2008－OS－108 (18)
2008／4／24

複数拠点に分散配置されたクラスタの効率的な管理手法

高宮安仁[†] 弘中 健[††]
斎藤秀雄[††] 田浦健次朗[††]

複数の拠点に配置されたクラスタを連結し、1つの分散計算機環境として提供する試みが国内外で広まりつつある。こうした環境で長時間に及ぶ科学技術計算を安定して実行可能にするためには、複数拠点を1つの均一な実行環境として迅速に構築するとともに、メンテナンスに伴うダウンタイムを最小限にする必要がある。しかし、既存研究では複数拠点のセットアップに対応していないという問題や、環境更新ごとにOSの再インストールを必要とするため、ダウンタイムが増大するといった問題があった。我々は分散環境上に安定した計算機環境を効率良く構築するためのツールとしてLucieを開発している。Lucieでは、全拠点を並列にセットアップするための機構や、再インストールを行うことなくソフトウェア障害からの復旧やアップグレードを行う機構を備えている。評価では、グリッド全体のセットアップ性能を確認するため、代表的な分散計算機環境の1つであるInTriggerを用いて全5拠点、86ノードを同時にセットアップした。結果、拠点数の増加が拠点ごとのインストール性能に影響を与えず、複数拠点を並列に効率良くセットアップできることを確認した。

An efficient management method for multi-site distributed cluster environments

Yasuhito Takamiya,[†] Ken Hironaka,[††] Hideo Saito[††]
and Kenjiro Taura[††]

With arising of long running scientific jobs executed in grid environment, it is needed to construct a distributed multi-cluster envrionment as one uniform execution envrionment rapidly while minimizing downtime caused by software maintainances. Although there were considerable amount of efforts against grid management frameworks, no one aimed at parallel setup of multiple sites, or upgrading of the whole software environment without re-installation of OS. In this paper, we propose a new management tool for multi-cluster environment called Lucie, which allows rapid construction of stable multi-cluster computing platform over distributed environment. With Lucie, one could recover from software faults or upgrade the whole software stacks in parallel without the need of re-installation of OS. In our benchmark, we applied Lucie for construction of InTrigger grid environment. The result showed that the increase of number of sites parallely setup does not affect the other ones, so we could setup multiple sites efficiently.

1. はじめに

地理的に離れた複数の拠点にそれぞれクラスタを配置し、分散した計算機環境を構築する試みが国内外で始まっている。これは、グリッド[1]と呼ばれる計算環境の一形態である。代表的な例として、フランスのGrid'5000[2]、オランダのDAS-3[3]、日本のTSUBAME[4], InTrigger[5]などが挙げられる。

グリッドは大規模な解析を必要とする高エネルギー物理学や天文学，および生物学などさまざまな分野の研究開発用プラットフォームとしてすでに実用段階に入っており、利用率が高い。例えば、東京工業大学キャンパスグリッド[6]の2005年12月から2006年1月間の使用率は約80％と非常に高く、常に200～300ユーザに利用されている状況であった[7]。

グリッドを実用に耐え得るプラットフォームとして安定して提供するためには、さまざまな条件をクリアしなくてはならない。例えば、長時間に及ぶ科学技術計算を安定して実行させるためには、ノードのメンテナンスによるダウンタイムを最小に抑える必要がある。また、さまざまなジョブやユーザのニーズに応えるためには、すべての拠点のソフトウェア環境を統一することで拠点間の差異を無くし、また要求に応じてソフトウェアのインストールや更新を行う必要がある。

従来こうしたグリッド環境の管理では、従来のサー

† NEC システムプラットフォーム研究所 NEC System Platforms Research Laboratories
†† 東京大学 The University of Tokyo

バ管理手法と同じく、専任の管理者による属人的な管理方法 – オンサイトでの手作業によるインストールや設定 – が採られていることが多かった。この理由として、グリッド環境は長い間実用的では無かったため、管理手法やツールの層が厚くなく、またノード数もそれほど多くなかったためサーバ管理の延長で管理されていたという点があった。また、グリッドごとのハードウェアおよびソフトウェア設定の違いが大きいため、事実上の標準と呼べるツールが無かったということがある。

