オーバレイスケジューラJojo3の提案

産業技術総合研究所 グリッド研究センター

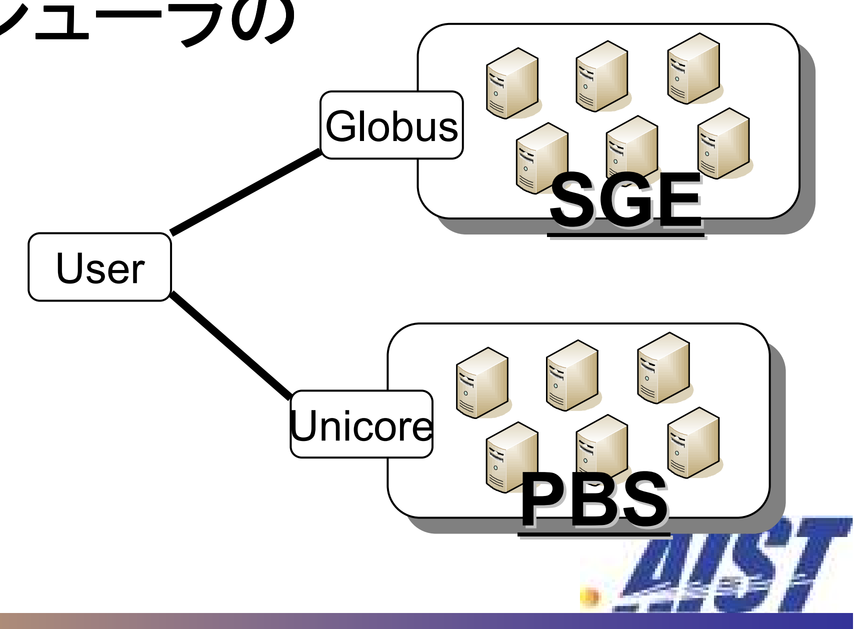
中田秀基, 田中良夫, 関口智嗣

背景

- グリッド技術とクラスタ技術の発展
  - 個々のユーザがアクセス可能な資源量の増加
- にもかかわらず有効に活用できていない
  - 有効利用できるのは簡単な独立ジョブのみ
  - 問題:分散プログラミングが困難
    - 分散プログラミングの本質的な困難性
    - ジョブ起動，通信の確立の困難性
      - 環境の不均質性，非対称性

背景(2)

- アプリケーションレベルスケジューラ
  - 特定のアプリケーションパターンに特化したフレームワーク
  - 下位レイヤを隠蔽．容易にプログラミングが可能
  - Ex. GridRPC, Condor MW, jPoP
- アプリケーションレベルスケジューラの実装は困難
  - さまざまな下位レイヤに対応しなければならない



