Vol.2010-OS-115 No.19
2010/8/4

仮想マシンとSpecCデバイスモデルを統合したデバイス故障エミュレータの実現

小泉仁志[†1] 坂西隆之[†1] 塙敏博[†1,†2]
佐藤三久[†1,†2] 三浦信一[†2]
石井忠俊[†3] 高見澤秀久[†4]

仮想マシンを用いたフォルトインジェクション手法は，ハードウェアのエミュレータ部分に故障を注入することで，OSを含めたシステムの耐故障性テストを行うことができる．我々はこれまでに仮想マシン環境QEMUを元にフォルトインジェクション機能を追加したFaultVMを提案してきた．本研究では，FaultVMに対して新たに設計されたデバイスを容易に追加するため，システム記述言語SpecCを用いてデバイスモデルを記述し，そこから生成されるシミュレータとFaultVMとを統合したFaultVM-SpecCを提案する．これによって，仮想マシンと協調動作する柔軟なフォルトインジェクション機能を実現する

Custormizing Virtual Machine with Fault Injector by Integrating with SpecC Device Model

Hitoshi Koizumi,[†1] Takayuki Banzai,[†1]
Toshihiro Hanawa,[†1,†2] Mitsuhisa Sato,[†1,†2]
Shin'ichi Miura,[†2] Tadatoshi Ishii[†3]
and Hidehisa Takamizawa [†4]

In this paper, we present the customization facility of virtual machine with fault injection function, FaultVM, to integrate new device models written in SpecC into FaultVM. We have been proposing FaultVM which employs the fault injection facility based on the virtual machine environment QEMU. Fault injection facility using virtual machine technology allows to verify fault tolerance function in high-availability systems by injecting the fault into the hardware emulator of a virtual machine. We adopt SpecC, the system description language, to describe the behavior of devices. The simulator generated from the description in SpecC is linked and integrated into FaultVM. It also makes the definition and injection of fault in the devices flexible.

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，ソフトウェア規模が増大するに伴ない，それらの信頼性の検証は複雑化かつ増大し、十分な検証を行えないことが多い．一方，高信頼性が必要とされる多くのシステムでは，ソフトウェアの不具合によるシステム全体の障害を防止するだけでなく，ハードウェアの障害にも対処する必要があり，資源の冗長化などによる耐故障機能を備えている．このようなディペンダブルシステムをテストする場合には，ハードウェアデバイスの故障の際にシステムが正常に動作を継続できるかどうかを検証する必要がある．しかし，ハードウェアに対する耐故障性を検証するために，一部を意図的に故障させることや，非現実的な負荷を与えることなど，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を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，分散並列システムの信頼性をテストする環境として，さらに，クラウド環境を利用して，シナリオに基づいたテストの自動化を実現するD-Cloudを提案し，フォルトインジェクションを含むテストをシナリオベースで実現できることを示してきた[1),2)]．

本研究では，FaultVMにあらかじめ用意されたデバイスだけでなく，新たに設計されたデバイスについても容易にフォルトインジェクションによるテストを行うことができるように，仮想マシンとSpecCデバイスモデルを統合したデバイス故障エミュレータを提案する．システム記述言語の一つであるSpecC言語により記述された新たなデバイスをフォルトインジェクション可能な仮想マシン環境FaultVMと統合したFaultVM-SpecCとして実

†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
†3 株式会社インターデザイン・テクノロジー
InterDesign Technologies, Inc.
†4 東芝ソリューション株式会社
Toshiba Solutions Corporation

現することにより，新たに記述されたデバイスに対して，仮想マシンと協調動作させ，柔軟にフォルトインジェクションを行うことが可能になる．

本稿の構成は以下の通りである．まず，2 章において本提案の関連研究について述べる．3 章において FaultVM-SpecC の設計について述べ 4 章において FaultVM と SpecC の統合について述べ，5 章で faultVM-SpecC を用いたフォルトインジェクション手法の提案をした後, 6 章でまとめと今後の課題について述べる．

2. 関 連 研 究

24 時間 365 日継続してサービスを提供し続ける必要があるシステムの中には, その停止が多くの人的・経済的な損失をもたらすものもある．このようなシステムは障害発生によるサービスの停止が起こらないように設計しなければならない．しかし, いくら部品の品質を向上させても物理的な故障は避けられないものである．そのため, ある程度の物理的故障が発生したとしても, サービスを継続できるようにシステムを冗長構成にした耐故障システムが用いられる．このようなシステムにおけるテストは，実際にシステムを稼働し, 故障を発生させ, それに対する振る舞いを分析する必要がある．しかし, システムの MTBF は年単位であり, 自然に故障が発生するのを待つには時間がかかりすぎる．また，デバイスの一部に対して，故意に物理的な障害を与えるのは極めて困難である．そのため, 故障を人工的にシステムに注入する，フォルトインジェクション機能が有効になる．

