Ninf-G2: 大規模環境での利用に即した高機能、高性能GridRPCシステム

産業技術総合研究所　グリッド研究センター

田中　良夫　　中田　秀基

朝生　正人　　関口　智嗣

Grid Technology Research Center AIST

AIST

背景

- グリッドはテスト段階から実用化へ
  - 要素技術が揃ってきた／固まってきた
  - インフラが整備されてきている／されそう
- 大規模アプリケーションのグリッドでの実行への期待
  - システム、アプリ双方の協力が必要
  - try & error で知見・経験を積み、それをシステム・アプリにフィードバック
- とはいえ…
  - どうやってプログラムを書く？
  - どうやってたくさんのリソースを使いこなす？
- MPIだけじゃない
  - GridRPCの方が適したものはたくさんある。適材適所

GridRPC

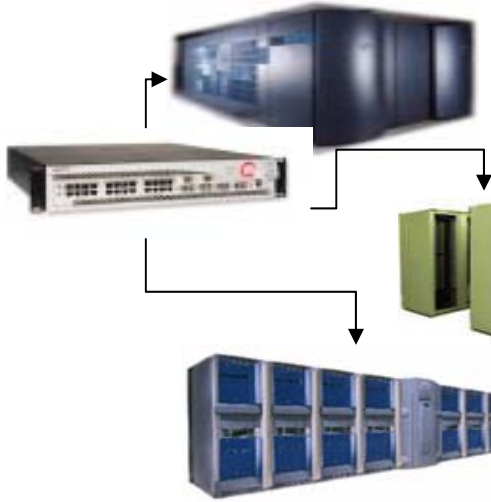
ユーザ

② 結果の通知

Internet

① 関数呼び出し

スーパーコンピュータの遠隔利用



特殊なライブラリの遠隔呼び出し

グリッド上の複数の高性能計算機を利用した大規模計算

ポータル、ＡＳＰのバックエンドとして利用

現在GGF GridRPC WGにてAPIの標準化を進めている

GridRPCの現状

- **Ninf、NetSolveプロジェクトは1994年頃スタート**
  - **当初はクライアント･サーバ型(1対1)の計算モデルを想定**
  - **クラスタの普及に伴い、マスター･ワーカ型(1対多)の計算モデルが主流に**
- **既存のシステムおよびその利用**
  - NetSolve
    - **細胞生理学向けモンテカルロシミュレーション**(MCell)
  - OmniRPC
    - HMCS-G (GRAPE-6 + CP-PACS)
  - Ninf
    - **数値、組み合わせ最適化問題** (BMI, SDPA)
    - **レプリカ交換モンテカルロ** (HPC Challenge @ SC2002)
  - Ninf-G
    - ApGrid Testbed**上での気象シミュレーション** (Grid Demo WS)
- **大規模環境･大規模アプリケーションでの利用？**

Ninf-G2

- 大規模環境での利用に即したGridRPCシステム
- GridRPC APIを用いて、グリッド上で動作するアプリケーションを簡単に開発
- 複数（数台～１０数台程度）の大規模な（数１０～数１００プロセッサ程度の）クラスタ群に関数単位で計算を振り分ける並列プログラムの容易な開発・実行
- 総数１０００プロセッサ規模のグリッドにおいて効率良く動作することを目標とする

これからの話の流れ

- GridRPC APIと参照実装
- Ninf-G2の設計
  - 設計に際して特に考慮した点
  - 設計および実装の方針
- 予備評価
  - 複数のRPCを効率良く行なう方法
- まとめと今後の予定

GridRPC API (RPCの手順)

- 初期化

```
grpc_initialize(config_file);
```

- 関数ハンドルの作成
  - リモートライブラリとの接続を抽象化したもの

```
grpc_function_handle_t  handle;

grpc_function_handle_init(
  &handle, host, port, “lib_name”);
```

