GridRPC を用いたタスクファーミング API ライブラリの設計

谷 村 勇 輔
産業技術総合研究所 グリッド研究センター

GridRPC プログラミング

- 典型的な GridRPC を用いたタスク並列プログラム

```
grpc_init()                                          → システムの初期化処理
for(i=0; i<N; i++)
   gprc_function_handle_init(handle, host, func)     → ハンドルを作成
    :
for(i=0; i<N; i++)                                     （ループ）
   gprc_call_async(&handle[i], &session[i], …)       → ノンブロッキング RPC
    :
grpc_wait_all()                                      → RPC の完了待ち
    :
grpc_function_handle_destruct()                      → ハンドルの破棄
grpc_finalize()                                      → システムの終了処理
```

- GridRPC API の利用により，アプリケーション開発のコストが削減
  - アーキテクチャの違いを気にしない
  - 計算機間のデータ通信を気にしない
  - 標準化によりクライアントプログラムを共通に利用できる

これまでの事例研究

- 実際のアプリケーションを用いた事例研究
  - 大規模実行
    - 500 CPU を用いた実験（SC'03）
    - プログラム：気象予測シミュレーション
  - 長時間実行
    - アジア太平洋のグリッドテストベッドを用いた実験（2004）
      - 3ヶ月間の断続的な実行（連続では約１週間）
    - プログラム：TDDFT
  - （大規模＋長時間）実行
    - 1800 CPU を用いて10時間実行（SC'04）
    - プログラム：QM/MD
- アプリケーション開発について学んだこと
  - 大規模・長時間実行を達成するには，タスク割り当ての自動化や耐障害機能などを何らかの形で実装する必要がある

長時間実行の実験の成果

- 耐障害性をもつアプリケーション開発に関する留意点を明らかにした
  - エラーコードの適切なハンドリング
  - ハートビートを利用した障害検知
  - バックグラウンドでのサーバプログラムの再起動および初期化
- GridRPC の上位 API 設計への課題と方針を示した
  - プリミティブな機能のみを備えた「エンドユーザ API」では規定されていないスケジューリングや耐障害機能を提供
  - 汎用化できるコンポーネントの抽出
  - GridRPC においてミドルウェア API を整備
- ターゲットとする上位 API
  - Ex. Task Farming, Task Sequencing, . . .

