Vol.2012-OS-122 No.19
2012/8/2

分散障害管理のためのアクターベースのスクリプトフレームワーク

菅谷 みどり[2]　岡本 悠希[1]　中田 晋平[1]　井出 真広[1]　若森 拓馬[1]　倉光 君郎[1]

概要：近年の情報システム基盤は, サービスに合わせて構成の変化を余儀なくされており, それに伴い障害管理システムも複雑化や変化への対応を柔軟に行うことが求められている. しかし, こうした要求に十分に耐えうる障害管理システムは十分に提供されていない. 本研究では, D-Script フレームワークを提案する. D-Script フレームワークは, 障害管理における柔軟性, 拡張性, スケーラビリティ, 安全性への対応を実現することを目的としている. 実現のために, 非同期分散型のアクターモデルによるタスク管理と, 同期のためのログプールを基盤とした機能を提供する. 論文では, 基本アイデアとプロトタイプによる実験の内容について述べる.

キーワード：分散システム, 障害管理, アクター, スクリプト, Konoha 言語

An Actor-based Scripting Framework for Distributed Fault Management

Midori Sugaya[2]　Yuki Okamoto[1]　Shinpei Nakata[1]　Masahiro Ide[1]　Takuma Wakamori[1]　Kimio Kuramitsu[1]

Abstract:
In these days, infrastructure of information systems has been needed to change their configuration according to changing services. For these systems, fault management systems are also needed to adopt to change, however, there are still difficult to manage these situations. This paper presents the idea and design on script-based extensible and flexible framework for fault management. The framework will present scalable and safety control scripting framework with actor and log-based architecture. Using our prototyped implementation, we will show how D-Script handles faults in a case of Web server systems.

Keywords: Distributed Systems, fault management, acotr, script, KonohaScript

1. はじめに

近年, 情報システムはユーザの多様なニーズに適合するために複雑化し, 相互に接続してサービスを提供するようになっている. こうしたシステムは, 社会基盤としてより影響力を持つものとなっているため, 障害へ対処することは, ますます重要な課題となっている.

分散システムにおける障害管理 (fault management) は, 障害検知, 要因特定, 復旧において様々な手法が提案されている [5][4][2]. しかし, 近年のシステムは, 状況変化を包含したり, アドホックな構成などが発生しうることや, 分散したサーバ上でサービスを共有するといった性質があることからも [12], 障害管理は依然として困難な課題である [11].

例えば現場では, 今でも分散システムの障害管理は, シェルスクリプトで行われているケースが多い. シェルは, 多くのサーバ系システムで採用されている Unix 系のシステムあればどこでも実行できる実行環境の利便性や, 障害管理に必要な OS のシステムのリソースへのアクセス透過性が高く, 障害管理で必要となる OS を操作するコマンドを直接利用できるメリットがある. また, シェルスクリプトからは, 障害管理で重要な同期処理についても, NFS などのネッ

[1] 横浜国立大学 工学府情報工学専攻
Yokohama National University, Kanagawa, 240—8501, Japan
[2] 横浜国立大学, 未来情報通信医療社会基盤センター (MICT)
Yokohama National University, MICT, Kanagara, 240–8501, Japan

トワークファイルシステムを利用することで透過的に分散サーバ上のデータを同期させて扱う仕組みを構築することができる. さらに, 通信についても OS 内部ではプロセス間通信, OS 間ではセキュアな SSH などの通信を組み合わせることでサーバ間の移動の透過性が実現できるなどのメリットもある. このようなメリットから, 依然としてシェルスクリプトでの障害管理は多く行われているが, 先に述べた近年の分散システムへ十分対処できるものとはいえない.

理由として, まずシェルスクリプトはシステム構成の変更やスケーラビリティに柔軟に対応することが困難であることがあげられる. 例えば, 新しいホストの追加といったシステムの構成変更のたびに, 新しいホストのためのスクリプトを記述, 追加しなくてはならない.

また, 手続き型言語であるため, オブジェクト指向言語などで可能な集合を扱うための抽象度の高い記述を柔軟に行うことが困難である. 複数サーバを扱うためには, サーバごとにスクリプトを記述し, それらを組み合わせて利用するため, システムの変更に合わせてコードが複雑化する傾向がある.

さらに, 実行が完了したことを保証する機能や, 同期機構が言語側にないため, プログラマが記述する内容に依存したモデルになりがちで, それらが不十分である場合には, さらに深刻な障害を招きかねない. こうした機能的欠如の多くは外部のライブラリやツールと連携して補うことで対処が行われる事が多いが, 統一的に扱うための手段が十分でないため, 長期的な運用の中でスクリプトが複雑化し, バグの温床になる可能性がある.

