システム評価 4－3
（2002．8．21）

Javaアプリケーションサーバの特性解析

保田 淑子　川本 真一　濱中 直樹
（株）日立製作所　中央研究所
〒185-8601　東京都国分寺市東恋ヶ窪1-280

企業がインターネットビジネスでの優位性を維持し新規サービスを展開し続けるためには，機能仕様のみならず性能目標も満足するアプリケーション(AP)サーバを短期間に開発する必要がある。本稿では，性能目標を短期間に達成するAPサーバ開発に向け，様々なOSおよびJava仮想マシン上で稼動するAPサーバの性能特性を解析した。特性解析では，OSとJava仮想マシンの組み合わせにより性能が最大48%も異なることを明らかにした。さらに性能差原因の特定に向けた評価モデルを構築し，性能差の主要因がガベージコレクションの内部処理方式であることを明らかにした。解析結果を考慮してOS及びJava仮想マシンを選択することにより，プログラマはAPサーバの開発期間を短縮できる。

Analysis of a Java application server

Yoshiko Yasuda, Shinichi Kawamoto, and Naoki Hamanaka
Central Research Laboratory, Hitachi, Ltd.
1-280, Higashi-Koigakubo, Kokubunji-shi, Tokyo 185-8601, Japan

To keep competitive advantages of Internet business and to develop new services one after another, application servers must be developed in a short period at low cost. With the aim of solving this problem, we analyzed workloads of a Java application server on multiple platforms using the SPEC JBB2000 benchmark and a new evaluation model. The experimental results indicate that there is a large performance gap between the multiple platforms. The evaluation model reveals that the gap is mainly caused by the difference of inter-process methods in the garbage collection. By selecting an appropriate OS and a Java virtual machine to the application server based on the experimental results, programmers can develop the Java application servers in a short period.

1. はじめに

ブロードバンド時代の到来により，企業情報システムの整備が急速に進展し，マルチプロセッサPCサーバを核にしたインターネットコマース対応の階層型Webシステムの需要が増大している。その中で，階層型Webシステムの中間層に位置するアプリケーションサーバは，企業情報システムのビジネスロジックを担う重要な部分である。企業がインターネットビジネスでの優位性を保ち，新規サービスを展開し続けるためには，機能仕様のみならず性能目標をも満足するアプリケーションサーバを短期間に開発する必要がある。

アプリケーションサーバを短期間に開発するため，オブジェクト指向型プログラミング言語Javaが多用される。Javaは，プラットフォームに依存しないプログラム開発が可能であることから，階層型Webシステム開発向きの言語である。しかしながら，Javaは発展段階ということもあり，同一のハードウェアでもそれに搭載するOSやJava仮想マシンの差異でアプリケーションサーバの性能が異なる可能性がある。選択するプラットフォームによっては，性能目標を達成できず，結果としてアプリケーションサーバの短期開発を阻害する要因になる。そこで，短期間で性能目標をクリアするアプリケーションサーバを開発するために，様々なOSやJava仮想マシン上で開発されたアプリケーションサーバがどのような性能特性を持つのかという点について解析することが課題となる。

本稿では，上述の課題を解決するために，Javaで開発したアプリケーションサーバの性能特性を解析した結果について述べる。特性解析では，

OS および Java 仮想マシンの差異が性能に及ぼす影響を調査し，評価モデルを構築することにより性能低下要因を特定する。

2. Java アプリケーションサーバ性能評価

2.1 評価プログラム

アプリケーションサーバの性能評価プログラムとして，SPEC JBB2000 を用いた[1]。SPEC JBB2000 は，オンライントランザクション処理性能を測定する世界標準ベンチマークである TPC-C[2]を約 1/10 にスケールダウンしたものに相当し，ビジネスロジック部分を Java 言語で記述する。評価指標はトランザクション処理で扱う商品データベースの規模を示すウェアハウス数（Java スレッド数に相当）を増加させた場合のスループットであり，180 秒間の計測時間において，一秒あたりに処理したトランザクション(ops)数である。マルチプロセッサ環境では，評価プログラムは単一の Java 仮想マシン上で実行され，各プロセッサに均等に Java スレッドが割り当てられる。