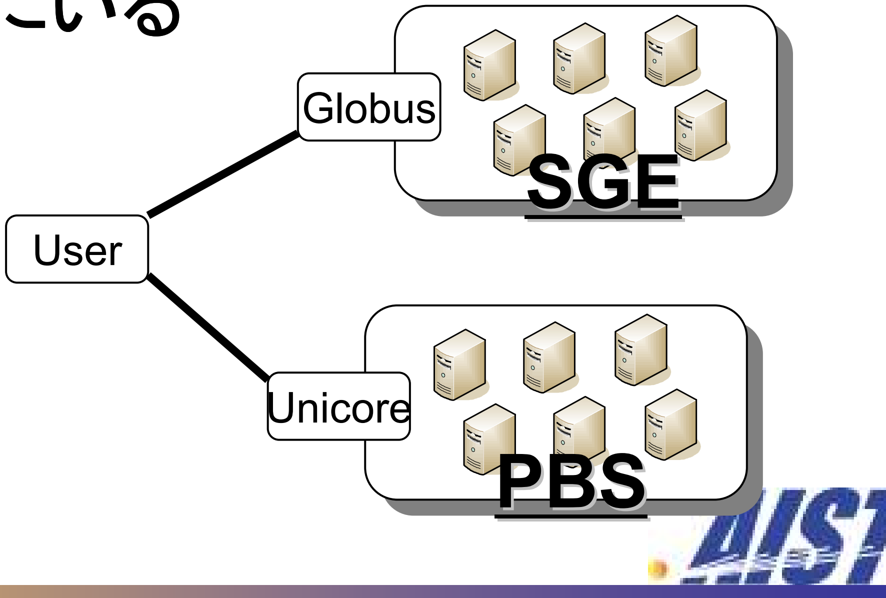
発表の目的

- オーバレイスケジューラの提案
  - 下位レイヤの複雑さを隠蔽
  - さまざまなアプリケーションレベルスケジューラの実装を容易に

- Jojo3 の紹介
  - オーバレイスケジューラの実装
  - Executor アプリケーションレベルスケジューラ

典型的なグリッドの構成

- 複数のクラスタがネットワークで接続
  - クラスタはキューイングシステムで管理されている
  - クラスタはプライベートネットワークのみ
- 各クラスタには外部からジョブ投入が可能
  - なんらかのグリッドミドルウェア
- クライアントはNATの背後にいる



なぜグリッド上のプログラミングは面倒なのか

- さまざまなグリッドミドルウェア
  - 資源のモニタリング，ジョブ起動の作法が違う
- ジョブが起動するまでにどれだけ時間がかかるかわからない
  - バックエンドのキューイングシステムの状態に依存
- ジョブ間のネットワーク接続が提供されない
  - ネットワークの非対称性をプログラマが解決しなければならない．

Appli.	Appli.	Appli.	Appli.	Appli.	Appli.

Globus2	Globus4	UNICORE	TORQUE	SGE

Grid Middleware　　　Local Scheduler

アプリケーションレベルスケジューラ

- グリッドの複雑さを隠蔽してアプリケーションプログラミングを容易に
  - Ex. Ninf-G5, Condor-MW

Appli. Appli. Appli. Appli. Appli. Appli.

Application Level Scheduler (ex. Ninf-G)

Application Level Scheduler

Application Level Scheduler

Globus2 Globus4 UNICORE TORQUE SGE

Grid Middleware

Local Scheduler

オーバレイスケジューラの導入

- アプリケーションレベルスケジューラと，グリッドミドルウェア，キューイングシステムの間に，新たな層を導入
- アプリケーションレベルスケジューラ実装を容易に

Appli. Appli. Appli. Appli. Appli. Appli.

Application Level Scheduler | Application Level Scheduler | Application Level Scheduler

Overlay Scheduler

Globus2 | Globus4 | UNICORE | TORQUE | SGE

Grid Middleware　　Local Scheduler

オーバレイスケジューラの機能

- 資源をホモジニアスに提供
  - グリッドミドルウェア，キューイングシステムの実装によらず一元的に管理
  - 各計算資源のモニタリング情報を提供
- ジョブの起動
  - どの計算資源でも同一のインターフェイスで即座に起動
- ジョブ間通信
  - 任意のジョブの間で通信を提供

Application Level Scheduler

Job
Job
Job

Daemon
Daemon
Daemon
Daemon
Daemon

Overlay Scheduler

Globus
Grid Engine
SSH
SSH

Base Level Scheduler

Jojo3の設計

- 機能
  - 使用可能なノード群の管理
  - ジョブの起動と管理
  - ジョブ間通信の提供
- アーキテクチャ
  - ルートとなるノードから再帰的にデーモンを起動してデーモンのネットワークを構成
  - デーモンはジョブに対してインターフェイスを提供

Jojo3の構成

Job Group

A	Job	Job	Job	
B	Job	Job		Job
C	Job		Job	Job
	Jojo3	Jojo3	Jojo3	Jojo3
	ノード	ノード	ノード	ノード

アプリケーションレベルスケジューラの挙動

- ルートノード上のジョブ
  - 使用可能なノードを検索
  - 使用可能なノード上にワーカジョブを起動
  - ワーカジョブからの接続を待つ
- ワーカジョブ
  - 自分を起動したジョブに対して接続

Jojo3の実装

- Java言語で記述
  - JDK1.4 で記述
    - SE5,　SE6 の機能は使用せず.
    - デフォルトで1.4レベルの機能しか提供していないLinuxディストリビューションがあるため.
  - アプリケーションは言語の制約なし
    - テキストベースの言語中立なインターフェイスを提供
    - Java, C, PythonのAPIを提供予定

