Lucie: 大規模クラスタに適した高速セットアップ・管理ツール

高宮安仁[†] 真鍋篤[†††] 松岡聡[†,††]

コモディティクラスタリングシステムにおける，ノード数規模の急激なスケールアップに伴い，クラスタのセットアップおよび保守コストは増大しつつある。また、各ノードのローカルディスク上に数 GB 単位のデータベースファイルを必要とする、いわゆるデータインテンシブアプリケーションがクラスタ上で用いられつつあるが、インストール時における、大規模データのセットアップに着目したツールは重視されてこなかった。そこでわれわれは、大規模クラスタ向けセットアップ/ 管理ツールである Lucie, およびデータ配布ツール Dolly+ を開発している。Lucie では、インストール用メディアを用いないネットワークブート/ インストール機構、および用途に応じたインストーラ自体の再構成といった拡張機構を実現する。また、 Dolly+ では、仮想リング構造による高速データ転送によって、耐故障性を保ちつつ、インストール時に高速に数 GB 単位のデータを全ノードへ配布することができる。本稿では、Lucie のインストール性能、およびインストール時の Dolly+ と Lucie による大規模データ配布性能について評価をおこなった。結果、性能はノード数によらずほぼ一定であり、本研究の将来的な目標であるプラグアンドプレイクラスタリングの基礎技術として有用であることがわかった.

Lucie: A fast installation and administration tool for large-scaled clusters

Yasuhito Takamiya,[†] Atsushi Manabe [†††]
and Satoshi Matsuoka[†,††]

Rapid increase in the number of nodes for commodity clustering is mandating the handling the potential cost of setup and maintenance clusters as the norm. Moreover, with arising of data intensive applications which requires several GBs of data on each cluster nodes, it is revealed that there were no installation tool aimed at installation-time setup of such large-scaled data. In this paper, we propose a new cluster installation/administration tool called Lucie which allows network boot/installation mechanism with no specific installation media and configurability which allows reconstruction of installer itself on demand. Additionally, we propose a new data distribution mechanism called Dolly+ which deploys fault tolerant, high-speed virtual ring topology data transferring. With Dolly+, one could distribute several GBs of images to all cluster nodes in installation-time maintaining fault tolerance. Our several benchmarks show that Lucie and Dolly+ can install and setup the whole cluster in constant time. This result shows that Lucie and Dolly+ are scalable and efficient, and could well serve as a basis for 'Plug-and-Play' clustering.

1. はじめに

近年、コモディティクラスタリングシステムにおける、ネットワークハードウェアやユーザレベル通信技術の進歩によって、OS 上、およびネットワーク上でのレイテンシの克服、および広バンド幅の獲得が達成されている。このため、数千〜数万ノード超の大規模クラスタが実現・計画されている。しかし、実際には以下に挙げるような運用面における問題のため、実現は困難である。

新規にクラスタを運用開始する場合、すべてのノードへ OS やソフトウェアをインストールし、設定する必要がある。このための設定項目は数十〜数百項目におよぶため、すべての設定ファイルを手作業で作成するのは困難である。また、単一ノード用インストーラを用いて逐次的にインストールした場合、ノード数が増大するにつれ、作業時間は線型増加してしまう。加えて、さまざまなクラスタ保守作業

- ノード故障時の新品ノードとの交換/復旧
- ノード間で設定ファイルに不整合が起こった場合の、問題個所の特定/訂正作業
- ソフトウェアへのパッチ適用やバージョンアップなど、定期的な更新作業

はいずれも、各ノードへのリモートログインによる逐次的な手作業では手間がかかり、作業上の誤りを犯しやすい。

われわれは、大規模クラスタ用インストーラ・管理

† 東京工業大学 Tokyo Institute of Technology
†† 科学技術振興事業団 JST
††† 高エネルギー加速器研究機構 KEK

ツールとして、Lucie[7),22)] を開発している。Lucie は、

- クラスタ全体の完全自動ネットワークインストール
- インストール時の、耐故障大規模データ高速配布
- 設定パッケージの配布
- 復旧エージェント機能

といった自動管理機構を提供している。ユーザは、従来の手作業による復旧作業をおこなうかわりに、Lucie によるクラスタ全体の自動再インストールをおこなうことによって、数分で復旧できる。また、設定パッケージの利用によって、新規クラスタの導入作業、ソフトウェア更新作業を効率化できる。

本稿では、Lucie インストーラをもちいて、112 ノード構成クラスタのセットアップ時間を計測した。結果、セットアップ時間が 8 分弱とその有効性を確認した。また、耐故障大規模データ配布機構をもちいて、1GB のノード全体への配布性能を計測した。結果、ノード数増加に対して所要時間は一定であることを確認した。また、Lucie を実際に東京工業大学松岡研究室の Presto クラスタ群[2)] に適用し、合計約 500 ノードのクラスタ運用に有用であることを確認した。

