Vol.2009-OS-112 No.19
2009/8/6

プログラムテスト環境を提供するクラウドコンピューティングシステムの検討

坂西　隆之[†1]　　小泉　仁志[†1]
神林　亮[†1]　　佐藤　三久[†1,†2]

本稿では自動的にテスト環境の構築，並列プログラムテスト，デバイスに対する故障エミュレーションが可能な並列分散システムテスト環境を提供するクラウド D-Cloud を提案する．高い信頼性確保のためには異なる入力による網羅的なテスト実行やハードウェア故障に対する耐故障性のテストなど様々なテストが存在し，それらを実行するには非常に時間と手間がかかる．D-Cloud ではフォルトインジェクションが可能な仮想マシンを用いて，仮想デバイスレベルでの故障についてのテストが可能であるだけでなく，仮想マシンをクラウドとして管理することにより，多くの計算資源を柔軟に利用することができ，多くのケースについてのテスト作業を自動化することができる環境を提供する．システムの構成やフォルトインジェクションを含むテストのシナリオの記述について検討し，システムの設計した．

Design of a Cloud Computing System for Program Testing Environments

Takayuki Banzai,[†1] Hitosi Koizumi,[†1]
Ryo Kanbayashi[†1] and Mitsuhisa Sato[†1,†2]

In this paper, we propose a cloud computing system, D-Cloud, which provides parallel software testing environments for reliable distributed and single systems by virtual machines with fault injection facility and automating a series of tests. Recently, as the importance of high dependability in software system is increasing, exhaustive testing of software systems is getting costly and time-consuming. D-cloud enables the test on the fault tolerance by device fault at virtual machine level, and allows exhaustive tests to be run automatically by flexible management of virtual machines by cloud computing technology. We designed the D-Cloud system including the description language in XML for system configuration and scenario of fault injections.

1. はじめに

近年，社会の高度情報化に伴い，さまざまな情報システムにおいて高い信頼性の確保が重要となってきている．しかし信頼性を確保することは容易ではなく，ソフトウェア開発においてはソフトウェア規模の増大，テストを行うための作業量の多さなどから信頼性確保のためのプログラムテストが十分に行えないことが多い．そしてプログラムテストの実行はテストを行うための環境を全てのテストノードで構築する必要性がある．また様々な入力に対して正確に動作が行われていることの確認など非常に手間がかかるだけでなく，大量の計算リソースを必要とする．

高信頼性を確保する必要のある多くのシステムはソフトウェアの不具合の問題だけでなく，ハードウェアの障害に対処する必要があり，ハードウェア故障を前提とした耐故障機能を備えている．このようなシステムをテストする場合，ハードウェアデバイスの故障にテストプログラムが対応しているかを検証する必要がある．しかしハードウェアの故障に対する耐故障性を検証するためにハードウェアの一部を故意に故障させること，非現実的な負荷をかける必要があることなど実際には難しい場合が多い．

特にネットワークで結合された複数のシステムからなる分散システムでは，実際にハードウェアにおいて故障させることが難しいだけでなく，現象を再現し，原因を追究することが極めて困難である．そこで，我々は並列分散システムの信頼性をテストするための環境として仮想マシンを用いたフォルトインジェクタが有効であると考え，Fault VM[1)] を提案した．しかし Fault VM はフォルトインジェクションを行うのみあり，テスト環境の構築などは行われないためテスト者が各自で行わなければならずテスト者の労力は依然として大きい．

そこで本稿では，分散システムのプログラムテストにおいて自動的にテスト環境構築，システム動作確認，ハードウェア故障に対する耐故障機能のテストが実行可能な並列分散システムテスト環境 D-Cloud を提案する．D-Cloud は，クラウドコンピューティングシステム Eucalyptus[2)] を用いての計算資源の管理をベースとして，テストシステムの構築，フォルトインジェクションによるハードウェア故障エミュレーション，様々なテストケース・シナ

†1 筑波大学大学院システム情報工学研究科
University of Tsukuba,Graduate School of Systems and Information Engineering
†2 筑波大学計算科学研究センター
University of Tsukuba,Center for Computational Sciences

リオの自動化機能を提供するクラウドコンピューティングシステムである．D-Cloud を用いることにより短期間で効率的なプログラムテストが可能となる．また時間的，経済的コストを抑えるだけではなく，システム全体の信頼性の検証も可能であり，信頼性確保の実現につなげることができる．