2.1 フォルトインジェクション手法

ソフトウェアベースのフォルトインジェクションはアプリケーションレベルの耐故障性テストでよく用いられる．この手法はソフトウェアによりハードウェア故障をエミュレーションすることで実装される．最も基本的な方法としては, アプリケーションプログラムのコードの修正による手法が挙げられる．しかし, この手法では, ソースコードが公開されていない場合は利用できないことや, テストのための修正により実際に動作するプログラムとテストに用いるプログラムが異なる等の問題が挙げられる．この問題点を解決する手法として BOND[3], Xception[4] が提案されている．

BOND の実装では故障の注入としてシステムコールをフックし, 誤った返り値をアプリケーションに渡す手法が用いられている．これにより, 動作するアプリケーションプログラムや OS を修正することなく故障の注入を行うことができる．故障注入のタイミングの指定方法は 2 種類あり, 1 つはテストするアプリケーションが動き出してからの時間を指定する方法である．もう 1 つは専用のロガーエージェントを用いて, メモリアクセスと API 呼び出しの回数をカウントし, その回数が一定数に達したら行う方法である．後者の方法は再現性の確保を目的に実装されている．

Xception は特定のアドレスにハードウェアブレークポイントを設置し, メモリやレジスタの値を修正することで故障の注入を実現する．従って故障の注入は指定したアドレスが呼び出された時点で故障の注入が行われる．BOND と同様にアプリケーションプログラムを修正することなく故障の注入が可能である．しかし，これらの 2 つの手法は, ソフトウェアでアクセスできない領域には故障を注入できないという問題点がある．

一方，仮想マシンを用いて，ハードウェアのエミュレータに対して故障を注入する手法が提案されている．Singh らは，ハードディスクの耐故障性を測定するために QEMU 上に多種多様な故障モデルを作成し，シナリオベースでのテストを実行している[5]．また，FAUmachine[6] では仮想マシンにより様々な種類のデバイスに対して故障の注入を実現可能にしている．このように，フォルトインジェクションに仮想マシンを用いることにより，以下の利点があると考えられる．

(1) フォルトインジェクションを行う際, 場合によってはハードウェア，ソフトウェアを含む システム全体に深刻なダメージを与えテストが続行不可能になる可能性がある．しかし，その度に環境を構築するのは時間がかかる．一方, 仮想マシンはスナップショット機能やイメージファイルのバックアップを取る等の方法を用いることでスムーズにシステムをテスト開始時点の状態に戻すことができる．

(2) ハードウェアの動作がソフトウェアで記述されているため, 仮想計算機のプログラムにソフトウェア的にアクセスすることで仮想マシン上で動く OS に物理的な故障が発生した場合と同様の影響を与えられる．

2.2 FaultVM

我々はこれまでに，仮想マシンを用いたフォルトインジェクションツールとして FaultVM の提案を行っている．FaultVM は QEMU[7),8)] をベースに実装されており，仮想マシンがエミュレーションするデバイスに対してアクセスを行うことでフォルトインジェクション機能を実装している．故障の注入の方法として仮想デバイスの I/O のフックとハードウェアの状態の書き換えによる 2 通りの実装をした．前者の方法は各デバイスの I/O をフックし，誤った値をゲスト OS やデバイスに返す．また、ハードウェアの状態を書きかえることで，故障をエミュレートする方法も実装している．

しかし，FaultVM を含む仮想マシンを用いたこれらの手法では，予め実装されたデバイスにのみフォルトインジェクションが可能であり，新たに設計されたデバイスに対してテス

トを行いたい場合には，仮想マシンのソースコードを変更し，仮想マシンに合わせて対応するデバイスのシミュレータを新たに記述する必要があった．そこで，本研究では，新たにデバイスを追加する際にも仮想マシンに対する変更を不要にする，デバイス故障エミュレータを提案する．

3. FaultVM-SpecC の設計

前章で述べた通り，仮想マシンによるフォルトインジェクション手法では，OS に対して物理的な故障を注入したのと同等の効果を得られるが，既存の手法では，新たに設計されたデバイスを追加しフォルトインジェクションを実現することは仮想マシンの修正を伴うため困難であった．そこで，本研究では，フォルトインジェクション可能な仮想マシン FaultVM と，システム記述言語 SpecC で記述したデバイスモデルによるシミュレータを統合したデバイス故障エミュレータ FaultVM-SpecC を提案する．