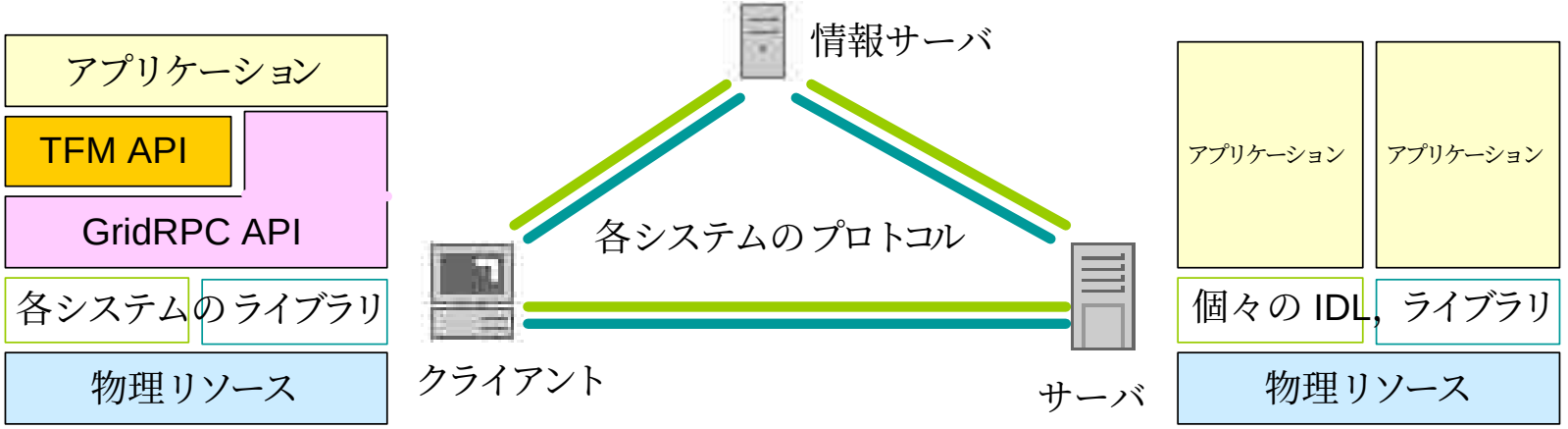
研究目的

- 上位 API を提供するライブラリを検討して，アプリケーション開発のコストを削減する
- 上記の実装の枠組みや，実装に必要な共通機能を整理して標準化への指針を示す

- タスクファーミング（TFM）
  - 入力データやパラメータを変えながら同一プログラムを並列に実行すること
  - タスクファーミング機能により，以下のフローを簡単に記述できるようになる
    - 計算データ，パラメータの範囲を与える
    - タスクを投入する
    - 結果を回収する

TFM API と GridRPC

- TFM API は，クライアント側の GridRPC API を用いたプログラミングを容易にする
  - 現状，互換性のないバックエンドを意識させない
- TFM API は，それぞれの GridRPC システムで実装可能であること



これまでのグリッドの TFM API

- NetSolve Request Farming
  - MCell ではパラメータ探索を行う際のミドルウェアとして利用されている
  - Farming API を利用すると性能が悪い
  - 全てのタスクを１度に待つ方法しか存在しない
    - 画像処理用のアプリケーションなどでは不便