本稿の構成は以下のようになっている．2 章において D-Cloud の目的・概要について述べ，3 章において設計について述べる．4 章において記述するシナリオについて述べ，5 章において関連研究について述べる．6 章においてまとめと今後の課題について述べる．

2. フォルトインジェクション機能を持つ並列分散システムテスト環境 D-Cloud

2.1 目的・特徴

D-Cloud は，高信頼なシステムの開発環境として，フォルトインジェクション機能を持つ仮想マシンの提供を行う．そしてクラウドコンピューティングシステムによって管理される計算資源上において，同時に多くのシステムのテストを可能とする．D-Cloud の特徴を以下にまとめる．

- 仮想マシンレベルでフォルトインジェクション機能を実装することにより，システムを実際のマシンに近い環境でハードウェア障害に対する耐故障機能をテストすることができる．
- クラウドコンピューティングシステムを用いることにより，テストに必要な計算資源を柔軟に管理することができる．また，大量の計算資源が利用可能な場合には，テストを並列に行い，高速化が行える．
- 仮想マシンを複数用いることにより，高信頼システムとして重要な分散システムのテストが，容易に行える．
- クラウドコンピューティングシステムと連携して，テストを自動化する環境を提供し，テストの効率化が行える．

2.2 仮想マシンの利用

D-Cloud では，テストプログラムの実行に仮想マシンを利用する．

仮想マシンはハードウェアデバイスのエミュレーションも行っているため，デバイスに対してフォルトインジェクションを行うことができる．これにより，プログラムコードによるソフトウェアフォルトインジェクションのようにプログラムの変更を必要とせず，ハードウェア故障のエミュレーションが実行できる．

さらに仮想マシンを利用することにより，ユーザレベルのプログラムのテストだけでなく，カーネルレベルのプログラムテストが可能となる．実際のシステムではカーネルレベルのバグの多くはカーネルパニックとなり自動的に OS が停止してしまう．このような状態になってしまうとその OS 上で操作することができず，デバックに必要な情報が得られないことが多い．しかし，仮想マシンを利用することによりゲスト OS 上でカーネルパニックが発生してもホスト OS に対して影響がないため，システム全体が停止することはなくなる．また定期的にゲスト OS のスナップショットを取得することにより，任意のスナップショットに OS の状態を戻すことができるため，予期せぬエラーによりカーネルパニックが発生しても，スナップショットを用いて任意の状態に戻すことができる．

2.3 計算資源の管理

信頼できるシステムの開発を行う場合，一般的に潜在するバグを検出・修正するためになるべく多くのケースについて迅速にテストを行えることが望ましい．そして多くのテストを行うためには大規模な計算リソースを効率的かつ柔軟にマネージメントする必要がある．

D-Cloud では，クラウドコンピューティングシステムにより，仮想マシンの計算資源を柔軟に管理することができる．例えば高信頼化が必要なシステムの多くは，複数のホストからなる分散システムとなる場合が多いが，複数の仮想マシンによる並列システム構成も可能となる．

2.4 システム構築・検証テストの自動化

D-Cloud では，検証作業を支援するために対象となるテストシステムの構築とプログラムテスト，フォルトインジェクションを自動的に行うことができる．

D-Cloud はテスト者が記述した XML 形式のシナリオを用意することで，シナリオに沿ったテストシステムの構築，テスト実行，フォルトインジェクションを行う．シナリオに複数のテスト環境を記述することで，異なるテスト環境を作成し同時に実行することができる．よって対象となるシステムのディペンダビリティ支援機構が有効に働くかどうか網羅的にテストすることができ，実システム上でテストを行うのに比べて短時間で多数のテストを行うことが可能となる．

3. D-Cloud の設計

D-Cloud は仮想化ソフトウェア QEMU[3] と AmazonEC2[4] のオープンソースでの実装である Eucalyptus を利用する．**図 1** に D-Cloud の構成を示す．

