ソフトウェアテストシンポジウム 2005
JaSST'05: Japan Symposium on Software Testing 2005

Ballista method によるプログラム強靭性評価手法

田中 洋† 松本 大† 鈴木 郁子† 福永 拓男‡ 中西 恒夫‡ 福田 晃‡

†シャープ株式会社 〒632-8567 奈良県天理市櫟本町 2613-1
‡九州大学大学院システム情報科学研究院 〒816-8580 春日市春日公園 6-1
E-mail: †{hiroshi.tanaka, matsumoto-masaru, suzuki.ikuko}@sharp.co.jp

‡{297ga, tun, fukuda}@f.csce.kyushu-u.ac.jp

あらまし ソフトウェア受入れテストの一つとして強靭性自動評価手法である Ballista method を導入した．オリジナルの Ballista はオペレーティングシステムをテストするものであるが、これをユーザプログラムに適用できるように改良し，その効果，問題点，および実現性について検討した．結果，受入れテストの時間短縮やテスト操作ミスの削減等の効果を確認した．
キーワード Ballista method, 強靭性, 自動評価手法, ソフトウェア, 受入れテスト, ユーザプログラム

A New Approach to Program Robustness Test with Ballista method

Hiroshi Tanaka† Masaru Matsumoto† Ikuko Suzuki†

Takuo Fukunaga‡ Tsuneo Nakanishi‡ Akira Fukuda‡

†SHARP Corporation 2613-1 Ichinomoto-cho, Tenri, Nara, 32-8567, Japan
‡Kyushu University 6-1 Kasuga Koen, Kasuga, Fukuoka, 816-8580, Japan

Abstract We applied the Ballista Method which is known as an automatic robustness testing method to out-sourced software project. The Ballista method originally developed for testing operating systems is modified to examine user programs. The result shows that it reduces the first testing time and human errors in testing procedures.
Keyword Ballista method, robustness, automatic test, out-sourced software, user programs

1. はじめに

大規模化・複雑化したソフトウェアの開発期間を短縮するため，一策としてプログラム開発の外部委託やオープンソースの導入を図るケースが増えている．しかし，その受入れテストには多大な時間を要する．

本研究では，受入れテストの効率化を目的として，オペレーティング・システム（以下，OS）の強靭性をほぼ自動的に評価する Ballista を導入し，ユーザプログラムにも適用できるように改良を図ったので報告する．

以下，2 章ではオリジナルの Ballista について説明し，3 章ではこれをユーザプログラムに対応するための改良方法を提案し，4 章ではユーザプログラムへの適用結果を示し本研究の効果や課題を考察する．

2. Ballista とは

2.1. 概要

Ballista は Carnegie Mellon 大学の Koopman 教授のグループにより開発された POSIX 互換 OS の強靭性の試験ツールである．強靭性試験は,境界値や異常値と考えられる値（以後，この値をテストパラメータと呼ぶ）を引数としてシステムコールを呼出し，OS が正常に終了する（Pass）か，アボート（Abort）するか，無限ループに陥る（Restart）か，を確認することで行う．つまり、あるテストパラメータでシステムコールを呼出して正常に処理されることを確認するものではなく，予想外の値を引数として呼出されても，ハングアップしたりせず正しくエラーとして終了することを確認するのものである．

このように，Ballista は返り値を期待値と比較し判定するものではないので，テストとしては十分とは言えない．しかし，期待値を設計する必要がなく，関数をブラックボックスとしてテストできるので，プログラムがテストするに十分な完成度であるかを大雑把に把握する受入れ検査には適している．

2.2. Ballista の構成

Ballista は C，C++，Perl で実装されており，テスト

制御部とテストユニット，テストデータの 3 つからなる．

2.2.1. テスト制御部

テストユニット(後述)の生成，テストユニットの実行，テスト結果の出力，テストユニットの削除等の機能があり，一連の強靭性テストの制御を行う．

2.2.2. テストユニット

テスト対象関数毎にテスト制御部が生成するライブラリであり，Ballista 実行時に動的ライブラリとしてロードされる．テストユニットには以下のものが含まれている．

