C-010

SAGA によるグリッドアプリケーションとその性能評価
SAGA based Grid Application and the perfomance evaluation

河井 裕* 岩井 剛 佐々木 節 渡瀬 芳行
Yutaka Kawai Go Iwai Takashi Sasaki Yoshiyuki Watase

Abstract

現在の各グリッドミドルウェアは各地域において独立に並行して開発され、運用されている。そのため、異種グリッドミドルウェア間の相互運用性の整備することがこれからの課題である。*KEK* では既に *NAREGI, Torque* の *SAGA* アダプタのプロトタイプを開発し、*2* つの異なるミドルウェア上でのジョブ実行を可能にする *SAGA* ベースのアプリケーションを示した。*SAGA* は、ミドルウェアより上位に位置するインターフェースであるため、オーバーヘッドによるコストの影響を考慮する必要がある。本論文では、開発した *SAGA* アダプタを用いて *SAGA* ベースのアプリケーションの性能評価を行い、*SAGA* のオーバーヘッドによるコストを示した。

1 はじめに

グリッドコンピューティングの技術は現在広く利用されているが、それぞれのグリッドミドルウェアが各地域において独立に並行して開発され、運用されている。したがって、異種グリッドミドルウェア間の相互運用性の整備がこれからの課題となっている。例えば、インターネット上におけるバッチジョブの分散処理を利用するために必要な機能は限られている。それにも関わらず、異なるミドルウェア毎に異なる利用法を強いられている現状がある。各ミドルウェアの技術の違いを吸収し、統一されたインターフェースが用意されることで、エンドユーザーは、各ミドルウェアの技術の違いを気にすることなく、自らのアプリケーションの開発が行うことが可能になる。なお、この研究は RENKEI –RESources liNKage for E-scIence– プロジェクト [8] のサブテーマの一つにもなっている。

SAGA –A Simple API for Grid Applications– [9] は、OGF [6] で標準化が進められている仕様であり [3]、異なるミドルウェアを利用するための統一的なインターフェースを提供する。KEK では既に NAREGI –National Research Grid Initiative– [4, 5], Torque [11] のそれぞれの SAGA アダプタのプロトタイプを開発し、2 つの異なるミドルウェア上でのジョブ実行を可能にする SAGA ベースのアプリケーションを示した [2]。

SAGA は、ミドルウェアより上位に位置するインターフェースであるため、オーバーヘッドによるコストの影響を考慮する必要がある。本論文では、SAGA の環境構築について述べると共に、SAGA ベースのアプリケーションを用いることによって生じるオーバーヘッドを評価した。

2 SAGA Architecture

現在存在している様々な種類のグリッドミドルウェアは、それぞれ独自のコマンドや特徴を備えており、アプリケーションを開発する際にはそれらを意識してアプリケーションを開発しなければならない。そのため、異なるミドルウェア間でアプリケーションの互換性を保つことができず、そのことが開発者に対して大きな負担となっている。KEK での研究の目標は、SAGA を用いて統一的なインターフェースを提供することによって、開発者の生産性を上げることである。

SAGA のレイヤー（以下、単に SAGA と呼ぶ）は図 1 のように位置し、様々な種類のミドルウェアの違いを吸収することが可能になっている [10]。SAGA から各ミドルウェアにアクセスするには、SAGA アダプタが必要である。アプリケーション開発者は SAGA の API を使用することで、異なるミドルウェアに対して同じ手続きでア

* Computing Research Center(CRC), High Energy Accelerator Research Organization (KEK)