D-Cloud を構成する主な要素を示す．

(1) フォルトインジェクション可能な QEMU ノード

Vol.2009-OS-112 No.19
2009/8/6

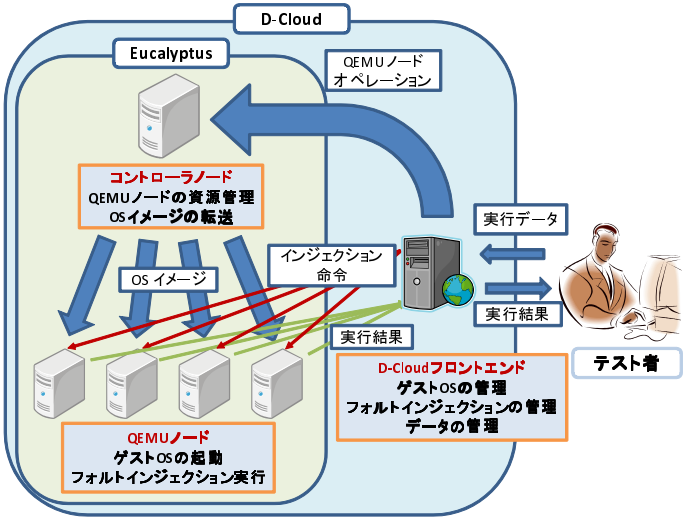

図 1 D-Cloud の構成

(2) QEMU ノードを管理する Eucalyptus 環境

(3) テスト者と QEMU ノードとのデータ転送，QEMU ノードへフォルトインジェクション命令を行う D-Cloud フロントエンド

以下に上記 3 要素について述べる．

3.1 QEMU を用いたフォルトインジェクション

D-Cloud では仮想化ツールとして QEMU を利用している．QEMU の使用には以下の利点がある．

- ハードウェアデバイスをエミュレートしている．
- オープンソースで提供されている．
- モニターモードがありゲスト OS の挙動がホスト OS から確認することができる．

QEMU は全てのデバイスを仮想的にエミュレートしており，任意のデバイスに対して故障のエミュレートが可能である．ソースコードが公開されているため，インジェクションが可能であるように改変を行うことができる．またモニターモードを使用することによりゲスト OS のメモリ情報の取得，レジスタ情報の取得，ゲスト OS の動作を制御することなどが可能である．

表 1 フォルトインジェクションの種類

デバイス	内容	要素
ハードディスク	指定されたセクタがエラーを返す	badblock
	指定されたセクタが readonly になる	readonly
	ECC 符号により誤りを検出する	ecc
	誤りを含むデータを返す	corrupt
	ディスクの反応が遅くなる	slow
ネットワーク	パケットの 1 ビットがエラーになる	1bit
	パケットの 2 ビットがエラーになる	2bit
	CRC 符号により誤り検出する	crc
	パケットをロスする	loss
	NIC を認識しなくなる	nic
メモリ	1 ビットがエラーとなる	1bit
	2 ビットがエラーとなる	2bit
	指定したアドレスのバイトがエラーとなる	byte

3.1.1 フォルトインジェクションの種類

表 1 に現在検討中の D-Cloud において注入できるフォルトの対象デバイスと種類を示す．ハードディスク，ネットワークデバイス，メモリを対象デバイスとしており，対象デバイスとエラー内容をシナリオに記述することによりフォルトインジェクションを行う．

3.1.2 フォルトインジェクションの再現性

故障による耐障害機能のテストでは再現性が重要である．再現性があればエラーの原因の特定が容易となり，エラー修正に大きく寄与することができる．特にハードウェア故障においては，多くの場合が影響を予期できず，再現不可能である．したがって再現を試みたとしてもその時間的，経済的コストは大きなものになる．しかし VM によって故障のエミュレーションを行うことにより定められた時間，場所に故障を発生させることが可能となり，実際に故障を発生させる場合と比べコストと手間を大幅に減少させることができる．

3.2 Eucalyptus を用いた計算資源管理

D-Cloud では Eucalyptus を用いて，VM の管理を行う．Eucalyptus は仮想化ツールを使用することにより，柔軟に計算資源の管理，提供を行うことができるクラウドシステム構築基盤である．

Eucalyptus の計算資源管理はゲスト OS を起動するための QEMU ノードとゲスト OS をコントロールするコントローラノードの 2 種類で構成されている．実行の流れは以下のようになる．

(1) OS イメージをコントローラノードにアップロードし，その OS イメージを登録

(2) コントローラノードが OS イメージを QEMU ノードに対して転送
(3) QEMU ノードが転送された OS イメージを起動

コントローラノードは OS イメージの登録，管理，QEMU ノードの資源管理を行う．登録された OS イメージは一覧として表示されるため，その中から任意の OS イメージを選択することができる．そしてゲスト OS の起動要求が来ると QEMU ノード全体の資源情報を得て，資源が開いているノードを選択し OS イメージの転送を行う．そのため使用者が QEMU ノードの資源を意識する必要がなく，コントローラノードにより資源管理が行われる．

