1997).
6) Grimshaw, A. S., Wulf, W. A. and the Legion team: The Legion Vision of a Worldwide Virtual Computer, *Comm. ACM*, Vol. 40(1), pp. 39–45 (1997).
7) Sato, M., Nakada, H., Sekiguchi, S., Matsuoka, S., Nagashima, U. and Takagi, H.: Ninf: A Network based Information Library for a Global World-Wide Computing Infrastracture, *Proceedings of HPCN'97 (LNCS-1225)*, pp. 491–502 (1997).
8) Berman, F., Wolski, R., Figueira, S., Schopf, J. and Shao, G.: Application-Level Scheduling on Distributed Heterogeneous Networks, *Proceedings of the 1996 ACM/IEEE Supercomputing Conference* (1996).
9) 竹房あつ子, 合田憲人, 中田秀基, 小川宏高, 松岡聡, 佐藤三久, 関口智嗣, 長嶋雲兵: グローバルコンピューティングシステムのシミュレーションによる評価, 情報処理学会論文誌, Vol. 40, No. 5, pp. 2192–2202 (1999).
10) NWS Java API. `http://ninf.etl.go.jp/~nakada/nwsjava/`.
11) Osculant. `http://beta.ee.ufl.edu/Projects/Osculant/`.
12) Chapin, S. J. and Spafford, E. H.: Support for Implementing Scheduling Algorithms Using MESSIAHS, *Scientific Programming*, Vol. 3, pp. 325–340 (1994).
13) Grid Forum. `http://www.gridforum.org/`.
14) 高木浩光, 松岡聡, 中田秀基, 関口智嗣, 佐藤三久, 長嶋雲兵: Java による大域的並列計算環境 Ninflet, 並列処理シンポジウム JSPP'98, pp. 135–142 (1998).

(平成11年9月1日受付)
(平成12年2月4日採録)

竹房あつ子（正会員）

昭和 48 年生．平成 8 年お茶の水女子大学理学部情報科学科卒業．平成 10 年同大学大学院理学研究科情報科学専攻修士課程修了．平成 12 年同大学院人間文化研究科複合領域科学専攻博士課程修了．博士（理学）．同年日本学術振興会特別研究員．並列分散処理，グローバルコンピューティング，スケジューリングに興味を持つ． ACM 会員．

合田　憲人（正会員）

昭和 42 年生．平成 2 年早稲田大学理工学部電気工学科卒業．平成 4 年同大学大学院修士課程修了．平成 8 年同大学院博士課程退学．平成 4 早稲田大学情報科学研究教育センター助手，平成 8 年同特別研究員．平成 9 年東京工業大学大学院情報理工学研究科助手，平成 11 年同大学院総合理工学研究科専任講師，現在に至る．博士（工学）．並列・分散処理方式，並列化コンパイラ，マルチプロセッサ OS，グローバルコンピューティングシステムの研究に従事．電子情報通信学会，電気学会， ACM， IEEE-CS 各会員．

松岡　　聡（正会員）

昭和 38 年生．昭和 61 年東京大学理学部情報科学科卒，平成元年同大学大学院博士課程中退．同大学情報科学科助手，情報工学専攻講師を経て，平成 8 年より東京工業大学情報理工学研究科数理・計算科学専攻助教授．平成 10 年より科学技術振興財団のさきがけ研究員を併任．理学博士．オブジェクト指向言語，並列システム，リフレクティブ言語，制約言語，ユーザ・インタフェースソフトウェアなどの研究に従事．現在進行中のプロジェクトは，世界規模の高性能計算環境を構築する Ninf プロジェクト，オブジェクト指向言語の大規模並列クラスタ計算機上の実装，計算環境に適合・最適化を目指す Java 言語の開放型 Just-In-Time コンパイラ OpenJIT，各種ユーザインターフェースなど．平成 8 年度情報処理学会論文賞，平成 11 年情報処理学会坂井記念賞受賞．ソフトウェア科学会， ACM,IEEE-CS 各会員． IEEE Concurrency Magazine の Editor．

中田　秀基（正会員）

昭和 42 年生．平成 2 年東京大学工学部精密機械工学科卒業．平成 7 年同大学大学院工学系研究科情報工学専攻博士課程修了．博士（工学）．同年電子技術総合研究所研究官．現在同所主任研究官．並列プログラミング言語，オブジェクト指向言語，分散計算システムに関する研究に従事．

長嶋　雲兵（正会員）

昭和 30 年生．昭和 58 年北海道大学大学院博士後期課程修了．理学博士．同年岡崎国立共同研究機構分子科学研究所助手．平成 4 年お茶の水女子大学理学部情報科学科助教授．平成 8 年同教授．平成 10 年工業技術院物質工学工業技術研究所理論化学研究室長．平成 11 年工業技術院産業技術融合領域研究所計算科学グループ長．理論化学，並列分散処理，性能評価の研究に従事．日本化学会，日本応用数理学会， IEEE， ACM 各会員．