2.2 評価環境

表 1 に評価環境を示す。評価では，マルチプロセッササーバに，OS として Linux あるいは Windows2000 を搭載する。さらに，その上に Sun Microsystems 社あるいは IBM 社が開発した Java 仮想マシン(JVM)を搭載して評価プログラムを実行する。実行結果を解析することにより，様々なプラットフォームにおける Java アプリケーションサーバの性能特性を明らかにできる。

表 1：評価環境

#	項目	L-Sun	L-IBM	W-Sun	W-IBM
1	CPU	2 × Pentium III 550MHz			
2	ﾒﾓﾘ	1GB (256MB DIMM ×4)			
3	OS	Linux*1		Windows*2	
4	JVM	Sun*3	IBM*5	Sun*4	IBM*6
5	JVMｵﾌﾟｼｮﾝ	-Xms 256m -Xmx 256m			

*1 Redhat Linux6.2 Kernel 2.1.14-5.0SMP
*2 Windows 2000 server
*3 Sun JDK1.3.1-b24 for Linux [3]
*4 Sun JDK1.3.1-b24 for Windows [3]
*5 IBM JDK1.3 for Linux [4]
*6 IBM JDK1.3 for Windows [4]

2.3 評価結果

表 1 の評価環境において，評価プログラムを実行し性能を測定した。図 1 に示すように，いずれのプラットフォームにおいても，ウェアハウス数が 2 の場合に最大スループットとなる。またプラットフォームの組合せにより性能が大きく異なる。図 2 に，表 1 に示した組合せ(L-Sun/L-IBM/W-Sun/W-IBM)における処理性能比をまとめる。図 2 に示すようにプラットフォームの組合せにより処理性能が最大 48%も異なる。次章では，図 2 に示す処理性能比①～④の原因を特定する。

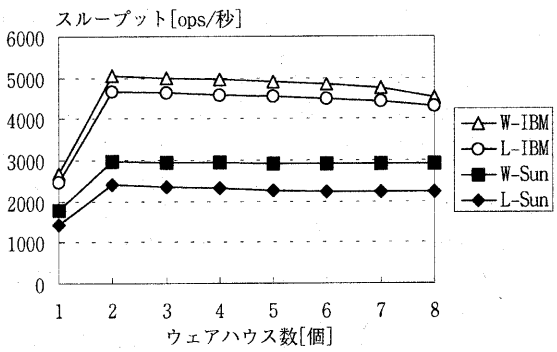

図1：SPEC JBB2000の性能評価

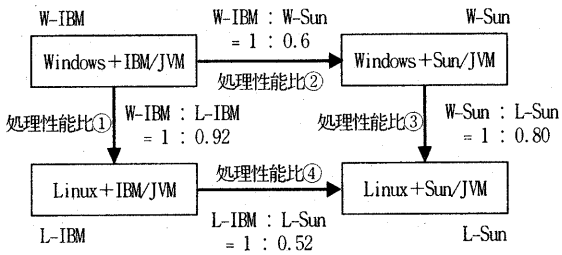

図2：プラットフォームの差異による処理性能比

3. 性能支配要因の特定

3.1 性能支配要因の特定手法

前節で挙げた処理性能比①～④の原因を特定するにあたり，最初に Java の実行時オプションを利用した。また，Java の実行時オプションでは対応できない詳細解析には，大容量サイクルベースバストレーサ[5]を利用した。大容量サイクルベースバストレーサは，プロセッサが発行するメモリアクセス情報である CPU バスリクエスト[6]をリアルタイムに 180 秒間記録する装置である。この装置により，サーバ内部の挙動をリアルタイムに解析することができる。

3.2 処理性能比の原因特定

(1) 処理性能比① (W-IBM と L-IBM の性能差)

