事前予約機能を持つ ローカルスケジューリングシステムの 設計と実装

1.産業技術総合研究所、2.東京工業大学、3.数理技研、4.エス・エフ・シー

中田秀基[1,2]、竹房 あつ子[1]，大久保 克彦[1,3]、岸本 誠[1,4]、

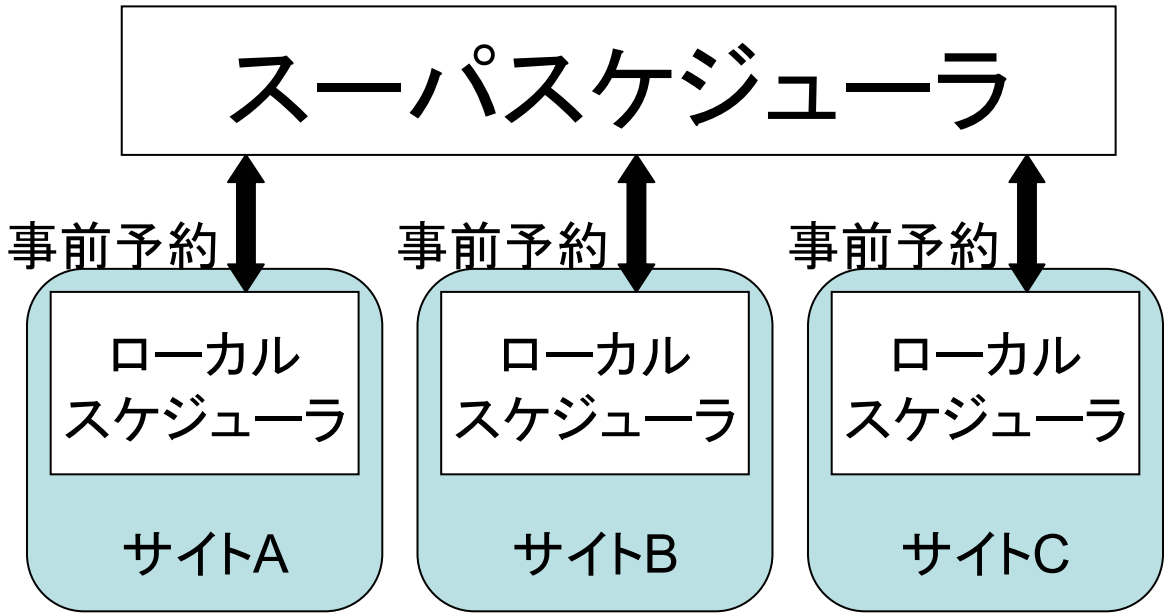
工藤 知宏[1]、田中良夫[1]、関口智嗣[1]