- Ninf 試作 API
  - コールバックを利用することでメモリの消費を抑える
  - ユーザからの要求
    - コールバック記述が難解である
    - サーバの初期化を行いたい
    - データのブロードキャストがほしい

TFM API に対する要請

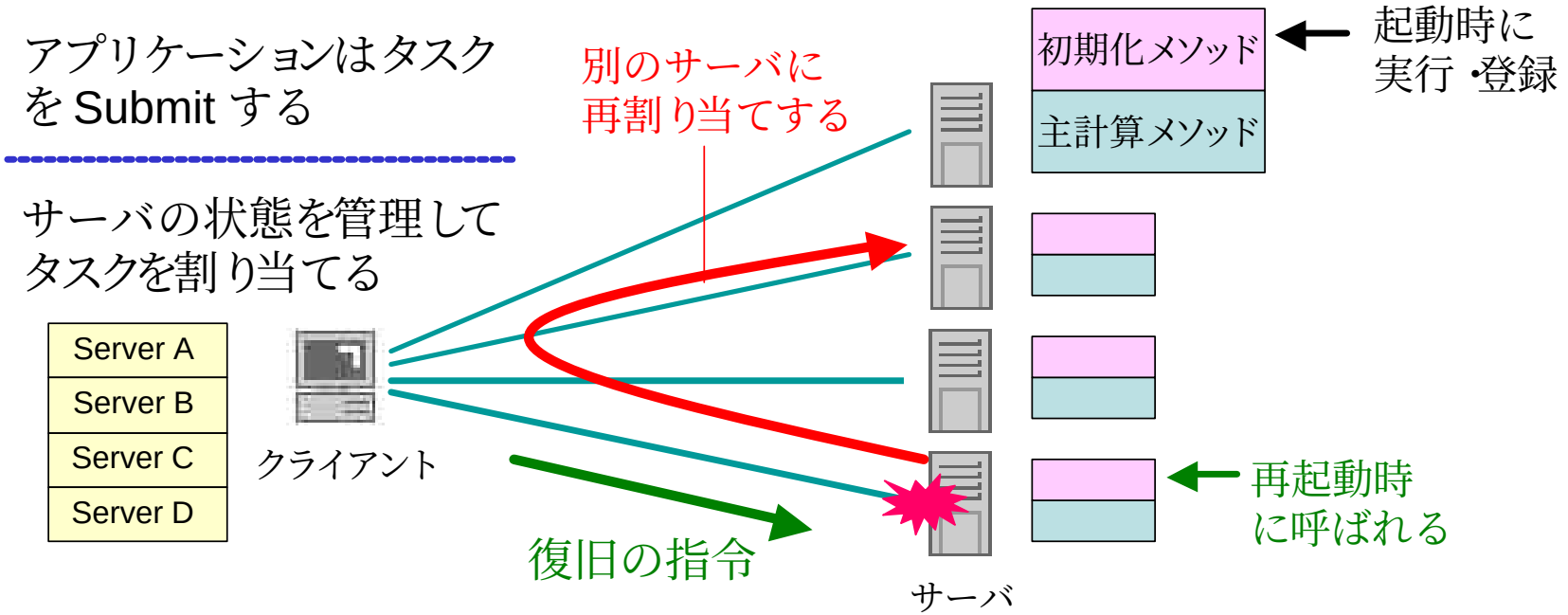
- タスクを自動的に計算機に割り当ててほしい
  - 実行先（ホスト名）を指定したくない
  - 性能や安定性を考慮して適切に割り当ててほしい
    - 例．割り当て順序の調整，不安定なサーバを削除，冗長投入
- 障害処理は TFM 内部で実装されてほしい
  - タスク実行は成功するまで複数回トライしてほしい
  - 遠隔プログラムが自動的に起動・再起動されてほしい
- TFM のパラメータ生成，結果処理部分だけを実装したい
  - 特定のアプリ向けの TFM ツールを上位に作りたい
    - 例．対話的な実行，パラメータを半自動的に生成
  - GridRPC の上に TFM を実装する枠組みが整理されてほしい
    - エンドユーザ API の記述性の向上
    - ミドルウェア API の提供
  - Farming 環境を初期化する API がほしい