・テストドライバ：

テスト実行時にテスト対象関数を呼出す役割で，テスト制御部によって自動生成される．

・テストパラメータ：

テストする関数を呼出す時に引数にセットされる値．テストパラメータセット(後述)から，テスト対象関数の引数の型に対応したものが設定される．

・システムライブラリ：

UNIX のシステムライブラリである．

2.2.3. テスト制御用データ

テストすべき関数やそれにあたるべきデータなどでテストユニットを生成する時に参照される．

・callTable

テストするべき関数の一覧表であり，関数の返り値の型，引数の型等の情報が関数１つにつき１行で記述されている．オリジナルでは，POSIX のシステムコールの情報が用意されている．

・テストパラメータセット

テストを実行する際に、関数の引数の値として設定されるデータのセットである．引数の型別に，境界値や異常値などのテスト実行用の値が用意されている．基本データ型のみ用意されている．

2.3. Ballista の動作フロー

テスト制御部の処理を中心に Ballista の動作フローを以下に説明する．

1. テスト制御部が，callTable からテストするべき関数の情報を順次（1 行づつ）読み込む
2. テスト制御部が，テスト対象の関数を呼出すためのテストドライバを生成する
3. テスト制御部が，テスト対象関数の引数の型に対応したテストパラメータをテストパラメータセットから取得する
4. テスト制御部が，3 のテストドライバ，4 のテストパラメータ，システムライブラリを一つのテストユニットとして生成する
5. テスト制御部が，4 のテストユニットを動的ライブラリとしてロードする
6. テスト制御部が，テストドライバを実行する．この時，テストする関数の引数にはテストパラメータが動的に与えられる
7. テストパラメータの数だけ 6 を繰