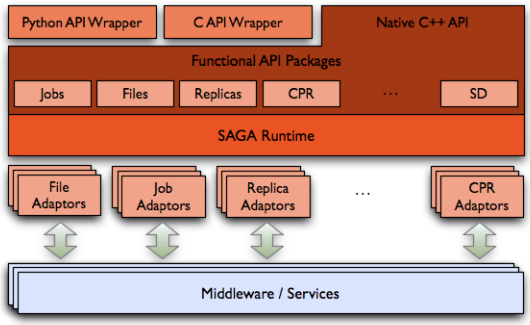

図 **1. The design of SAGA.**

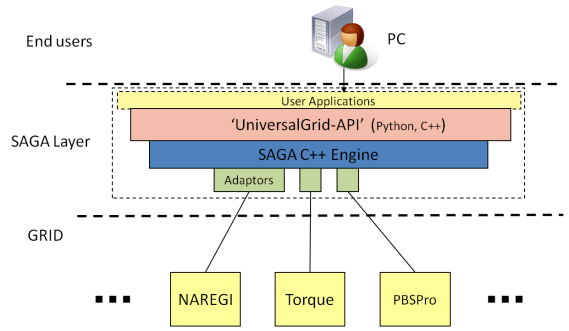

図 **2. SAGA-based user environment.**

クセスすることができる。

KEK では、表 1 に示す SAGA アダプタのプロトタイプを開発中である。SAGA には、C++ と Java の二種類の実装が存在するが、KEK では主に C++ の実装を用いている。

図 2 は、より具体的な SAGA 環境構成を示している。SAGA はエンドユーザーとグリッドの層の中間に位置している。エンドユーザーは、それぞれのミドルウェアの差異を知ることなしに、SAGA 及び SAGA アダプタを経由して各ミドルウェアのリソースを使うことが可能である。

例えば、表 1 のうち、SNA, STA, SPA の SAGA アダプタを使用した場合、ワークフローは図 3 のようになる。SAGA Adaptor Host と呼ばれるホストサーバーに SAGA library と SNA、STA、SPA がインストールされている。SAGA Adaptor Host には、NAREGI、Torque、PBSPro のそれぞれのクライアントもインストールされている必要がある。（Torque と PBSPro のコマンド名はほぼ同じであ

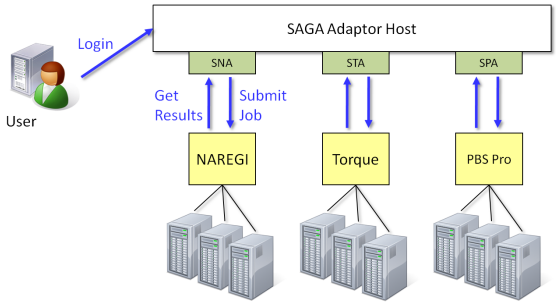

図 **3. Workflow diagram in the user environment based on SAGA.**

るが、プログラム自体は別物であるため、区別してインストールしなければならない。）エンドユーザーは、SAGA Adaptor Host を経て、NAREGI や Torque、PBSPro のミドルウェアにアクセスすることができる。

しかし、上記のような SAGA の利便性の代償として、SAGA を経由することによるオーバーヘッドを避けることはできない。そのため、そのオーバーヘッドが実用的な範囲のコストであるか否かを評価する必要がある。また、SAGA は Python binding も提供しているため、C++ と Python の両方の API の使用によるオーバーヘッドも考慮して測定しなければならない。

3　SAGA 内部の処理

先述のとおり、KEK では既にいくつかの SAGA アダプタのプロトタイプを開発しており、その中の一つである Torque 用の SAGA アダプタである STA を使用して性能評価する。

STA を経由した場合のアプリケーションの実行の概略は図 4 のようになる。Torque システムへのジョブ投入は、Torque 固有のコマンド、qsub [7] を実行することによって行われる。qsub コマンドは Boost.Process [1] によって実行されているが、その Boost.Process は STA によって呼ばれる。SAGA は Python binding も提供しているため、Python の SAGA API も使用可能である。

SAGA C++ の実装の内部は以上のような設計になっているため、qsub を直接用いてジョブを投入した場合、SAGA C++ API を使用した場合、Python API を使用した場合、の順でコストが増えていくのがわかる。これら 3 つの場合における性能測定を次に示す。