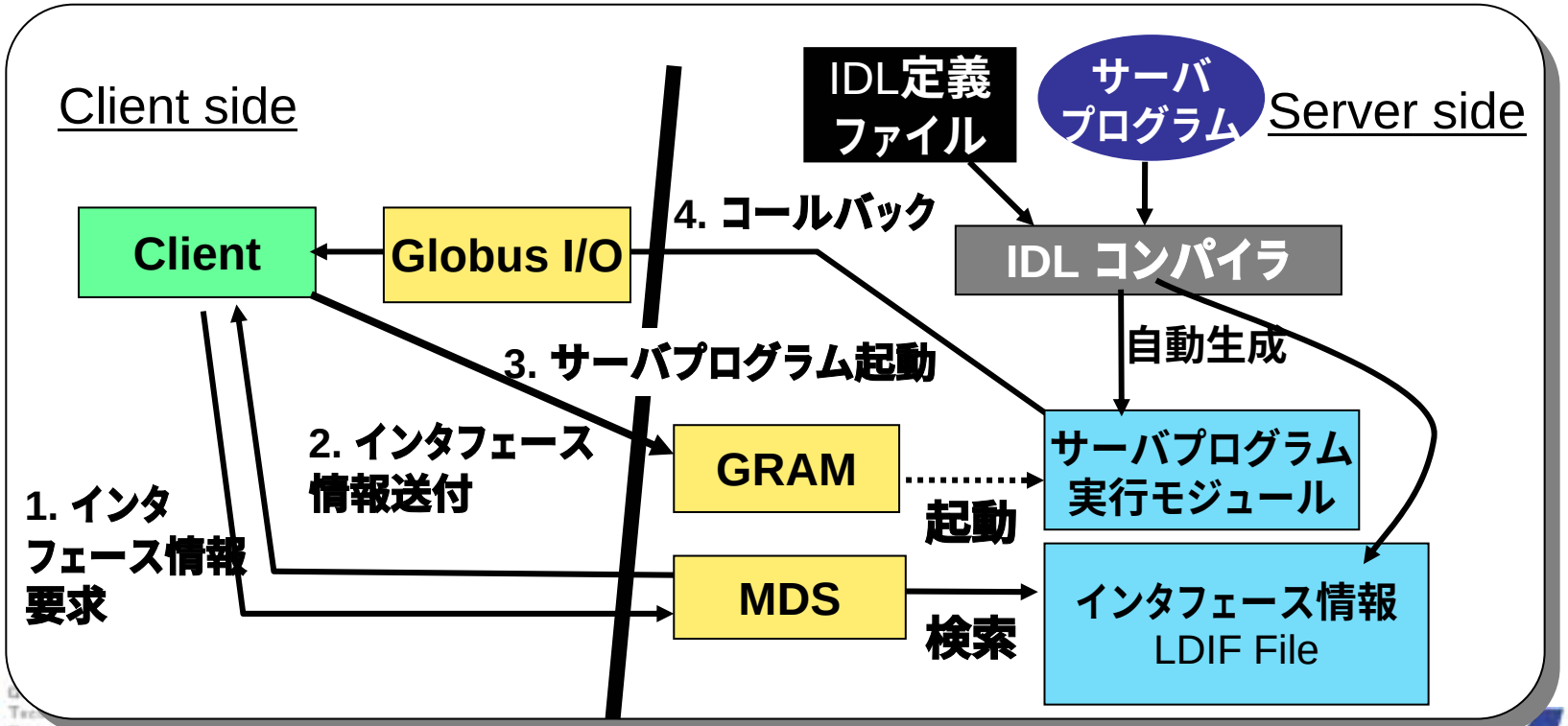
- RPC
  - 関数ハンドルを引数に遠隔手続き呼び出し

```
grpc_call(&handle, args…);
          とか
grpc_call_async(&handle, args…);
```

Ninf-G: GridRPC System on the Globus Toolkit

- **GridRPC APIの参照実装**
- **Globus Toolkit上に構築**
  - MDS: **スタブインタフェース情報管理**
  - GRAM: **サーバプログラムの起動･終了管理**
  - Globus I/O: **クライアントーサーバ間のセキュア通信支援**

関数ハンドル作成時の処理



Ninf-G2の設計

設計方針 (1/2)

- 大規模クラスタ利用におけるオーバーヘッドを抑える
  - 主に初期化のオーバーヘッド
  - Ninf-Gの場合、認証およびMDS検索
- 複数のサーバを利用するための機能を充実させる
  - サーバごとに異なる属性を指定、非均質性を隠蔽
  - 属性例：ポート番号、ジョブマネージャ、プロトコル
- 大規模アプリケーションの実行に対応する
  - ハートビート
  - モニタリング
  - 中間結果の表示

設計方針 (2/2)

- データ送受信を効率化する
  - 冗長なデータの送受信を省く機能の実装
  - 効率の良いデータ送受信
- システムを肥大化させない
  - 何でもかんでも詰め込むことはしない
    - 耐故障性機能
    - スケジューリング、ブローカリング
    - 障害については、エラーコードを適切に上げる
  - GridRPC APIはRPCのコアな機能のためのAPI
    - (必要に応じて) 上位のAPIを提供