3.1 概　　要

(1)　フォルトインジェクション機能を持つ仮想マシン FaultVM-SpecC は耐故障性システムのテストが一つの目的である．前章で述べたとおり，仮想マシンを用いることで特殊なデバイスの使用や実際のデバイスを故障させることなくデバイス故障と同等の効果をシステムに与えることができる．仮想マシンの I/O アクセスや割り込みを含むハードウェア記述を利用することで，ソフトウェアで設計されたデバイスを追加し，テストに用いることができる．これにより，物理的なデバイスを用いて耐故障性システムを動作させる前にテストを行い，実際に運用を開始してから発覚する設計ミスを減らし，製造コストを下げることができると考えられる．そこで本研究では，これまで開発されてきた，フォルトインジェクション機能を持つ仮想マシン FaultVM を拡張してデバイスモデルを追加してテストを行える機能を実装する．

(2)　新たに追加するデバイスのシミュレーション

FaultVM-SpecC では新たにデバイスの追加を行い，フォルトインジェクションを用いたシミュレーションを行う．本研究ではデバイス記述のためにシステム記述言語である SpecC を用いて行う．SpecC の今回の実装の詳細については 3.2 節で述べる．

(3)　シナリオに基づくフォルトインジェクション

あらかじめシナリオを準備し，それに沿ってフォルトインジェクションをシナリオベースで実行することで，テストプロセスの自動化や，複雑なテストパターンの生成を容易に行うことが可能になる．また，テスト中に異常が見つかった場合, その原因を探るためにテストの高い再現性確保が必要になる．この再現性を高めるためには細かなタイミングで故障の注入を行えるようにする必要がある．詳細については，5 章で述べる．

3.2 デバイスモデルの記述

組込みシステムの分野等では，新しく設計されたデバイスや特許を持ったデバイス，またプリンタなどのように機械的な動作を伴うデバイスが頻繁に用いられる．このようなデバイスを QEMU 上でエミュレーションするためには, C 言語で新たにそのデバイス向けに記述を追加する必要がある．しかし，C 言語は時間管理や並列動作，入出力などの概念がなく，本質的にデバイスの設計にむいていないため, C 言語による実装は非効率的あり，かつ, 実際のデバイスの動作を正確にシミュレーションを実現できているかを判断することが難しい．さらにフォルトインジェクション機能を実装する場合, デバイス毎に記述方法が異なるためフォルトインジェクション機能の実現にはより多くのコストが必要になると考えられる．

そこで我々は, デバイスモデルの記述のためにシステム記述言語 SpecC を導入する．さらに，SpecC[9),10)] を用いて，デバイスの種類に依らず，汎用的なフォルトインジェクション機能を実現する．

SpecC

SpecC 言語は，ハードウェアやリアルタイム処理を伴うシステム向けに，並行実行や同期処理，例外処理などを C 言語を拡張することにより記述可能にしたシステム記述言語 (SDL : System Descrition Language) である．SDL は，ハードウェア記述言語（HDL : Hardware Description Language）のようにレジスタ転送レベルのような詳細な記述は必要なく, ハードウェアの機能や仕様レベルで抽象化した記述を可能にする．特に C 言語の拡張である SpecC 言語は，記述が比較的容易であり，可読性が高いことが知られている．詳細な動作の記述には HDL が有意であるが，抽象度の高い SDL よりもシミュレーションに多くの時間を要する．OS を含むシステムテストの一部としてフォルトインジェクション機能を実現するためには，デバイスの一機能に異常が発生した場合のシステムの挙動のテストで良く, そのため HDL でなくとも、SDL での記述性で十分である．

SpecC 言語では，interface, channel, behavior という概念が追加されている．behavior 文は，個々の機能ブロックを表し，各 behavior 中には機能ブロックが持つ各操作に相当するメソッドが記述される．他の behavior との間のやり取りは channel 文を用いて記述し，各 channel は interface 文で示されるインタフェースを用いて操作される．さらに，behavior 間などで時刻の同期を行う手段として，event, wait, notify, waitfor 等が用意されている．event 文でイベントを定義しておき wait 文で，当該 behavior はそのイベントを待ってブ

ロックされる．別の behavior が notify 文によりイベントを発行することでブロックが解除され，次の処理に移ることができる．一方，waitfor 文を使うと，指定した時間だけブロックされる．SpecC シミュレータでは，これらの SpecC 記述は一旦 C++言語に変換されてコンパイルされ，ランタイムライブラリとリンクされた上でシミュレータとして実現される．シミュレータ内では，ランタイムスケジューラが，上で述べた behavior 毎にスレッドを生成し，wait, notify, waitfor 文が呼び出される度に，それに応じて適切なスレッドにコンテキストが切り替わる．