Grid Technology Research Center AIST

AIST

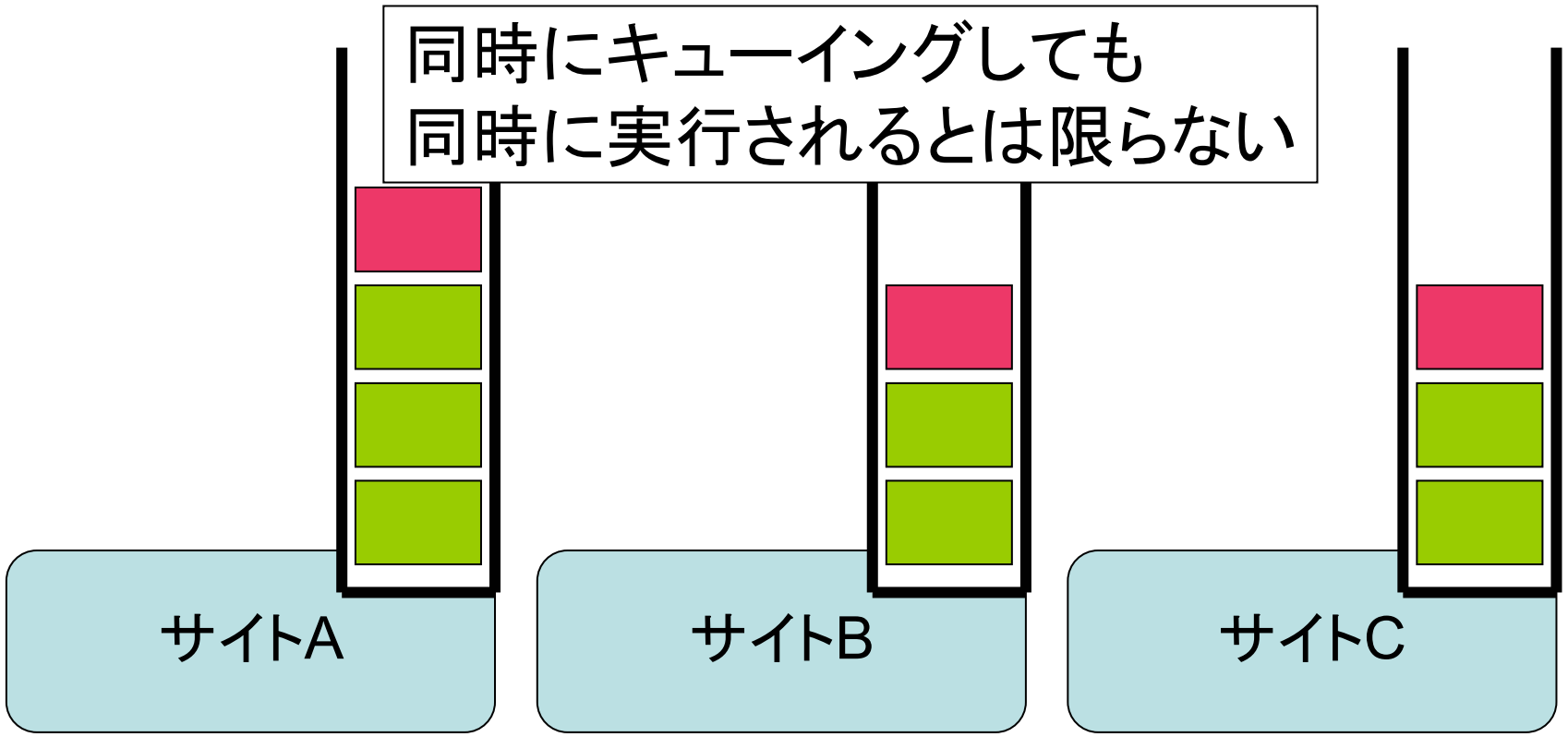
背景

- グリッドによる複数のサイトをまたいだ大規模計算
  - 資源のコアロケーション(同時確保)が必要