本研究では, こうした問題に対応するため, 障害管理のためのスクリプトフレームワーク D-Script を提案する. D-Script フレームワークは障害管理における (1) 柔軟性, 拡張性の実現, (2) スケーラビリティの実現, (3) 安全性の実現を行うことを目的とする. 目的の実現のために, アクターモデルをスクリプト言語に導入し, 利用可能とした. また, ログを統一的に扱うこと, また, ログを通じた同期処理を実現するための基盤として LogPool を研究/ 開発し, これらを協調させて動作させるための統一的なフレームワークを提供するものとした.

論文では, 本提案についての基礎的なアイデア, 設計, 実装について述べる. また, 基礎的な評価を行い, これらの機能が十分利用可能であることを示した.

論文の構成は以下の通りである. 第 2 節では，提案する D-Script フレームワークについて, 要求, 方針, 設計について述べる. 第 3 節では, 実装について述べる．第 4 節では，提案手法に対する実証実験を行い，第 5 節で，関連研究について述べる．最後に，第 6 節にて，まとめと今後の課題について述べる.

2. D-Script Framework

2.1 要求

近年のシステムは, 状況変化を包含する, アドホックな構成が発生しうる, 分散したサーバ上でサービスを共有するといった性質がある [12]. こうした要求に対応するためには, システムのスケールの変化, アドホックな構成の許容, 分散したサーバ上でのサービスの管理を行うための障害管理のための仕組みが必要とされている.

2.2 目的

本フレームワークの目的は, 要求に示したような変化を前提とした分散システム上で, 統一的な障害管理のためのスクリプトフレームワークを提供する事である.

具体的な設計方針としては (1) 柔軟性, 拡張性 (2) スケーラビリティ (3) 安全性の 3 つを指針とする. 実現のために, 安全で拡張性の高いスクリプト言語の利用, 柔軟な記述, 実行の安全性を保証する仕組みの導入, の 3 点が重要となる. 以下に, これらの目標を実現するための概念, アプローチを述べる.

2.3 D-Script Framework

上記の 3 つの性質を満たす D-Script フレームワークの提案においては, その名の通りスクリプト言語を用いる.

- **柔軟性, 拡張性の実現 (Script-of-Script):** スクリプトはグルー言語と呼ばれるように, 様々な外部のライブラリ機能を柔軟に追加できる利点がある. そのため, 障害対処に必要なライブラリ機能を拡張しながら, スクリプトを記述することができる. 本研究では既存の監視ツールや, OS コマンド, システムコマンドなどをつなぎあわせ, 柔軟な障害対処を行えるものとする.
- **スケーラビリティの実現:** 分散システムにおいて, システムの拡張は頻繁に発生する. こうした場合, その都度新しいシステムやホストに対してプログラムを記述するのは, 手間である. 一つのプログラムのインスタンスを利用して, 効率的に複数のサービスを管理できる事が望ましい. また, 並行して複数の処理を実行することにより, 管理の効率性を向上させる必要がある.
  本フレームワークでは, 障害対処のスクリプトの実行はアクターモデルにより実現する. アクターモデルは, 共有メモリを持たず, 非同期のメッセージパッシングを前提としたモデルである [7]. コード移動の利便性に加え, 競合やリモートプロシージャコールのブロッキングなども発生しない事から, 分散プログラムの基礎的な拡張性を向上させる. 我々はアクターモデルを用いることで, スケーラビリティや拡張性を向上させるものとした.

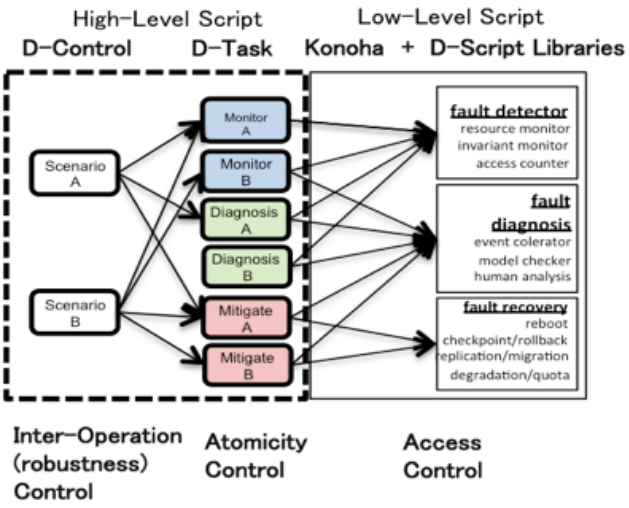

図 1 D-Script 概念モデル; Script-of-Script

- **安全性の実現:** 障害対処は, 一貫性や, アトミック性が重要である. 例えば, ある回復処理が完全に行われた事が保証されない限り, 次の処理を行うべきではない. この事から, 安全性が必要な処理については, タスクの実行は同期を取りながら行う要がある. しかしながら, アクターのプログラムのセマンティスクに同期処理を追加する事は設計上得策ではない.
  そこで我々は, ログプールを利用した同期処理を支援することでスクリプトの実行の安全性の保証を行うこととした. ログプールは, 各サーバのログデータを高速なメモリ上に統合的にプールする仕組みを提供する. インデックスサーチでメモリ上のデータに高速にアクセスできる事から, 同期性能を低下させずに, かつ, アクターの非同期実行のセマンティクスと協調した処理を可能とする.