TFM API 設計の指針（1）

- **タスクの実行先を自動的に決める機能をサポートする**
  - 実行先を指定できる機能（ただし，IDで指定）は残す
- **タスクの割り当てレンジを調整する機能をサポートする**
  - メモリ容量を考慮して最大値をユーザが指定可能
  - 実行時間を見積もり，割り当てレンジを自動調整
    - タスク当たりの計算量が均一であり，計算機の性能がヘテロである場合に，最終的な実行時間が最小になるようにする
- **障害に関係なく全てのタスクが実行されるようにする**
  - 内部で自動的にタスクの再実行を行う
    - タスクの実行先が指定されている場合は除く
- **未処理のタスク数少ない場合に冗長投入を行う機能をサポートする**
- **障害により停止したサーバを復旧させる機能をサポートする**
  - 定期的に復旧を試みるようにする
  - 全ての復旧処理はバックグラウンドで行う
- **起動した遠隔プログラムを個々に初期化する機能をサポートする**
  - 初期化メソッドは保存しておき，障害復旧の際にも呼び出す

TFM API 設計の指針 (2)

- 初期化メソッド，主計算メソッドを定義してタスクファーミングを行う場合の例
  - 初期化メソッド： 遠隔プログラムの初期化

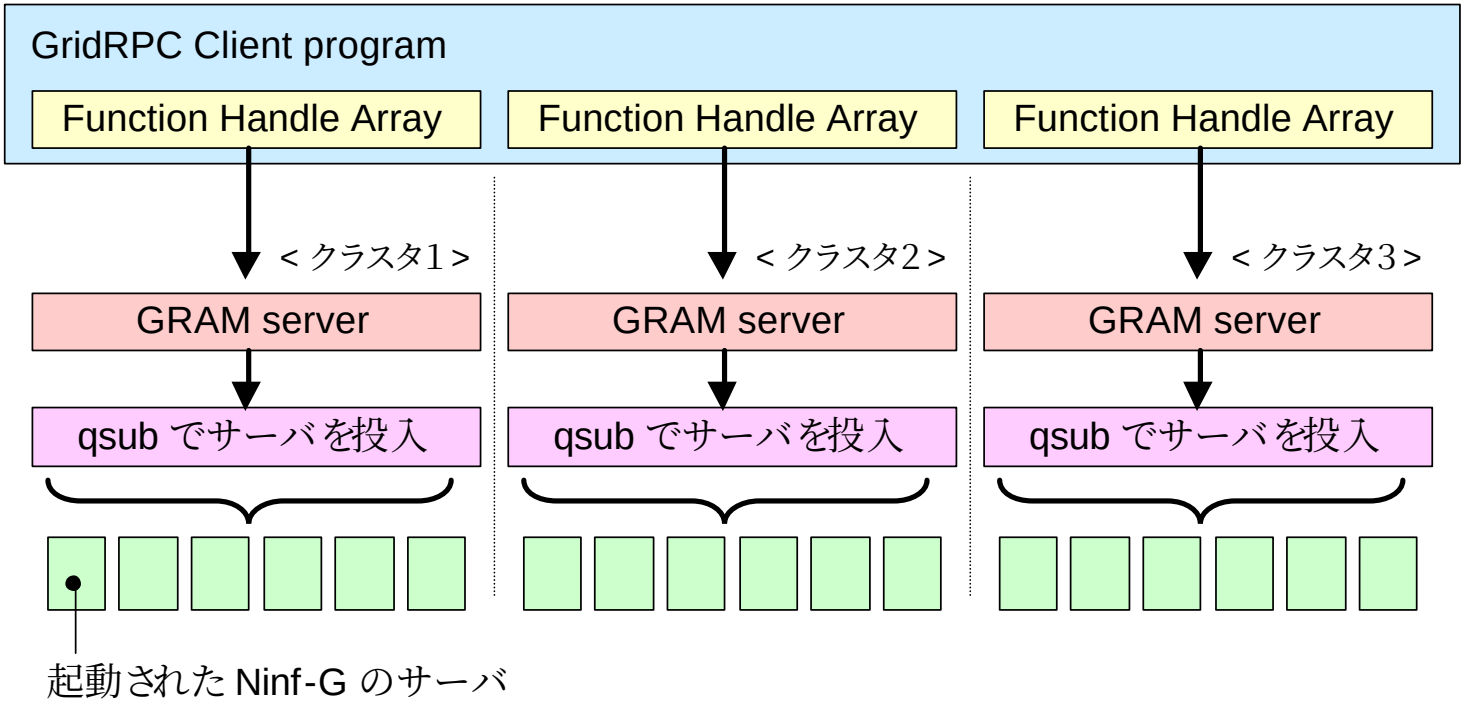


実装の概要

- 上位ミドルウェアとして共通のコンポーネントを作成
  - GridRPC サーバ（起動された遠隔プログラム）の管理
  - タスクの管理
  - 障害検知
  - 障害で停止したサーバのバックグラウンドでの復旧
- タスクファーミング API の検討
- GridRPC および Ninf-G を用いて実装
  - 実装に用いた Ninf-G 独自の機能
    - リモートオブジェクト機能
      - 複数の RPC 間でリモートのデータを共有する仕組み
    - RPC の実行情報の取得機能
    - 送信データの転送確認を待たずに非同期呼び出し関数から返る機能
    - サーバの一括起動の機能

