2008－OS－108 (25)
2008／4／24

スナップショットを用いたデバッグ環境の構築

菊地 誠　　阿部 洋丈　　梅村 恭司
豊橋技術科学大学 情報工学系

本研究では，プログラムのエラーの再現が難しい場合や複数のプログラムが並列実行している場合などのデバッグに，スナップショットを利用することを検討した．スナップショットはある時点での PC の状態をそのまま保存したものでありプログラム実行中の状態を保存できる．そのため，再現の難しいエラーの状態を保存しておくことができ，エラー原因の情報収集に有用である．複数のプログラムが並列実行している場合においては複数の PC とディスプレイを利用し，スナップショットを他の PC へネットワーク転送して様々なプログラムの実行状態を同時に分析することを検討した．本稿ではスナップショットを用いたデバッグ環境の構築方法について示し，作成したデバッグのシナリオにより本環境でデバッグできることを確認する．また，本環境においてスナップショットの各操作に要する時間の測定実験より，本環境で用いる PC の性能やスナップショットのネットワーク転送の方法を検討する．

Implementation of Debugging Environment Using Snapshots

Makoto Kikuchi, Hirotake Abe and Kyoji Umemura
TOYOHASHI UNIVERSITY OF TECHNOLOGY

In this research, we considered using snapshots for debugging program errors that is difficult to reproduce or program works with some other programs. It is useful for debugging because snapshot can save state of difficult error of repeatability. When program works with some other programs, we considered for running state of program was analyzed at the same time with multiple PCs and displays. In this paper, we describe on method for implementation of debugging environment using snapshots, and we check that we can debug error with scenario on this environment. In addition, we consider for performance of PC to use on this environment and method for transfer by experiment.

1 はじめに

プログラムのデバッグにおいて，エラーの再現が難しい場合や複数のプログラムが並列実行している場合などでは，エラー原因を調べにくいことがある．エラーの再現が難しい場合では，例えばネットワーク状態の遷移がプログラムの実行時に影響する場合，エラーが起こった時と同一の状態を再現するのが難しい．よって，デバッグの際にエラー原因を調べるのが困難となる．また，複数のプログラムが並列実行している場合では，複数のソースコードを変更してデバッグするためにデバッグの経過が分かりづらい．これらのような場合においてもデバッグを行えるように，本研究ではスナップショットを用いてデバッグに利用することを考えた．本研究でのスナップショットとは，ある時点での PC の状態をそのまま保存したものであるためプログラム実行中の状態，すなわちエラーが起こっている状態を保存することができる．よって，プログラムの実行状態をスナップショットとして保存することで再現の難しい場合のエラー原因の情報収集ができる．さらに複数の PC，ディスプレイを使用してスナップショットを他の PC へネットワーク転送し，複数のプログラムが並列実行している場合でも同時に様々な実行状態を見比べながらエラー原因を分析できる環境の構築を行った．

本環境の構築にあたって，複数の PC をネットワーク接続した．スナップショットについては，Linux 上で仮想的に Linux を動作させる User Mode Linux（UML）[6]を用い，この UML のスナップショットを，ScrapBook for UserMode Linux（SBUML）[7]により保存，差分圧縮，復元する．また，この SBUML のコマンドラインは複雑であるため，新たにスナップショットを他の PC へ転送できる機能を加えてスナップショットを保存，差分圧縮，転送，復元できるツール作成を行った．

本環境のデバッグ機能を検証するためにデバッグのシナリオを作成し，検証を行った．このシナリオではクライアント，プロキシ，サーバのネットワークプログラムが１つの PC 上で並列実行している場合に起こるエラーを，スナップショットの転送を用いてデバッグできることを

確認した.

デバッグでは作業に要する時間は重要であるため, 本環境でスナップショットの操作に要する時間を測定する実験を行った. デバッグ性能の測定実験では, まず本環境において性能の異なる 3 台の PC を用いてスナップショットの保存, 差分圧縮, 復元にかかる時間を測定した結果より PC の性能による処理時間の違いを示す. 続いての測定実験では, 差分圧縮によるスナップショットの転送での有効性を検討し, 本環境においては有効であることを示す.

2 デバッグ環境の構築方法

本環境の構築にあたり, 使用したハードウェアやソフトウェアの使用方法について述べる.