2. 要　　請

新たに購入した Vanilla PC をクラスタノードとしてセットアップする場合、もしくは故障したクラスタノードを復旧する場合を考える。短時間でクラスタノードとして完全に動作させるためには、

(1) ハードディスク上へのパーティション、ファイルシステムの **自動的な** 作成
(2) クラスタでもちいるソフトウェアや OS の、各ノードへの**自動**インストール・設定
(3) ソフトウェアが使用するデータセットの、各ノードへのセットアップ

が必要である。

2.1 インストールの自動化

クラスタ用のインストーラ/管理ツールでは、単一ノード用インストーラにおける逐次的な部分を極力排除する必要がある。たとえば、クラスタのセットアップでは、同時に多数ノードのセットアップを完了させる必要があるので、1)〜2) で必要となるキーボード入力操作は、完全に不要でなければならない。

加えて、インストーラは起動用メディアを必要としないことが重要である。これは、セットアップするノード数が増えるにつれ、準備しなければならないメディアの数も増えるためである。

2.2 インストール時のデータ配布

3) でセットアップすべきデータセットのファイルサイズは、特にデータインテンシブアプリケーションでは巨大化の傾向がある。たとえば、ホモロジー検索ソフト blast[9)] のデータベースファイルは、数ギガ〜数十ギガバイトに及ぶ。このようなファイルのコピー方法として、rsync や tar による、ホスト毎の逐次的なコピー方法を用いた場合、ノード数が増加するにつれ、セットアップ時間も線型増加してしまう。

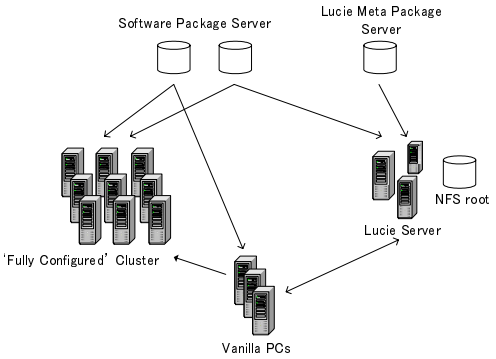

図 1　Lucie アーキテクチャ

そこで、データコピーの高速化のためには、IP multicast など、ボトルネックの存在しにくい、ブロードキャスト的なコピー機構をもちいる必要がある。また、配送にもちいるネットワークデバイスとして、Gigabit-Ethernet や Myrinet などの高速なデバイスをもちいる必要がある。また、データセットのコピー操作自体、ユーザが個別におこなうのではなく、1)〜3) の操作と同時に、インストール時に自動的におこなうことが重要である。

2.3 信　頼　性

データコピーの信頼性も重要である。コピー対象のノード数の増加につれ、データコピー中ノードの故障確率も上昇する。たとえば、データ配布トポロジとして、カスケードトポロジを選択した場合、ノード故障の影響はカスケードに沿って子孫ノードすべてへ及んでしまい、復旧が難しい。このため、耐故障性のある配布トポロジの選択、および故障時の復旧機構が必要である。

3. Lucie

上記の要請を満たすためのツールとして、われわれは Lucie を開発している。

Lucie は、インストールを制御する Lucie サーバ、Lucie サーバへ Lucie メタパッケージを提供するメタパッケージサーバ、クラスタノードや Lucie サーバーへソフトウェアパッケージを提供するパッケージサーバ、Lucie クライアント (インストールされるクラスタノード) から構成される (図 1)。

Lucie サーバ　は、Lucie クライアントへネットワーク情報を送信する BOOTP/DHCP サーバ、NFS 経由でのルートファイルシステムを提供する NFS サーバ、ブートイメージである pxeboot, Linux カーネルを提供する TFTP サーバから構成される。それぞれのサービスはそれぞれ別ホスト、もしくは単一ホスト上で動作する。Lucie によるインストールでは、インストール処理の各段階において、Lucie サーバ上の各種設定が Lucie クラ

イアントへ順次送信されることによっておこなわれる。

メタパッケージサーバ は、クラスタの設定を論理的に分割したパッケージ (メタパッケージ) をネットワーク経由で Lucie サーバへ配布する。

パッケージサーバ は、クラスタでもちいるソフトウェアバイナリパッケージをクラスタノード、Lucie サーバへネットワーク経由で配布する。配布にもちいるプロトコルとして、ftp, http, NFS, CD-ROM, rsh, ssh が選択可能である。

Lucie によるインストールは、次の 2 段階に分けられる。第 1 段階では、NFS 上のディレクトリをファイルシステムとしてマウントし、Linux を起動するディスクレスブートをおこなう。第 2 段階では、ディスクレスシステム上から Lucie インストーラ本体を起動し、ローカルディスクへのインストール作業がおこなわれる。