図 1 に, D-Script フレームワークの概念モデルを示した. D-Script は 2 つのレイヤー構造から成る. 上位の言語は, 上位のビジネスプロセスなど連携したモデルを提供する. 下位の言語は, 外部のさまざまな機能を利用するための FFI(Foreign Function Interface) により提供される汎用的な機能を提供する. 上位の言語は D-Script DSL, 下位の言語は, 信頼性の高い言語機能を必要とすることから, 静的型付けを持つ Konoha スクリプト言語 [8] を用いる.

2.4 基本モデル

本節では D-Script フレームワークで用いる基本的な概念について述べる.

障害対処においては, 通常, (1) 障害検知 (fault detection), (2) 要因の特定 (fault diagnosis), (3) 障害回復 (fault recovery) の 3 つの機能が重要となる. この中で, (2) の要因の特定は主に人間が行う業務であるが, (1)(3) はスクリプトによる機械的な支援が必要な業務である. アクターを用いて, これらの対策を記述するために, D-Script フレームワークにおける要素技術の関係の定義と設計を行った. まず, アクターが実行するためのタスクを D-Task とし, これらが (1)(3) のスクリプトを実行するものとした. また, 同期を行うための機能を LogPool とし, これと連携してタスクを制御するための機能を D-Control とした. また, これらの実行環境の提供を DSE (D-Script Engine) とした. これらの基礎的なインタラクションを図 2 に示した. 以下, これらの機能の詳細について述べる.

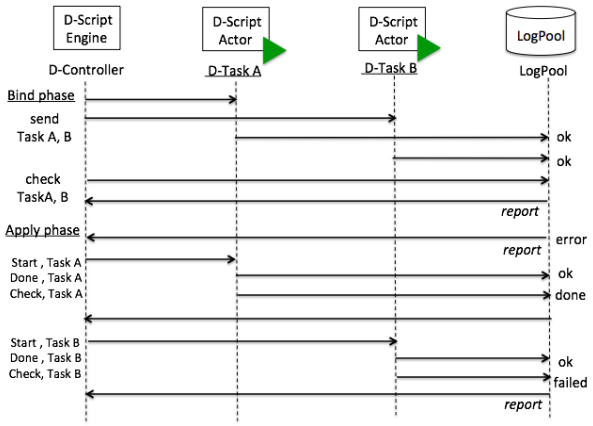

図 2 D-Control, D-Task, LogPool のインタラクション

- **D-Task** D-Task はより上位のプロセスからみた場合, 下位レベルの機能を特定するスクリプトである. アクターは, D-Task を実行する. 本フレームワークでは, D-Task の実行はアトミックに行われることを保証する. そのため, D-Task は Successed, Failed のいずれかの状態しか持たない. 外部から停止命令がきた場合, D-Task は Failed となる.
- **D-Control** D-Control は, D-Task の制御フローを指示する抽象単位である. 先に述べた通り, 本フレームワークでは, タスクのシーケンシャルな実行と, それらの実行を並列に実行するための同期を支援する仕組みを提供する. D-Task は, 実行が終了する度に, LogPool に実行結果を記録する. D-Control は LogPool から受け取った実行結果をもとに, 全ての D-Task の実行が行われた事を確認し, 実行が完了していた場合は, 制御フローに従った次のタスクもしくはタスク集合の実行を開始する. 複数タスクのうち, 一つでも失敗している場合には, アトミック性を保証するために, 続く処理全てを失敗させる. D-Control と, D-Task, LogPool のインタラクションのシーケンスを図 2 に示した.

2.5 D-Script Engine (DSE)

D-Script のフロントエンドは, アクターベースの D-Task の分散環境での実行を制御する. 実行の安全性を保証するために, スクリプトの動的/静的なチェックをフロントエンドで行うことで, 実行の安全性を保証する.