2.1 使用したハードウェア

本環境では, 本研究室にある図 1 のような複数の PC とディスプレイを用いることを想定している. 図 1 の各 PC をネットワーク接続し, 後述するツールを用いてスナップショットを他の PC へ転送する. プログラムの実行状態を複数のディスプレイで見比べながらデバッグできる環境となる.

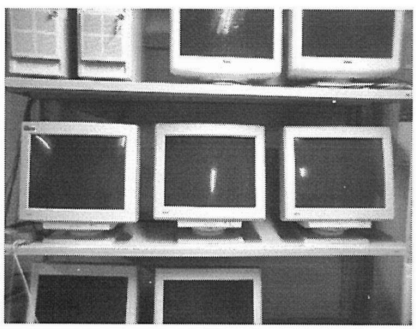

図 1 本研究室のハードウェア環境

2.2 使用したソフトウェア

2.2.1 User Mode Linux (UML)

本環境では, Linux 上で仮想的に Linux を動作させる User Mode Linux (UML) を用いた. UML を動作させる都合上, ホストとなる OS は Fedora Core1 (以下ホスト OS とする) とし, UML では Red Hat7.2 を動作させた. UML の機能を以下へ示す.

- ディスクイメージファイル上に構築された Linux 環境である.
- ネットワーク機能をフルサポートする (Universal TUN/TAP ドライバ経由).
- hostfs 機能により, ホスト OS 上のファイルへのアクセスができる.
- 差分ファイルへのディスクイメージ変更履歴を取得する.

本環境では UML で動作させる Red Hat7.2 上で, デバッグ対象となるプログラムを実行させる.

2.2.2 ScrapBook for User Mode Linux (SBUML)

本環境では, UML のスナップショットを ScrapBook for User Mode Linux (SBUML) を用いて保存, 復元させる. スナップショットの保存において, SBUML では元となるスナップショットと作業したところまでの状態の差分をとることで大幅にファイルサイズを小さくすることができる (差分圧縮). すなわち, 元となるスナップショットを各 PC 上に配置しておけば, ファイルサイズの小さいこの差分を取った部分のみをネットワークで転送すれば, スムーズにプログラムの実行状態を移送できる. この差分圧縮によるネットワーク転送での有効性については後述する.

2.2.3 GUI ツールの作成

SBUML を用いて, 新たにスナップショットを転送する機能を加え, GUI で UML の起動, 終了やスナップショットの保存, 差分圧縮, 転送, 復元, 削除といった操作を行うツールを作成した. 作成したツールを図 2 へ示す. また, 図 2 のツールにおける各ボタンの機能を以下へ示す.

(1) boot UML (UML を起動)

UML を起動する SBUML のコマンドを実行し, スナップショットを読み込んで UML を起動する.

(2) save snapshot（スナップショットを保存，圧縮）

クリックにより表示されるダイアログで，スナップショットの起動に用いたマシンディレクトリ，起動するスナップショット名，保存するスナップショット名を入力し，スナップショットを保存と差分圧縮する SBUML のコマンドをそれぞれ実行する．なおマシンディレクトリとは，SBUML がスナップショットを起動させる時に UML のシステム状態を一時的に保存しておくディレクトリである．

(3) transfer snapshot（スナップショットを別の PC へ転送）

クリックにより表示されるダイアログで，保存しているスナップショット名と転送先の PC 名を入力し，システムコマンドを実行してスナップショットを転送する．

(4) restore snapshot（スナップショットを復元）

クリックにより表示されるダイアログで，保存しているスナップショット名を入力し，UML を起動する SBUML のコマンドを実行する．

(5) delete snapshot（スナップショットを削除）

クリックにより表示されるダイアログで保存しているスナップショット名を入力し，システムコマンドを実行してスナップショットを削除する．

(6) end UML（UML を終了）

全ての SBUML マシン及びディレクトリのクリーンアップ，SBUML に関連のある全プロセスを終了させる SBUML のコマンドを実行する．

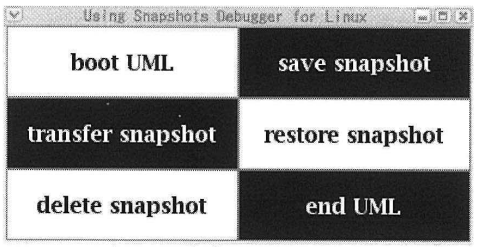

図 2　作成したツール

2.3　本環境のまとめ

以上により構築した本環境の概要図を図 3 へ示す．PC 上で起動している UML のスナップショットを，作成したツールにより保存，差分圧縮し，他の PC へネットワーク転送する．この転送されたスナップショットを各 PC 上で復元し，プログラムの実行状態を複数のディスプレイで見比べながらデバッグする．