3.1 Lucie の起動

Lucie では、インストーラの実行環境として、CD-ROM 等のメディア上にあらかじめ書き込まれた Linux のディスクイメージではなく、NFS サーバ上の chroot 環境を NFS root としてもちいる。

KickStart[12] 等の一般的インストーラでは、インストーラ実行環境として、メディア上にあらかじめ圧縮された Linux ディスクイメージをもちいる。この場合、Myrinet のように標準でないドライバを必要とするハードウェアや、ユーザ独自の設定ツールなどをインストーラから利用することは難しい。これは、実行環境のカスタマイズには、インストールメディアイメージを新たに作り直し、ノード台数分コピーする必要があるためである。

一方、実行環境として NFS root を用いた場合、インストーラの実行環境は NFS サーバ上にふつうのファイルシステムとして存在する。ユーザは、追加ファイルを NFS root へインストールすることによって、インストール中の任意ジョブの起動や、カーネルへの動的なモジュール追加といったカスタマイズができる。

Lucie の起動は、以下の手順で実行される (図 2)。

(1) PXE/MBA 機構[1),5)] 対応 NIC の BIOS が起動し、PXE ブート ROM を読み込んだ後、DHCP 要求によってネットワーク情報を取得する。この後、TFTP で pxeboot (preboot execution environment boot) を取得し、起動する。

(2) pxeboot から TFTP で Linux カーネルイメージを取得し、これを起動する。

(3) カーネルは、DHCP/BOOTP 要求によりネットワーク情報、NFS root の IP アドレス、およびパスを取得する。この後、NFS root を root ファイルシステムとしてマウントし、Linux を起動する。

Lucie クライアントはルートファイルシステム

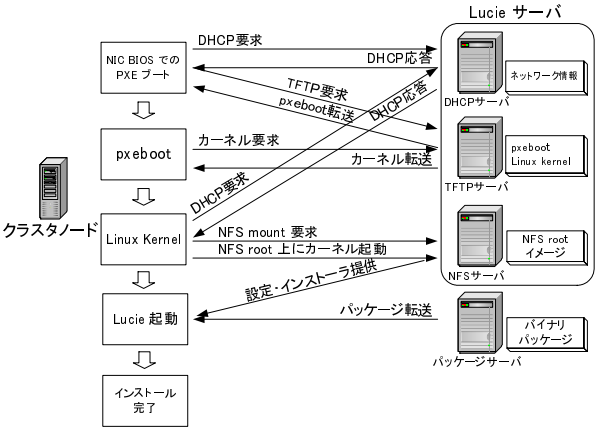

図 2 ネットワークブートによる Lucie の起動

```
define host{ # ホスト定義テンプレート
        lucie_group     lucie1
        services        mpi, c-develop, c++-develop
        name            hosttemplate
        register        0
}
define host{ # ホスト cnode00 の定義
        host_name       cnode00
        address         192.168.10.1
        mac_address     00:50:56:40:40:b6
        use             hosttemplate
}
define host{ # ホスト cnode01 の定義
        host_name       cnode01
        ...
}
define lucie_group{ # Lucie サーバグループの設定
        domain_name     is.titech.ac.jp
        gateway         192.168.10.254
        subnet          255.255.255.0
        dns1            192.168.0.2
        name            lucie1
        lucie_server    lucie1.is.titech.ac.jp
        dhcp_server     dhcp1
        http_server     ghost1
}
define host_group{ # クラスタの設定
        hostgroup_name prestoi-cluster
        members         cnode00, cnode01, ...
        lucie_group     lucie1
}
```

図 3 クラスタ構成の定義ファイル (一部)

をマウント後、/etc/init.d/rcS を起動する。この /etc/init.d/rcS は Lucie 独自のものと置き換わっており、Lucie インストーラを開始する。

3.1.1 Lucie サーバの構築

Lucie では、Lucie サーバ群を構築するためのコマンドとして lucie-setup を提供している。lucie-setup コマンドは、ユーザ設定をもとにして、NFS , TFTP, および DHCP/BOOTP サーバの設定を自動的におこない、Lucie インストーラと NFS root を生成する。

図 3 はクラスタ名、クラスタを構成するノード、NIC の MAC アドレスやネットワーク情報の設定である。NFS の設定では、これと chroot 環境の Linux のバージョン、ディストリビューション名、root パスワード、インストールするパッケージリスト等から、NFS 設定ファイルと NFS root を生成する。