(静的チェック)： D-Task などの D-Script フレームワークで提供されるスクリプトの実行は, 2 層コミットプロトコルを用いている. まずスクリプトが実行される前に, 各サーバ上のアクターに送られる. このフェーズを bind フェーズと呼ぶ. bind フェーズでは, 各サーバ上でスクリプトの静的型付けのチェックが行われる. もし, 全てのタスクがチェックをパスすれば, 次の apply フェーズで実行される. もし, チェックが通らなければ, 実行はキャンセルされる. D-Script は, ロールベース制約をもつ静的型付け言語である Konoha スクリプト言語で記述されており, bind の際にアクセスコントロールが, 型チェックの一部として行われる仕様としている.

(動的チェック)： D-Script を用いるの利点として, 実行結果が自動的にログプールに記録されることがあげられる.D-Script のフロントエンドは,D-Task の実行が正常に終了する事を監視しており, 正常に終了した場合には, 制御フローに従って, 次のタスクを起動する. もしタスクの実行が途中で失敗した場合は, ポリシーに基づいたエラーのハンドリングを行う.

2.6 D-Script Actor

D-Script フレームワークは, 分散環境におけるスクリプトの並列実行を実現するためのアクターモデルにより実現する. D-Script アクターは, スクリプトエンジンとして, D-Task の本体となるスクリプトを送受信する機能を提供する.

アクターの安全な実行は, 先に述べた動的/静的なチェックにより行う. アクターが DSE から起動される際には, これらに対応した次の二つのオプションが利用可能である.

CHECK: 受信したスクリプトを実行開始前にチェックする. 本チェックは, シンタックス, 型, アクセス権限を静的にチェックする.

RUN: 受信したスクリプトを動的にそのターゲットとなるアクターのリソースにバインドして実行する. 実行時のエラーは, LogPool に記録する.

2.7 D-Script LogPool

アクターは非同期にメッセージをやり取りする. しかし, 並列処理を行うシステムで, 同期をとりながら再起動を行いたい場合など, これらのアクターを一つの処理に同期して動作させたい場合に対処する必要がある. こうした場合, D-Script のフロントエンドでは, ログプールにポーリングして, D-Task の実行結果を取得し, タスクの実行を確認してから次の命令に移ることを保証することで同期を実現する.

D-Scrip フレームワークでは, このようにログベースで同期を取るための仕組みとしてログプールを設計した. ログプールは, ログのインメモリストレージであり, 関連する key でのランダムアクセスが可能である. ログのストレージは一時保存のみ行い, 最低限必要な数時間経つと, 削除される. 障害解析の際に必要となる重要なログは, 削除される前に永久ストレージにコピーする. コピーの操作は, D-Script のフロントエンドで実行される.

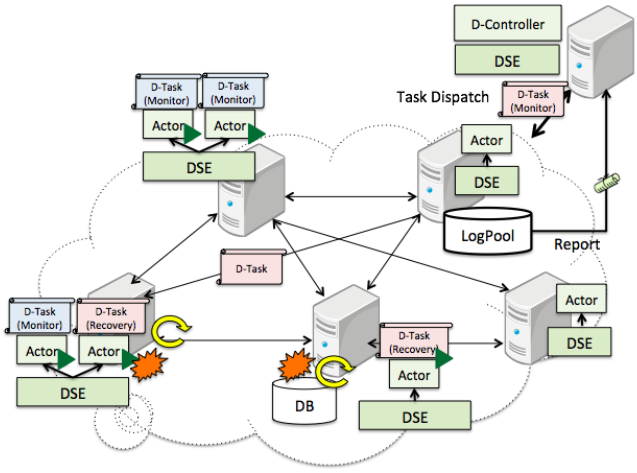

図 3 D-Script 全体アーキテクチャ

D-Script フレームワークでは, ログはタスクがスイッチするためのキーとなるイベントとなる. 一般的には, ログのフォーマットは, そのログを生成するアプリケーションにより様々で, 場合によってはエラーメッセージが自然言語で記述されていることもある. ログをイベントとして扱うために 我々は, タイムスタンプ, イベント, 自動生成された情報（ユニークなシーケンシャル番号）を含む共通のフォーマットを定義する. ログに必要となるその他の情報は JSON 形式のデータフォーマットで記述されるものとした.

3. 実装

3.1 アーキテクチャの概要

D-Script フレームワークを利用して, 障害管理システムを構築した場合のアーキテクチャの全体像を図 3 に示した. 障害管理システムの最小構成は検知と回復機能の連携であるが, これらを動的に行うために, 変更可能なシナリオに基づいてアクターを制御する D-Controller を中心とし, D-Task(Monitor) のディスパッチにより, 障害検知を行う. タスクは運用中にシナリオの変更を通じて, 動的に追加などが可能である. モニタは LogPool に統一的にログを保存する. 障害発生時には, 統一的に収集されたログをクエリにより検索しながら, 要因特定を行う. 要因特定後は, 対策シナリオをもとに D-Task (Recovery) を作成し, アクターとしてディスパッチして実行する. 回復スクリプトは, 実行をログに記録し, 同期を取りながら, 複数のサーバの回復処理を行う. これらが安全に行われることを D-Script フレームワークにより保証する.