3.3 D-Cloud フロントエンド

計算資源管理システムと連動しゲスト OS の管理，テスト環境の構築，テスト者とゲスト OS とのデータ転送を行うのが D-Cloud フロンドエンドである．実行する動作を以下に示す．

(1) テスト者からのテストシナリオ，テストプログラム，入力データ，実行スクリプトの受け取り
(2) 計算資源管理システム対してのゲスト OS 立ち上げ命令
(3) 起動したゲスト OS に対してテストプログラム，入力データ，実行スクリプトの配布
(4) 対象ゲスト OS に対してのフォルトインジェクションの命令
(5) 出力データ，ログ，スナップショットの管理

D-Cloud フロントエンドはまずテスト者から実行するために必要なデータを受け取る．そして利用者からの実行依頼に対してゲスト OS の制御命令をコントローラノードへ通知し，コントローラノードが QEMU ノード上のゲスト OS イメージを制御する．テスト終了後に QEMU ノードから転送された出力データ，ログ，スナップショットを保存しておき，テスト者がダウンロードできる状態にする．

テスト者はテストの結果としてログ，出力データ，スナップショットを得ることができる．エラーが確認されるようであれば，保持してあるスナップショットを取得し，そこからエラー原因を探ることができる．QEMU は任意のタイミングでメモリ情報，レジスタ情報などを取得できる．そのためスナップショットを利用してエラー直前やエラー発生時の状態に復元し，障害の原因の追及を行うことが可能である．

D-Cloud はフォルトインジェクション可能な QEMU を使用しており，ゲスト OS に対して任意のタイミングでフォルトインジェクションを実行することができる．

QEMU ノード上で起動するゲスト OS はテストを行うため環境を整えておかなければならない．テスト者によってテスト環境を構築してある OS イメージを利用する手法であれば，環境を改めて構築する必要はない．ユーザーレベルのプログラムテストであれば，特別なカーネルを使用する必要がないため，予め Eucalyptus 上で用意しておいた OS イメージを利用し，アプリケーションなどの環境を後ほどインストールする手法でプログラムテストが行える．

仮想マシンの環境構築ソフトウェアとして Rocks[5] や OREGrid[6] が存在するが，それらは OS インストールから環境構築を開始するため，D-Cloud の利用としては適当ではなく，アプリケーションのみのインストールが行える lucie[7] の利用を検討している．

カーネルレベルのプログラムテストはカーネル自体がテスト対象であるため，基本的には Eucalyptus 上で用意しておいた OS イメージを利用することができない．そのためテスト者が各自でテストを行いたいカーネルで起動する OS イメージを作成する必要がある．

4. システム構成とシナリオ記述

システム構成とシナリオは XML 形式で記述する．**図 2** にシナリオ全体の記述例を示す．テストを実行するためのシナリオの要素として以下の 4 つの要素が存在する．

- テストを行うホストのハードウェア環境設定
- テストを行うホストのソフトウェア環境設定
- フォルトインジェクションの定義
- テスト，フォルトインジェクション，ファイル転送の実行内容設定

4.1 ハードウェア環境設定タグ

ハードウェア環境設定は machineDefinition タグ内において記述する．**表 2** にタグの内容を示す．

machineDefinition タグでは環境定義の名前，起動する OS イメージ，CPU，メモリ，NIC の指定が必要である．Eucalyptus は提供するイメージごとにそれぞれ個別の ID を割り振っておりテスト者は利用したい OS イメージを選択し ID タグ内に記述する．テスト者が用意した OS イメージを使用する場合は定められた ID を記述することにより，その OS イメージを Eucalyptus にアップロードし使用する．CPU コア数，メモリ容量，NIC の個数は同一イメージにおいて可変であるため，同一 ID の OS イメージにおいて任意の値で起動することができる．

図 2 の記述例ではこのマシン定義の名前を machine-A と設定し，プロセッサコア数 1，メモリ容量 512MB，1 個の NIC で構成された id-1 のイメージが定義されている．

