計算機アーキテクチャ　147−7
ハイパフォーマンスコンピューティング　89−7
（2002．3．7）

Grid 環境における大規模クラスタ向けジョブマネージメントアーキテクチャの実装及び性能評価

岩崎　聖[†1,†2]　松岡　聡[†1]　曽田哲之[†3]
平野基孝[†3]　建部修見[†4]　関口智嗣[†4]

我々は Grid Data Farm(Gfarm) システム用のジョブ起動アーキテクチャの設計・実装を行っている．Gfarm システムは数千から数万ノード規模の PC クラスタで構成され，ノード間の通信・認証に GSI を用いている．このため，Gfarm システムでジョブを起動する際，ナイーブな実装を用いるとノード数に比例した GSI 認証コストが発生し，数千プロセスからなるジョブの起動に数千秒かかることが予想される．本稿で述べるアーキテクチャでは，あらかじめ確立済みのコネクションを用いることで起動要求伝達時の認証コストを回避する．実装中のシステムでジョブの起動に要する時間を計測した結果，15 ノードで 3.5 秒，63 ノードで 6 秒と想定したスケーラビリティは得られなかったが，これはジョブ起動プロトコルに問題があり，プロトコルを改善することでさらなるスケーラビリティが得られると考えている．

Implementation and Evaluation of a Scalable Job Management Architecture for Large-Scale PC Cluster on the Grid Environment

Satoru Iwasaki,[†1] Satoshi Matsuoka,[†1] Noriyuki Soda,[†3]
Motonori Hirano,[†3] Osamu Tatebe[†4] and Satoshi Sekiguchi [†4]

In this paper we describe the design and implementation of the job launch architecture for Grid Data Farm(Gfarm) system. Gfarm system is composed of PC clusters with ten thousands of nodes on the Grid. Gfarm system uses GSI for communication and authentication between nodes. Because of this, if an ingenuous method is used to start a job on the Gfarm system, the GSI authentication cost which is in proportion to the number of nodes occurs, and expects that the start of the job which consists of thousands of processes takes several thousand seconds. We avoid the authentication cost by using the connection which has been established in advance. Our system shows that the job launching time is 3.5 second with 15 nodes and 6 second with 63 nodes. We think that we can achieve more scalability by improving job-launching protocol.

1. はじめに

Grid Data Farm(Gfarm)[1)~3)] はスイス CERN[4)] の LHC ATLAS 実験によるペタバイト規模の実験データ解析環境として研究が進められている．Gfarm システムでは広域に分散した数千ノードから数万ノード規模の PC クラスタがその中核をなしている．並列ジョブの起動に要する時間は，大規模なクラスタ型計算機ではシステムの性能や使い勝手に大きな影響を与える．しかしながら，既存のクラスタ用並列システムではアプリケーションの起動時間の問題についてあまり着目されておらず，rsh などを用いた単純な機構で並列ジョブを起動するものが多かった．数十から百・二百ノード規模のクラスタを少数の信頼できるユーザが利用する環境では，ジョブの起動に要する時間も許容できる範囲であったと考えられる．しかし，Gfarm システムは世界各地の競合する研究者達が共同で利用する大規模クラスタシステムであり，ファイルアクセスやシステム内で行われる通信に厳重なセキュリティが求められる．このため Gfarm システムでは全てのシステム間通信を GSI[5),6)] を用いて行う．GSI では公開鍵暗号方式を利用したユーザ・ホスト認証を行い，複数のリソースへのシングルサインオンや SSL によるセキュアな通信が提供されるが，コネクションの確

†1 東京工業大学
Tokyo Institute of Technology
†2 E-mail: iwasaki@matsulab.is.titech.ac.jp
†3 株式会社 SRA
Software Research Associates,Inc.
†4 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology

立時に 1 秒程度と重い認証コストが生じるデメリットがある．そのため，従来の rsh を利用したモデルの様にナイーブに計算ノードに接続・認証・ジョブ起動要求を行った場合，数千プロセスからなる並列ジョブの起動に数百あるいは数千秒かかってしまうという問題がある．