表 **1. SAGA Adaptors under development in KEK.**

Name	Full Name	Middleware
SNA	SAGA NAREGI Adaptor	NAREGI (National Research Grid Initiative)
STA	SAGA Torque Adaptor	Torque
SPA	SAGA PBSPro Adaptor	PBSPro (Portable Batch System Professional Edition)
SIA	SAGA iRODS Adaptor	iRODS (The Integrated Rule-Oriented Data System)
SGFA	SAGA Gfarm Adaptor	Gfarm (Grid Data Farm)

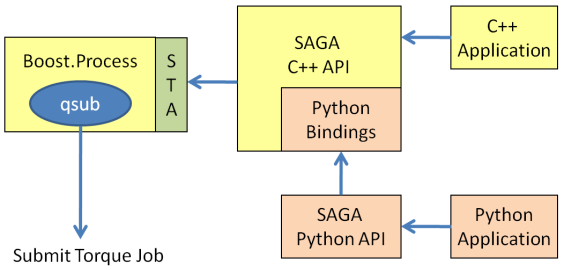

図 **4. Call mechanism in SAGA with STA**

```
#! /bin/sh
#PBS −N saga−app
#PBS −d /home/ykawai/tmp
#PBS −l walltime=300
#PBS −q @dg02.cc.kek.jp
#PBS −M yutaka.kawai@kek.jp
/bin/hostname
```

図 **5. PBS script**

```
#!/bin/sh

for i in ‘seq 1 500‘
do
        (time ./jobtest) 2>> time_c.log
        (time ./jobtest.py) 2>> time_py.log
        (time qsub pbs_script.txt ) 2>> time_q.log
done
```

図 **6. Shell script to execute time commands.**

4　性能評価

性能評価は以下の 3 ケースを行った。

(A) qsub コマンドを直接実行する場合
(B) SAGA C++ API を用いた C++ アプリケーションを実行する場合
(C) SAGA Python API を用いた Python アプリケーションを実行する場合

それぞれに共通の条件として、Job Description が 3 つのケース全てにおいて同じになるように構成した。ケース (A) では、図 5 の PBS スクリプトを用いている。ケース (B), ケース (C) で使用したソースコードは紙面の都合上割愛するが、それぞれ図 5 と同じ Job Description が生成されるように作られている。

時間測定は、time コマンドを用い、図 6 のようなシェルを実行し、3 つのケースをほぼ同時に測定した。それぞれ 500 回のジョブの投入を連続して行っている。

測定結果は、図 7 になる。表 2 は、それぞれのケースの平均経過時間と標準偏差をまとめたものである。

5　考察

今回の性能評価では、ジョブを投入するまでにかかる SAGA のオーバーヘッドを測定する必要があったため、hostname を返すのみの低負荷なジョブを投入した。より計算資源を必要とする高負荷のジョブを投入することも考えられたが、それはミドルウェアに対して負荷が高くなるのみで、クライアント側のオーバーヘッドには影響がないため避けることにした。もっと明確に言えば、ジョブ投入にかかる時間はジョブの内容とはほとんど関係がない。なぜならば、ジョブの内容は Job Description に記述されるものが全てであり、Job Description の生成コス

表 **2. Average and Standard Deviation of the performance results.**

Case	Average	Standard Deviation
(A) qsub	80.17 ms	108.01
(B) C++	472.16 ms	148.76
(C) Python	539.10 ms	165.03