3.2 Lucie インストーラ

Lucie インストーラは個々のインストール処理をおこなうモジュールへモジュール化されており、ユーザによって指定された順に実行する (図 4)。 主なモジュー

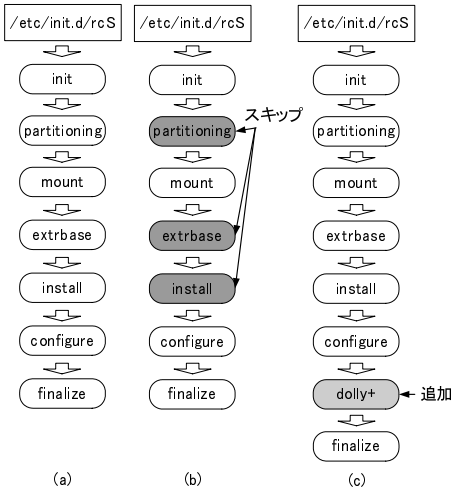

図 4 Lucie インストーラの構成

ルの機能を以下に示す。

init Lucie インストーラの環境変数や tty 等の初期化をおこなう

partition ローカルディスクのパーティショニング、フォーマットをおこなう

mount ローカルディスクを NFS root の ramdisk 上へマウントする

extrbase Linux の最小基本システムを含む tar.gz をローカルディスク上へ展開する。

install ローカルディスク へ chroot し、指定されたソフトウェアパッケージをインストールする。

configure 各ノードに応じた設定ファイルの書換えをおこなう。

finalize インストーラの終了処理をおこなう。

図 4(a) は通常インストール、すなわちすべてのインストール処理をおこなう場合のモジュール設定である。図 4(b) では、フルインストールに比べて実行時間の短い、修復インストールをおこなうインストーラが定義されている。ここでは、パーティショニング、Linux 基本システムのインストール、およびソフトウェアパッケージインストールモジュールの実行を省略している。

3.2.1 インストーラモジュールの追加

インストーラの拡張機構として、ユーザは、用途に応じた独自のモジュールを定義し、実行することができる。Lucie では、ユーザ定義モジュール作成支援のための、Ruby 言語で記述された helper class やスクリプト群を提供している。例として、インストール中に Lucie クライアントへリモートログインし、インストールモジュールのデバッグをおこなうための、remotelogin モジュールを挙げる。

remotelogin モジュールでは、インストール時の sshd/rshd 起動処理を記述する。ユーザは Lucie サーバ上の NFS root へ ssh/rsh サーバを追加インストールすることにより、インストール中にノード上で sshd/rshd を立ち上げ、リモートログインできる。

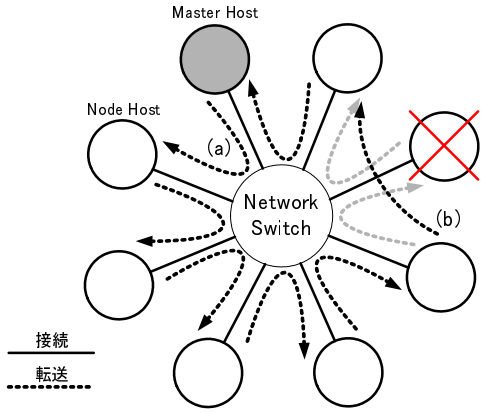

図 5 Dolly+ での論理ホストリング機構

3.3 大規模データのノード間配布

我々は、大規模データのノード間耐故障高速配布ツールである、Dolly+[4] を開発している。Lucie では、インストール時の大規模データのノード間を、Dolly+ をもちいて実現している。

Dolly+ は、複数台のホスト間で、ネットワークを介してファイルやディスクイメージをコピーする機能を持つ。ネットワーク経由データコピーソフトウエアの一種である Dolly[3] をベースにしており、Dolly のデータコピー機能に加えて、以下に述べるデータコピーの高速化、耐故障性機構の拡張機能を持つ。

3.3.1 データコピーの高速化

ネットワーク経由データコピーの実現方式として、サーバ・クライアント方式による実現を考える。対象のホスト数が 100 ～ 1000 台以上と多く、また転送するファイルサイズが数 GB と非常に大きい場合、サーバボトルネックが発生し、著しく性能が低下する。

Dolly+ では、こうしたボトルネックの発生を避けるために、コピー対象ホストをマスターホスト (コピーされるイメージを持つホスト) を先頭としたリング状に接続し、直列転送をおこなう (図 5(a))。このようにして、マスターホストでのボトルネック発生を回避し、全二重ネットワークスイッチの性能を最大限に引き出す。

そのほかの高速化機構として、ホスト内での $network \rightarrow memory, memory \rightarrow disk, memory \rightarrow netowrk$ 間のデータコピーでは、マルチスレッドによるパイプライン転送をおこなう。図 6 中の円 Server, Node 1, Node 2 はリング通信を構成する各ノードを示し、各ノード上の 3 つの四角は $network \rightarrow memory, memory \rightarrow disk, memory \rightarrow netowrk$ のコピーをおこなう各スレッドをあらわす。マスタノード上で $disk \rightarrow memory$ のコピーをおこなうスレッド (図 6: Server 上のスレッド 8) は、転送するファ