Vol.2009-OS-112 No.19
2009/8/6

```
<jobDescription>
        <machineDefinition>
                <machine>
                        <name>machine-A</name>
                        <cpu>1</cpu>
                        <mem unit="MB">512</mem>
                        <nic>1</nic>
                        <eucaid>id-1</eucaid>
                </machine>
        </machineDefinition>
        <systemDefinition>
                <system>
                        <name>system-A</name>
                        <host>
                                <hostname>hostA</hocstname>
                                <machine>machine-A</machine>
                                <config>configA</config>
                        </host>
                </system>
        </systemDefinition>
        <injectionDefinition>
                <injection>
                        <name>injection-A</name>
                        <fault>
                                <location>hdd</location>
                                <target>sda</target>
                                <kind>badblock</kind>
                                <start>all</start>
                                <end>all</end>
                                <times>permanent</times>
                                <percentage>100</percentage>
                        </fault>
                </injection>
        </injectionDefinition>
        <testDescription>
                <run>
                        <name>test-1</name>
                        <system>system-A</system>
                        <script>
                                <putFile>sample1.tar</putFile>
                                <exec id="test1">test1.sh</exec>
                                <when after="test1" delay="0min"><doSnapshot>snap1</doSnapshot></when>
                                <when after="test1" delay="2min"><doInjection>injection-A</doInjection></when>
                                <when after="test1" delay="3min"><doSnapshot>snap2</doSnapshot></when>
                                <when after="test1" delay="10min"><getSnapshot/></when>
                                <when after="test1" delay="10min"><getFile>output</getFile></when>
                                <when after="test1" delay="10min"><halt/></when>
                        </script>
                </run>
        </testDescription>
</jobDefinition>
```

図 2 シナリオ記述例

表 2 machineDefinition タグ

タグ名	内容
machine	環境の設定
name	タグの名前
cpu	CPU 数の指定
mem	メモリのサイズ指定
nic	NIC の個数の指定
eucaid	OS イメージの ID

表 3 systemDefinition タグ

タグ名	内容
system	環境の設定
name	タグの名前
host	ホスト毎の設定
hostname	ホストの名前
machine	ハードウェア環境の指定
config	環境構築の設定ファイル

4.2 ソフトウェア環境設定タグ

ソフトウェア環境設定は systemDefinition タグ内において記述する．**表 3** にタグの内容を示す．

systemDefinition タグでは環境定義の名前，ホスト名，machineDefinition タグ内で指定したマシン名，lucie の設定ファイルを指定する．Eucalyptus 上の OS イメージを使用する場合は，テストプログラムを動作させるための環境を構築する必要がある．そのため追加インストールするアプリケーションの種類などをあらかじめテスト者が定めておき，config ファイルとしてアップロードを行う．テスト者が用意した OS イメージを変更せずに使用する場合は config を空行にしておくことで，追加インストールが行われないで起動する．また machineDefinition で定義したタグの名前を記述し，ハードウェア環境の設定を引き継ぐ．

図 2 の記述例ではシステム全体の名前を system-A と定め，machine-A で設定したハードウェア環境のマシンがホスト名 host-A として起動する．その後 lucie により configA の設定ファイルを用いてテストプログラムが動作するための環境が構築される．

図 3 に分散システムの記述例として HA サーバの構成例とシナリオ記述例を示す．HA サーバとはウェブサーバの負荷分散を目的としたサーバシステムであり，ロードバランサが HTTP リクエストをラウンドロビンでバックエンドのウェブサーバに振り分ける．そしてロードバランサが稼働中にエラーで停止してしまった場合は待機しているノードが稼働を開始することができ，耐故障性の機能を備えている．

分散システムにおいてはホスト毎に設定の記述を行う．host タグを複数作成することによりホスト毎の記述内容に対応したゲスト OS が起動する．図 3 の例においては HTTP リクエストを出すクライアントとして client を起動する．clinetset というハードウェア環境設定タグにおいて clientconf という設定ファイルにより環境の構築を行う．またロードバランサとして loadbalancer0，loadbalancer1 を起動する．balancerset というハードウェア環

```
<system>
        <name>system-B</name>
        <host>
                <name>client</name>
                <machine>clientset</machine>
                <config>clientconf</config>
        </host>
        <host>
                <name>loadbalancer0</name>
                <machine>balancerset</machine>
                <config>balancerconf</config>
        </host>
        <host>
                <name>loadbalancer1</name>
                <machine>balancerset</machine>
                <config>balancerconf</config>
        </host>
        <host>
                <name>webserver0</name>
                <machine>serverset</machine>
                <config>serverconf</config>
        </host>
        <host>
                <name>webserver1</name>
                <machine>serverset</machine>
                <config>serverconf</config>
        </host>
</system>
```