- 多くのサイトはFCFS(First Comes First Served)＋優先順位のキューイングシステムで運用



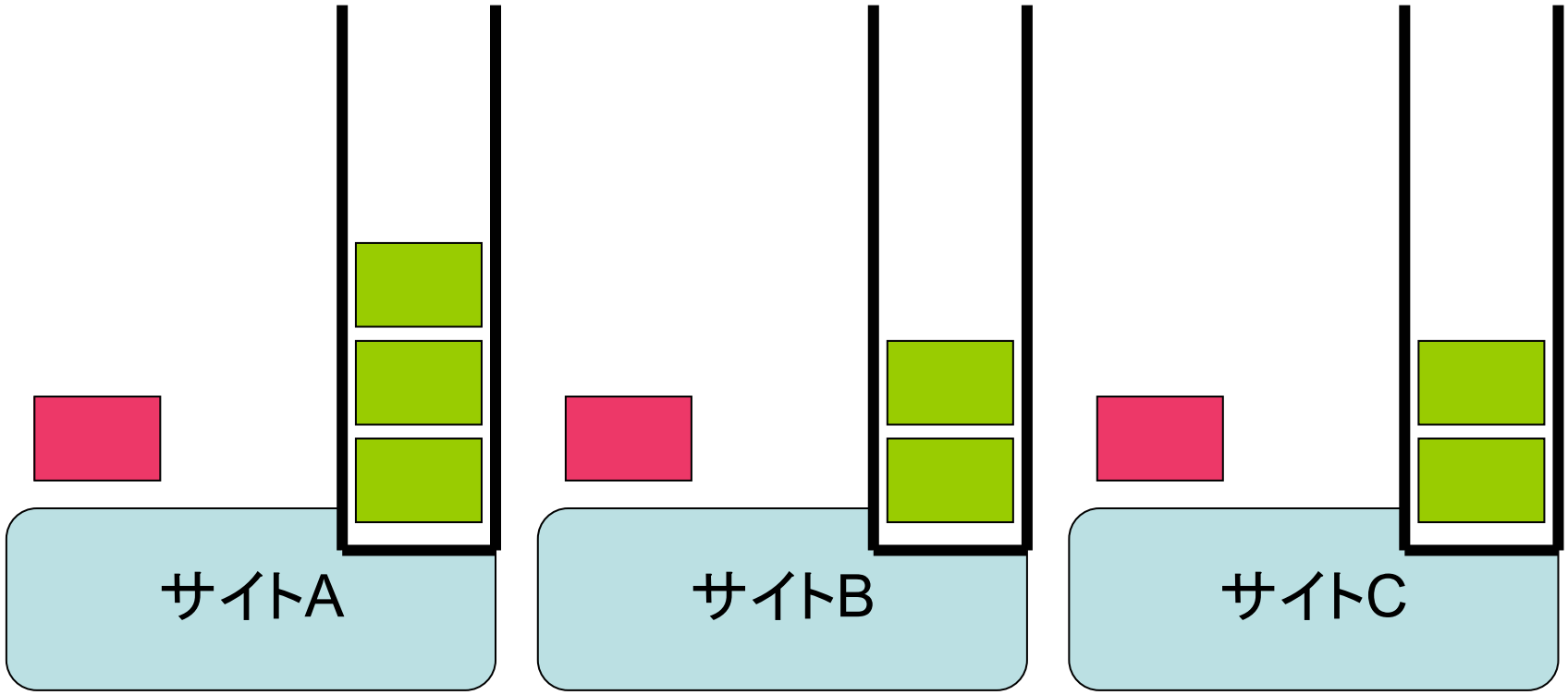
計算資源のコアロケーション (1/2)