Listing 1 Actor Request / Response Protocol

```
1  < Request >
2  {
3  "method" : {"tycheck", "eval"},
4  "context" : context−id,
5  "taskid" : task−id, // D−Task ID (識別に必要 Log)
6  "type": {"d-script", "sh"},
7  "logpool" : "192.168.0.XX",   //固定の場合は不要
8  "script": { (script) },
9  }
10 < Response >
11 {
12  "id" : task−id,
13  "status": {"done", "error"},
14  "status␣symbol" : "DSE_OK",
15  "status␣detail": "OK", // error など．message
16 }
```

3.2 DSE (D-Script Engine)

3.2.1 基本構成

D-Script の実行基盤となる D-Script Engine(DSE) は, D-Script において, 複数の要望に対応する必要がある. 例えば, 本研究では, アクターの実行前の静的チェックと, 実行後の動的チェック, また実行時の動作をセキュアに行うためのアクセスコントロール, 分散システムのコントロールなどの機能を提供する.

3.2.2 アクターの起動とチェック

DSE はアクターを起動する際には, D-Task として記述されたスクリプトを, 相手ホストにメッセージとして送信し, 送信先ホストの DSE は, D-Task を 実行時評価し，実行する. 送信先ホストの DSE は, 実行時評価の前に, スクリプトの型検査を行い, 問題があれば実行しない. この際にアクセス権限検査も行う. これらの結果, 実行が可能な場合だけスクリプトを実行する. また, 実行した結果は LogPool に送信する.

3.2.3 アクターの同期

D-Task(Recovery) に同期が必要な場合には, 次の手順にて行う. (1) D-Task は実行の開始と終了を LogPool に記録する. (2) D-Controller は, LogPool に送られたこの実行ログを見て, 成功していれば, DSE に次のリクエストを送信する. (3) DSE は D-Controller からのリクエストを受けて, 評価，実行する. DSE は，D-Task が同期処理中の場合は, 他のリクエストを受け付けないようにロックを取得する. 同期処理情報を受け取るため, D-Controller は現在実行中のフローの識別子を送信する. 以下に,D-Controller を介した Request/Response のプロトコルのフォーマットを以下に示す.

3.3 LogPool

LogPool はログの収集, 一時保存, 検索/転送の機能を提供する. 本節では, はじめに LogPool が扱うデータモデルについて述べ, LogPool の実装について述べる.

Listing 2 LogFormat

```
1 { Time : 2012−03−22−03:55:22+10:00,
2   TraceID : HTTP−REQUEST,
3   Host : localhost:1099,
4   http−status−code : 200 }
```

3.3.1 LogPool Data Model

LogPool はアプリケーションが発行したログをアプリケーションから発行されたログを最初に受け付けた時間 (Time), あるログポイントを一意に認識できる ID(TraceID), ログの出力したホスト, ポートの情報を付与した Key-Valke 形式で扱う.

3.3.2 LogPoolEngine

Logpool はロガーとしての役割とログを検索する DB としての役割を持つ. Logpool は監視対象のアプリケーションに組み込みログを出力する TraceEngine, アプリケーションからログを受け取り, メモリ上にログを保存する PoolEngine, メモリ上に保存されているログを検索し, 結果をクライアントにストリームとして返す QueryEngine の 3 つから構成されている.

TraceEngine は syslog など既存のロギングライブラリと同様にアプリケーションに組み込む形で利用する. TraceEngine は静的に型付けされた Key-Value 形式で表現されたデータを Logpool に転送する. PoolEngine は TraceEngine から送られてきたログデータについて, それぞれ TraceID ごとに最新のログをメモリ上に保存する. また, システム運用に際し, 重要なログについてメモリからファイルに出力し, 永続化を行う. 最後に, QueryEngine はクライアントから送られてきた LogPool 専用のクエリ言語を用いてメモリ上のログを検索し, 結果をクライアントに返すものとした.

4. 評価

D-Script のプロトタイプ実装の評価を行った. 以下に詳細を述べる.

4.1 実験評価環境

実験評価環境として, 我々が開発し実サービス提供を行っている Web オンラインプログラミング環境 [1] を用いた. 本環境は, 横浜国立大学, 早稲田大学のプログラミング演習の授業, 自習に利用されている. サービスの提供を行っているシステム構成を図 4 に示した. 本サービスでは, 予期せぬアクセス負荷により, 過去に Web サーバ, DB サーバのアプリケーションが応答しない障害が発生した. 本実験では, 実際の障害時と同じ状況を再現し, それに対して, 障害対策のスクリプトを実行し, その結果を示すものとした.