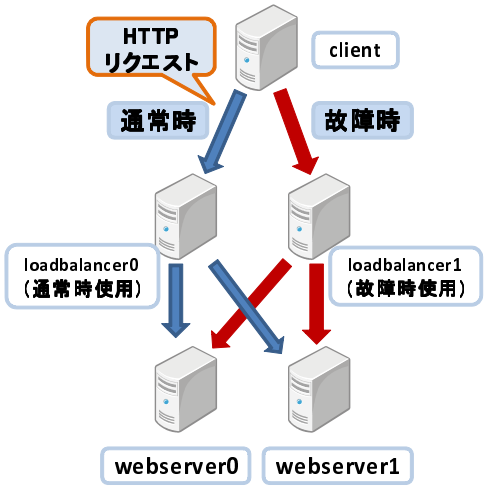

図 3 HA サーバの構成例とシナリオ記述例

境設定タグにおいて balancerconf という設定ファイルにより環境の構築を行う．またウェブサーバとして webserver0，webserver1 を起動する．serverset というハードウェア環境設定タグにおいて serverconf という設定ファイルにより環境の構築を行う．

4.3 フォルトインジェクション定義タグ

フォルトインジェクション定義は injectionDefinition タグ内において記述する．**表 4** にタグの内容を示す．

injectionDefinition タグではインジェクションの名前，デバイスの種類，対象デバイス，インジェクションの種類，範囲，回数，発生確率を指定する．範囲においてはインジェクションするデバイスの種類がハードディスク，メモリの場合は記述する必要があるが，ネットワークの場合はパケットが対象であるため記述する必要はない．回数についてはフォルトインジェクション回数指定もしくは永続的実行のどちらかを設定する．

表 4 injectDefinition タグ

タグ名	内容
injection	インジェクション内容の設定
name	インジェクションの名前
fault	インジェクション毎の設定
location	デバイスの種類の指定
target	デバイス指定
kind	インジェクションの種類
start	開始セクタ，アドレス
end	終了セクタ，アドレス
times	インジェクションの回数
percentage	インジェクションする確率

表 5 testDefinition タグ

タグ名	内容
run	各ゲスト OS で実行される内容
name	テスト全体の名前
system	テスト実行環境の指定
script	実行内容
putFile	ゲスト OS に転送するファイル
exec	ユーザーが記述したスクリプトを実行
when	動作タイミング
doInjection	定義したインジェクションを実行
doSnapshot	snapshot を作成
getSnapshot	snapshot を取得
getFile	管理ノードに転送する出力ファイル
halt	テストの終了

図 2 の記述例では injection-A という名前でハードディスクドライブ sda に対してのデータがエラーを返すようなフォルトインジェクションを定義した．start，end タグに all と入れることにより sda 内全てのデータが対象となるため sda 内全てのデータが永続的に 100%の確率でエラーを返す設定となる．

4.4 実行内容タグ

実行内容は testDescription タグ内において記述する．**表 5** にタグの内容を示す．

testDescription タグにはテスト全体の名前，使用するテスト環境，実行内容が記述される．実行内容の指示はユーザーが記述したスクリプトを指定することにより行われる．またスプリプト実行タイミングを基準にして時間を設定し，フォルトインジェクションの実行，スナップショットの作成，ファイルの転送，テストの終了のタイミングの指示を行う．

図 2 の記述例ではテスト名として test-1 と定め，テスト実行環境として systemDefinition タグで指定した system-A を使用する．実行内容は hostA 内において sample1.tar を D-Cloud フロントエンドから転送し，テスト者があらかじめ記述したスクリプト test1.sh を実行する．後ほどの時間指定の基準とするため id 属性を付属し，タグ when で使用する．test1.sh 実行後すぐに snap1 という名前でスナップショットを作成，2 分後に injectionDefinition タグで指定した injection-A を実行，3 分後に snap2 という名前でスナップショットを作成，10 分後に出力された output とスナップショットを D-Cloud フロントエンドに転送する．そしてホストをシャットダウンする．