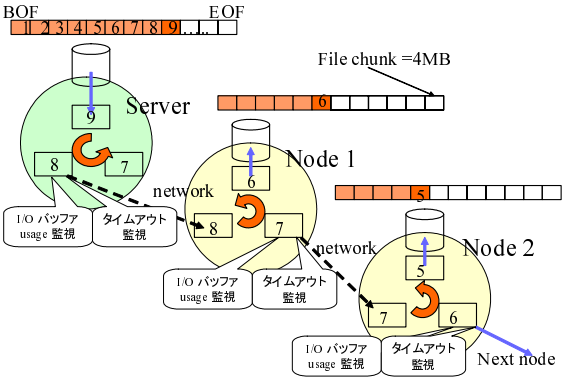

図 6 Dolly+ でのマルチスレッド処理と耐故障性機構

イルを chunksize=4MB 単位に区切り、パイプライン転送する。

3.3.2 耐故障性機構

ホストを直列接続する場合、接続されたホストのいずれか一台に異常が生じた場合の転送停止を回避する必要がある。Dolly+は耐故障性機構として、I/O バッファ監視と転送タイムアウトの監視をおこなう (図 6)。

I/O バッファ監視として、上流ノードから下流ノードへの chunk 転送スレッド (図 6: Server のスレッド 8, Node1 のスレッド 7, Node2 のスレッド 6) は、I/O バッファの使用割合を監視する。もし、I/O バッファが一定割合以上使用されていない場合、下流ノードが何らかの障害を発生したと判断し、より下流の健全なノードへ転送相手を変更する。

同様に、同スレッドは chunk の転送タイムアウトを監視する。もし一定時間以上転送が完了しない場合、下流ノードが何らかの障害を発生したと判断し、より下流の健全なノードへ転送相手を変更する。

異常を検出した場合、異常ホスト (図 5: ×印ノード) を自動的に論理ホストリングから排除し、論理ホストリングを再構成し、転送を続行する (図 5(b))。

3.3.3 Lucie との連携動作

Dolly+ を Lucie と連携動作させることにより、インストール時、Dolly+ をもちいて大規模データを各ノードへ配布することができる。これは、インストール時に各ノード上で Dolly+ クライアントを起動する Dolly+ モジュールを作成し、これをインストーラ定義へ追加することにより実現している (図 4(c))。

3.4 設定パッケージ

Lucie では、関連する設定ファイル同士をひとつのバイナリパッケージとしてパッケージングし、作成・配布する機能を提供している。これを Lucie メタパッケージ (LMP) と呼ぶ。ユーザは、「MPI 実行環境の設定」「リモートシェルの設定」などの LMP を Lucie サーバへインストールすることにより、Lucie サーバの設定を簡便におこなうことができる。

```
#!/bin/sh -e

# Source debconf library.
. /usr/share/debconf/confmodule
db_version 2.0

db_title 'Configuring Ganglia LMP'

# Latitude and Longitude
db_input medium lmp-ganglia/latlong || true
db_go

# URL
db_input medium lmp-ganglia/url || true
db_go
```

図 7 フロントエンド定義ファイル (一部)

3.4.1 Lucie Meta Package の構成

Lucie メタパッケージは、以下の要素から構成される。LMP をインストールすると、以下のすべての要素が、インストーラへ反映される。

パッケージリスト 各ノードへインストールされるパッケージのリスト、およびパッケージサーバの URI 情報。

設定ファイルテンプレート インストール時にディレクトリ構造を保ったまま各クラスタノードへコピーされる、/etc ファイルのテンプレート。

設定スクリプト インストール時に実行される、/etc 設定ファイル編集用のスクリプト。

カスタマイズ用フロントエンド ファイルテンプレート、設定スクリプトをカスタマイズするための、フロントエンド定義ファイル (図 7)。

エージェントスクリプト LMP によって設定された設定ファイル、および各デーモンの動作状況について、sanity check をおこなう cron スクリプト。

パッケージ依存情報 LMP 間の依存関係、競合関係などの依存情報。

3.4.2 Lucie Meta Package の作成・配布

LMP 開発者は、設定ファイルから生成した LMP バイナリをパッケージサーバへ配置し、ソフトウェアパッケージと同様にネットワーク配布する。LMP 生成のためには、入力として上記の構成要素が必要である。

LMP 構成要素のうち、パッケージ依存情報は、それぞれの LMP が提供するパッケージリスト、設定ファイルテンプレート、および設定スクリプト間の依存関係から自動生成される。Lucie は、依存関係を追跡し、rpm 等のバイナリフォーマットへ変換するツール lmp を提供している。