仮想マシンとデバイスモデルの同期

FaultVM-SpecC では仮想マシン FaultVM と SpecC により記述されたデバイスモデルの協調動作を実現する必要がある．ここでは高見澤らにより提案された QEMU による SW/HW 協調シミュレータの構築手法[11] に基づき実装する．この手法では本研究における FaultVM-SpecC と同様に，ハードウェアモデルとして SpecC によるシミュレータ，ソフトウェア開発環境として QEMU を対象にしている．本研究では，これと同様に FaultVM を SpecC における 1 つの behavior とみなして，1 スレッドとして動作させる．SpecC シミュレータとの同期には，QEMU における Translation Block (TB) の区切りを利用する．**図 1** に，FaultVM と SpecC のシミュレータとの協調動作の仕組みを示す．

QEMU ではターゲット CPU の命令を独自の中間言語に変換してからホスト CPU 上で実行する．この中間言語への変換は，ターゲット CPU から jmp 命令を受け取るか, 一定サイズの命令を変換するまで継続して行われる．このホスト CPU 用に変換されたブロックは TB と呼ばれ, 一度変換された TB は, QEMU が備えるキャッシュに保存されて再利用される．そのため，再度ブロックを参照する際は変換に要する実行時間を削減することができ可能で，高速化を実現している．各 TB の末尾に TB の実行に要するシミュレーション時間を通知するための制御コードを追加する．シミュレーション実行エンジンはその値を元にデバイスモデル側で前回の実行時に指定された waitfor の値から実行すべきスレッドがないか調べ，ある場合にはそれを実行する．実行すべきスレッドが無ければ, 再び QEMU 側に実行が移される．また TB 実行中 I/O アクセス要求があれば当該 behavior に対応したスレッドでそれを処理し，逆にデバイスモデルで発生した割り込み通知が QEMU 側に伝えられる．

4. FaultVM と SpecC デバイスモデルの統合

本実装では FaultVM を SpecC のシミュレータ上のスレッドの 1 つとして動作させる．本章では，より詳細な実装と，フォルトインジェクション機能の実現方法について述べる．

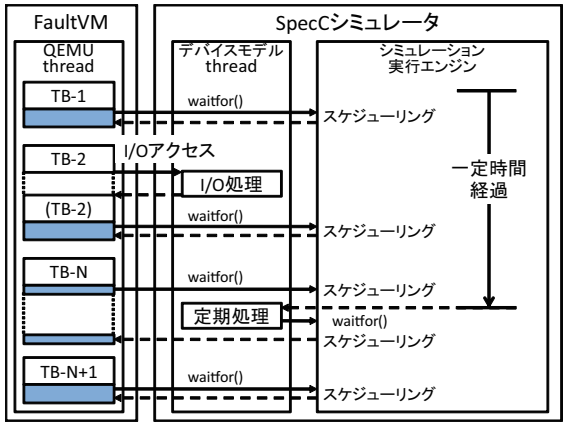

図 1 SpecC シミュレータと FaultVM の同期方法

本実装では，SpecC シミュレータとして，オープンソースで提供されている SpecC リファレンスコンパイラ (SCRC) Release V2.1 を用いる．オープンソースであるため，コンパイラやランタイムライブラリの改変を容易に行うことができる．シミュレーションのためのデバイスモデル記述には詳細な動作を記述する必要は無く，より抽象度を挙げて記述することでシミュレーション時間を短縮することができる．SCRC を用いて実行されるシミュレータはコンパイラによって SpecC デバイスモデルは一旦, C++言語に変換され, pthreads ライブラリにより，FaultVM および各 behavior はそれぞれがスレッドとして動作する．

文献 11) では Windows プラットフォーム上で，独自のシミュレーション実行エンジンを用いて実装されているが，本実装では Linux 上で実装を行い，SCRC が持つシミュレーション実行エンジンを利用して同期を実現する．

4.1 SpecC の利用

SCRC シミュレータにはスケジューラが実装されており, behavior 間の処理の待ち合わせや, パイプライン実行, 一定時間の処理の停止・再開等の並列動作はスケジューラで制御してシミュレーションを行っている．これらの並列動作は pthread で記述されているが SpecC 上の スケジューラにより同時に動作するスレッドがただ一つになるように制御されている．これは，シミュレーション時刻の管理を容易にするためであると考えられる．この SpecC によるコンテキストスイッチは waitfor や wait, notify 関数が呼び出された場合に行われる．特に waitfor によるコンテキストスイッチは本提案の実装において，デバイスと QEMU の