大規模クラスタ利用におけるオーバーヘッド抑制への対応

- 複数の関数ハンドルをまとめて効率良く作る機能を導入し、そのためのAPIを提供
  - Ninf-Gでは関数ハンドルの作成に際してGRAM呼び出し
  - 複数の関数ハンドルを効率良くまとめて作成
  - 関数ハンドルの配列の生成・破棄をするAPIを提供
    - grpc_function_handle_array_initialize_np();
    - grpc_function_handle_array_default_np();
  - 実装方法は後述
- MDSを利用せず引数情報を取得する機能を提供
  - LDIFファイルをクライアント側に置く
  - リモートライブラリの起動に必要な情報だけクライアント側に置く

複数のサーバを利用するための機能の提供

- 記述性の高いクライアントコンフィグレーションファイルの利用
  - タグを使って構造化
  - jobmanager, port, protocolなどの属性をサーバごとに指定可能
- サーバコンフィグレーションファイル
  - サーバ側の設定(リモートライブラリの動作に必要な情報)を設定
    - ファイル転送の際に一時的に使用するディレクトリ
- 関数ハンドル作成時のタイムアウト設定
  - サーバの環境、設定、状態の非均質性に対応
  - クライアントコンフィグレーションファイルのSERVERセクションのjob_timeoutで設定

大規模アプリケーションへの対応

- ハートビート機能を提供
  - リモートライブラリがクライアントにハートビートを送付
- コールバック機能を提供
  - リモートライブラリがクライアント側の関数を呼び出す
    - 計算の途中結果の表示
    - 計算の一部をクライアント上で実行
    - 対話的な処理
- セッションキャンセル機能の実装
  - セッション：クライアントがリモートライブラリを呼び出し、その実行終了を認識するまでの処理
  - GridRPC APIはセッションをキャンセルするためのAPIを提供している
    - grpc_error_t grpc_cancel(int sessionID);
  - サーバ側のAPIとして、キャンセル要求が来ているかどうかを調べるAPIを追加
    - ngstb_is_canceled()
  - キャンセル機能を実現するためには、リモートライブラリの中で定期的にngstb_is_canceled()を呼び出し、クライアントからキャンセル要求が来ていないかどうかをチェックする

データ送受信の効率化

- リモートオブジェクトの実装
  - 状態を保持させることにより、冗長なデータ送受信を省く
  - リモートオブジェクトを定義するIDL，リモートオブジェクトを操作するクライアントAPIを提供
- バイナリプロトコルの実装
  - クライアントとリモートライブラリとの間でのデータ送受信に、XMLまたはバイナリのいずれかを利用可能とする。
    - クライアントコンフィグレーションファイルのSERVERセクションのprotocolで指定
- 送信データの圧縮 (検討中)
  - データを送信する際に必要に応じて送信データの圧縮を行なう
  - クライアントコンフィグレーションファイルで送信データの圧縮について指定
    - compress　　off or zlib
    - compress_size サイズ
  - compress_size以上のデータを圧縮単位とする。