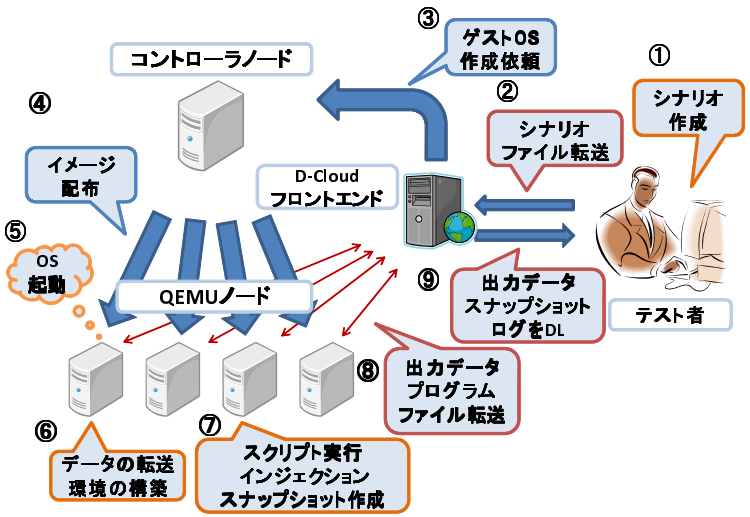

図 4 実行の様子

4.5 D-Cloud の動作

図 4 に D-Cloud の実行の流れを示す．

(1) テスト者はテストシナリオ，環境設定ファイル，実行スクリプトを記述する．
(2) 作成したシナリオ，設定ファイル，スクリプト，ファイルを D-Cloud フロントエンドにアップロードする．OS イメージを作成した場合は OS イメージも同様にアップロードする．
(3) D-Cloud フロントエンドが Eucalyptus コントローラに対してシナリオに記述されたゲスト OS の作成を依頼する．すでに Eucalyptus コントローラ上で用意されている OS イメージであればその OS イメージを利用し，テスト者がアップロードした OS イメージであれば転送を行う．
(4) Eucalyptus コントローラが利用可能な QEMU ノードを選び OS イメージを転送する．
(5) 転送された OS イメージが QEMU ノード上で起動する．
(6) 起動したゲスト OS に対しプログラムテストに必要なファイルを転送する．そしてゲスト OS がテストプログラムを実行するための環境構築を行う．
(7) テストシナリオに沿って，スクリプトの実行，フォルトインジェクションの実行，スナップショットの作成を行う．
(8) テスト終了後，出力データ，ログ，スナップショットを D-Cloud フロントエンドに転送する．
(9) テスト者が必要に応じて出力データ，スナップショット，ログをダウンロードする．

また D-Cloud は以下の 4 つのサービスをコンポーネントにより設計される．

- ポータルサービス:ユーザーに対してウェブポータルを提供する．利用可能な OS イメージの表示，データ転送のための環境を構築する．
- テスト環境構築サービス:設定ファイルに従い，テスト環境の構築を行う．machineDefinition タグ，systemDefinition タグを使用する．
- ジョブ実行サービス:シナリオに従いプログラムテストを実行する．testDescription タグ，injectionDefinition タグを使用する．
- データ管理サービス:実行ノードに対して入力データの転送，テスト者に対してログ，スナップショットなどの転送を行う．

5. 関 連 研 究

テストを複数ノードで行い，高速化，高信頼性を図る手法は多く提案されている．GridUnit[8] はソフトウェアテストを行う際に，OurGrid[9] を用いてテストの分散化を図り，グリッド上の多重ノードにおいてソフトウェアテストを行っている．テストは JUnit[10] を使用しており，JUnit の実行単位を区切り，分散化を行っている．GridUnit はソフトウェアテストを JUnit 上でのみでしか実行できず，OS が停止してしまう恐れのあるカーネルレベルでのプログラムテストを行うことができない．

ETICS[11] はテストを行う際の環境の構築，ソフトウェアコンパイル，ソフトウェアテストを自動で行い，Condor[12] を用いてグリッド上の多重ノードにおいてテストを行っている．ETICS は環境の構築，ソフトウェアコンパイルなどを自動で行い，ソフトウェアテストを行える状態まで自動的に実行されるという点は我々の研究と同じである．しかし GridUnit 同様に実 OS 上で動いているためにカーネルレベルでのテストを行うことができない．そしてスナップショットといったような OS 本体の再現性を実現することができない．

クラウド環境を用いたプログラムテスト環境として Open Solaris Test Farm がある．Open Solaris Test Farm[13] では，ウェブから仮想マシンのインスタンスを作成し，その上でテストを実行する．しかし使用するツールが限定的で任意のテスト環境の構築が行えない，分散システムのテストを想定していない点で本研究と異なる．

フォルトインジェクションを行う手法も多く提案されている．ソフトウェアによるフォルトインジェクションをソースコードを改編することにより実行する研究[14] が提案されている．しかし本研究はハードウェアによるフォルトインジェクションを目指しているため，それらの研究とは異なる．また VM を用いてインジェクションを行う研究も開始されており，QEMU を用いてハードディスクにフォルトインジェクションを行う研究[15] や FAUmachine[16] を用いて複数デバイスに対してフォルトインジェクションを行う研究[17] が存在する．しかしこれらの研究はそれぞれハードディスクのみを対象としている点，単体システムのみについて対象としている点で本研究とは異なる．