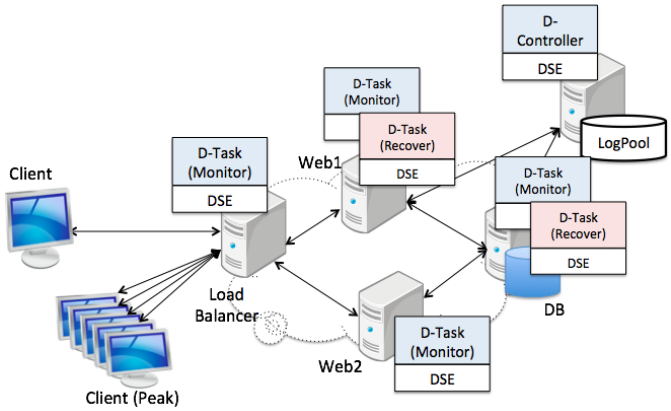

図 4 システム構成

表 1 マシン性能

Host	CPU	Mem	OS
D-Controller	Intel Core i7	4GB	MacOS X (10.7.2)
Web1	AMD Phenom II X6	16GB	Ubutu 11.10
Web2	AMD Athlon Dual Core	16GB	Ubutu 11.10
DB	Intel(R) Core2 Quad Core	4GB	Ubutu 11.10

4.2 システム構成

次にシステム構成について述べる. クライアントは常時アクセス可能な状態で Web サーバを利用する. サーバ側ではクライアントからのアクセスは, ロードバランサで受け, 負荷に応じて Web サーバ 1, Web サーバ 2 に均等に振り分ける. Web サーバは Apache を用いて, そこに Web サービスおよび, プログラムのバックエンドとなるエンジンを実行する. サービスは, クライアントの Web ブラウザの JavaScript 側に言語のフロントエンド機能を持たせ, コード生成と実行をサーバで行うことで負荷を減らす構成となっている.

サービス利用の際に使用されたデータの保存用の DB サーバはバックエンドに配置した. DB サーバは, 2 つの Web それぞれのデータを統一的に扱うために, NFS でディスクを共有している. 表 1 に, 使用したマシンの性能をまとめた.

4.3 D-Script 実験

以下に D-Script の実験について, 基礎的なタスクの実行時間, 障害時のインタラクションを含むタスクの実行について述べる.

4.3.1 実行ログの LogPool への記録

タスクは, D-Controller から, DSE/Actor に送信され, そこで実行される. D-Task を実行した際に, ログプールに記録されるデータを List 3 に示す. TraceID は, LQL のクエリに利用する ID となっており, 文字数等のメタデータとして追加することで, 検索時に役立てることを想定している. また, 時間はマイクロ秒で取得している. Context id は, 現在は 0 を代入している. リスト 3 では 2 つのログを示している

Listing 3 ステータスのログ

```
1 [server_read_callback:45]
2 'TraceID': 'task' 4, 'time': '199265921' 7, 1, '
      context': '0' 'status': 'starting'
3 7, 4, 'TraceID': 'task' 4, 'time': '199265921' 7, 1, '
      context': '0' 'status': 'starting'
4 ...
5 [server_read_callback:45]
6 'TraceID': 'task' 4, 'time': '199266026' 7, 1, '
      context': '0' 'status': 'done'
7 7, 4, 'TraceID': 'task' 4, 'time': '199266026' 7, 1, '
      context': '0' 'status': 'done'
```

が,1 つ目の D-Task は ‘starting’ でタスクが開始した事を示している. 後の 105ms 後のログでは ‘done’ でタスクの終了を示している.

4.3.2 D-Task の実行時間

今回, システム上で, 実際に D-Controller から D-Task をリモートホストに送信し, リモートホスト上の DSE/Actor 上で D-Task が起動する, という基礎的な時間を計測した. この結果, 10 回の平均が 69.28(ms) であった. アクターを 10 個に増やし, 同様の実験を行ったところ, 平均は 128.74ms であった. タスク数が 10 倍になっているのに対して, 性能低下は 1/2 に留まっていることからタスク数に比例して性能低下が生じるという事にはなっていない事が分かる. しかし, 現状のアクターの実装は, 1 つのアクターが 1 つのポートを利用するなど, 必ずしも効率の良い実装となっていない. この点については, 今後改善の必要がある.

4.3.3 障害時のインタラクションを含む処理

実システムで D-Script を利用する事を想定して実験を行った. 実際の障害時には, Web1, DB サーバがハングアップする障害が生じた. この場合, システムのハードウエアリセットしか手段がなくなってしまい, サービス停止時間が延長してしまう.