各スレッドの同期をとるために使用される．以下，waitfor 構文について述べる．

waitfor を呼び出すことにより指定した実行時間処理が進むまで現在実行しているスレッドの動作を停止する．SpecC は実行時間をタイムスタンプで管理している．waitfor は int 型の値を引数に取り, 現在のタイムスタンプ＋引数の小さい順にキューに入れられる．スケジューラは現在処理しているスレッドが無くなると，この実行待ちキューからスレッドを取り出し, タイムスタンプを更新する．

また, SpecC スケジューラは実行待ちのスレッドはランダム・キューにより管理するが, 乱数発生に使用するシードの値が固定されているためコンパイルするプログラムに変更がなければ何度実行しても同じ順序でスレッドが処理される．このため, シミュレーションは再現性を保つことができる．

4.2 QEMU に対する機能追加

本研究では仮想マシンとして QEMU-0.10.6 のソースコードを改変して実装を行う．qemu-0.10.6 ではターゲット CPU とホスト CPU の間の命令変換を TCG(Tiny Code Generator) と呼ばれる命令変換機構を用いて行っている．3.2 節で述べたシミュレーション時間の経過をスケジューラに通知する制御命令はこれにより生成される TB の末尾に埋め込む．

また，QEMU とデバイスモデルを協調動作させるためにはポートマップド I/O やメモリマップド I/O の入出力のインタフェースと割り込みを処理する機能を追加しなければならない．今回はターゲット CPU を x86 系として，以下の処理関数を実装する．

- メモリマップド I/O のための処理関数
- ポートマップド I/O のための処理関数
- 割り込み通知機構

I/O 処理関数について説明するために, QEMU 内部の実装を述べる．QEMU ではデバイスに対する I/O 処理関数はそのデータサイズとアドレスに応じて 2 次元配列で管理される．この 2 次元配列には例えば，ioport_write_table[3][MAX_IOPORTS] がある．1 つ目の添え字はデータサイズ (byte, word, dword) を表し，2 つ目の添え字は使用するアドレスを示している．各デバイスの I/O 処理関数は処理するデータのサイズとアドレスに応じて，この二次元配列に格納される．これにより，ゲスト OS から in/out 命令が出た場合，そのアドレスから使用するデバイスが特定でき，データサイズからその後の処理を特定できる．

QEMU とデバイスモデルとフォルトインジェクタ (FI) スレッドの協調関係と，I/O 処理関数と割り込み通知機能の動作を**図 2** に示す．この内，FI スレッドの役割については 5 章で詳しく述べる．図 2 中では QEMU とデバイスモデル間のインタフェースにおけるやり取りが点線で表されている．今回の実装ではテスト者が新たにデバイスを追加した場合, QEMU 内で追加したデバイスの I/O 処理関数として，アドレスとデータを SpecC デバイスモデル側に中継するだけの関数を実装する．デバイスモデル側では中継されてきたアドレスとデータを受け取り，設計者が定義した処理を行う．本研究ではテスト者が QEMU の内部を修正することなくデバイスシミュレーションを行えるようにするため，デバイスが利用する I/O アクセスの方法を問わず，デバイスの接続が可能になるように実装する．具体的には，QEMU 側にメモリマップド I/O とポートマップド I/O アクセスと全てのデータサイズの全ての I/O アクセスについて，デバイスモデル側とのインタフェースを実装する．このため, デバイスの設計者は SpecC 側にデータの中継先として, 全ての I/O アクセスに対して処理関数を記述する必要がある．割り込み処理は QEMU が実装している割り込みコントローラを利用する．つまり, デバイスモデルの割り込み通知は, QEMU 内部の割り込みコントローラを呼び出す関数を実装する．また，ポートアドレスや割り込み番号の指定はテスト者が明示的に記述を行う．