- FCFS
  - キューイングされた順番で実行



計算資源のコアロケーション (2/2)

- 事前予約
  - キューとは独立に時間スロットを確保



FCFSと事前予約の関係はどうなるの？

- 事前予約だけ，というわけにはいかない
  - 各サイトのユーザはFCFSになれている
  - 実行時間の正確な見積もりは難しい
- FCFSジョブの実行中に予約時間がきたら？
  - 実行中のジョブを殺す？
  - 終了するまで待つ？
- 予約は早い者勝ち？
  - フェアシェアは？
  - ユーザの優先順位は？

FCFSと整合性を持つ事前予約機能の検討が必要

⇒ スケジューリングポリシを自由に変更できる環境がほしい

事前予約をサポートするローカルスケジューラ

- 有料のものにはいくつかある
  - PBS Professional, LSF
  - スケジューリングポリシを変更できないため検討の環境としては不適
- フリーのスケジューラもある
  - OpenPBSと協調動作するMaui Scheduler
  - ソースの改変によるポリシ変更も可能
    - APIが切られているわけではないので，実際には難しい

研究の目的

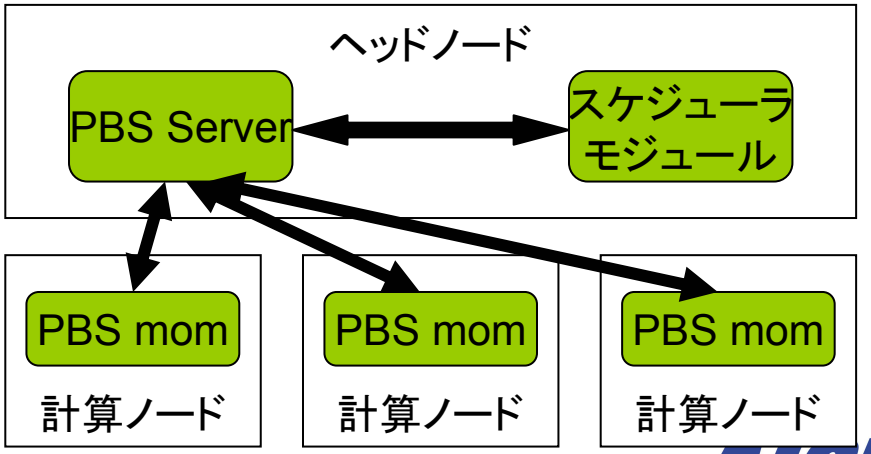
- 事前予約ポリシを検討するテストベットを構築
  - 既存ローカルスケジューラ TORQUE に対してスケジューリングAPIを整備
  - そのAPIを用いて，事前予約機能をテスト実装
- 上位レイヤであるスーパースケジューラとの連携のためのインターフェイスを整備
  - WSRF 標準に基づく予約インターフェイス
    - Globus Toolkit 4 (GT4) を用いた認証
  - 予約スロットを指定してのGRAMからのジョブ投入

発表の概要

- TORQUEの概要
- システムの提案
  - Java API
  - 事前予約可能スケジューラの実装
- 連携インターフェイス
  - WSRFによる予約インターフェイス
  - GRAMからのジョブ投入
- 予備評価
- 結論と今後の課題