Web サーバ, DB サーバのアプリケーションの応答が無い場合の緊急対処として, 我々は Web アプリケーション, DB サーバアプリケーションを再起動する処置を D-Task により行うものとした. この際, 両者を同時に実行するのではなく, Web サーバが正しく再起動した事を確認した上で, DB サーバを再起動する, という安全性のための LogPool を介した同期の手順を踏むものとする. List 4 に Web サーバに送られる D-Task の簡略化したサンプルを示した. D-Task は D-Controller から Web1 に送られ, DSE/Actor 上から start() メソッドで起動後, 処理を行い, ログに記録を残して終了する. D-Controller からのリスタート命令があった場合は, apache (httpd) をリスタートし, その結果をログに記録する. 実際には, ここまで抽象度の高い記述での実現はまだ十分ではないが, 機能的には同等のスクリプトを用いて実験を行った.

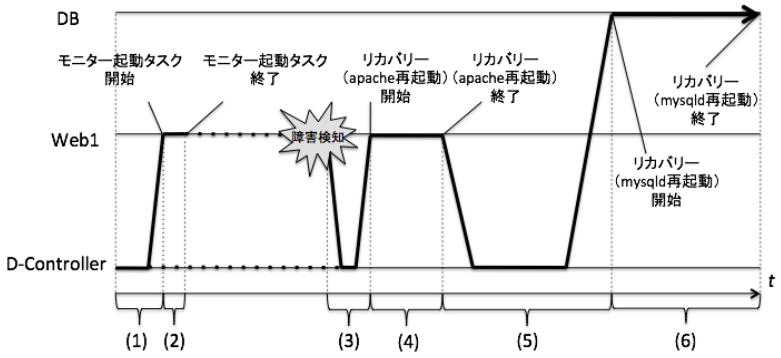

図 5 障害時の D-Script のタスクのフロー

Listing 4 障害対応時の D-Task 例

```
1  act {
2  @Resource (WebServer)
3          task RestartWebServer() {
4          import dscript.httpd.*;
5          mng = new HTTPDManager();
6          mng.start();
7          }
8          LogMonitor.sync();
9          mng.restart();
10 }
```

表 2 D-Task の障害対応の時間 (平均)

計測区間	時間 (ms)
(1)	1130.73
(2)	0.582
(3)	11.66
(4)	1202.53
(5)	7635.57
(6)	5582.30

障害対策を行う際の, タスクのシーケンスを図 5 に示した. また, それぞれの実行を数回繰り返した時間の平均を表 2 に示した. 尚, ホスト間の時間の同期は NTP で取り, データは LogPool に集約したものを用いた.

(1) から (6) は D-Controller, Web1, DB サーバ上の D-Task が障害対応時に行った対応にかかった時間を示している. (1)-(6) までの詳細を以下に述べる. (1) は DSE 上の D-Controller が起動してから, D-Task を Web1 に送付し, Web1 上では送付されたモニタ起動用 D-Task によりモニタが起動された時間を示している. (2) は, D-Task がモニタを起動してから, 終了するまでの時間である. (3) は Web1 上の CPU 利用率の上昇を検知し, LogPool に敷居値を超えた事が記録された時間を起点として,Logpool が D-Controller に障害を通知し, D-Controller がリカバリのための D-Task を Web1 に送付し, それが実行を開始する時間を示している. (4) は, Web1 のリカバリタスク (apache の再起動) の起動から終了までの時間である. 時間 (5) は, Web1 のリカバリタスクが終了したことが LogPool に記録され, Logpool が D-Controller に通知をし, DB 上で実行するリカバリタスクを DB に送付する. DB 上でスクリプトが起動するまでの時間. (6) mysql が起動してから, リカバリが終了するまでの時間とした.

結果からも明らかなように, Web1 でのリカバリタスクが終了してから, 次の DB ホスト上でのリカバリタスクの実行である計測区間 (5) に, 他の計測時間より長い 7 秒ほどの時間がかかっている. これは, (5) の処理では, Web1 から D-Controller への同期が LogPool を介して行われていることによる. LogPool は, ログのチェックを数秒おきに行っており, ログに'done' が記録されてから, 最悪数秒待たされた後に, D-Controller に結果が送られる. また, スクリプト自体の処理の問題もある. apache のリカバリは http のコネクションが途切れる信頼性の問題があるが, そもそも TCP/IP は再送が前提なので, 再起動時のコネクション切れは, それほど大きな問題にならない. これに対して, DB は再起動時にメモリ上のバッファのシンクや一貫性の保持のための確認などが発生するため, 再起動までの処理に時間がかかる結果となったといえる.