しかし、グリッド環境を効率的に利用するためのミドルウェアの成熟やハードウェアの低価格化に伴い、今後グリッド環境の普及が一層促進される傾向である。

我々の目標は、グリッド環境、とくに複数拠点に分散配置されたクラスタ同士を接続して構成されるグリッドの管理に適した管理手法やツールのデザインを示し、効率的な管理のノウハウを公開することである。また、管理のための一般的なガイドラインを示すことによって、グリッド管理手法を共通化し、グリッド同士の相互運用や設定の共有を促進することが重要である。

こうした目標を達成するため、我々は複数拠点に分散配置されたクラスタを効率的に管理するためのツールである、分散クラスタ管理ツール Lucie を開発している。Lucie では全拠点の並列セットアップによるダウンタイムの最小化、拠点やノード情報を利用した遠隔セットアップの自動化、Web インタフェースによる運用の簡易化などを実現している。

以下では、グリッド管理を難しくするいくつかの要因について述べ、そこから導かれる Lucie の設計を述べる。次に、設計の詳細について述べ、InTrigger 上での評価結果を報告する。

2. グリッド管理の課題

グリッドの管理の課題は、グリッドの構成要素であるクラスタに由来するものと、グリッド自体の特徴に由来するものに分類できる。

クラスタの特徴に由来する課題として、ノード数が多いことに起因する設定ミスが挙げられる。クラスタは多数のノードから構成されるため、1 台の計算機を管理する場合と比べて、単調な作業の繰り返しによる作業漏れや、作業者のスキルのばらつきなどによるミスをより起こしやすい。

この解決法として、すべての設定を自動化することで、繰り返しによる人為的ミスを排除するための設定記述言語が提案されている[8),9)]。これらの記述言語では、あらかじめソフトウェアごとの設定と設定間の依存関係を記述しておくことで自動的にすべてのノードの設定を行う。

クラスタ同士を接続したグリッドの特徴として、単体のクラスタと比較して非常にノード数が多いことが挙げられる。このため、単体クラスタでは表れにくかった次のような課題が生じる。

インストールの効率化 グリッド全体を同時に更新する場合など、非常に多数のノードへのインストールを効率的に行うために、インストール時の無駄なデータ転送を減らすといった効率化が必要である。また、インストールに失敗する箇所があっても、全体としては影響を受けることなくインストールを完了させることが重要である。

ソフトウェア環境の均一化 グリッド全体で均一な実行環境を提供するために、設定の変更や更新をノードの状態によらずすべてのノードへ反映させなくてはならない。たとえば、グリッド環境では常にすべてのノードが正常稼動しているとは限らないため、障害から復旧したノードには自動的に設定を適用する必要がある。

ダウンタイムの削減 ユーザが増加するにつれて、メンテナンスに伴うダウンタイムの影響はより深刻となる。このため、ソフトウェアのインストールや障害時の復旧方法としては、ノードの再起動を伴わない方法が好ましい。また、インストール処理をリアルタイムで監視し、エラーが起こった場合には直ちに通知を行うことで、障害の影響を小さくし、復旧までの時間を短縮する必要がある。

そのほかの課題として、次の 4 つがある。

ネットワークが一時的に利用不能となった場合でもキャッシュなどを用いてインストールを続行する必要がある。これは、分散クラスタ管理ツール中に単一故障点があると、ネットワークの切断によってシステムのインストールが不安定になったり、不可能になる可能性があるためである。

インストール時に、キーボードやマウス操作といったすべての対話的操作を排除し、インストールをネットワーク経由で自動化する必要がある。これは、グリッドは物理的に分散して配置されるため、電源の ON/OFF や BIOS 設定の変更などといったインストールに必要な操作を直接行うことができないためである。