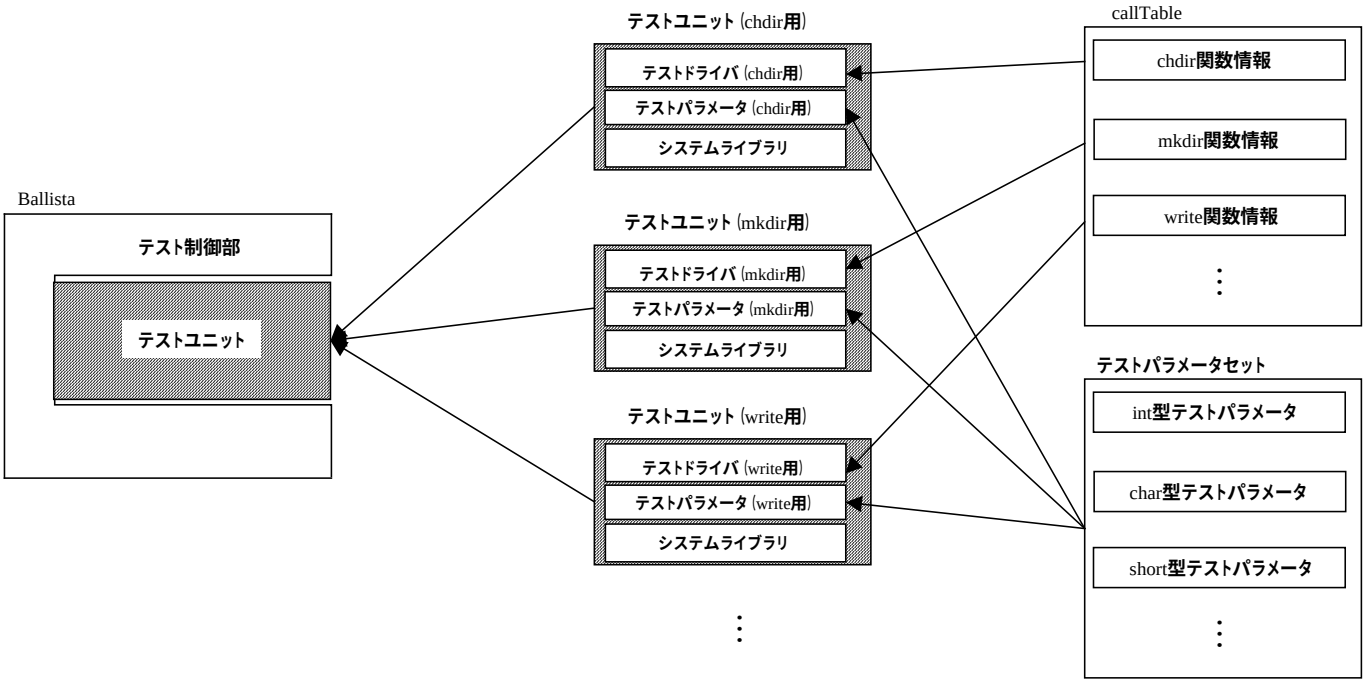

図 1　Ballista 動作フロー

返す

8. callTable 内の全テスト対象関数について１～７を繰返す

以上の動作フローを図１に示す.

3. Ballista の改良

Ballista はテストすべき関数の引数に複数のテストパラメータを動的に変化させ自動的に与える仕組みである. この仕組みを維持しつつ Ballista を改良して, ユーザプログラムの強靭性評価への適用を試みた.

3.1. ユーザ関数への対応

Ballista でテストする関数はシステムコールである. これをユーザプログラムの強靭性評価に適用するには, 以下の改良と対応が必要となる.

3.1.1. callTable の自動生成

Ballista は callTable を基にテストを自動実行するが、オリジナルの Ballista では, テスト対象関数をシステムコールに固定している. しかし, ユーザ関数はシステムコールと違い, 様々な仕様で設計される. そこで, ユーザ関数の関数定義からテストに必要な関数情報を抽出して callTable を自動生成するスクリプトを開発した.

また, callTable の引数の型と関数の返り値の型に情報は, typedef された型を基本型にマッピングしている. これにより, オリジナルの Ballista のテストパラメータセットをそのまま用いることができる.

3.1.2. ユーザ関数ライブラリ

オリジナルの Ballista はシステムライブラリをテストユニットに含み, システムコール関数を呼出す. ユーザ関数についてもライブラリから呼出すため, ユーザ関数を一つのライブラリ化し, テストユニット生成の時にはこれをリンクするよう変更した.

3.1.3. ユーザ定義ヘッダファイル

オリジナルの Ballista のテストドライバは, テストする関数の定義に標準ヘッダファイルをインクルードしている. これを, ユーザ定義ヘッダファイルをインクルードするように変更した.