このように, 対象となる処理に応じてスクリプトの実行までの時間は変化するが, 基礎的な D-Controller から起動された D-Task が, リモートホスト上の DSE/Actor で起動するまでの時間は, 約 50ms から 1 秒程度との結果となり, シェルを使ってリモートログインしてプログラムを実行するよりも, 安全で早くかつ, 効率が良いといえる. しかしながら, 現状まだプログラム上の改善の余地があるため, 継続的に開発を進める予定である.

5. 関連研究

分散システムの拡大と複雑化により, フォルトやエラー伝搬の管理はますます困難となっている. そのため, 分散システムにおけるフォルト管理はより重要性を増していると同時に, 様々な研究が示されている.

分散フォルト管理システムの中でも PeerReview[5], Friday[4], PinPoint[2] などのシステムは静的なフォルト管理システムである. これらのシステムでは, ある特定のフォルトは設計時にその対処方法を決める. D-Script は, フォルトやエラー伝搬の管理は, 静的なフォルト検知システムや回復システムを用いながらも, 運用時に拡張を追加しながら, 動的に行う.

分散システムの管理という観点では, 数多くの言語ベースのアプローチ; ポリシーベース [9], アクターベース [3], SMNP[6], CIM-SPL[10] などのコントロールプロトコルベースのアプローチが提案されている. こうしたシステムは, 必ずしもフォルト管理のために設計されていないが, フォルト検知, 動作中リモートシステムのコンフィグレーションなどを行うことができる. 一般に, これらの提案されている言語は, ドメインスペシフィックな問題をモデル化することにより設計されている. 例えば, D-SMART のポリシー言語は [9], 検知と回復動作の構造から導かれたルールベースのポリシー言語である. それゆえ, システムの動作は, このモデリングの設計に強く依存している.

これに対して, D-Script フレームワークは, システム設計/実装の不完全と運用時にスクリプトにより徐々にプロセスを改善することの橋渡しをするための仕組みを提供するものであるといえる.

6. 結論

本論文では, 障害管理の課題に対応し, 動的かつ柔軟に拡張が可能で安全に実行できる D-Script フレームワークを提案した. D-Script フレームワークは (1) 拡張性, 柔軟性 (2) スケーラビリティ (3) 安全性を提供する障害管理のためのスクリプトフレームワークを提供する. 評価では, 基本的な操作が連携して行える事を示した. 今後さらに実装の改善を重ね, より障害への対応力を高める予定である.

謝辞 本研究は,JST/CREST「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」領域の一部として行われた.

参考文献

[1] Aspen. Web Programming Environment. `http://konoha.ubicg.ynu.ac.jp/aspen/`.

[2] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, O Fox, and Eric Brewer. Pinpoint: Problem determination in large dynamic internet services. In *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, pages 595–604, 2002.

[3] John Field and Carlos A. Varela. Transactors: A programming model for maintaining globally consistent distributed state in unreliable environments. *SIGPLAN Not.*, 40:195–208, January 2005.

[4] Dennis Geels, Gautam Altekar, Petros Maniatis, Timothy Roscoe, and Ion Stoica. Friday: Global comprehension for distributed replay. In *IN NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION (NSDI*, 2007.

[5] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: practical accountability for distributed systems. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 175–188, New York, NY, USA, 2007. ACM.

[6] Elias Procpio Duarte Jr. and Luis Carlos Erpen De Bona. A dependable snmp-based tool for distributed network management. In *DSN*, pages 279–284. IEEE Computer Society, 2002.

[7] Rajesh K. Karmani, Amin Shali, and Gul Agha. Actor frameworks for the jvm platform: a comparative analysis. In *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java*, PPPJ '09, pages 11–20, New York, NY, USA, 2009. ACM.

[8] Kimio Kuramitsu. Konoha - implementing a static scripting language with dynamic behaviors. In *Workshop on Self-sustaining Systems (S3) ACM*. ACM Press, 2010.

[9] Hanan L. Lutfiyya, Michael A. Bauer, Andrew D. Marshall, and David K. Stokes. Fault management in distributed systems: A policy-driven approach. *J. Netw. Syst. Manage.*, 8(4):499–525, December 2000.

[10] Li Pan, Jorge Lobo, and Seraphin Calo. Extending the cim-spl policy language with rbac for distributed management systems in the wbem infrastructure. In *Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, IM'09, pages 145–148, Piscataway, NJ, USA, 2009. IEEE Press.

[11] Wenchao Zhou. Fault management in distributed systems. In *Technical Report (CIS) in Universirty of Pensylvania*, January 2010.

[12] 所眞理雄. オープンシステムサイエンス. In C. Shannon and J. McCarthy, editors, *http://www.sonycsl.co.jp/message/opensystemsscience.html*, pages 43–98. Princeton University Press, 1956.