グリッド内の全拠点のネットワーク情報データベースを構成し、インストール時にこれを参照できる必要がある。これは、ホストベース認証や IP フィルタリ

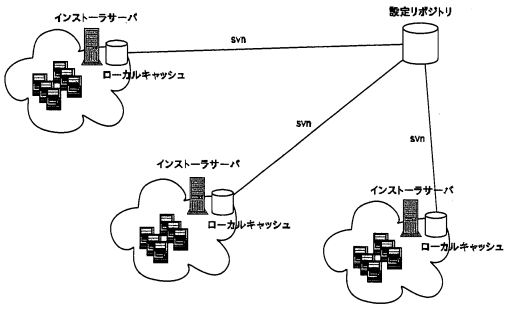

図 1 Lucie の全体像

ングの設定など、グリッド内のノード同士の通信と外部との通信を区別するためである。

拠点間でのハードウェアおよびネットワーク構成の違いを吸収する必要がある。これは、グリッドは複数拠点のマシンが混在するため、拠点間で大部分の設定は共通であるもの、ハードウェアやネットワーク構成に多少のばらつきが生じるためである。

3. 分散クラスタ管理ツール Lucie

こうした課題を解決するため、我々は単体クラスタ管理ツール Lucie[10] のグリッド対応版を開発し、フリーソフトウェアとして公開している[11]。次節以降ではこれまでに述べた課題を踏まえた上で、Lucie の設計方針と解決方法について説明する。

3.1 Lucie の全体設計

Lucie は集中管理型のシステムであり、複数の拠点に分散配置されたクラスタすべてを一箇所から集中管理する。

Lucie はグリッド全体の設定を保持する設定リポジトリと、個々の拠点を管理するインストーラサーバ、および拠点を構成するノード群から構成される (図 1)。管理者は、ノードの追加/削除やインストール、設定の更新といったコマンドによってグリッド全体を管理する。

3.1.1 インストールの効率化

Lucie ではインストールを効率的に行うため、拠点単位で並列にインストールを行う。拠点ごとのインストールが完全に独立して行われるため、インストールの遅い拠点やインストールに失敗した拠点は他の拠点のインストールに影響を与えない。

また、インストール時の無駄なデータ転送を削減しインストールを高速化するため、すべてのノードは拠点内のプロキシを通じてソフトウェアパッケージをダウンロードする。この際、一度ダウンロードされたソフトウェアパッケージはローカルキャッシュ上にキャッシュされる。これによって、拠点の外とのデータ転送量を削減しインストールを効率化するとともに、一時的なネットワークの切断時にもインストールを続行できる。

3.1.2 ソフトウェア環境の自動的な均一化

全拠点を常に最新の状態かつ均一に保つ仕組みとして、インストーラサーバによる設定リポジトリのポーリングがある。インストーラサーバは定期的に設定リポジトリの更新をポーリングし、更新があった場合には差分をチェックアウトする。また、チェックアウトした差分を全ノードへ反映する。このため、管理者はこの設定リポジトリへ変更をコミットするだけで、変更点を自動的に全拠点へ行き渡らせることができる。

3.1.3 ダウンタイムの削減

Lucie では設定が壊れた場合に OS の再インストールを行う代わりに、設定全体のうちで壊れた部分と影響を受ける箇所を判定し修復することで、リブートによるダウンタイムを削減する。Lucie ではインストール後の設定用のサブシステムとして、設定記述言語 Puppet[9] を用いている。Puppet ではソフトウェアごとの設定と、設定間の依存関係を宣言的に記述することで、障害時には障害を起こした設定と依存する部分のみ再設定する。これによって、復旧に必要な OS の再インストール回数を削減している。

3.1.4 ネットワークエラーへの対処

一時的なネットワーク切断によるインストールエラーを避けるため、各拠点のインストーラサーバは設定リポジトリから取得した設定をローカルキャッシュへキャッシュする。設定リポジトリとの接続が切れた場合にはローカルキャッシュを用いることで、設定リポジトリが単一故障点となるのを避けている。

3.1.5 物理的操作の排除

ノードをネットワークブートによってインストールし、インストール完了後にローカルブートに切り替えるためには、通常 BIOS 画面でのブートデバイス変更操作が必要である。これを無くすために、Lucie では次のようなブートデバイス切り替え方式を用いている。

グリッド管理者はノードの設置時に一度だけ BIOS 設定を行い、ブートデバイスをネットワークブート (PXE) に指定する。この設定によって、ノードはデフォルトで PXE によるネットワークブートを試みる。

インストーラサーバでは、ノードがネットワークブート時に TFTP プロトコルで取得するブート設定ファイルを切り替えることによって、ブートデバイスの切り替えを行う。たとえば、ノードのインストール