6. まとめと今後の課題

本稿では，自動的にテスト環境の構築，並列プログラムテスト，ハードウェアデバイスに対するフォルトインジェクションが行える並列分散システムテスト環境 D-Cloud の提案を行った．また Eucalyptus と QEMU を利用した全体の構造の設計，単一ホストにおけるプログラムテストと HA サーバテストにおけるシナリオの記述方法を示し，実際の実行の流れについて述べた．D-Cloud を用いることにより，XML のシナリオで記述するホストに対応した環境，異なる入力において，並列にプログラムテストを実行することが可能となる．また，ハードウェアデバイスが故障した状況のシステムの動作確認も行うことができる．これによりプログラムテストの効率化，フォルトインジェクションによる信頼性のテストを同時に行うことが可能となり，システム全体の信頼性確保を実現することができる．

今後はさらに全体構造の設計，シナリオ設計の詳細を検討し D-Cloud の実装を行う予定である．

謝辞 本研究の一部は科学技術振興事業団・戦略的創造研究推進事業 (CREST) 研究プロジェクト「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」，研究課題「省電力でディペンダブルな組込み並列システム向け計算プラットフォーム」による．

参 考 文 献

1) 神林亮，佐藤三久：仮想マシンを用いた分散システムの耐故障性評価環境の検討 . 情報処理学会第 70 回全国大会
2) Daniel Nurmi,Rich Wolski,Chris Grzegorczyk, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov:The Eucalyptus Open-source Cloud-computing System CCA-08 Cloud Computing and its Applications . http://open.eucalyptus.com/
3) QEMU. http://www.nongnu.org/qemu/download.html
4) Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/
5) Philip M. Papadopoulos , Mason J. Katz , Greg Bruno, NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters, Proceedings of the 3rd IEEE International Conference on Cluster Computing, p.258
6) 高宮安仁, 山形育平, 青木孝文, 中田秀基, 松岡聡: ORE Grid：仮想計算機を用いたグリッド実行環境の高速な配置ツール, 先進的計算基盤システムシンポジウム SACSIS2006, pp541-550
7) 高宮安仁, 真鍋篤, 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．先進的計算基盤システムシンポジウム SACSIS2003, pp365-372
8) Alexandre Duarte,Walfredo Cirne,Francisco Brasiliro,Patricia Machado:GridUnit software testing on the grid . Proceedings of the 28th international conference on Software engineering pp 779-782
9) Nazareno Andrade, Walfredo Cirne, Fransisco Brasileiro, Paulo Roisenberg:OurGrid An Approach to Easily Assemble Grids with Equitable Resource Sharing . Lecture Notes in Computer Science Job Scheduling Strategies for Parallel Processing pp61-86
10) E Gamma,K.beck:JUnit A cook's tour . Java Report pp27-38
11) Marc-Elian Begin Contact Information, Guillermo Diez-Andino Sancho, Alberto Di Meglio, Enrico Ferro, Elisabetta Ronchieri, Matteo Selmi and Marian urek: Integration and Testing Tools for Large Software Projects:ETICS. Lecture Notes in Computer Science Rapid Integration of Software Engineering Techniques pp81-97
12) Francisco J. Gonzalez-Castano, Javier Vales-Alonso, Miron Livny, Enrique Costa-Montenegro, Luis Anido-Rifon:Condor grid computing from mobile handheld devices . ACM SIGMOBILE Mobile Computing and Communications Review pp117-126
13) Open Solaris Test Farm. http://opensolaris.org/os/community/testing/testfarm
14) Seungjae Han, Shin K.G, Rosenberg H.A:DOCTOR an integrated software fault injection environment fordistributed real-time systems. Computer Performance and Dependability Symposium, 1995. Proceedings., International pp204-213
15) Siddharth Kumar Singh:Hard drive fault injection testbed using QEMU virtual machine hypervisor. http://profile.iiita.ac.in/sksinghb03/qemureport.pdf
16) FAUmachine. http://www3.informatik.uni-erlangen.de/Research/FAUmachine/
17) S.Potyra. V.Sieh. M.Dal Cin: Evaluation Fault-Tolerant System Designs using FAUmachine, EETS'07