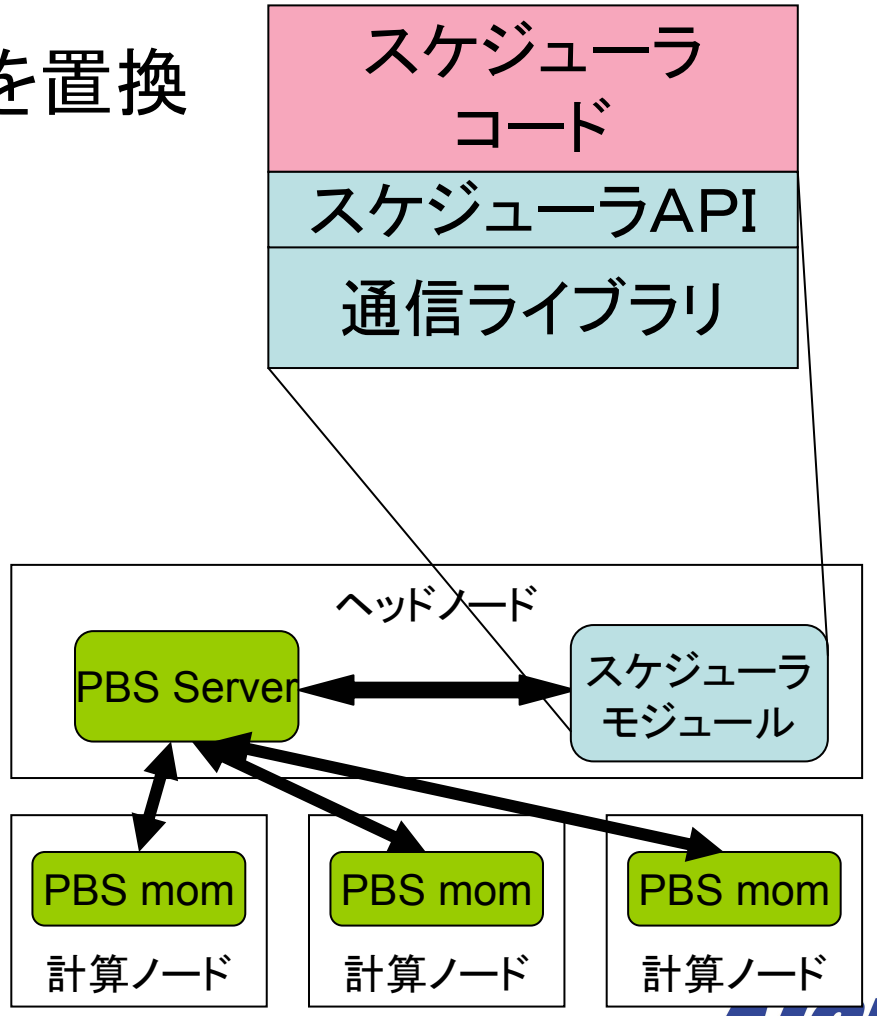
TORQUEの概要

- OpenPBSの一種
  - c.f. OpenPBS：メンテナンスされていない
  - 改変，再配布が可能
- 3つのデーモンから構成
  - PBS Server
    - セントラルサーバ
    - キュー・計算ノードを管理
  - スケジューラモジュール
    - PBS Server からのリクエストでジョブを計算機に配置
  - PBS Mom
    - 計算ノード上で機能
    - ジョブの起動・モニタリング



提案アーキテクチャ

- スケジューラを独自実装
  - デフォルトスケジューラを置換
  - Javaで実装
    - 開発用API を整備
- PBS Server との通信
  - 通信プロトコル
    - 比較的シンプルな
      テキストプロトコル
  - 認証
    - 特権ポートを利用



スケジューリングAPI

- PBSの内部状態を示すJava のクラスを提供
  - ServerStatus
  - QueStatus
- PBSの状態取得と制御を行うインターフェイスを提供
  - PBSInteface

PBSInterface の概要

```
public interface PBSInterface  {
   void      setSocket(Socket socket);
   Socket getSocket();
   void      authenticateUser(String userName,    int    localPort);
   void      disconnect();

   ServerStatus                 statusServer();
   BatchReplyStatusNode     statusNode();
   BatchReplyStatusQueue   statusQueue();
   BatchReplyStatusSelect   selectStatus(String queueName);

   void runJob    (String jobId, String destination);
   void runJob    (String jobId,  Collection<NodeInfo> nodes);
   void deleteJob(String jobId);
   void holdJob   (String jobId, HoldJobType holdType);
   void rerunJob (String jobId);
   void modifyJob(String jobId,  String attr,  String value);
}
```

接続関連

情報取得

操作

簡単なスケジューラの実装

```
public class SimpleFifoScheduler {
  public static void main(String[] args) {
    // start scheduling server
    PBSServerConfig servConf = new PBSServerConfig();
    ScheduleStarter starter =
      new ScheduleStarter(servConf);
    PBSInterface pbs = new TorqueImpl();

    // get scheduling order, and run
    ScheduleOrder order;
    PBSSchedulerCommandType cmd =
            PBSSchedulerCommandType.NULL;
    do {
      order = starter.waitOrder();
      Socket socket = order.getPBSServerSocket();
      pbs.setSocket(socket);
      cmd = order.getSchedulerCommand();

      if (cmd.mustRunSchedule()) {
        try {
          schedule(pbs);
        } catch (PBSException e) {}
      }
      socket.close();
    } while (cmd != PBSSchedulerCommandType.QUIT);
  }
```

```
  private static void schedule(PBSInterface pbs) {
    ServerStatus server = pbs.statusServer();
    if (!server.isReadyToUse() || server.getQueuedJ() == 0)
      return; // no jobs to schedule

    Collection<NodeStatus> nodes = pbs.statusNode().getAllStatus();

    for (QueueStatus queue : pbs.statusQueue()) {
      if (!queue.isReadyToRun() || queue.getQueuedJobs() == 0)
        continue;    // no jobs to run

      for (JobStatus job : pbs.selectStatus(queue.getName())) {
        if (!job.isReadyToR
          continue;    // cannot run now
        for (NodeStatus node : nodes) {
          if (!job.isRunnableOn(node))
            continue;    // node is down

          String jobId = job.getJobId();
          String destination = node.getName();
          pbs.modifyJob(jobId, "comment", "Job started on " + new Date());
          pbs.runJob(jobId, destination);
          return;
        }
      }
    }
  }
}
```