を行う際には、設定ファイル内でルートデバイスとしてNFSを指定し、NFSのパスとしてインストーラサーバ上のインストーラを指定する。また、インストール後のローカルディスクからのブートを行う際には、ルートデバイスとしてローカルディスクを指定する。これによって、いったんBIOSの設定を行うだけで、インストーラサーバからインストール/ローカルブートを切り替え可能である。

4. 実　　装

本節ではLucieを構成する各コンポーネントについて実装の詳細を説明する。

4.1 設定リポジトリ

設定リポジトリに管理される情報として、すべての拠点とノード情報、インストーラの設定情報、およびPuppetによるソフトウェア設定がある。

拠点とノード情報はYAML形式[12]のテキストファイルとして次のように保存されている。

```
hongo:
  dnsdomain: logos.ic.i.u-tokyo.ac.jp
  nodes:
    hongo000:
      macaddress: 00:01:80:62:64:30
      install_address: 157.82.22.10
      global_address: 157.82.22.10
      ncpu: 1
      arch: i686
    hongo001:
      macaddress: 00:01:80:62:64:31
                ...
```

YAMLのトップレベル情報として、グリッド内に存在する拠点が定義される。拠点定義内では、拠点内の全ノードで共通な情報と、拠点に含まれるノードが定義される。ノード定義内ではノードのIPアドレス、MACアドレス等ノード固有の情報が記述される。

インストーラの設定情報としては、インストールするカーネルのバージョンやソフトウェアパッケージの取得元など、Puppetを起動して設定を行うのに必要な最小限の環境を構築するための情報が含まれる。

Puppet設定では、ソフトウェアごとの設定が宣言的に記述される。たとえば、次のPuppetスクリプトはsshの設定を行うスクリプトである。

```
package { 'ssh':
    ensure => latest,
    before => Service[ssh],
}
```

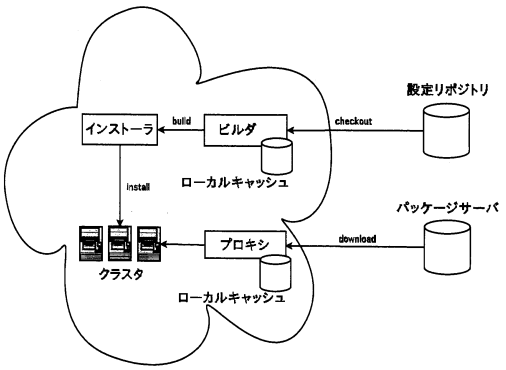

図2　インストーラサーバの構成

```
service { 'ssh':
    name   => sshd,
    ensure => running,
}
```

この例では，sshサービスが起動される前に，sshパッケージがインストールされている必要があることを宣言している。また、sshサービスが落ちていた場合自動的に復旧することを宣言している。

設定リポジトリはバージョン管理システムの一種であるSubversion[13]のリポジトリとして提供される。新しく拠点を追加するためには、installer addコマンドの引数としてリポジトリのurlを指定するだけで、自動的にインストーラサーバを構築できる。

```
% ./installer add my_installer \
        --url svn://mygrid.com/config \
        --username foo --password xxxxx
```

4.2 インストーラサーバ

各拠点で動作するインストーラサーバは、クラスタノードをインストールするインストーラ、インストーラをビルドするビルダ、ソフトウェアパッケージのダウンロード用プロキシ、設定リポジトリの内容およびソフトウェアパッケージをキャッシュするローカルキャッシュから構成される(図2)。

インストーラは拠点内の全ノードをネットワーク経由で並列にインストールするネットワークインストーラである。

ビルダは定期的に設定リポジトリの変更をポーリングし、変更があった場合には変更点をローカルキャッシュへ保存する。また、ローカルキャッシュを用いてインストーラをリビルドする。

プロキシはhttpプロキシとして動作し、各ノードか

らのソフトウェアパッケージダウンロード要求をプロキシする。また、一度ダウンロードされたソフトウェアパッケージをローカルキャッシュへ保存する。

ローカルキャッシュは、設定リポジトリからチェックアウトした設定情報や、一度ダウンロードされたソフトウェアパッケージをキャッシュとして保存する。

4.3 ハードウェア構成の違いの吸収

表 1 に実験環境である InTrigger の 2008 年 3 月現在のスペックを示す。インストール設定に影響を与える拠点間での違いとして、1) CPU アーキテクチャの違い (64/32 bit) 2) IPMI の有無 3) ネットワークアドレス (プライベート/グローバルアドレス) の違い 4) ディスク容量の違い、が挙げられる。