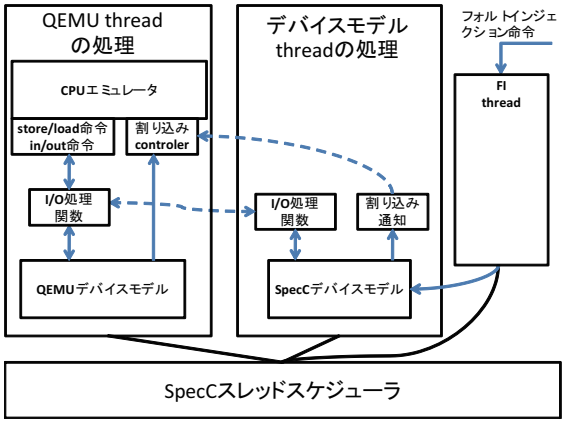

図 2 I/O 処理関数と割り込み通知機構の実装

また，具体的なデバイスモデルの記述例を**図 3** に示す．デバイス名をここでは DEV1 としている．この例で記述される behavior，channel と interface 等の意味は 3.2.1 節で解説している．CPU_IO インタフェース（1 行目）と CPU_COMM チャネル（17 行目）では，FaultVM 上の CPU とデバイス間の通信を定義している load4,store4 で記述されているのは 4 バイトのメモリマップド I/O アクセスのインタフェースであり，in4，out4 の記述は

バイトのポートマップド I/O のためのインタフェースを示している．また，デバイスモデルから QEMU への割り込み通知は CPU_INTR インタフェースで定義される．このインタフェースで定義される irq_assert 関数は QEMU 内部の割り込みコントローラと直結しており，割り込みを通知する．

5. FaultVM-SpecC によるフォルトインジェクション機能

本章では SpecC で記述されたデバイスモデルに対するフォルトインジェクション機能について提案する．異常動作を模擬するためには，各 behavior から出力される値，または入力する値を誤った値に強制的に変更することで実現する．これにより，behavior の動作不良と同等の効果が得られると考えられる．フォルトインジェクションの実現にはフォルトインジェクションを行う特殊な通信チャネルを用意する．これを使って新たに通信チャネルを定義し，behavior 間の通信に使用することにより，デバイスごとにフォルトインジェクタの実装を用意する必要がない．このため，テスト者は SpecC でデバイスモデルを記述するだけでフォルトインジェクションテストを行うことができる．

5.1 フォルトの種類

フォルトインジェクションを用いる場合, その故障の期間として永続的故障, 断続的故障, 一時的故障の分類が用いられる[12]．永続的故障はコンポーネントの損傷や疲労, 不良品の使用が原因の回復不能な故障を表している．この故障を取り除くための方法は故障部分を交換するか, 修復可能であれば修復するのみである．一時的故障は不安定な電力の供給や, 電磁エネルギー, 放射線の影響等によって現れる．この故障は一般的に短期間のものであり, 回路の状態はすぐに正常な状態に戻るが, システムにはエラーが残る可能性がある．一時的故障は永久的故障より頻繁に生じることや, その検出が遥かに難しいことが知られている．断続的故障はシステムが動作している最中, ハードウェアの状態の不安定さや変化により生じ, 活動状態と休止状態を繰り返す故障の事を指す．これはハードウェアの設計ミス等によって引き起こされる．FaultVM-SpecC では，これら 3 種類の故障モード全てのエミュレーションを可能にする．

5.2 フォルトインジェクション機構の実現

フォルトインジェクション機能を用いたテストをシナリオベースで実行可能にすることでテストを自動化することができ, 複雑なテストパターンの生成も容易に実現できるようになる．さらに, テストのタイミングの記述に SpecC におけるタイムスタンプを用いることで, SpecC ではクロック毎，FaultVM(QEMU) では TB 毎の間隔という細かい単位でタイミングを記

```
1  interface CPU_IO {
2    unsigned char  load1(unsigned int addr);
3    unsigned short load2(unsigned int addr);
4    unsigned long  load4(unsigned int addr);
5    unsigned char   in1(unsigned int addr);
6    unsigned char   in2(unsigned int addr);
7    unsigned char   in4(unsigned int addr);
8    void store1(unsigned int addr,
9                unsigned char  data);
10   void store2(unsigned int addr,
11               unsigned short data);
12   void store4(unsigned int addr,
13               unsigned long  data);
14   void out1(unsigned int addr,
15               unsigned char  data);
16   void out2(unsigned int addr,
17               unsigned short data);
18   void out4(unsigned int addr,
19               unsigned long  data);
20 };
21
22 interface CPU_INTR
23   void irq_assert(unsigned int irq);
24 };
25
26 channel CPU_COMM(CPU_IO dev1, ... )
27         implements CPU_IO {
28   ...
29   unsigned load4 (unsigned int addr) {
30     if (dev1_range_is_ok(addr)) {
31       return dev1.load4(addr);
32     }
33     ...
34   }
35   void store4 (unsigned int addr,
36                unsigned int data) {
37     if (dev1_range_is_ok(addr)) {
38       dev1.store4(addr, data);
39     }
40     ...
41   }
42 }
43
44 behavior CINTR(CPU_INTR intr)
45          implements CPU_IO {
46   ...
47   void irq_assert(unsigned int irq) {
48     /* interrupt to CPU */
49   }
50 }
51
52 behavior DEV1(CPU_INTR intr)implements CPU_IO {
53   ...
54   unsigned long load4(unsigned int addr) {
55     ... /* CPU reads from DEV1 */
56   }
57   void store4(unsigned int addr,
58               unsigned int data) {
59     ... /* CPU writes   to DEV1 */
60   }
61   void main(void) {
62     while(1) {
63       waitfor(duration);
64       intr.irq_assert(INT_DEV1);
65     }
66 };
67
68 behavior Main(void) {
69   CPU_COMM Comm(Dev1,  ...);
70   CPU_MODEL Cpu(Comm);
71   CINTR Cintr(Cpu);
72   DEV1 Dev1(Cint);
73   ...
74   int main(void) {
75     par {
76       Cpu.main();
77       Cintr.main();
78       Dev1.main();
79       ...
80     }
81     return 0;
82   }
83 };
```