本稿では，この問題を解決するための Gfarm 用並列ジョブ起動・管理アーキテクチャの設計及び実装について述べる．本システムでは各ノード上で並列ジョブ起動用のデーモンを動作させ，並列ジョブの起動命令をデーモン間にあらかじめ確立したコネクションを用いて伝達することで GSI の認証コストを削減しスケーラビリティ向上を図る．

現在実装中のシステムを東工大松岡研究室の Presto III クラスタ 64 ノードを用いて並列ジョブの起動に要する時間の計測実験を行った結果，15 ノードで約 3.5 秒，63 ノードで約 6 秒と，想定していた程のスケーラビリティは得られなかった．これは現在のジョブ起動プロトコルに問題があるためであり，今後プロトコルを改善することでさらなるスケーラビリティが得られると考えている．

2. Grid Data Farm(Gfarm)

Grid Data Farm は，CERN の LHC ATLAS 実験における観測データの解析環境の構築を目標に，産業技術総合研究所 (AIST)，高エネルギー加速器研究機構 (KEK)，東京大学素粒子物理国際研究センター (ICEPP)，東京工業大学が共同で研究を進めているペタスケールデータインテンシブ計算環境である．

Gfarm システムは Grid 技術とコモディティクラスタ技術を中心に構築される．システムのコア部分のアーキテクチャは，数百 GB から数 TB のローカルディスクを持つ PC クラスタで構成される．

Gfarm で扱うデータは，クラスタノードのローカルディスクからなる Gfarm filesystem 上にフラグメント化され分散配置される (**図 1**)．Gfarm filesystem 上のデータファイルは gfarm file と呼ばれる．gfarm file 名と，各データ断片の置かれたノードの位置やデータ断片の物理ファイル名との対応は Gfarm Metadata データベースに記録される．このデータベースには gfarm file のファイルサイズ・ファイル更新日時・アクセス制御などといったメタ情報も収められる．

Gfarm 上で動作するアプリケーションは，基本的に処理対象となるデータの断片をローカルに持つノードにスケジューリングされ，SPMD モデルで実行される．アプリケーションからは Gfarm 並列 I/O API を用いて gfarm file 名を指定してデータにアクセスする．

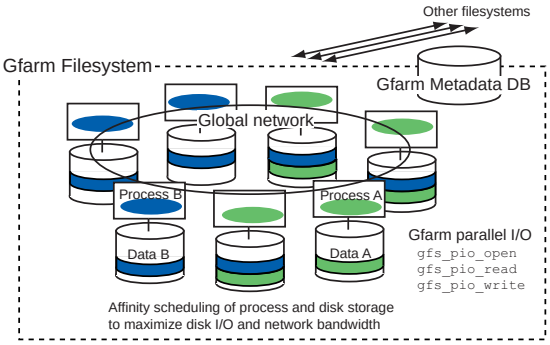

図 1 Gfarm アーキテクチャ

3. ジョブ起動アーキテクチャの設計・実装

本節では，現在実装を行っている Gfarm システム用のジョブ起動アーキテクチャの設計・実装について説明を行う．

3.1 設 計 方 針

Gfarm システム用のジョブ起動アーキテクチャを設計する上では，主に以下の点が重要な方針となる．

- **スケーラビリティ**
  Gfarm システムは数千から数万ノードのクラスタで構成される．このため，クラスタの規模が巨大になってもスケーラブルにジョブの起動が行えるアーキテクチャにする必要がある．
- **セキュリティ**
  Gfarm システムは広域上の複数のクラスタで構成され，また世界各地の様々なユーザが利用するケースを考慮している．競合する研究者達が同じクラスタを共同で利用することも考えられ，あるユーザのみがアクセス権限を所有するデータが他ユーザに不正にアクセスされるようなセキュリティホールが存在してはならない．
- **対故障性**
  Gfarm システムのような大規模クラスタ型計算機においては，一部のノードが動作不良を起こしたり，故障が発生したりすることは避けられない．システムを構成する一部のノードがダウンした場合に，システム全体が機能しなくなるような設計は避けなければならない．