我々は、こうした設定の違いを吸収したインストール設定を記述するため、全拠点間での共通設定と拠点固有の設定を切り分け、それぞれを Puppet のクラスとして次のように定義した。

```
# 拠点間 共通設定
node default {
  include ssh
  include ganglia
    ...
  include site_$sitename
}

# 拠点固有設定
class site_chiba {
  case $arch {
    i686:   { include network_hongo0_type }
    x86_64: { include network_hongo1_type }
  }
  include iptables_global
}
```

この例では、全拠点に共通な設定が node default として定義されており、chiba 拠点に固有の設定は site_chiba クラスとして定義されている。YAML データベースより、Lucie によって変数 $sitename に自動的に拠点名 ('chiba') がセットされる。このため、node default 内で chiba 拠点の定義が include される。同様に、chiba 拠点内の CPU アーキテクチャによる違いは、変数 $arch の値によって場合分けされる。

4.4 インストールログの収集と進捗管理

Lucie ではインストールジョブの投入に ssh を用いることで、インストールログの収集と進捗管理を行う。

一般的なインストーラのインストール方法を大きく分類すると、1) 各ノードが自律的に独立してインストールを進める方法 2) サーバからインストールジョブを投入する方法、の 2 通りが考えられる。

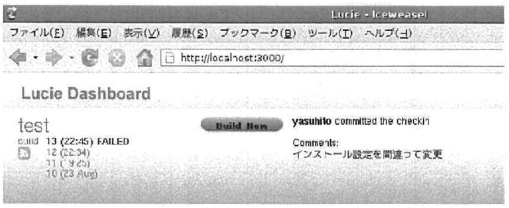

図 3 Dashboard のスクリーンショット

1) の方法では、ノードのブート時に各ノードがインストーラを開始し、インストールが終了するとインストールした OS を起動する。インストールのログはインストール終了時などにまとめてサーバ側に送付する。

この方法では、インストールの進捗管理をサーバが行わなくて良いためノード台数に対してスケーラブルであるが、インストールが途中で失敗した場合にインストールログを失うため、失敗の原因がわかりづらい。

2) の方法では、インストールの各処理をサーバから全ノードへ向けて順々に投入し、処理ごとにログを収集する。

この方法ではサーバがインストールの進捗管理を行う必要があるためサーバの負荷が高いものの、インストールの詳細なログをサーバ側で収集できるため、インストール状況を把握しやすい。

Lucie では 2) の方法を用い、インストールジョブの投入に ssh を用いた。また、ログの収集ではインストールジョブの実行結果を ssh 経由でログファイルに書き込むようにした。

4.5 管理用 Web インタフェース

拠点やノード情報のリアルタイム監視とグリッド運用の簡易化を実現するツールとして、Lucie では Dashboard と呼ばれる Web インタフェースを提供している。Dashboard では全ノードのインストールログをリアルタイムで閲覧可能であり、エラーを起こしたノードを赤く表示するなどといった機能がある。これによって、ダウンタイムを低減することが期待できる。また、Web インタフェースを提供することによって、運用コストを低減することが期待できる。