3.1.4. コンパイラによる関数名エラーへの対応

テストドライバは C++で自動生成されるのでオリジナルの Ballista ではシステムコールを C++の名前で

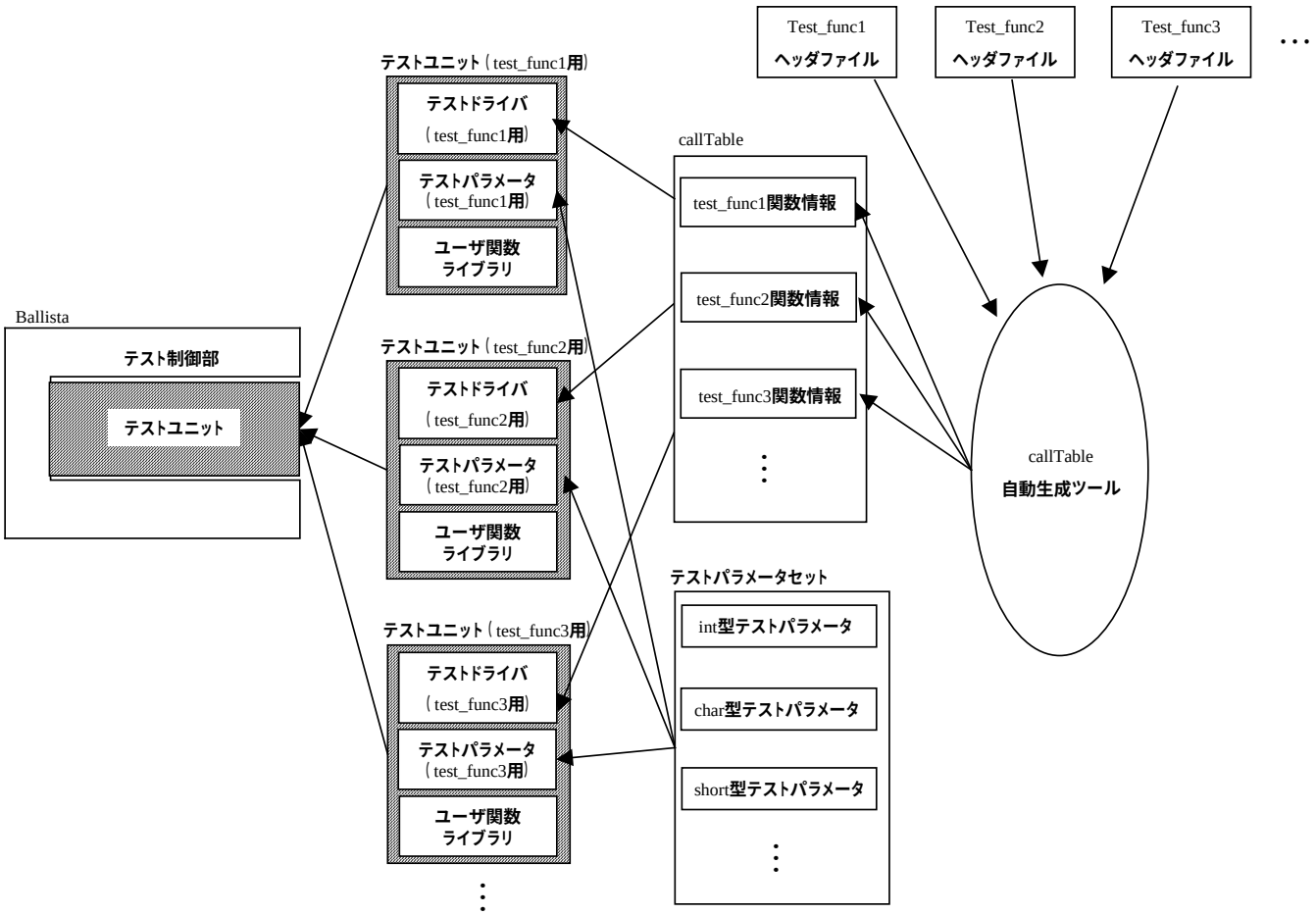

図 2 改良後 Ballista 動作フロー

呼出している．一方，ユーザ関数は K&R 規格の C にも対応するために，gcc でコンパイルする必要がある．gcc と g++では関数名の差異が生じ，関数名エラーでテストドライバがユーザ関数を呼出せない．そこで、テストドライバ側を修正し，ユーザ関数が C である事を明示的に宣言できるようにした．

これらの対応を表 1 にまとめる．

表 1 ユーザプログラムへの対応

項目	対応方法
callTable	ユーザ関数のヘッダファイルから callTable を自動生成するツールを作成 同時に typedef された型を基本型にマッピング
ライブラリ	ユーザ関数を共有ライブラリ化 これをリンクするように，テスト制御部を修正
ヘッダ	テストドライバは，ユーザ定義ヘッダーファイルをインクルードするように，テスト制御部を修正
関数名	テストドライバがユーザ関数を C 言語の関数名で呼び出せるよう，テスト制御部を改良

また、改良後の Ballista 動作フロー図を図 2 に示す．

3.2. ポインタ引数への対応

前述の改良で Ballista をユーザプログラムに適用し強靭性を自動評価できる環境は整った．しかし，OS とユーザプログラムでは評価の観点が異なる．特に，ポインタ引数の処理については，改良の余地がある．

3.2.1. ポインタ引数の処理

Ballista では，テスト対象関数の引数がポインタの場合，OS のクラッシュを引き起こすようなアドレス値をテストパラメータとして与えている．これをユーザ関数のポインタ引数に適用しても， ポインタ引数の不正なアドレス参照となりアボートする．これでは OS の強靭性評価にしかならない．ユーザ関数の強靭性評価には，ポインタ引数に有効なアドレスを参照させる事が必要である．