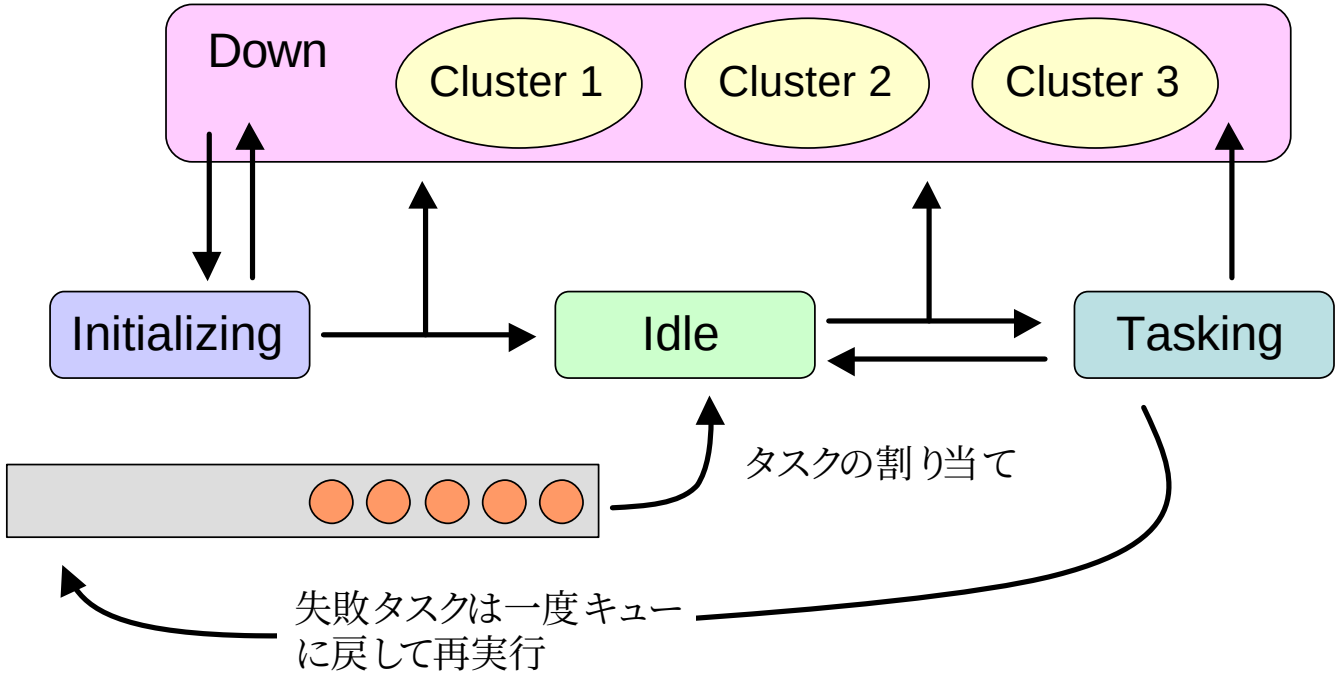
実装：一括起動のサーバを個々に管理

- 各 Function Handle Array（FHA）に属する Handle を FHA に関係なく管理する
  - FHA はサーバを高速に起動するための Ninf-G の独自機能
    - Globus GRAM のオーバーヘッドが生じるのを防ぐため
  - NetSolve では既にサーバが起動していることが前提



実装：タスクとサーバの状態管理

- **Down の状態は各 Handle_Array 毎に管理する**
  - 復旧操作は Handle Array 毎に行うため
- **Idle プールにおいて定期的にサーバの順序を並び替える**



実装：Argument Array API の利用

- **任意の RPC 引数列を構築するための API**
  - GridRPC の上位 API を提供するためのミドルウェア用の関数群
  - Ninf では arg_stack として実装していた（va_list は扱えなかった）

タスク投入

```
int grpcg_submit(char * func, …){
    :
  va_start(ap, func);
  grpc_arg_array_init_with_va_list(<handle>, arg_array, ap);
  va_end(ap);
    :
}
```

va_list から arg_array を作成可能

実行先として同じ型のハンドルを渡す

格納しておいた引数列を渡す

タスク実行

```
int grpcg_i_dispatch(){
    :
  grpc_call_arg_array_async(<handle>, &session_id, arg_array);
    :
}
```

提案 API (1)

- **タスクファーミング API ライブラリの初期化，終了**

int grpcg_init(char * conf, sched_attr_t * sched, ft_attr_t * ft);

int grpcg_fin();