各種ログをリアルタイム監視するための仕組みとして、Ajax と DHTML によって 15 秒ごとにログを非同期的に取得し、表示を動的に更新するようにした。

表 1 InTrigger を構成する各拠点の仕様

拠点名	設置機関	CPU	メモリ	ディスク	ネットワーク	ノード数
chiba0	国立情報学研究所	Pentium M 1.86GHz	1GB	300GB	グローバル	70
chiba1		Core2 Duo 2.13GHz	4GB	1TB		58
hongo0	東京大学	Pentium M 1.86GHz	1GB	70GB	グローバル	70
hongo1		Core2 Duo 2.13GHz	4GB	500GB		14
imade_p	京都大学	Core2 Duo 2.13GHz	4GB	500GB	プライベート	28
imade_g		Core2 Duo 2.13GHz	4GB	500GB	グローバル	2
kyoto_p	京都大学	Core2 Duo 2.13GHz	4GB	500GB	プライベート	31
kyoto_g		Core2 Duo 2.13GHz	4GB	500GB	グローバル	4
okubo	早稲田大学	Core2 Duo 2.13GHz	4GB	500GB	グローバル	14
suzuk	東京工業大学	Core2 Duo 2.13GHz	4GB	500GB	グローバル	36
mirai	はこだて未来大学	Xeon 5410 2.33GHz	16GB	2TB	グローバル	6
kobe	神戸大学	Xeon 5410 2.33GHz	16GB	2TB	グローバル	11
合計			1160GB	382.9TB		344

これによって、UI をブロッキングすることなく、ブラウザの更新ボタンを押下しなくても定期的に自動更新することができる。

また、管理用のサーバにログインする手間を省き運用を簡易化するために、インストーラの再構築やノードの追加・削除☆など、コマンドラインツールと同等の操作を Web ブラウザ上から行うことができる。

これらの機能は明示的なインストールが不要である。これは、Lucie デーモンが Dashboard の機能を提供する http デーモンとしても起動するためである。このため、Apache など外部 http サーバの明示的なインストールや設定が不要であり、導入コストの削減が図られている。

5. 関連研究

システムの設定を記述するための言語や単体クラスタのインストールツールがいくつか提案されている。

5.1 Cfengine

Cfengine[8] はシステム管理用の言語であり、puppet と同様に宣言的にシステムの設定を記述することができる。サーバ・クライアント型のアーキテクチャを採用しており、サーバ上の設定ファイル (マニフェスト) が各ノードへ適用される点も Puppet と同様である。Cfengine 自体はインストール機能を持たないため、他のインストーラと組み合わせることでクラスタのセットアップを行うことができる。

Cfengine の問題点として、マニフェストが一元化されていないことによる不整合性が挙げられる。Cfengine ではマニフェストがマスターサーバに一元化されておらず、各ノードがあらかじめ一部のマニフェストをローカルに持っておく必要がある。このため、セットアップが複雑であり、タイミングによってはマスターサーバとノードのマニフェストが不整合を起こす可能性がある。

一方 Puppet では、ノードを設定するのに必要な情報はマスターサーバ (インストーラサーバ) のホスト名のみであり、このホスト名は Lucie によってノードへ自動的に注入される。このため、マニフェストは常にマスターサーバ (設定リポジトリ) に一元管理されることとなり、整合性が保たれる。

他の Cfengine の問題点として、拠点やノードの情報を設定に反映させる場合の煩雑さが挙げられる。Cfengine では、他の拠点情報に依存する設定 (iptables など) を行うためには、拠点およびホスト情報のデータベースとデータベースにクエリを送る仕組みを Cfengine 上に実現する必要がある。しかし、これに必要な拡張を行うためには C 言語による実装が必要である。

一方 Lucie では、Lucie が管理する拠点およびホスト情報のデータベースを puppet 内の変数として参照できる。このため、拠点情報を反映したマニフェストを容易に作成することができる。

5.2 NPACI Rocks

NPACI Rocks[14] は単体クラスタ用のインストーラであり、単一ノード用のインストーラである anaconda を用いてクラスタ全体をネットワークインストールする。クラスタに障害が起こった場合には障害の解析や部分的な回復などを行わず、必ずクラスタ全体を再インストールすることで、クラスタ全体を健全な状態に保つ。また、Rolls と呼ばれる設定テンプレートを提供しており、これを用いることによってクラスタやグリッド向けアプリケーションのインストールと設定を簡単に行うことができる。

再インストール方式の問題点として、再インストー

☆ 現時点では未実装

ルによるハードウェアへのダメージとダウンタイムの増大が挙げられる。特にハードディスクのフォーマット処理はディスクや電源への負担が大であり、インストールを繰り返し行うことで故障率が高まる。また、再インストール中はクラスタ全体が利用不能となってしまうため、ダウンタイムが増大する。