そこで，ユーザ関数を呼出すテストドライバで変数を確保し，この変数へのポインタアドレスを関数のポインタ引数の値として呼出すように改良した．

この改良を行う事で，ユーザ関数のポインタ引数には常に有効なアドレス値が渡される．つまり，OS が不正なアドレスを参照してアボートする事はなく、結果がアボートであればユーザプログラムの不具合と判断できる．

3.2.2. ポインタが指し示す変数の値

有効なポインタを用意しても，そのポインタが指し示す変数の値が同じであれば，この値を参照するユーザ関数の強靭性はテストできない．

そこでポインタが指し示す変数にテストパラメータを動的に変化させて与えるように改良した．これにより，テスト対象関数の引数がポインタであってもユーザプログラム強靭性を自動的に評価できる．

図 3 にテストすべき関数の引数がポインタである場合の，OS とユーザプログラムでのテストの違いを説明する．

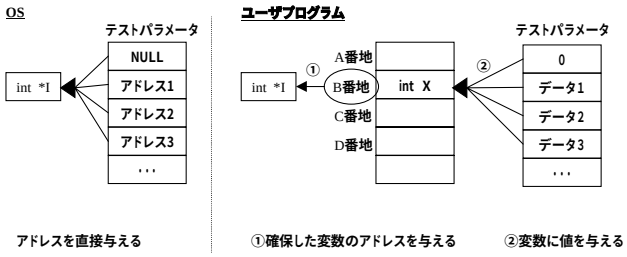

図 3 ポインタ変数とテストパラメータ

4. 結果と考察

4.1. 結果

まず，従来の Ballista を Linux2.4.9 と Linux2.2.20 へ適用した．その結果を表 2 に示す．

表 2 改良前の Ballista の適用結果

System	Linux 2.4.9	Linux 2.2.20
POSIX Fns. Tested	221	224
Fns. with Catastrophic Failure	0	5
Fns. with Restart Failure	38	37
Fns. with Abort Failure	130	132
Fns. with No Failure	78	78
Number of Tests	341264	347219
Abort Failures	56431	58225
Restart failures	1559	1898
Normalized Abort + Restart Rate	16.99%	17.32%

次に，改良した Ballista を，DES 処理ライブラリを構成する 5 つの関数に適用した．各関数の引数に与えたテストパラメータは

- 0
- 2^n, (2^n -1), (2^n +1)
- 型の境界値

などの値である．表 3 に int 型の引数に与えるテストパラメータの例を示す．

結果を表 4 に示す．

ここでは，エラーを発見することができなかったが，これがテストデータの不十分さによるものかどう

かは分析する必要がある.

表 3 int 型のテストパラメータの例

種別	値
0	0
2^n	2, 16, 64, 256, 2^{12}, 2^{13}
2^n –1	1, 15, 127, 255, (2^{14}-1)
2^n +1	17, 65, 129, 257, (2^{13}+1)
short 型最小値-1	SHRT_MIN-1
short 型最大値	SHRT_MAX
int 型最小値+1	INT_MIN+1
int 型最大値	INT_MAX

表 4 改良した Ballista の適用結果

DES Fns tested	5
Number of tests	8,391
Number of failures	0

実行時間は関数の規模等に依存するが，平均では，両者とも，1 秒あたりに 20 件のテストを処理した.

4.2. 考察

Ballista をユーザプログラムの強靭性評価に適用できるよう改良した.

テストすべき関数をシステムコールからユーザ関数に変えるために，callTable の生成やユーザ関数のライブラリ化などの準備が必要であるため，改良後の Ballista の実行手順は図 4 のようになる．しかし，これらの準備工程は自動化されているので，オリジナルの簡便さを損なってはいない.