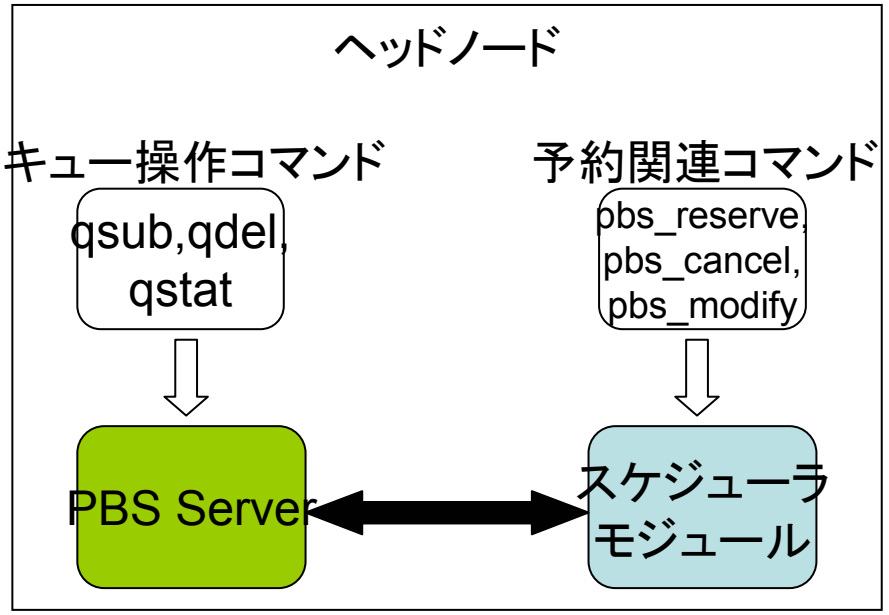
計算ノード情報を取得

キューからジョブを取得

ジョブをノードに割り当て

予約機構の実現

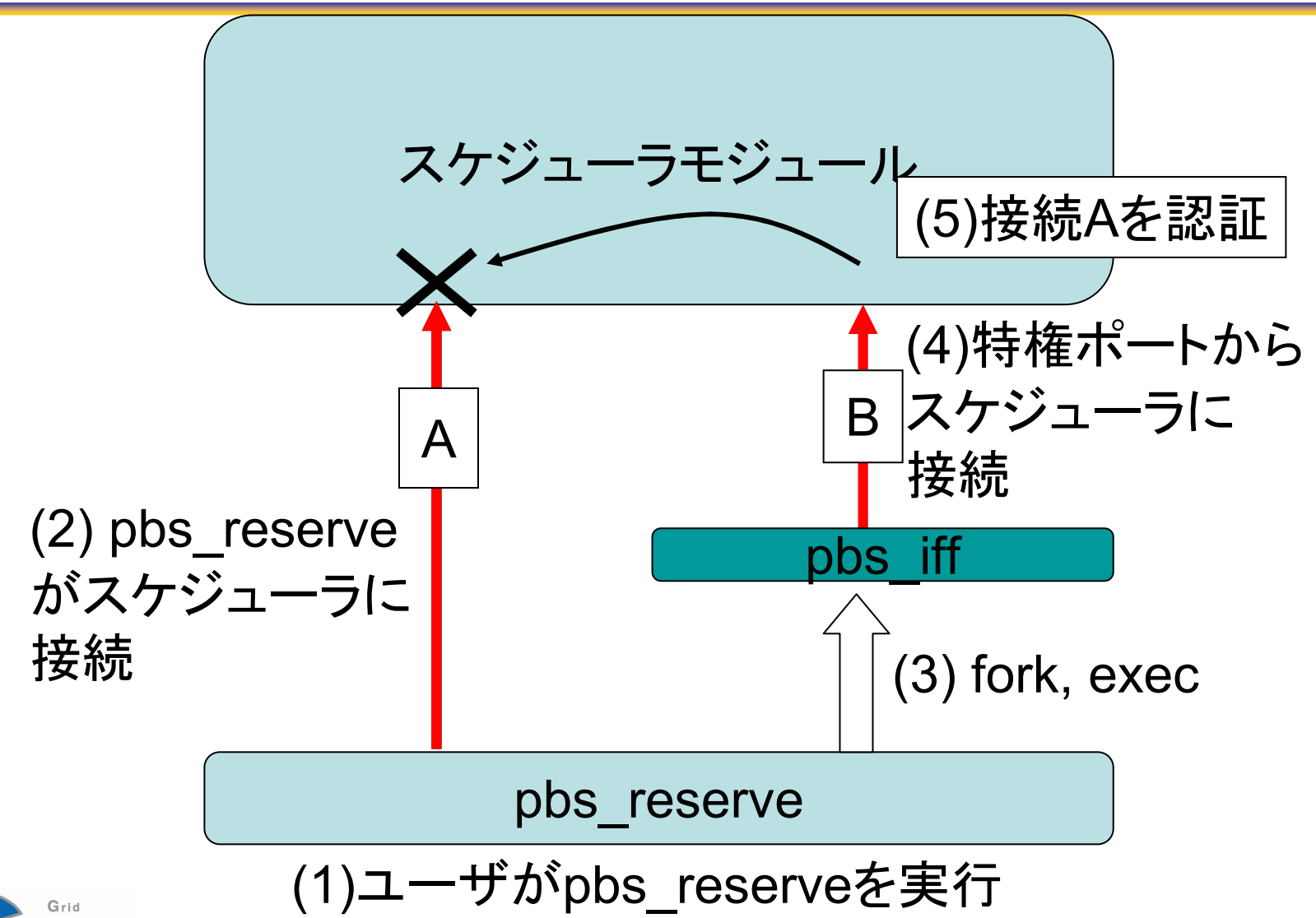
- スケジューラモジュールに予約テーブル管理機構を導入
  - スケジュール時にテーブルを参照
- 予約コマンドを提供
  - スケジューラモジュールにアクセス
  - RMIで通信
- 予約テーブルを永続化
  - db4objects を使用