パッケージ依存情報以外の設定項目例として、フロントエンド定義ファイルを挙げる。フロントエンド定義ファイルは LMP インストール時にカスタマイズ用 GUI を表示し、LMP のカスタマイズ機能を提供する (図 8)。フロントエンド定義ファイルでは、フロントエンド各入力画面でのメッセージと、各入力画面の状態遷移をシェルスクリプトで定義する (図 7)。Lucie は、フロントエンド用ツールキットとして、汎用設定マネージメントツール debconf[17] を用いている。

3.4.3 エージェントスクリプト

LMP 開発者は、オプションとして、LMP 自体に

図 8 LMP フロントエンド GUI

sanity check 用の cron スクリプトを含むことができる。これを エージェントスクリプトと呼ぶ。

エージェントスクリプトは、LMP でインストールされた各デーモン、および設定ファイルの状況を定期的にチェックする。不整合を検出した場合、スクリプトによる復旧や、障害状況の XML 形式によるマルチキャスト送信をおこなう。Lucie サーバ上のモニタリングデーモンはこの XML を受け取り、管理者への通知、スクリプトによる復旧手順を実行する。

Lucie はエージェントスクリプト開発用に、プロセステーブル内容の監視処理、およびモニタリングデータマルチキャスト処理記述用の Ruby 言語による class library を提供している。

3.4.4 Lucie Meta Package の利用

Lucie Meta Package は通常のバイナリパッケージとして提供されているため、rpm コマンドなど、各種パッケージマネージャを用いたインストール・アンインストール等の操作ができる。

パッケージマネージャは、インストールしようとしている LMP 同士の競合を検出した場合、警告を発生しインストールを中止する。apt など、依存関係の自動解決機能を持つパッケージマネージャでは、インストールしたい LMP、および依存する LMP を追跡し、両者を自動的にネットワークインストールする。

4. 実　　験

図 9 に東京工業大学松岡研究室での Lucie を用いたクラスタ運用形態を示す。図中の灰色ホスト上では Lucie が動作しており、それぞれ PrestoII PrestoIII ion クラスタ、PrestoI クラスタ、OBI testbed クラスタを管理している。 今回、実験にもちいる環境として、PrestoIII クラスタ (CPU: Athlon MP 1900+ × 2 (SMP), Memory: 768MB, HDD 40GB, OS: Linux 2.4.18 × 255 nodes, Network: 100base-T connected with switching hub, Myrinet2000) を用いた。実験では、Dolly+ によるファイル配布性能、および Lucie によるインストール性能を調査した。

4.1 Dolly+ によるファイル配布性能

図 10 に、ネットワークとして ethernet(100 base-T) または Myrinet(GM[8]) ドライバ使用) を用いた場合の、Dolly+ による 1GB のファイル配布性能、およびノードを 250 台まで増加させた場合の予測性能を示す。なお、Dolly+ では、配布終了時にすべてのスレーブノードからマスターノードへ終了通知が送信され、マスターノード上にログを出力する。よって、Dolly+ が配布に要する時間として、マスターノード上での配布開始時刻と終了通知受信時刻の差を用いた。

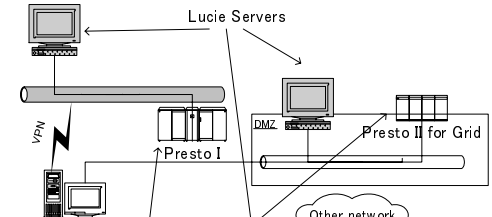

図 9 クラスタ構成図

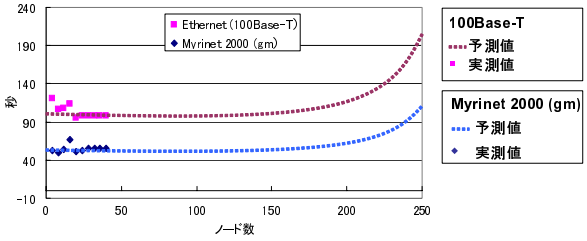

図 10 Dolly+ によるファイル配布性能

グラフより、Dolly+ を用いた場合、配布対象のノード数が増加した場合でも要する時間はほぼ一定であり、100base-T では約 110sec/1GB、Myrinet(GM) では 60sec/1GB であることがわかる。Lucie ではインストール時に、動的に GM モジュールを Linux カーネルへ読み込むことができるため、データ転送にもちいるデバイスとして、Myrinet を利用できる。このため、インストール時のデータ配布性能を向上できることがわかる。

Dolly+ による転送時間がほぼ一定である理由として、Dolly+ の仮想リング構造により、通信のソフトウェア的なボトルネックを回避していることが挙げられる。予測値は以下のように求められる。Dolly+ ではパイプライニングをおこなっていることから、1GB(ファイルサイズ) /4MB(chunsize)=250 段のパイプライニングで 1 ファイル分の転送遅延が生じると予測でき