デーモンモジュールのステージング

- ブートストラップとなる小規模のJarファイルのみ転送して起動
  - カスタムクラスローダ
  - 100K byte程度
- 残りのクラスファイルは動的に起動ホストから取得される
  - 事前のインストールは不要
  - 各ノードにはJava環境のみを期待

本体

起動ノード

本体

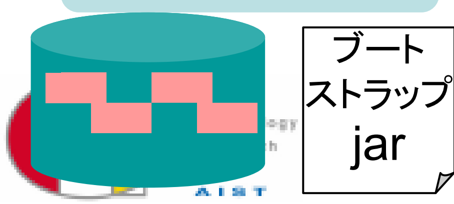
クラスローダ

ノード

本体

クラスローダ

ノード

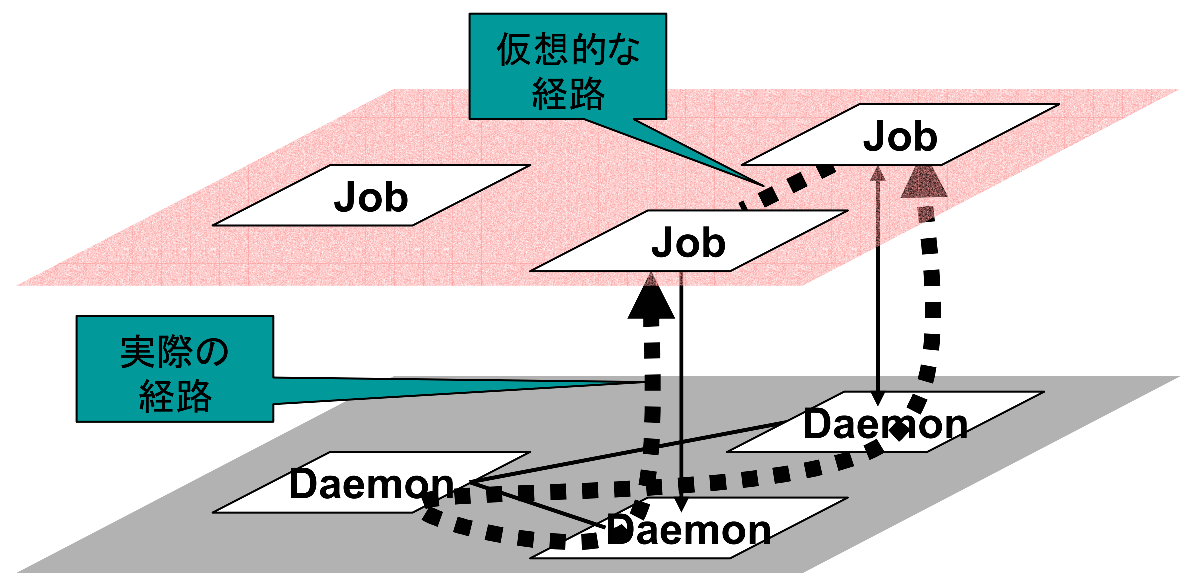


ブート
ストラップ
jar

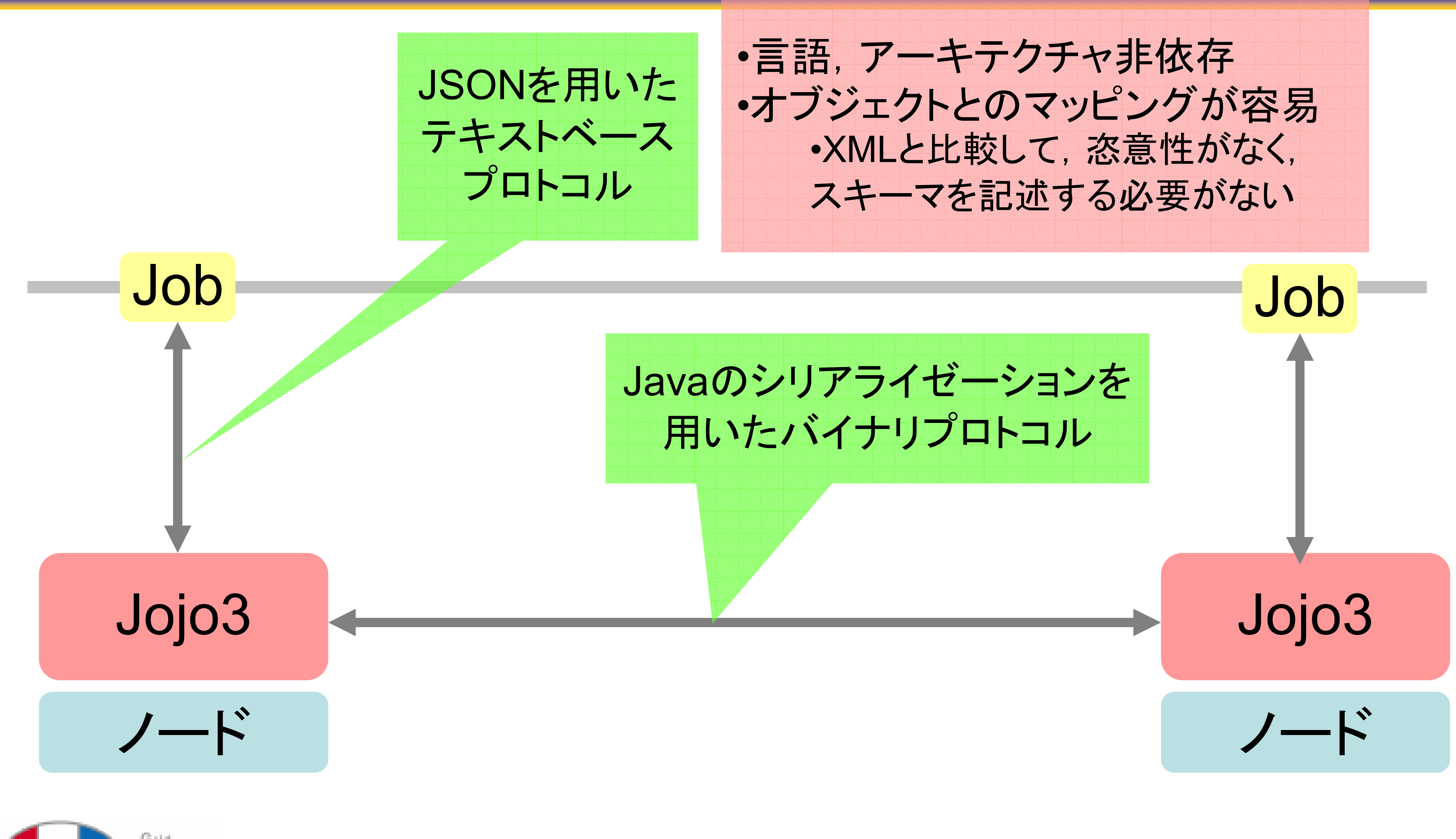
ブート
ストラップ
jar

ジョブ間通信の実装

- デーモン間の通信網を利用
  - 直接接続するわけではないので，ネットワークが非対称な環境でも動作



プロトコル



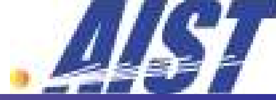
JavaクライアントAPIの設計

- ノンブロッキング
- イベントリスナを登録
  - イベントが生じたらリスナをコールバック
  - クライアントプログラムがpollする必要がない

Java Client API

```
class Client {
  // コンストラクタ
  Client(UpdateListener listener);

  // 自分のジョブ識別子取得
  String getJobId();

  // 自分の所属するジョブグループの識別子取得
  String getJobGroupId();

  // ジョブグループの起点ジョブの識別子取得
  String getRootJobId();

  // ノード群の情報取得
  NodeInfo [] getNodeInfo();

  // 自分の属するジョブグループの他のジョブの状態取得
  JobState [] getJobState();

  // ジョブの接続要求
  String invoke(String nodeId, JobDescription job);

  // 他のノードへの接続要求
  String connect(String jobId);
}
```

```
interface UpdateListener {
  // 接続成功
  void connected(String conId, Connection con);

  // 接続失敗
  void connectionFailed(String conId);

  // ジョブグループに属するジョブの状態変化
  void jobStateChanged(String jobId,
                       JobState jobState);

  // ノードの状態変化
  void nodeStateChanged(NodeInfo nodeInfo,
                        NodeState nodeState);
}
```