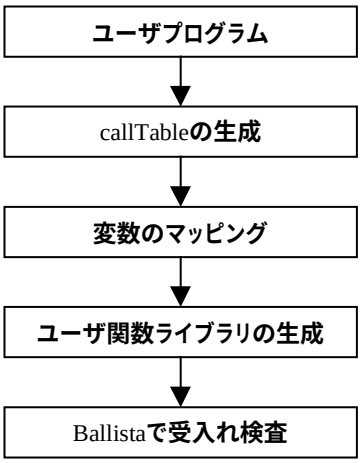

図 4　受入れテスト手順

また，ポインタ変数に対するテストの強化を図った．有効なアドレスを与え，その内容を動的に変える機能を追加したことで，種々のデータでのユーザプログラムの受入れテストが可能になった.

以上の点から，本研究での改良は，Ballista の利便性を維持しつつ，機能を強化するものであったと考える.

次に，改良した Ballista による受入れテストの効果を考察する.

第 1 に，受入れテストの時間が大幅な短縮である．従来,必要であったテストドライバのコーディングや，テストデータの準備が必要なく，さらに平均 1/20 秒でテストが実行されるからである．ただし，Ballista は正しく動く事を判定するものではなく，あくまで丈夫である（落ちない）事を判定しようとするものである．したがって，従来のテストを代替するものではない．この点で全テスト時間の削減効果を大きく算定する事は危険である.

第 2 に,受入れテストの品質の均一化である.Ballista Method によれば、汎用のテストドライバを利用し、基本的な型に対応したテストパラメータセットが用意されているため、これらを利用することで一定の受入れテストは必ず実施され，人為的な漏れはすくなくなる．特に，導入後の受入れテストの手順は，図 4 に示すように殆ど自動化されるので，担当者の操作ミスは削減されるとともに，同じ受入れテストを簡単に繰返して実行でき不具合現象の再現性も高まる．ただし，上記の時間の削減効果と同様に，これも受入れ検査に限定しての効果である.

第 3 に，テストサイクルの円滑化である．Ballista による受入れテストで，テストを本格的に始める前にプログラムの完成度を大まかに捉えられるからである.簡単にアボートするような完成度の低いプログラムをテスト部門に引き渡す事は，非常に危険である．再起動が多くなり本格的なテストでの手戻りが多くなる．何よりも，テスト担当者のモチベーションを低下させる．これを回避するために受入れ検査で強靭性を評価する事は重要である.

これらを鑑み，Ballista は受入れテストのファーストステップとして効果的であると考える.

5. おわりに

Ballista をユーザプログラムに適用できるように改良し，特にポインタ変数に関する機能強化を行った．結果，OS のみでなくユーザプログラムに対しても，一連のテスト準備から実行まで強靭性評価をほぼ自動実行でき，受入れテストにおける時間の短縮とテスト操作ミスの削減の効果が得られた.

本取組みでは，引数の値は Ballista で用意されている値を使用している．この値をさらに検討して，より効果的なパラメータセットとする取組みを行っている.

Ballista は決められたパラメータを引数としてテス

トを自動実行し強靭性を評価するだけであるが，テストに先立ち，動作がある程度保証できる事は，テスト担当者のモチベーションを下げない点で有効と考える．また，期待値を与え実行結果と自動照合できるような仕組みは Ballista を改良するよりは，別の仕組みで実現する方が良いと考える．

また今後の課題としては，構造体・共有体・関数ポインタを使用しているユーザプログラムへの対応がある．

6. 謝辞

本研究は，文部科学省知的クラスター創成事業の一環として実施している．各関係機関および（株）SRA 西日本，（株）ネットワーク応用技術研究所の関係者の方々に感謝致します．

文　献

[1] Phillip Koopman and John DeVale, "The Exception Handling Effectiveness of POSIX Operating Systems," IEEE Trans. on Software Engineering, Vol.26, No.9, pp.837-848, September 2000.

[2] T. Fukunaga, T. Nakanishi, M. Matsumoto, S. Yamazaki, T. Kitasuka and A. Fukuda, “Empirical Study on Robustness of Operating Systems to Exceptional Inputs: A Report on Re-experiment of CMU Ballista Method,” Proc. of The First CLUSS International Workshop on Next Generation Embedded Software, pp.175-187, Fukuoka, Japan, Dec.2003.