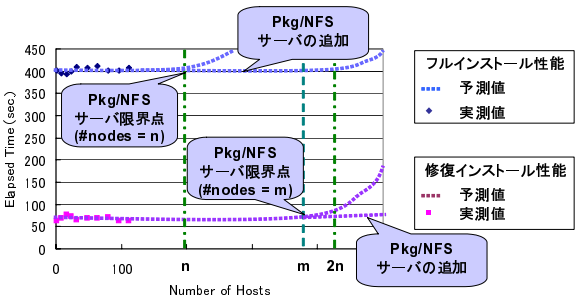

図 11 Lucie によるインストール性能

る。このため、250 ノードでのファイル転送時間は、2 ノードの場合でのファイル転送時間の 2 倍になると予測できる。

4.2 Lucie によるインストール性能

図 11 に、Lucie を用いてフルインストール、再設定のインストールをおこなった場合に要する時間 (実測値) とノード数を増加させた場合の時間 (予測値) を示す。なお、フルインストール実験では、212 パッケージ、合計約 300 MB (パッケージとして圧縮時の容量) をインストールした。また、すべてのノードについて、TFTP で配布されるインストール用カーネルをハードディスクへあらかじめ書き込んでおき、全ノードをリブートすることでインストーラを開始した。これは、TFTP の実装として広く使用されている atftp[19] の問題のため、10 台程度以上のノードを同時にブートした場合、数台のノードでの TFTP 失敗を確認したためである。

グラフより、Lucie 配布対象のノード数が増加した場合でも要する時間はほぼ一定であり、フルインストールをおこなった場合では約 400 秒、また再設定のみをおこなうインストールでは約 60 秒であることがわかる。

ノード数の増加がインストール時間へ影響しない理由として、最もボトルネックになりやすい部分である、各クライアントが Lucie サーバへアクセスする時間 (パッケージ取得時、最小構成 Linux のダウンロード時) は高々 30 秒程度であり、インストール時間全体に対する割合が低いためであると考えられる。予測値として、同時にインストールするノードを増加させた場合、NFS サーバもしくはパッケージサーバの性能限界 (図 11: ノード数 = n or m) によりインストール時間が増加すると考えられる。そこで、パッケージサーバや NFS サーバをミラーリングし、負荷分散させることによって、サーバ台数 × n or サーバ台数 × m までスケーラビリティを確保できると考えられる (図 11: ノード数 = 2n)。

5. 関 連 研 究

クラスタ用セットアップ・管理ツールの大部分は、単一ノード用自動インストーラをクラスタへ適用したものがほとんどである。Lucie が達成しているような、インストール用メディアを必要としないネットワークブート/インストール機構、設定パッケージ機構、インストーラ自体の拡張機構、および高速データ転送などといった、クラスタに特化した機能を達成したものは少ない。

RedHat KickStart[12] は、RedHat 社が配布する自動 Linux インストーラである。特長として、RedHat Linux の標準インストーラを用いて通常インストールをおこなうことにより、同内容のインストールをおこなう KickStart 設定ファイルが自動生成される点、ネットワークブート/インストールに対応している点が挙げられる。欠点として、KickStart は単一ノード用の自動インストーラであるため、複数ノードをグループ化し、同時に設定/運用するといった機能が無い。

SystemImager[16]、SystemInstaller[15]、System Configurator[14] および PowerCockpit[11] は、ディスクイメージのコピーによって複数ノードをセットアップするツール群である。ユーザは、ひな形となるマスタノードをセットアップし、システムイメージを作成する。作成したシステムイメージを rsync[20] やマルチキャストをもちいてコピーノードへ配布する。コピーベースなシステムの欠点として、ディスクイメージの更新には、rsync アルゴリズム等による差分更新といった高速化ができる一方、初回コピーに時間がかかる点がある。また、ヘテロなシステムを構成する際、異なるノード構成の数だけディスクイメージを作成しなければならない。

NPACI Rocks[18] は Lucie と同様、再インストールによってクラスタノード上のソフトウェアアップグレードや復旧といった操作を簡便におこなうことを目的としたツールである。NPACI Rocks では、KickStart 設定ファイルの断片 (スニペット)、およびスニペット間の継承関係を定義した、XML KickStart を提供している。ユーザは、クラスタノードで利用したいサービスを提供するスニペットを選択し、これを kpp, kgen と呼ばれるプリプロセッサ、トランスレータへ入力することよって RedHat KickStart のインストールメディアを作成できる。この方式の欠点として、スニペット間の競合関係を開発者自身が把握する必要があるため、XML ファイルが巨大になった場合、生成される KickStart ファイルに矛盾が生じやすい。NPACI Rocks のインストーラ自体は KickStart であるので、KickStart 以上の機能は無い。このため、インストーラ自体の拡張や、インストーラ動作環境のカスタマイズなど、Lucie が実現しているいくつかの機能はない。

Beowulf 型、および SCore 型[13] の標準クラスタ用構築ツールとして、Oscar[10] や Score EIT (Easy Installation Tool)[13] がある。これらのツールでは、ツールベンダ提供の標準設定を実現するクラスタを