3.2 設　　計

現在実装中のジョブ起動アーキテクチャの概要を**図 2** に示す．本システムではクラスタを構成する各ノード上でジョブ起動用のデーモン (gfpmd, gfarm process management daemon) を稼動させ，gfpmd 間で階層化リング構造のコネクションを確立しておく．この階層化リング構造は Gfarm システムを構成するクラスタが属する各サイト毎に構築する．ジョブを起動するには，トップレベルのリングに属するノードに起動要

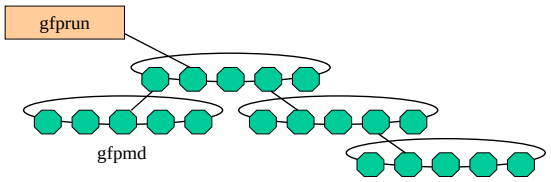

図 2 gfpmd アーキテクチャ

求を伝達する．現在は gfprun というフロントエンドプログラムをコマンドラインから実行することでリング内の gfpmd に接続・認証・起動要求を伝達する．gfpmd に伝達された起動要求は，リング状コネクションを介してクラスタ全体に伝達される．

システムを構成するコンポーネント間の通信のセキュリティを確保するため，システム内の全ての通信に GSI を利用する．これにより各コンポーネント間でコネクションを生成する時にはホスト証明書を用いた認証がノード間で行われ，通信は SSL で保護される．あらかじめ確立済みのコネクションを用いて起動要求の伝達を行うため，ジョブ起動要求をクラスタ内で伝達する際には認証コストは発生しない．ジョブの起動要求時に発生する認証コストは，各サイトへの gfprun - gfpmd 間のコネクション生成のコストだけで済む．

gfpmd 間で構築するコネクションをリング状構造にした理由は主に対故障性向上のためである．リング状トポロジでは，あるノードに障害が発生した場合にそのノードの左右のノード間でコネクションを張り直すことでリング構造を保つ．Gfarm システムは数千から数万ノードと大規模であり，スケーラビリティを確保するためには通信トポロジを階層化する必要がある．通信トポロジを階層化した場合，障害が発生したノードが含まれる階層以下の全てのノードに影響を与えるため，通信トポロジの再構成コストは最小限にとどめなければならない．リング以外のトポロジ，例えばスター型の構造では，中央のノードで障害が発生した場合，スターに含まれるノード数に比例した数のコネクションを生成し直す必要があり，通信に GSI を用いる gfpmd では不利である．

階層化リング構造では，1 階層のリングのサイズを 50 ノードとした場合，3 階層のリング構造で $50 \times 50 \times 50$ のおよそ 12 万台規模のクラスタをサポート可能である．リングの端から端までの経由ノード数は高々 150 hops と，低遅延でメッセージ伝達が可能であると考えている．

階層化リング構造の欠点としては，リング内で伝達されるメッセージがループし続けないようにするなどの配慮が必要となりややプロトコル処理が複雑になる．gfpmd では，メッセージヘッダ内に各階層におけるメッセージ送信者の情報を収める．各ノードでは隣のノードからメッセージを受信した際にこのリング内送信者を調べ，自分が送り出したものであった場合にはメッセージの種類に応じた適切な処理 (例えば単純に破棄する，など) を行う．

並列ジョブを構成する各プロセスの標準出力は gfprun コマンドを起動したシェルの標準出力に転送する．また，gfprun へ与えられた標準入力やシグナルは各プロセスに伝達される．しかしこれらの伝達をリング状コネクションを介して転送するのでは効率が悪く，リング内を流れる他のユーザの起動要求メッセージなどの伝達効率を下げることにもなる．このため，各並列ジョブ毎に専用の I/O tree を構築して標準入出力やシグナルの伝達を行う．この tree の構築を行う際に発生する認証コストは，tree の構築を各ノードで並列に行うことでノード数が増加してもある程度以下に抑えることができると考える．