また Cfengine と共通する問題として、拠点やノード情報を設定に反映させるための仕組みが無い点が挙げられる。これは、Rolls は基本的にはシェルスクリプトに変換される XML のセットであり、設定に必要な情報をデータベース等から取得する仕組みを備えていないためである。

その他の問題点として、インストールに失敗した場合の原因の解析が難しい点がある。NPACI Rocks ではインストールの進捗確認に rocks-console と呼ばれる X Window 上で動作するツールを提供しており、ノード単体のインストール進捗情報を X Window プロトコル経由で表示する。ノード単体のインストールの進捗画面は転送されるものの、ログ自体は転送されない。このため、どのノードがインストールに失敗したか、どの箇所で失敗したかなどの解析が難しい。

一方 Lucie では、ssh を用いてノードごとのインストールログをサーバ側にすべて蓄積するため、ログ解析が容易である。

5.3 広渕らのシステム

広渕らのシステム[15] では、複数拠点に配置されたクラスタをインターネットを介したイーサネット VPN によって単一のネットワークセグメントに接続することで、仮想クラスタを構築することができる。ネットワーク的に離れたクラスタをまとめて 1 つの仮想クラスタとして扱うことができるため、この上で NPACI Rocks などの単体クラスタ用インストーラを無改造で動作させることができる。

この方式の問題点として、インターネットを介したイーサネット VPN は、利用状況によっては不安定であったり十分な帯域が確保されない場合があるため、インストールに失敗したりインストール時間が 1 時間以上に増大する場合がある。これは、単体クラスタ用のインストーラは動作環境として LAN を想定しており、インターネット上ではインストール時のパッケージファイルの配布などでタイムアウトが起こるためである。

一方 Lucie ではクラスタの仮想化は行わず、拠点ごとに独立してインストールを行う。拠点内でのインストールは基本的に単体クラスタインストーラでの処理と同様であり、設定ファイルの生成に必要な他拠点の情報のみの抽象化を行う。また、ノードにダウンロードされるパッケージファイルは拠点のプロキシサーバにキャッシュされるため、効率的に配布することができる。このため、拠点数が増えた場合でも実用的な時間でインストールを完了することができる。

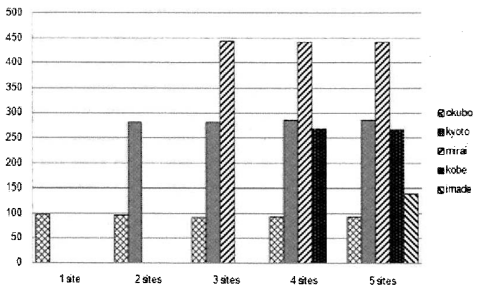

図 4 複数拠点のインストール時間

6. 性能評価

評価では、複数拠点の同時インストールが効率化されることを確認するため、InTrigger 環境のセットアップに要する時間を計測した。セットアップに用いるコマンドとして、node install コマンドを使用した。なおこのコマンドは拠点内のすべてのノードに対して、Puppet クライアントとして動作するのに最小限のセットアップ (ハードディスクの初期化、Linux のベースシステムとカーネル、および Puppet クライアントのインストールの後、リブート) を行う。

評価方法として、複数拠点へ並列に node install コマンドを発行し、各拠点で node install コマンドが完了するまでの時間を計測した。使用した InTrigger の拠点は okubo(14 ノード)、kyoto(30 ノード)、mirai(6 ノード)、kobe(11 ノード)、imade(25 ノード) の 5 つである☆。

図 4 に複数拠点のインストール時間を示す。同時にインストールする拠点数を増やした場合でも、それぞれの拠点のインストール時間はほぼ変化しないことがわかる。これは、それぞれの拠点でのインストールは他の拠点と完全に独立して行われるためである。たとえば今回、拠点 mirai の DNS の逆引きに失敗する問題のため、mirai のノード数は他拠点と比較して少ないにもかかわらずインストールに最長時間を要した。

☆ かっこ内は、各拠点のノードのうち実際にインストール可能であったノード数。

このように一部の拠点で障害が起きた場合でも、他の拠点へは悪影響を与えないことがわかる。

7. まとめと課題

複数拠点に分散配置されたクラスタを効率的に管理するための手法として、分散クラスタ管理ツール Lucie を紹介した。Lucie ではインストールの効率化やダウンタイムの削減、拠点やノード情報を利用した遠隔セットアップの自動化、Web インタフェースによる運用の簡易化などを実現している。