図 3 デバイスモデルの例

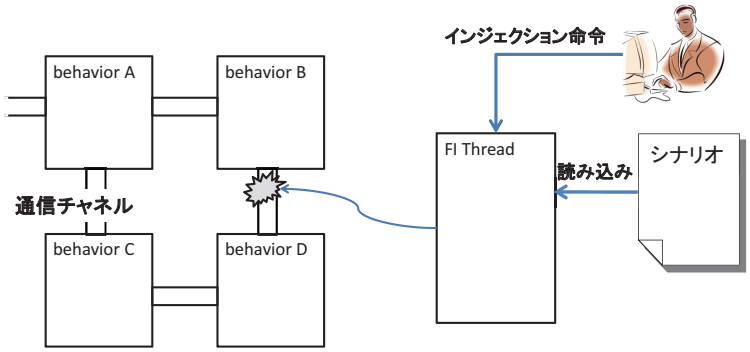

図 4 フォルトインジェクションの実行

述できるようになり, 高い再現性を確保できると考えられる．このため，FaultVM-SpecC ではフォルトインジェクタ用のスレッドを用意し FaultVM-SpecC と共に動作させ，シミュレーションクロックに合わせて故障を注入する．この実行の流れを**図 4** に示す．テスト者はフォルトインジェクション命令を発行し，フォルトインジェクタスレッドはそれを受け取るとシナリオの読み込みを開始する．その後，シナリオに従ってフォルトインジェクションテストを行う．フォルトインジェクションの指示については，起動時のコマンドライン引数だけでなく，QEMU が仮想マシンの制御に用いるコマンドにもフォルトインジェクション用のコマンドを追加して実装する．また，テストの実行タイミングを知るため，現在の SpecC のタイムスタンプを出力するコマンドを実装する．これにより，テスト者は QEMU による実行の状況をモニタしながら，対話的にフォルトインジェクションをテストすることが可能になる．

シナリオを読み込み，故障注入を指示するフォルトインジェクタは，SpecC の behavior の形で実装する．故障注入用の通信チャネルを用意し，フォルトインジェクションテストを行う場合は，デバイスモデルの記述の際に，そのチャネルを利用して記述を行う．また，フォルトインジェクション用のインタフェースとフォルトインジェクタとを接続するために SpecC コンパイラを改変し，内部で自動的に接続する．これにより，デバイスモデル開発者がフォルトインジェクタを意識せずにデバイスモデルを記述することが可能になる．

5.3 テストシナリオとの連携

5.1 節で述べた，永続的故障，一時的故障，断続的故障を模擬するためには，それぞれ以下のような情報を用いれば良い．

befavior=DEV_A channel=CH_A start=9000 val=0	befavior=DEV_A channel=CH_A start=9000 end=10000 val=1	befavior=DEV_A channel=CH_A start=9000 possibility=0.6 val=
(a)永続的故障	(b)断続的故障	(c)一時的故障

図 5 テストシナリオの記述例

- 故障を注入の場所 (behavior と channel)：テストしたい behavior 名，およびそれが使用する故障注入用の通信チャネル名を指定する．
- 故障注入の開始時刻 (start)
- 故障注入の終了時刻 (end) この値を指定しないと永続的故障になる．
- 故障注入の確率 (possibility) 永続的・断続的故障であれば，この値が 1.0 である．また，この値が 0 の場合は，故障が発生しないと考えることもできる．
- 故障時の値 (val): 故障時の値を指定する．一時的故障で値を指定しない場合には，ランダムに変化する．