半自動的に構築できる一方、インストールするソフトウェアの選択や追加といった、カスタマイズ機能が全般的に不足している。

6. まとめと今後の課題

われわれは、大規模クラスタ用の高速セットアップ/管理機構として、クラスタ用自動イ ンストーラ Lucie、および Lucie と協調動作し、インストール時に大規模ファイルの耐故障高速配布をおこなうツールである Dolly+ を開発した。

本稿ではインストール時のデータ配布機能について、Dolly+ を用いて配布対象のノード台数を変えて計測した。その結果、性能は台数によらずほぼ一定であり、40 ノードへ 1GB のデータ配布をする場合、約 110 秒 (100base-T), 約 60 秒 (Myrinet2000) であることを確認した。また、あらかじめ TFTP で配布されるイメージをディスクに書き込んだ 112 ノードを Lucie を用いてセットアップし、約 400 秒とその有効性を確認した。

問題点として、約 10 台以上のノードから TFTP リクエストをおこなう場合、まれに TFTP に失敗することがわかった。この問題は、ノードへ Linux がインストール場合、今回の実験でおこなったように TFTP で配布されるイメージをあらかじめ書き込むことによって回避できる。しかし、初回インストール時には各ノードへのインストールをシリアライズして実行しなければならないため、TFTP 実装の修正が必要である。

現在、エージェントによる障害の監視/自律的な回復の手段として、開発者の知識ベースによるスクリプトを用いた、アドホックな回復方法を用いている。今後の課題として、障害状況・障害検知方法・復旧方法の体系的な分類をおこない、ネットワークレイヤなどの各レイヤでの障害復旧処理の切り分け、および各レイヤ間での障害通知機構を実装する必要がある。

Lucie を用いたクラスタの動的な再構築, スケジューリングに関する試みとして, われわれは並列チェックポインティング/マイグレーション機能をもつ Parakeet MPI[21] を開発し、テストベッドとして ion プラグアンドプレイクラスタ[6] を構築している．将来は計算を止めないノードのプラグアンドプレイ機構として、クラスタ上の並列プロセスを Parakeet MPI により定期的にチェックポインティングし、障害発生時には Lucie を用いて故障ノードを新規インストールノードと交換、マイグレーション/リスタートにより並列プロセスを復活するといった機構についても調査する。

参 考 文 献

1) 3com support library - dynamicaccess managed pc boot agent (mba) downloads. http://support3com.3com.com/infodeli/tools/nic/mba.htm.
2) Cluster team @ matsuoka lab. web page. http://cluster-team.is.titech.ac.jp/.
3) Dolly - a program to clone disks. http://www.cs.inf.ethz.ch/CoPs/patagonia/dolly.html.
4) Dolly+ home page. http://corvus.kek.jp/~manabe/pcf/dolly/index.htm.
5) Intel wired for management (wfm). http://www.intel.com/labs/manage/wfm/index.htm.
6) ion cluster web page. http://cluter-team.is.titech.ac.jp/ion/index.html.
7) Lucie web page. http://lucie.sourceforge.net/.
8) Myrinet gm software for linux. http://www.myri.com/scs/linux.html.
9) NCBI blast home page. http://www.ncbi.nlm.nih.gov/BLAST/.
10) OSCAR: Open source cluster application resources. http://oscar.sourceforge.net/.
11) Powercockpit. http://www.mountainviewdata.com/us/powercockpit/.
12) Redhat linux kickstart information. http://wwwcache.ja.net/dev/kickstart/.
13) Score. http://www.pccluster.org/.
14) System configurator web page. http://www.systemconfig.sourceforge.net/.
15) System installer. http://systeminstaller.sourceforge.net/.
16) Systemimager. http://www.systemimager.org/.
17) Joey Hess. Debconf specification. http://kitenet.net/doc/debconf/specification.html.
18) Mason J.Katz, Philip M. Papadopoulos, and Greg Bruno. Leveraging standard core technologies to programmatically build linux cluster appliances. *CLUSTER 2002, IEEE International Conference on Cluster Computing*, April 2002.
19) Remni Lefebvre. Advanced tftp. http://freshmeat.net/projects/atftp/.
20) Andrew Tridgell. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, The Australian National University, 2000.
21) 高宮 安仁 and 松岡 聡. ユーザ透過な耐故障性を実現する MPI へ向けて. In **情報処理学会 電気通信処理学会 並列処理シンポジウム** *JSPP2002* **論文集**, pages 217–224, 2002.
22) 高宮安仁, 真鍋篤, 白砂哲, and 松岡聡. Lucie: 大規模クラスタに適した高速セットアップ・管理ツール. In **情報処理学会研究報告**, *Swopp 2002*, pages 131–136, 2002.