このように本環境では，同時に様々なプログラムの実行状態を分析しながらデバッグすることができる．

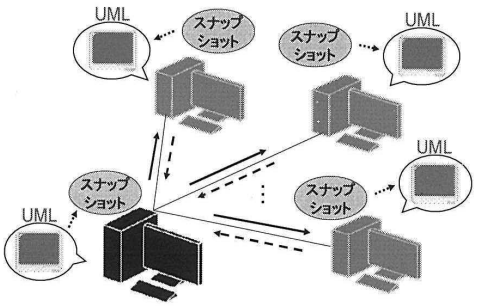

図 3　本環境の概要図

3　デバッグ機能の検証

本環境でデバッグができることを確認するためにクライアント，プロキシ，サーバのネットワークプログラムによるデバッグのシナリオを作成した．各プログラムは 1 台の PC 上で並列実行させ，クライアントプログラムで入力した文字列が，プロキシプログラムを介してサーバプログラムへ送信され，出力される．

3.1　意図しないエラーの内容

このシナリオでのエラー内容は，「クライアントプログラムで送信された文字列をサーバプログラムで受信すると，出力に勝手に改行が入ることがある．」である．

3.2　デバッグの問題点

どのプログラムがエラー原因となっているか

調べる際に複数のソースコードを変更するため，デバッグの経過が分かりづらいことが問題となっている．

3.3 デバッグのシナリオ

このケースでは，図 1 のうち 3 台の PC（PC1 - 3 とする），ディスプレイを用いた本環境でプログラムの実行状態を保存したスナップショットを他の PC へ転送し，見比べながらエラー原因を調べ，結果プロキシに文字数の制限があったことを特定する．このシナリオの詳細を以下に示す．

① PC1 上で，作成したツールを起動しツールの boot UML で UML を起動する．この時の PC 上のスクリーンショットを図 4 へ示す．
② 現状でのエラーを確認するため，クライアント，プロキシ，サーバプログラムを UML 上で並列実行し，クライアントプログラムから文字列を入力してサーバプログラムへ送信する．各プログラムは screen コマンドで切り替えて操作する．
③ サーバプログラムで出力された結果，受信した文字列に改行が入ってしまうことがあるエラーが生じている．
④ デバッグを開始するにあたって，バックアップ用にまず 3 つのプログラムを実行してエラーが起こっている状態でスナップショットを保存する．ツールの save snapshot でスナップショット名を original として保存する．この時の PC 上のスクリーンショットを図 5 へ示す．
⑤ 準備が整ったので，デバッグを開始する．まずクライアントプログラムでの入力が正確に行われているかを確認するために，入力した文字列を出力するコードを挿入する．
⑥ 3 つのプログラムを並列実行させた状態で，スナップショット名を client_printDebug として保存する．
⑦ client_printDebug を PC2 へ転送する．transfer snapshot で転送するスナップショット，転送先の PC 名を選択して転送する．この時の PC 上のスクリーンショットを図 6 へ示す．
⑧ PC2 へ移り，受信したスナップショット client_printDebug を復元する．ツールの restore snapshot をクリックし，client_print Debug を選択する．すると，プログラムの実行状態が復元できていることが確認できる．この時のスクリーンショットを図 7 へ示す．
⑨ クライアントプログラムから文字列を入力してみると，正確に入力が行われていることが確認できるが，サーバプログラムでは改行が入っている．
⑩ プロキシでの受信を確認するために PC1 へ戻り，ツールの end UML で一度起動していた UML を終了する．
⑪ バックアップ用に保存しておいたスナップショット original を復元する．
⑫ プロキシプログラムにクライアントプログラムから受信した文字列を出力するコードを挿入しスナップショット名を proxy_printDebug として保存し，PC3 へ転送する．
⑬ PC3 へ移り，プロキシプログラムでは正確に受信できていることを確認するが，サーバプログラムでは同じように改行が入っている．
⑭ これまでの結果よりサーバの受信が悪いのではないかと考え，PC1 へ戻りサーバプログラムのコードを見直してみるが，特におかしいところは見あたらない．そこでプロキシプログラムを介した時に，何か誤処理されているのではないかと検討をつけ，プロキシプログラムのコードを見直す．
⑮ PC3 へ移り，プロキシプログラムのソースコードを見て文字列操作がされていそうなコードを見つけ，その部分をコメントアウトする．
⑯ サーバで正確に文字列が受信されるようになった．結果として，プロキシで文字列を介した際，20 文字来たら改行するという操作がされていたことが分かった．
⑰ コードを修正し，スナップショット名を debug_completed として保存し，デバッグ完了となる．