予備評価

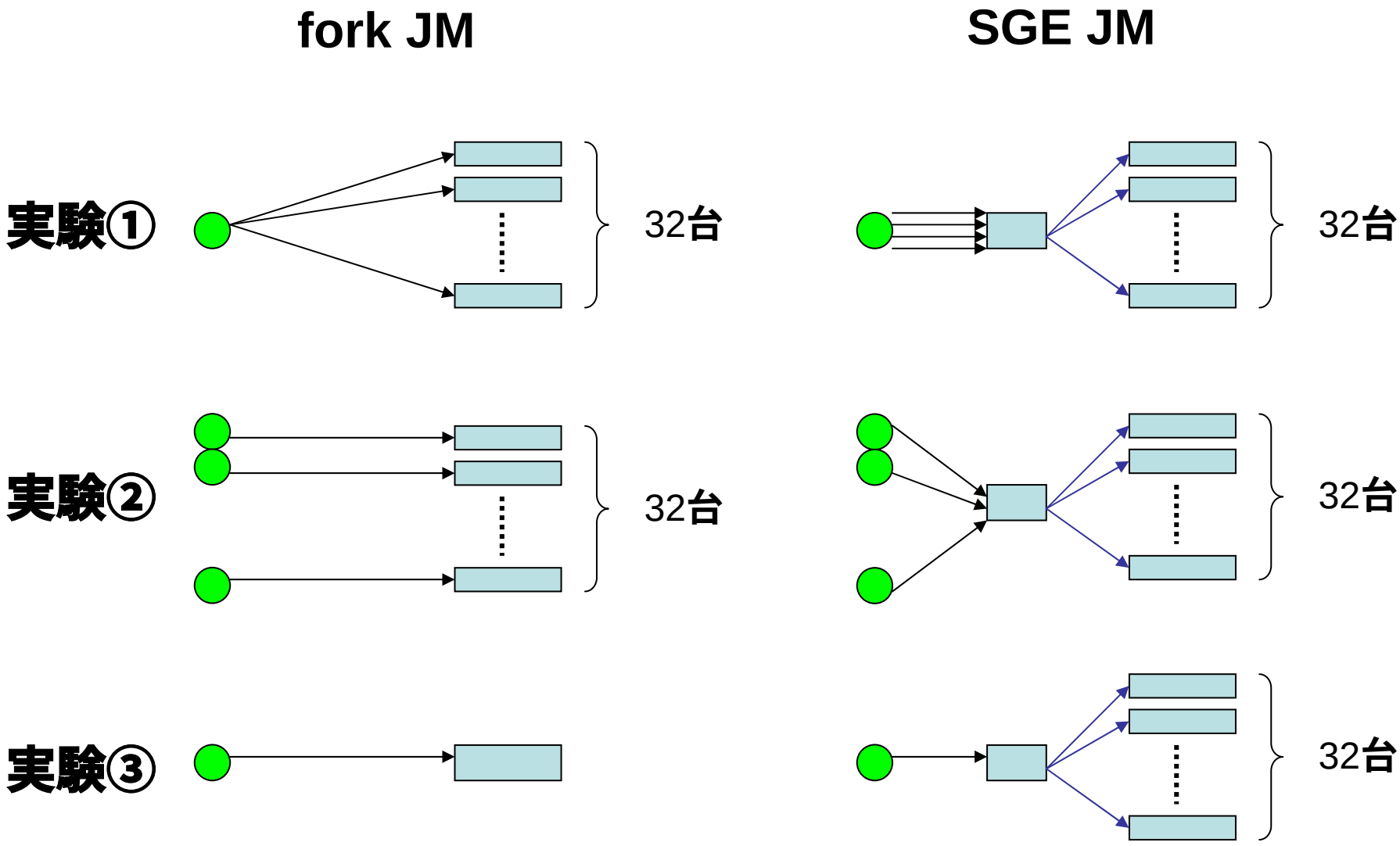
複数の関数ハンドルをまとめて作成する方法

- クライアント側でマルチスレッド並列
  - 生成された関数ハンドルの個別制御が容易
  - ある程度はこれでうまくいくことは経験済み
- GRAMのcount をNにする
  - 認証が1度で済む
  - GRAM的には1つのインタフェースになるので、関数ハンドルの個別制御が複雑

実験

- **実験方法**
  - 以下の3通りの方法で、32個のリモートライブラリの起動に要する時間（関数ハンドル作成を依頼してから、すべての関数ハンドルが作成され、リモートライブラリが起動されるまでの時間）を計測
    1. 1個のスレッドでgrpc_function_handle_init()を32回繰り返す
    2. 32個のスレッドを生成し、各スレッドでgrpc_function_handle_init()を呼ぶ
    3. 1度のGRAM呼び出しで複数のジョブを起動する機能を利用して32個のリモートライブラリを起動する
- **実験環境**
  - 32ノードのLinuxクラスタ
  - CPU: Dual P-III 1.4GHz
  - Interconnection: Gigabit Ethernet
  - Globus
    - Version: 2.2.4
    - JobManager: fork (all nodes), sge (front node)
  - Ninf-G Ver.1

実験①、②、③の振る舞い



実験結果

- 各テストを5回実行した際の計測値と平均値

	実験① 1thread		実験② 32threads		実験③ GRAM	
JobManager	SGE	fork	SGE	fork	SGE	fork
最長	59	29	46	10	13	27
	55	29	41	8	13	27
	54	27	41	8	12	27
	52	27	40	8	12	26
最短	51	27	39	8	11	26
平均	54.2	27.8	41.4	8.4	12.2	26.6

単位は秒

マルチスレッド化の効果

認証1度

他のモジュールに期待するもの

- 資源管理
  - スケジューラ、ブローカ
    - ローカルなスケジューラとリンク
  - (ファンシーな) リソース予約システム
- 耐故障性
- より使いやすく
  - リモートライブラリのシッピング、自動ビルド
- 上位階層
  - 指示文による自動分散化コンパイラ
  - Portal, PSE,…

まとめと今後の計画

- 大規模環境での利用に即したNinf-G2の設計および実装
  - 外部仕様：設計完了
  - 内部仕様：予備評価を行いつつ設計中
- Ninf-G2の仕様を持ちネタとしてGridRPC WGでのAPIの標準化活動に参加。現在はMLを通じて議論の真っ最中
- 今後の予定
  - SC2003にてVersion 0.9を配布
  - 年度内にはVersion 1.0を配布