3.3 gfpmd によるアプリケーション起動

gfpmd を用いて並列ジョブを起動する場合，現在は gfprun というフロントエンドコマンドを利用する．ユーザ認証には GSI を用いるため， gfprun コマンドを用いる前に grid-proxy-init コマンドであらかじめ proxy certificate を生成しておく．gfprun コマンドは以下のような書式で起動する．

```
% gfprun [-H hostlist] command [args...]
```

以下では gfpmd においてどのように並列アプリケーションの起動が行われるかについて詳細に説明を述べる．

(1) **ジョブを起動するノードの決定**

gfprun コマンドに，プロセスを起動するノードのリストを記述したファイルをオプションで指定する．ノードリストが指定されなかった場合は，コマンドライン中からアプリケーションの処理対象となる gfarm file 名を調べ，スケジューラにノードの割当をまかせる．スケジューラからジョブを割り当てるノードのリストが返る．また，Gfarm システム上でのジョブ ID の割り当てをジョブマネージャから受ける．

(2) **各サイトへの接続・認証**

ノードリストに含まれるサイト毎に gfpmd の最上位リングのノードに接続・認証を行う ☆(**図 3**)．接続先の最上位リングの gfpmd をそのジョブセッションにおけるセッションマスタと呼ぶ．

(3) **各ノードへジョブグループ割当要求**

gfprun は全サイトへの接続・認証が完了すると，それぞれのサイト毎にジョブを割り当てるノードのリスト及びジョブ ID をセッションマスタに通知する．セッションマスタは，リスト中の全ノードに向けてジョブグループの割り当てメッセージを送信する．このメッセージには割り当てるジョブグループのジョブ ID が含まれている (**図 4**，この図の例では 2,5,7,10 番ノー

☆ 現在はサイト毎に最上位リングを校正するノードのリストを用意しておき，そのリストを参照することで接続・認証するノードを決定する．

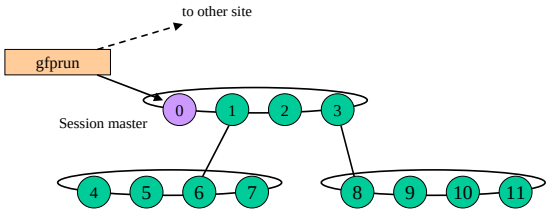

図 3 各サイトへの接続・認証

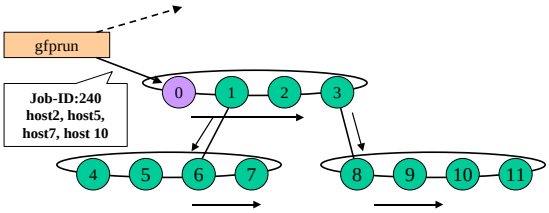

図 4 各ノードへジョブグループ割当要求

ドにジョブグループを割り当てている).

(4) **I/O tree 用ポートの bind および ack 収集**

ジョブグループ割り当てメッセージを受け取ったノードは，標準入出力およびシグナル伝達用 I/O tree を構成するためのソケットを動的に bind する．bind されたポート番号を，ジョブグループ割り当て要求に対する ack メッセージと共にセッションマスタに送り返す (**図 5**)．セッションマスタは全ノードからの ack を集め，どのノードがどのノード・ポートに接続することで I/O tree を構成するかを決定する．

(5) **I/O tree の構築**

セッションマスタが各ノードに対してどのノード・ポートに接続を行うべきか通知する．各ノードでは I/O tree 構築の指示メッセージを受け取ると，I/O tree 構成用のプロセス (プロセスマネージャ) を fork する．fork されたプロセスは指定されたノードに接続・認証を行う．この処理が終わると，最終的にセッションマスタを始点とした I/O tree の構築が完了する (**図 6**).