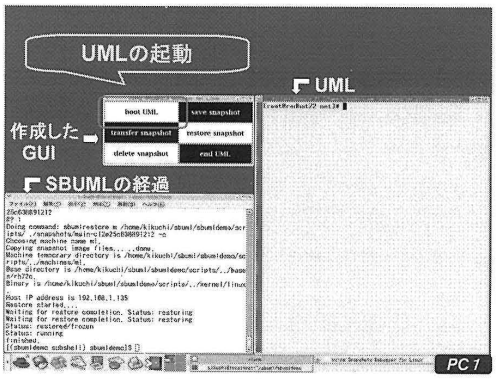

図 4　UML の起動

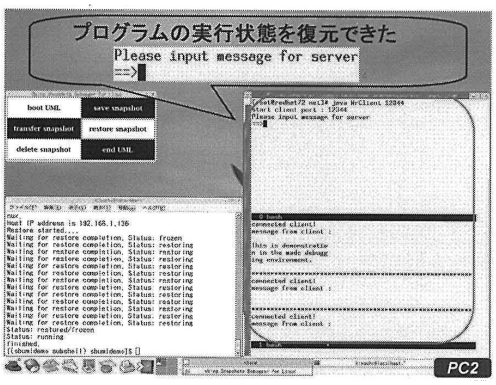

図 7　スナップショットの復元

以上のシナリオを実行することで，本環境でスナップショットを保存，差分圧縮，転送，復元し，デバッグできることを確認できた．また，スナップショットはプログラムの実行状態を保存することができ，実行状態を他の PC へ移送できることも確認できた．

図 5　スナップショットの保存

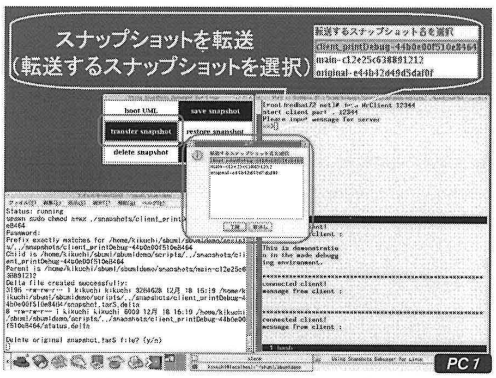

図 6　スナップショットの転送

4　スナップショットの操作時間

デバッグでは作業に要する時間は重要であるため，本環境でスナップショットの各操作に要する時間を測定する実験を行った．はじめの実験では，PC 性能により処理時間が大きく異なることを示す．続いての実験では，差分圧縮によるスナップショットの転送における有効性を示す．

各実験で使用したスナップショットはデバッグ機能の検証の，デバッグのシナリオで使用したものであり，以下に各スナップショットの内容を示す．また，実験に使用した各 PC の性能を表 1 へ，各スナップショットのファイルサイズで差分圧縮していない場合を表 2 へ，差分圧縮した場合を表 3 へ示す．

(1)　main

ケーススタディにおいて，デバッグ作業の元となるスナップショット．

(2)　original

スナップショット main の状態から，各ネットワークプログラムを並列実行中にエラーが起こっている状態を保存したスナップショット．

(3) client_printDebug

スナップショット main の状態から，クライアントプログラムの入力を出力させるコードを挿入して，各ネットワークプログラムを並列実行中に保存したスナップショット．

(4) proxy_printDebug

スナップショット original の状態から，プロキシプログラムが受信した文字列を出力させるコードを挿入して，各ネットワークプログラムを並列実行中に保存したスナップショット．

(5) debug_completed

スナップショット proxy_printDebug の状態からエラーコードを修正して，デバッグ完了した状態のスナップショット．

表 1 各 PC の性能

PC 名	CPU[GHz]	メモリ[Mbyte]
PC1	1.0	1024.0
PC2	1.4	512.0
PC3	3.0	1024.0

表 2 差分圧縮していない場合のスナップショットのファイルサイズ

スナップショット	ファイルサイズ[Mbyte]
main	414.9
original	417.0
client_printDebug	417.5
proxy_printDebug	415.2
debug_completed	420.1