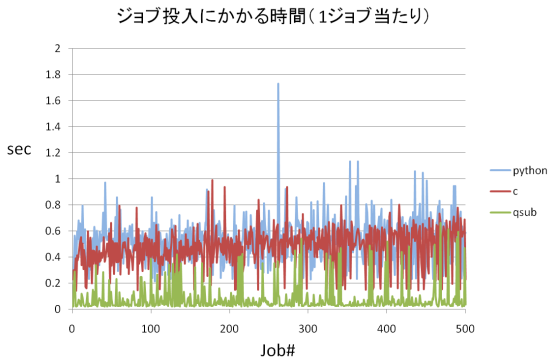

図 **7. The result of speed performance.**

トは、全てのジョブにおいてほぼ等しいからである。

図 7 を見ると、確かに一回のジョブ投入に必要なオーバーヘッドは多く、ケース (A) に比べてケース (B) でも平均で 4.9 倍、ケース (C) では 5.7 倍のコストがかかっている。しかし一方で、連続ジョブ投入した場合においても、オーバーヘッドのコストが累積していないこともわかる。つまり、表 2 の平均経過時間が、直接エンドユーザーが体感する遅延であると考えられる。エンドユーザーが実際に操作する状況の中で、一回のジョブ投入にかかる時間が平均にして約 0.5 秒という結果は、現実的に許容できる範囲である。

6 結論

本論文では、KEK で既に開発した SAGA アダプタである STA を用いて SAGA ベースのアプリケーションを用いることによって生じるオーバーヘッドの評価を行った。また、SAGA を利用するユーザーの環境構成や、SAGA の内部構成について触れた。それらの構成を考慮の上で、性能評価を行うにあたり、qsub を直に用いる場合、SAGA C++ API を用いる場合、Python API を用いる場合とに分けて比較した。

SAGA は、既存のミドルウェアより上位に位置するインターフェースであるため、そのオーバーヘッドによるコストの発生は自明であるが、ジョブ投入にかかる時間は現実的に許容できる範囲であることがわかった。

KEK では、マルチグリッドミドルウェア環境下において、アプリケーション開発者がシームレスに分散コンピューティングのインフラを利用できることを目的としている。ユーザーが異なるサイトのアプリケーションやデータを扱う場合にも、容易に利用可能な統一的な手続きを開発することが出来れば、世界的な規模での分散処理の実現が可能になる。SAGA 及び SAGA アダプタの開発が、その目的達成のための有効な手段の一つであることが言える。

参考文献

[1] Boost.Process. Online. http://www.netbsd.org/ jmmv/process/.

[2] T. S. Go Iwai, Yutaka Kawai and Y. Watase. SAGA-based user environment for distributed computing resources: A universal Grid solution over multi-middleware infrastructures. In *Accepted for International Conference on Computational Science, ICCS 2010, Amsterdam, Netherlands.*, 2010.

[3] S. Jha, H. Kaiser, Y. El Khamra, and O. Weidner. Design and Implementation of Network Performance Aware Applications Using SAGA and Cactus. In *Accepted for 3rd IEEE Conference on eScience2007 and Grid Computing, Bangalore, India.*, 2007. http://saga.cct.lsu.edu/publications/papers/confpapers/Design-and-Implementation-of-Network-Performance-Aware/saga_cactus_escience.pdf/view.

[4] S. Matsuoka, S. Shinjo, M. Aoyagi, S. Sekiguchi, H. Usami, and K. Miura. Japanese Computational Grid Research Project: NAREGI. *Proceedings of the IEEE*, 93(3):522–533, March 2005.

[5] NAREGI – NAtional REsearch Grid Initiative. Online. http://www.naregi.org/index_e.html.
[6] OGF – Open Grid Forum. Online. http://www.ogf.org/.
[7] TORQUE Resource Manager, qsub. Online. http://www.clusterresources.com/torquedocs21/commands/qsub.shtml.
[8] RENKEI – REsources liNKage for E-scIence. Online. http://www.e-sciren.org/index-e.html.
[9] SAGA – A Simple API for Grid Applications. Online. http://saga.cct.lsu.edu/.
[10] The SAGA C++ Reference API. Online. http://saga.cct.lsu.edu/cpp/apidoc/.
[11] TORQUE Resource Manager. Online. http://www.clusterresources.com/pages/products/torque-resource-manager.php.