予約関連コマンド

- pbs_reserve
  - 予約をリクエスト
  - 入力：開始・終了時刻,ノード数
  - 出力：予約ID
- pbs_rsvcancel
  - 予約キャンセル
  - 入力: 予約ID
- pbs_rsvstatus
  - 予約情報取得
  - 入力： 予約ID
  - 出力： 予約状態
- pbs_rsvmodify
  - 予約変更
  - 入力： 予約 ID, 開始・終了時刻,ノード数

TORQUEのセキュリティ



Grid Technology Research Center AIST

AIST

使用シナリオ

- まず予約

```
> pbs_reserve -s 12:00 -e 14:00 -n 1
Reserve succeeded: reservation id is 14
```

- 予約状態の確認

```
> pbs_rsvstatus
 id owner          start              end      duration state
 14 nakada      Feb 20 12:00   Feb 20 14:00     2h00m Confirmed
```

- 予約ID を指定してジョブをサブミット

```
> qsub -W x=rsvid:14 script
```

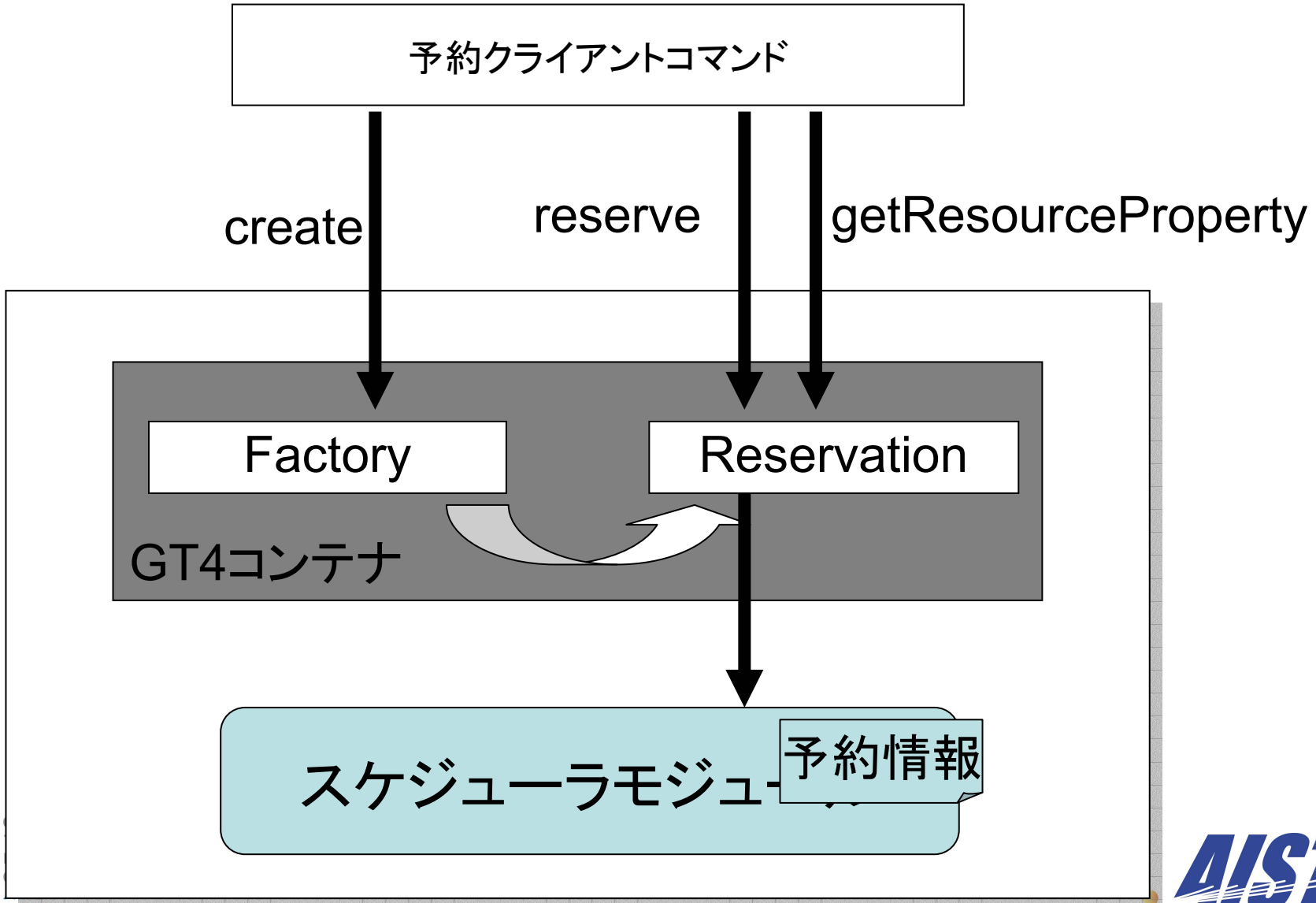
連携インターフェイス

- コアロケーションの実現には，予約インターフェイスのスーパスケジューラへの公開が必要
  - セキュリティ
  - 将来の標準化
- GT4を用いたWSRFインターフェイスで実現
  - GT4 によるセキュリティ
    - PKIによる認証
    - grid_mapfile による認可
  - WSRF (Web Services Resource Framework)
    - OASIS で標準化
    - WSDL でインターフェイスを定義

WSRFによるインターフェイス

- ファクトリサービス
  - Reservation サービス本体を作成
  - CreatePBSReservation
    - 予約期間，ノード数を指定
    - Reservation サービスへのEPR (ポインタ) を返却
- Reservation サービス
  - reserve : 実際に予約を実行
  - getStatus: リソースプロパティ内の予約情報を更新
  - cancel: キャンセル
  - modify: 予約変更
  - getResourceProperty:
    - 予約の状態を取得