表 3 差分圧縮した場合のスナップショットのファイルサイズ

スナップショット	ファイルサイズ[Mbyte]
original	2.3
client_printDebug	2.5
proxy_printDebug	0.5
debug_completed	4.4

4.1 PC 性能による処理時間の比較

PC 性能により，処理時間がどのくらい異なるかを比較する実験を行った．

性能の異なる PC3 台(PC1 - 3)を使用し，ケーススタディにおいて使用した各スナップショットの保存，差分圧縮，復元に要する時間を連続 3 回計測し平均をとった結果を，図 8 へ示す．

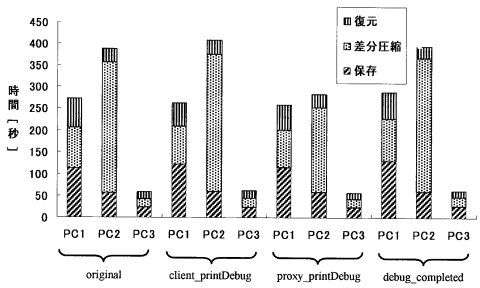

図 8 各 PC による，スナップショットの操作時間

図 8 において，まずスナップショット original の PC1 と PC2 を比較してみると，CPU 周波数の高い PC2 の方がスナップショットの保存，復元に要する時間が短いことが分かる．また，メモリ容量の多い PC1 の方がスナップショットの差分圧縮にかかる時間は短くなる事が分かる．さらに PC3 と PC1，PC2 を比較してみると，PC3 の処理時間が大幅に短いことが分かる．これらは他のスナップショットについても同様であることが分かる．続いてそれぞれの PC の処理時間をスナップショットごとに比較してみると，差分圧縮時間にばらつきがあることが分かる．これは，差分圧縮は元の状態から作業した状態までの差分をとるため，変更した内容が多いほど差分圧縮にかかる時間は大きくなるためであると考えられる．

以上の結果から，スナップショットの保存，差分圧縮，復元に要する時間は PC 性能に大きく依存することが分かる．性能の良い PC を使用することで，処理時間を短くできる効果を確認できた．

4.2 差分圧縮の有効性

本環境においてスナップショットを転送する際に，ファイルサイズを小さくするために用いている差分圧縮が，転送においてどのくらい有効であるかを検討する実験を行った．本来の，スナップショットの保存，転送，復元という操作において保存，圧縮，転送，復元と 1 ステップ処理を増やしても，UML のシステム状態の移送完了に要

する時間が短くなる事を示す.

PC性能による処理時間の比較の実験で用いた最も性能の良いPC3を使用し，スナップショットの差分圧縮の有無により，UMLのシステム状態の移送完了までに要する総時間を図 9 へ示す. 図 9 において，スナップショットの保存，差分圧縮，復元に要する時間は図 8 と同様である. スナップショットの転送に要する時間は，ファイルの転送速度がマシン性能によらないものとして別に測定し，100BASE-Tのネットワークで表2と表 3の各ファイルを2台のPCを用いて連続5 回測定し平均をとった結果である. なお，スナップショットの復元に要する時間も PC3 で行った処理時間であり，図 9の総時間はPC3が2 台あるものと仮定した上での測定結果となる. また図 9 の結果より，差分圧縮の有無による各スナップショットの操作に要する総時間の減少率を表 4へ示す.

表 4　差分圧縮によるUMLのシステム状態の移送完了までに要する総時間の減少率

スナップショット	減少率[%]
original	40.75
client_printDebug	41.68
proxy_printDebug	41.75
debug_completed	41.31

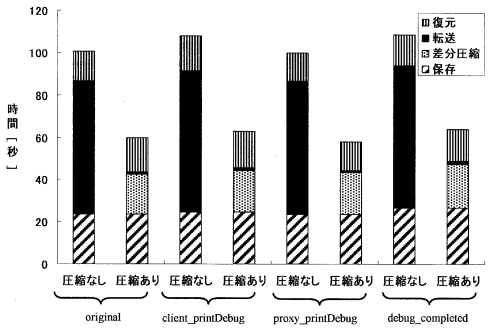

図 9　PC3におけるUMLのシステム状態の移送完了までに要する総時間

図 9 より，差分圧縮することでファイルサイズが小さくなるため転送に要する時間が大幅に短くなり，システム状態の移送完了までに要する総時間は表 4 から，差分圧縮してない場合よりも約 40%近く減ることが分かる. よって，本環境においては差分圧縮がとても有効であることが確認できた. 今回使用した100BASE-Tのネットワークよりも速度の速いネットワークで同様の比較は行っていないが，環境が良くなることで差分圧縮を用いなくてもシステム状態の移送完了までに要する総時間はさらに短くなると予想される.