評価では、拠点数が増加した場合でも Lucie によるインストールが効率的に行われることを示すために、InTrigger 内 5 拠点 86 ノードの同時インストール時間を計測した。結果、インストールに時間のかかる拠点を含めて同時インストールを行った場合でも、拠点ごとのインストールは独立して行われるため、拠点ごとのインストール時間は変化しないことを確認した。

今後の課題として、Puppet 言語の改良がある。Puppet 言語ではソフトウェアごとの設定を宣言的に記述し、設定間の依存関係を明示的に定義する必要がある。このため、従来のシェルスクリプトによる手続き処理的な設定方式と比較して厳密であり、依存関係を利用した自動復旧などを実現している。一方で、正しい設定を書くための難易度が高い。これを改良するため、Puppet の機能を失なうことなくマニフェストの一部にスクリプト的な手続き処理を記述できる仕組みが必要である。

また、新たに作成した設定をテストするための仕組みと、誤った設定をロールバックするためのトランザクション処理をサポートする必要がある。現在の仕組みでは、設定リポジトリに設定ファイルをコミットすると、自動的にすべてのノードに適用される。このため、万が一設定が間違っていた場合には、Subversion リポジトリのリビジョンを元に戻すといった擬似的なロールバックを行う必要がある。そこで、これらの一連の処理を Lucie で提供することによって、テストから実運用、障害時のロールバックをシームレスに行うための機構が必要である。

加えて、今回 InTrigger 環境を Lucie でセットアップする際に得られた知見をもとに、管理手法のノウハウやガイドラインを InTrigger や Lucie の Web 拠点等で公表していく予定である。

謝辞 本研究の一部は、『文部科学省科学研究費補助金「特別研究」情報爆発時代に向けた新しい IT 基盤技術の研究』による。また、Lucie の開発や実験の全般について、東京大学近山・田浦研究室および東京工業大学松岡研究室の皆様に支援をいただいた。加えて、InTrigger の各拠点を運用いただいている各研究室 (国立情報学研究所、京都大学角研究室・中島研究室、早稲田大学山名研究室、東京工業大学合田研究室、はこだて未来大学松原研究室、神戸大学永田研究室) の皆様にもご協力をいただいた。

参 考 文 献

1) I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications*, 15(3), 2002.
2) Grid '5000. https://www.grid5000.fr/.
3) The distributed asci supercomputer 3 (das-3). www.cs.vu.nl/das3/.
4) Tsubame. http://www.gsic.titech.ac.jp/.
5) Intrigger プラットフォーム. https://www.logos.ic.i.u-tokyo.ac.jp/intrigger/.
6) 東工大キャンパスグリッドプロジェクト. http://www.gsic.titech.ac.jp/TITechGrid/.
7) 松岡 聡. Tsubame の飛翔. グリッド協議会 第 18 回ワークショップ, 2006. http://www.jpgrid.org/event/2006/pdf/WS18_matsuoka.pdf.
8) Cfengine. http://www.cfengine.org/.
9) Puppet. http://reductivelabs.com/trac/puppet.
10) Y. Takamiya, A. Manabe, and S. Matsuoka. Lucie: A Fast Installation and Administration Tool for Large-scaled Clusters. In *Proceedings of Symposium on Advanced Computing Systems and Infrastructures (SACSIS2003)*, pages 365–372, May 2003.
11) Lucie web page. http://lucie.is.titech.ac.jp/trac/lucie/.
12) Yaml ain't markup language. http://yaml.org/.
13) Subversion. http://subversion.tigris.org/.
14) P.M. Papadopoulos, M.J. Katz, and G.Bruno. NPACI Rocks: Tools and techniques for easily deploying manageable linux clusters. In *Proceedings of 2001 IEEE International Conference on Cluster Computing*, Newport, CA, October 2001.
15) 広渕 崇宏, 谷村 勇輔, 中田 秀基, 田中 良夫, and 関口 智嗣. 複数サイトにまたがる仮想クラスタの構築. In **並列/分散/協調処理に関するサマー・ワークショップ** *SWoPP 2007*, 2007.