(6) **起動するコマンドの伝達，入出力管理**

gfprun コマンドは起動するコマンド，環境などをセッションマスタを介し I/O tree を通じてジョブグループ全体に伝達する．コマンドを受け取ったプロセスマネージャは，指定されたコマンドを fork および exec し，ジョブの実行を開始する．fork したコマンドの標準出力はプロセスマネージャがトラップし，I/O tree を通じて gfprun コマンドを起動したシェルに伝達される．また，gfprun に与えられた標準入力やシグナルも I/O tree を通して各プロセスマネージャに伝達され，並列ジョブのプロセスに送られる (**図 7**).

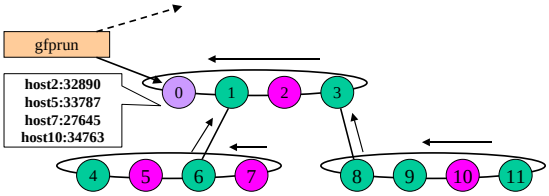

図 5 I/O tree 用ポートの bind および ack 収集

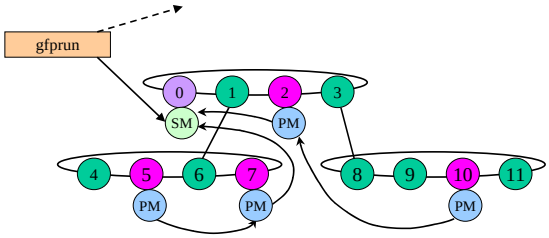

図 6 I/O tree の構築

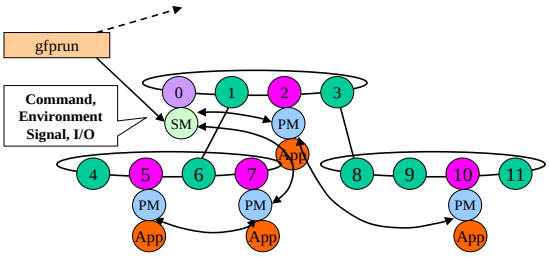

図 7 起動するコマンドの伝達，入出力管理

表 1 Presto III クラスタノード性能

CPU	Athlon MP 1.2GHz Dual
Motherboard	Tyan TigerMP(AMD 760MP chip set)
Memory	768MB DDR SDRAM (PC2100)
OS	Red Hat Linux Kernel 2.4.7
NIC	DEC 21140AF/Intel Ether Express Pro100

4. ジョブ起動性能評価・考察

4.1 ジョブ起動実験

前節までに述べたアーキテクチャにより，効率良く並列ジョブ起動が行えるかどうか調べるために実験を行った．現在実装中の gfpmd を用いてクラスタにジョブを投入し，ジョブの実行が完了するまでに要する時間を計測した．実験は東京工業大学松岡研究室の PrestoIII クラスタ上で行った．PrestoIII クラスタのノード性能を表 **表 1** に示す．

まず PrestoIII クラスタ 72 ノード上で gfpmd を起動し，単一のリング状コネクションを形成させた．この状態で，別のクラスタ管理ノードから gfprun コマンドを用いてジョブ投入を行う．今回の実験ではジョブの起動に要する時間を計測するため，実行するジョブとして単純な hostname コマンドを選択した．

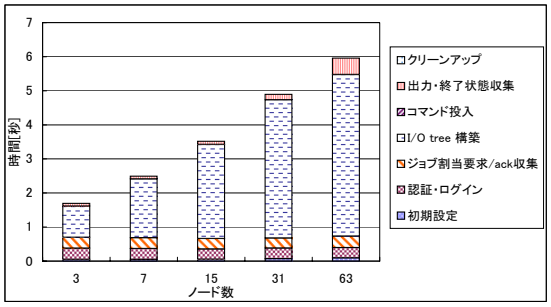

図 8 gfpmd ジョブ起動性能

```
% gfprun -H hostlist hostname
```

上記のようなコマンドを用い，ジョブを起動するホスト数を 3 ノードから 63 ノードまで変化させながら実験を行った．gfprun コマンドの各部分の実行に要した時間を計測した結果を **図 8** に示す．