記述例

```
public class Jojo3Worker {
    static class WorkerUpdateListener extends UpdateAdaptor {
            public void connected(String conId, Connection con) {
                    ExecutorWorker worker =
                            new ExecutorWorker(con.getInputStream(),
                                                                    con.getOutputStream());
            }
    }

    public static void main(String [] args)  {
            UpdateListener listener = new WorkerUpdateListener();
            Client client = ClientImpl.create(args, listener);
            client.connect(client.getRootJobId());
    }
}
```

アプリケーションスケジューラ実装例

- 簡単なマスタワーカ機構をJDK 5 で導入されたExecutorService として実装
  - もともとはRunnable, Callableに対するスレッドプール
  - Futureによる同期機構，複数のFutureに対する待ち合わせも実装されている
  - 分散化 – Runnable, Callable をシリアライズして転送，リモートで実行
- ユーザは，スレッドプール実装との差異をほとんど意識することなく利用することが可能

記述例

```
void solve(Collection<Callable<Result>> solvers) {
    CompletionService<Result> ecs
        = RemoteExecutors.getExecutorCompletionService();
    for (Callable<Result> s : solvers)
        ecs.submit(s);
    for (int i = 0; i < solvers.size(); ++i) {
        Result r = ecs.take().get();
        if (r != null)
            use(r);
    }
}
```

ExecutorService

- 実装コスト
  - 通常のソケット通信を用いて分散化したものに対して比較
  - マスタモジュールで100行程度
  - ワーカモジュールで50行程度

関連研究(1) Condor Glide-in

- Condor： ウィスコンシン大学で開発されたジョブスケジューリングシステム
  - クラスタを管理するキューイングシステム
- Condor Glide-in
  - 他のキューイングシステムで管理されている資源を一時的にCondorの管理下におさめ，ジョブを実行できるようにする仕掛け
  - バッチジョブとしてCondorのノード管理デーモンをサブミット．起動したデーモンは，Condorのセントラルマネージャにコネクトバックし，ジョブを受け付ける

関連研究(2)GXP

- GXP: [‘04 田浦]
  - Pythonで記述されたオーバレイスケジューラ
  - 主にsshを用いて自らのコードを転送して起動，ネットワークを構築
  - ジョブの起動と簡単な通信機構を提供
- ジョブに対するAPIは提供していない
  - 特定のファイルディスクリプタを用いてブロードキャスト
  - Javaのようにファイルディスクリプタを直接操作できない言語では利用できない.

関連研究(3)Jojo1，Jojo2

- 筆者らによる過去のシステム
  - Jojo3と実装の一部を共有
    - リモートジョブ起動機構，ステージングなど
  - 組み合わせ最適化問題に対するアプリケーションレベルスケジューラjPoPを実装するための基盤として開発
  - アプリケーションレベルスケジューラとの分離が不完全
    - 複数のアプリケーションレベルスケジューラで共有することができない
    - アプリケーションレベルスケジューラをJavaで書かなければならない

おわりに

- オーバレイスケジューラの提案
  - アプリケーションレベルスケジューラの実装を容易に
- Jojo3の設計と実装
- アプリケーションレベルスケジューラの例としてJavaのExecutorServiceを実装
  - アプリケーションレベルスケジューラの実装が，容易になることを確認

現状

- SSH, Globus GRAM, Condor に対する基本部分の実装が完了
  - キューイングシステムの取り扱いは不完全
  - リファクタリングが必要
- クライアントライブラリはJavaのみ

今後の課題

- 実装の完了
  - さまざまなキューイングシステムへの対応
    - Grid Engine, TORQUE
- 他の言語でのクライアントライブラリの実装
  - C言語, Python
- APIの検討
  - JavaのAPIの再検討
- さまざまなアプリケーションレベルスケジューラの実装
  - 統計処理パッケージRの並列化
  - 遺伝的アルゴリズムに特化したスケジューラ
  - Ninf-G5

Grid
Technology
Research
Center
AIST

AIST