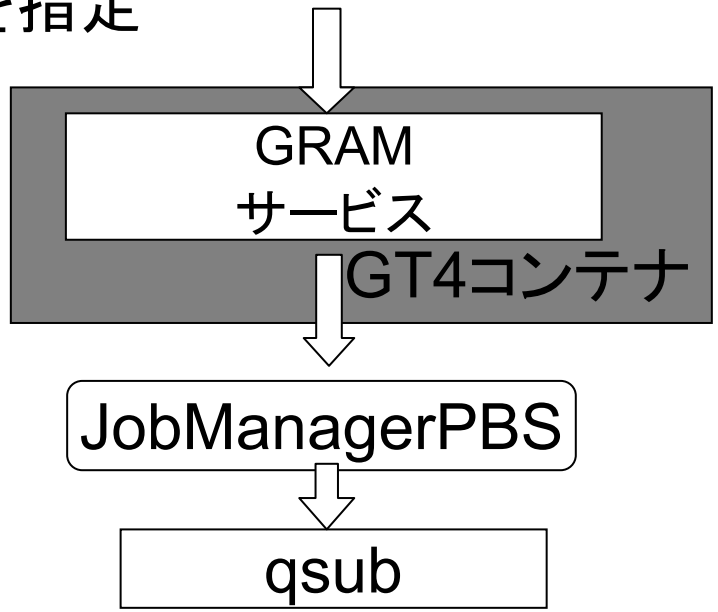
WSRFによる予約インターフェイス



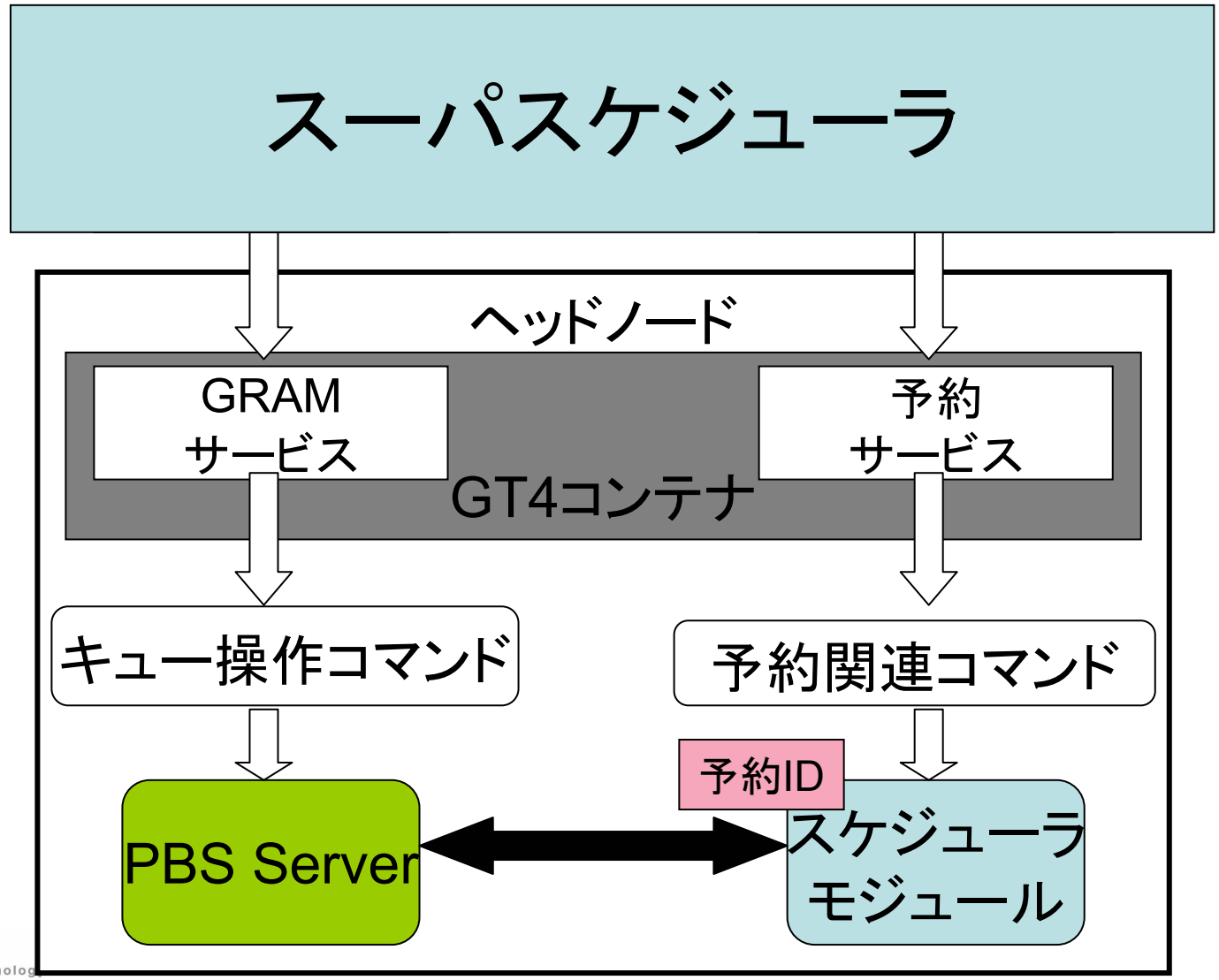
GRAMとの連携

- GRAM 経由で，予約IDを指定
  - ジョブ記述に予約IDを記述
    - GRAM 標準の拡張書式を利用
    - GRAM のPBS 用 Job Manager を変更
    - qsubのオプションに予約IDを指定

```
<extensions>
  <schedulerAttrs name="reservationID">
      xxxxxxxxxx
  </schedulerAttrs>
</extensions>
```



全体の流れ


スーパスケジューラ
ヘッドノード
GRAM
サービス
予約
サービス
GT4コンテナ
キュー操作コマンド
予約関連コマンド
予約ID
PBS Server
スケジューラ
モジュール

予備評価

- 予約と予約のキャンセルにかかる時間を計測
  - 直接アクセス・GT4経由アクセス
- 評価環境
  - すべてを1台の計算機内での実行
    - PBSサーバ，GT4コンテナ，クライアント
  - Pentium III 1.4 GHz, 2CPU, 2Gbyte

	予約	キャンセル
直接アクセス	0.78 s	0.68 s
GT4経由	1.7 s	1.3 s

- 直接アクセスのオーバヘッド
  - クライアント側でのRMI関連ライブラリのロード
- GT4経由のオーバヘッド
  - 認証・認可

おわりに

- 事前予約機能を持つローカルスケジューラを実現
  - Java APIによる拡張性
  - 事前予約機能検討のテストベッドに
- Globus Toolkit 4を用いた外部インターフェイス
  - 予約機能を外部に提供
  - GRAMと連携して，予約とジョブサブミッションを同じ枠組みで実現

今後の課題

- 予約機能とFCFS+優先順位との共存
  - フェアシェア
- Grid Engineのサポート
  - 同一のスケジューラをGrid Engine でも
  - c.f. Maui scheduler はGrid Engine のサポートを中止
- 実運用でのテスト
  - ネットワークとのコアロケーション
  - スーパースケジューラとの連携によるWSRFインターフェイスの妥当性評価

謝辞

本研究の一部は，文部科学省科学技術振興調整費
「グリッド技術による光パス網提供方式の開発」による.