この実験の結果，3 ノード時に約 1.5 秒，15 ノード時に約 3.5 秒，63 ノード時に約 6 秒という結果となり，ノード数が倍になるにつれジョブの起動に要する時間が増加してしまっている．ジョブ起動から実行完了までに要した時間の大部分を標準入出力・シグナル伝達用の I/O tree を構築する処理に要している．I/O tree の構築に要する時間が，ノード数の増加に伴って増加する結果となった．

4.2 考　　察

前節の実験結果では，ジョブ起動時間の大部分を I/O tree 構築時の GSI 認証に要する時間が占めており，ノード数の増加に伴って I/O tree 構築に要する時間も増加している．その他の部分に要する時間は，60 ノード程度までではほとんど差が見られなかった．この結果は，I/O tree の構築が期待していたようにはうまく並列に構築されていないことを表している．ノード数が 2 倍になるにつれ，I/O tree 構築のコストはおよそ 1 秒ずつ増加している．より大規模なノード数でもこのペースで増加すると仮定すると，I/O tree の構築には 256 ノードで約 7 秒，1024 ノードで約 9 秒，8192 ノードで約 12 秒要することになる．このぐらいの規模では I/O tree 構築以外の部分に要する時間も増加すると考えられ，現在の実装ではスケーラビリティの面でやや問題が残ってしまう．

今回の実験において I/O tree を効率良く構成できなかった原因は，セッションマスタが各ノードに I/O tree 構築を指示するメッセージを送り出す順番・タイミングや，プロセスマスタの I/O tree 構築部の実装に問題があるためだと考えられる．現在の実装では，セッションマスタが送り出す tree 構築メッセージは tree の始点に近いノードから順に一時に送り出される．メッセージを受け取ったノードではプロセスマネージャを fork する．fork されたプロセスマネージャは，非同期ソケットで指定されたノード・ポートに接続を試み，tree の下側のノードからの接続待ち受け用ソケットと共に select で監視しコネクションの確立完了を待つ．コネクションが確立した後，相手ノードとの認証を行う．マルチスレッドなどは用いていないため，特定のノードと認証を行っている間は，別のノードとの認証処理を同時に行うことはできない．tree の高さに比例して I/O tree の構築コストが増加したことから，tree の上側で先に認証処理が始まり下側のノードが上側のノードの認証処理が終わるまで待たされる，といった逐次的なパスが出来上がってしまっているのではないかと考えられる．

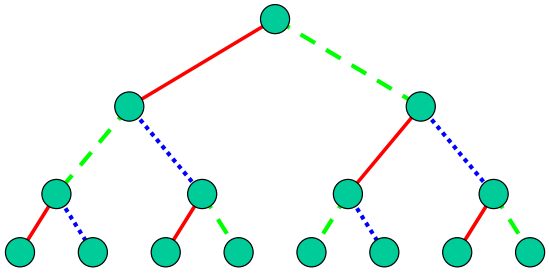

図 9 I/O tree 彩色

I/O tree を構築する際，各ノードで行われる接続・認証回数は高々 3 回である．tree を効率良く構築した場合には，ノード数に関わらず I/O tree 構築時のコストは GSI 認証 3 回分のコストにほぼ等しく抑えることが出来ると考えている．例えば，**図 9** で示した 3 種類のコネクションは，それぞれ並列に認証が可能である．tree の各コネクションの構築を 3 ステップに分けて並列に行えるように，tree 構築メッセージの順序やタイミング調節，あるいはプロトコルを変更することで，より効率良く I/O tree の構築が行えるものと考えている．

5. 関 連 研 究

MPI ライブラリの代表的実装の一つである MPICH[7),8)] では，MPD(multipurpose daemon)[9)~11)] と呼ばれる並列プログラムを高速に起動するためのシステムが提供されている．MPD では数百から千プロセス規模の並列プログラムの高速起動やシグナルの高速伝達，標準入出力・エラー出力の管理などの機能が実現されている．標準入力はノード全体，あるいは任意の特定のノードにのみ送ることが可能になっており，gdb を利用した並列デバッガなどのユーティリティも実現している．