- **遠隔プログラム（Ninf-G サーバ）の起動**

int grpcg_remote_init(int num_pe, char * func, …);

  - 起動時に初期化メソッドを登録可能

int grpcg_remote_init_n(int server_id, int num_pe, char * func, …);

  - 各サーバに ID を付与するとともに異なる初期化メソッドを登録可能

- **遠隔プログラムの終了**

int grpcg_remote_fin(int num_pe);

int gprcg_remote_fin_n(int server_id, int_num_pe);

提案 API (2)

- **タスクの投入**

int grpcg_submit(char * func, …);

int grpcg_submit_n(int server_id, char * func, …);

  - タスクの実行先（サーバ）を ID で指定

int grpcg_submit_r(void * ref, char * func, …);

  - 投入タスクに任意のポインタを関連づけておく

int grpcg_submit_nr(int server_id, void * ref, char * func, …);

- **タスクの完了待ち**

int grpcg_wait_all();

int grpcg_wait_any(int * task_id, void ** ref);

- **タスクのキャンセル**

int grpcg_cancel(int task_id);

API の利用例

- **NAS Grid Benchmark の ED のプログラミング例**

初期化メソッドの登録なしのサーバの起動 →

ホスト名を指定せずにタスクを投入. 障害が発生しても, 全てのタスクがどこかのサーバで実行される.

```
    :
rc = grpcg_init("server.list", &sched, NULL);
    :
grpcg_remote_init(NUM_PE, NULL);
    :
for(i=0; i<NUM_TASK; i++){
   grpc_sumit("SP.S", "SP", ..., &i, &width, &depth, ...);
}                                        SP.S の初期化パラメータ
rc = grpcg_wait_all();

grpcg_remote_fin(NUM_PE);
grpcg_fin();
   :
```

API の適用範囲

- 完全に独立したタスクを並列実行するアプリケーションに広く適用できる

- 適用しにくい場面
  - 「初期化 → 主計算」以外の依存関係をもつタスク群には適用しにくい
    - NAS Grid Benchmark の ED 以外のアプリケーションには適用できない．ワークフローを構築する必要がある．
  - クライアントとサーバ間のデータ転送やタスクの実行先を明示的に指定するプログラミングには向かない
    - エンドユーザ API を用いて実装すべき
    - １つのタスクをどう定義するかを検討すべき

GridRPC への要請

- Ninf-G の拡張機能の一部を GridRPC として規定されることが望ましい
  - RPC の実行情報取得 API の規定
    - タスクを適切にサーバに割り当てるために用いられる
      - 実行情報：引数データの転送時間，リモートでの計算時間など
  - データ転送のタイミングに関する規定
    - データ転送は上位ミドルウェアの内部で管理されるため，ノンブロッキング機能が提供されれば十分である
- より広く議論されるべき点
  - サーバの初期化機能を実装するために，リモートオブジェクトの仕組みを利用した
  - 性能を考慮して，複数のサーバを一括起動する仕組みを利用した

Argument Array API への要請

- 引数のポインタを格納するだけでなく，引数データの複製を行う機能が必要

- 理由１
  - アプリケーションユーザにデータ領域を書き換えないことを常に求める
    - セッション管理はライブラリ内部でなされるが，データの書き換えのタイミングはアプリケーションでケアすることになる
- 理由２
  - 上位ミドルウェアにおいて，タスクの冗長投入を行う機能が実装しにくい
    - OUT，INOUT の引数データの書き込みを制御する必要がある

まとめ

- GridRPC の上位ミドルウェアとしてタスクファーミング API の検討を行った
  - （標準化の最終段階にある）エンドユーザAPI
  - （標準化の議論が始まっている）Argument Array API
  - Ninf-G の独自機能

- 上位ミドルウェアを作成するために
  - GridRPC として規定されることが望ましい機能
    - RPC の実行情報の取得方法
    - データ転送のタイミング
  - Argument Array に必要な機能
    - 格納されるデータの複製機能，あるいは排他アクセス機構