5　関連研究

本研究のようにスナップショットを利用する方法には上里ら[1]の報告がある. また，トランスレータを利用したスナップショットの実現方法として下川ら[2]の報告がある. これらで利用しているスナップショットはひとつの実行プロセスの状態であるが，本研究で実現しているスナップショットはある時点での PC の状態をそのまま保存したものであるため，複数の実行プロセスの状態を保存することができる.

また，プログラムの実行状態を再現する別の方法として，実行を遡る逆実行というものがある. 逆実行に関しては多くの研究がされており，様々な手法がとられている. 丸山（一）ら[3]と丸山（真）ら[4]は逆実行を利用したデバッガを報告している. 本研究で行った方法と比較してみると，逆実行はプログラムの実行をある時点まで戻すというものであるが，本研究ではPCの状態全体を保存することでプログラムの実行状態を保存し，何回でも簡単に復元できるという大きな違いがある. 例えエラーの再現が難しい場合でもプログラムの実行状態を保存しておくことができ，デバッグに非常に有用であるといえる. また，丸山ら[5]はデータ再演の方法を提案している. データ再演とは，プログラムの実行状態を戻したところから再びプログラムを実行するまでを自動化することなので，本環境でもスナップショットの復元の後に再演する方法が実現できればデバッグ機能が向上するので，大いに有用になると考える.

6 まとめ

本研究では，プログラムの実行状態をUMLのスナップショットとしてSBUMLを用いて保存し，差分圧縮によりファイルサイズを小さくしてすばやくネットワーク転送する環境を構築した．これにより複数のプログラムが並列実行している場合に，複数のハードウェアを用いて実行状態を同時に見比べてデバッグできるということを，デバッグのシナリオにより確認した．またスナップショットの保存，差分圧縮，転送，復元に要する時間の計測実験を行い，性能の良いPCでは処理時間が早くなることや，差分圧縮することでスナップショットの転送に有効となることを確認した．

謝辞 この研究は，平成19年度科学研究費補助金（課題番号19500120）の研究成果の一部である．

7 今後の課題

本環境は，デバッグにおけるエラー原因の情報収集に有用となるよう，プログラムの実行状態を保存，転送，復元するという原理のみを実装した環境である．よって，デバッグ作業における変数値の出力などの作業は手動で行わなければならないため，今後の課題としては既存のデバッガをUMLに導入することにより，デバッグ作業を向上させることが挙げられる．また，スナップショットの操作に要する時間を短縮するため，性能の良いPCや速度の速いネットワークを使用することが挙げられる．

参考文献

[1] 上里献一，河野真治：Suciライブラリのスナップショット API を利用した並列デバッグツールの設計，琉球大学理工学研究科，日本ソフトウェア科学会第 20 回大会論文集（2003）．

[2] 下川僚子，梅村恭司：トランスレータを利用した機種非依存な実行移送方式，情報処理学会論文誌，Vol. 40，No. 6，pp. 62 - 72（1999）．

[3] 丸山一貴：プログラム実行点の概念に基づくデバッグパターンの抽出と自動化，東京大学大学院情報理工学系研究科，博士論文（2003）．

[4] 丸山真佐夫，山本繁弘，大野和彦，中島浩：巻き戻し実行をサポートする並列プログラムデバッガ，情報処理学会論文誌，Vol. 45，No. SIG3 (ACS5)，pp. 109 - 121 (2004)．

[5] 丸山真佐夫，津邑公暁，中島浩：データ再演法による並列プログラムデバッギング，情報処理学会論文誌，Vol. 46，No. SIG12（ACS11），pp. 214 - 224（2005）．

[6] User-Mode Linux (http://user-mode-linux.sourceforge.net/)

[7] ScrapBook for User-Mode Linux (http://sbuml.sourceforge.net/)．