文献[9)] では Chiba City[12)] クラスタ (256node dual PIII500MHz, FastEthernet 使用) 上で，数百ノードからなる並列プログラムの起動にかかる時間を評価する実験を行っている．本稿の実験と同様に，hostname コマンドを並列に起動し，実行が終了するまでの時間

を計測している．その結果，211 ノード上でコマンドを実行させた場合に約 2 秒，211 ノード上で各 2 プロセス実行させた場合に約 3.5 秒であったとしている．211 ノードの MPD daemon にメッセージを伝え終えるのには 0.13 秒しかかからなかったとしている．

MPD では標準の TCP/IP による通信を行っており，コネクション生成時には共有鍵方式の認証を行っている．GSI ほどの認証コストは発生しないため，非常に高速に並列ジョブの起動が可能になっている．

6. まとめ・今後の課題

本稿では，gfarm システム用の並列アプリケーションの起動を高速に行うためのアーキテクチャとして，Gfarm process management daemon, gfpmd の設計および実装について述べ，実装中のシステムを用いて並列ジョブの起動に要する時間の計測実験を行った．

3 ノードから 63 ノードまでノード数を変えながら起動時間を計測した結果，数十ノード規模でも GSI 認証による I/O tree 構築のコストは大きく，63 ノード時のジョブの起動には 6 秒程度の時間を要するという結果を得た．この結果は，既存のジョブ投入システムと比較した場合，高速であるとはいいがたい．しかしながら，この I/O tree 構築コストは構築の仕方を工夫することでノード数に関わらずある程度以下に抑えることが可能であると考えられる．

今後の課題として，現状の実装で I/O tree 構築に要するコストがノード数に比例してしまう原因を詳細に解明し，効率的に I/O tree の構築が行えるようにプロトコルや実装を変更を行う．また，より大規模なノード数での起動実験や，ノードが高負荷の場合の gfpmd 間メッセージの転送効率やジョブ起動効率などの評価を行う予定である．

参 考 文 献

1) Grid Datafarm. http://datafarm.apgrid.org/ .
2) O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, H. Sato, Y. Tanaka, S. Sekiguchi, Y. Watase, M. Imori, and T. Kobayashi. Grid Data Farm for petascale data intensive computing. *Technical Report ETL-TR2001-4*, Electrotechnical Laboratory, 2001.
3) 建部 修見, 森田 洋平, 松岡 聡, 関口 智嗣, 曽田 哲之. 広域大規模データ解析のための Grid Datafarm アーキテクチャ. 情報処理学会研究報告 2001-HPC-87, pp. 177-182, 2001.
4) http://www.cern.ch/
5) I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pp. 83-92, 1998.
6) R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. A national-scale authentication infrastructure. *IEEE Computer*, Vol.33(12), pp. 60-66, 2000.
7) W. Gropp and E. Lusk. Sowing MPICH: A case study in the dissemination of a portable environment for parallel scientific computing. *Journal of supercomputer Applications and High-Performance Computing*, Vol.11(2), pp. 103-114, 1997.
8) W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message-passing interface standard. *Parallel Computing*, Vol.22(6), pp. 789-828, September 1996.
9) R. Butler, W. Gropp, and E. Lusk. A scalable process-management environment for parallel programs. *Technical Report MCS-P812-0400*, ANL, April 2000.
10) Rusty Lusk, D. Ashton, A. Chan, B. Gropp, D. Swider, R. Ross, and R. Thakur. Scalable process management and interfaces for clusters. *Workshop on Clusters and Computational Grids for Scientific Computing*, Lyon, September 24-27, 2000.
11) R. Butler, W. Gropp, and E. Lusk. Components and interfaces of a process management system for parallel programs. *Parallel Computing*, Vol.27(11), pp. 1417-1429, October 2001.
12) Chiba City web site. http://www.mcs.anl.gov/chiba/ .