シナリオの記述例**図 5** を示す．

6. まとめと今後の課題

耐故障性システムのテストを容易に実現するためのツールとして FaultVM-SpecC を提案した．今回提案した FaultVM-SpecC を用いることによりテスト者は SpecC で新たにデバイスモデルを記述し, 容易にフォルトインジェクションを用いた耐故障性テストを行うことができる．また，SpecC で記述することにより，可読性が高く，機能を高い抽象度で記述できる．また，FaultVM 側とデバイスモデル側はインタフェース以外は独立しているため，テスト者はインタフェースを呼び出す以外 FaultVM の実装を意識することなく協調動作を行うことができ，テストのセットアップの容易化を実現できる．現在，QEMU とデバイスモデルのスレッド間の同期について実装を行い，I/O 処理関数，割り込み通知の実装が完了しており，フォルトインジェクタを実装中である．

今後の課題としては以下のことが挙げられる．

(1) 複雑なデバイスモデルを用いた妥当性の検証

本提案手法を用いることで妥当なテストを行うことができるかを十分に複雑なデバイスモ

デルを作成して証明する必要がある．また, テストの実行にかかる時間を計測し, 本提案のテストにかかる時間的・経済的削減の効果を検証する必要がある．

(2)　分散システムへの適応

本研究の目的はソフトウェア規模の増大に伴い複雑化したテストのコストを下げることである．そのため, その規模とテストの複雑さから分散システムへの適用は重要な課題の一つである．しかし, パケットの受信のタイミングのずれにより, 高い再現性を持ったテストの実現は難しい．そのため, パケットの同期を行うための仕組みを考える必要がある．

(3)　スナップショット機能の活用

今回, シナリオベースで故障の実行を可能なツールを提供した．しかし, 複数のテストの実行が途中まで共通していた場合, テストを終了し, 新たに再開するのではなくテストの途中から実行を行うことでより効率的を計ることができる．

このテストのリスタートを行うためにはスナップショット機能が利用できると考えられる．実際に仮想マシンのスナップショット機能を用いてソフトウェアテストを加速させる手法が論文[13]で提案されており，今回の手法にも適用可能であると考えられる．そのためには, SpecC の状態の保存やタイムスタンプの巻き戻しなどシミュレーション実行エンジンに QEMU のスナップショット機能と協調して動作する仕組みを作る必要がある．

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) T. Hanawa, T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, and M. Sato, Large-scale software testing environment using cloud computing technology for dependable parallel and distributed systems. In Proc. on 2nd International Workshop on Software Testing in the Cloud (STITC 2010), co-located with International Conference on Software Testing Verification and Validation (ICST 2010), pages 428-433, Apr. 2010.
2) T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, H. Kimura, T. Hanawa, and M. Sato, D-Cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology. In Proc. 2nd International Symposium on Cloud Computing (Cloud 2010) in conjunction with CCGrid2010, pages 631-636, May 2010.
3) Andrea Baldini, Alfredo Benso, Silvia Chiusano, Paolo Prinetto, 'BOND': An Interposition Agents Based Fault Injector for Windows NT. dft, pp.387, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'00), 2008.
4) J. Carreira, H. Madeira, and J. Silva, Xception: A Technique for the Evaluation of Dependability in Modern Computers. IEEE Transactions on Software Engineering, 24(2), 1998.
5) Siddarth Kumar Singh et al, Hard drive fault injection testbed using QEMU virtual machine hypervisor. `http://profile.iiita.ac.in/sksingh_b03/qemureport.pdf/.`
6) S. Potyra, V. Sieh, and M. D. Cin. Evaluating fault-tolerant system designs using FAUmachine. In Proc. 2007 workshop on Engineering fault tolerant systems (EFTS '07), page 9, 2007.
7) Fabrice B, QEMU, a Fast and Portable Dynamic Translator, USENIX 2005 Annual Technical Conference, FREENIX Track.
8) QEMU, open source processor emulator. http://www.qemu.org/.
9) D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, SpecC: specification language and methodology. Springer, 2000.
10) SpecC reference compiler. http://www.cecs.uci.edu/specc/reference/.
11) 高見澤 秀久，位野木 万里，川田 秀司，石井 忠俊，吉川 寿広，荒木 大, QEMU による SW/HW 協調シミュレータの構築. 情報処理学会第 72 回全国大会 2010 年 3 月.
12) J.A.Clark and and D.K.Pradhan, Fault Injection - A Method for Validating Computer-System Dependability. IEEE Computer, vol.28, no.6, pp.47-56, June 1995.
13) M. Subkrautetal, FastFaultInjectionwithVirtual Machines. International Conference on Dependable Systems and Networks, FastAbstracts, Edinburgh, UK June 2007.