W-IBM と L-IBM の処理性能比は 1:0.92 である。Java の実行時オプションである verbose オプショ

ンによりJVMにおけるヒープメモリの使用状態を調査した。その結果，ヒープメモリの使用状態はOSによらず同じであり，JVMレベルでは性能差がないことがわかった。評価で使用したLinuxのSMPカーネル2.1および2.2系は，マルチプロセッサ環境での性能が低い[7]。そのため，Windowsとの性能差が表面化したと考えられる。文献[7]によると，SMPカーネルを2.4系にすることで，この性能差を吸収可能である。

(2) 処理性能比② (W-IBMとW-Sunの性能差)

W-IBMとW-Sunの処理性能比は1:0.6である。この処理性能比の原因を特定するため，(1)と同様，verboseオプションにより，各JVMにおけるヒープメモリの使用状態を調査した。その結果，W-IBMの場合のみ，評価プログラム実行中にヒープメモリが足りなくなることが分かった。これは，W-IBMとW-Sunいおいて，JVMのガベージコレクション（以下GCと略する）方式が異なることに起因する。

W-IBMは，ヒープメモリ全体に対してごみ集めを行なうマーク・スイープGC方式を採用する。マーク・スイープGC方式は，GCスレッド以外を停止し，ヒープメモリ全体に対して使用オブジェクトにマークをつけ，マークされていないオブジェクトを削除した後，ヒープメモリを圧縮する。ヒープメモリに散在するごみオブジェクトを全て回収可能であるため，効果的なGCを行なうことができるが，ヒープサイズに比例してGC時間が増大するという欠点がある。

一方，W-Sunは一定サイズの領域に対してごみ集めを行なう世代別GC方式を採用する[8]。世代別GCは，(1)ほとんどのオブジェクトが生成後すぐに捨てられる，(2)古いオブジェクト程生存期間が長い，というオブジェクト指向プログラミングの特性を利用し，GCによるプログラム処理の中断時間を短縮するために，ヒープメモリを世代と呼ぶ複数グループで管理する。

図3に世代別GC方式を示す。世代別GCでは，ヒープメモリを新世代と旧世代の2つにグルーピングし，新世代を更にEden領域とSurvivor領域に分割する。アプリケーションプログラムが新規オブジェクトを生成すると，プログラムはJava仮想マシンのメモリ管理機構を呼び出し，新規オブジェクトにEden領域の空きメモリを割り当てる。新規オブジェクトに割り当てたEden領域のメモリが一定サイズを超えるとGCが発生する(マイナーコレクション)。新世代において新規オブジェクトに割り当てるための空きメモリがなくなると，ヒープ全体に対してGCを行なう（メジャーコレクション)。評価プログラムでは，十分にヒープメモリを確保するため，メジャーコレクションは発生していない。

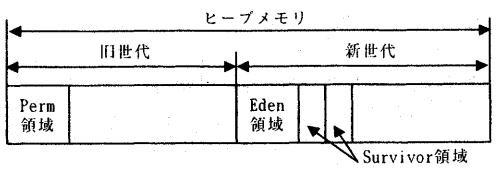

図3：世代別ガベージコレクション方式 [8]

続いて，W-IBMとW-Sunにおける評価プログラムの計測時間に占めるGC時間比を求めた。W-IBMでは，計測中2回GCが発生し，その時間は合計0.8秒である。それに対して，W-Sunでは，マイナーコレクションが複数回発生する。一回のマイナーコレクションは55ミリ秒と短いが，総GC時間は，11.6秒にもなることが分かった。これは，計測時間の6.4%に相当する。トレース解析結果から，W-Sunにおいても，W-IBMと同様GC中に通常のプログラム処理が中断し，総GC（プログラム中断）時間がIBM-JVMに比べ約15倍も長いため，性能が大きく低下することが判明した。

以上をまとめると，マルチプロセッサ環境であっても評価プログラムのように単一のJVM上で複数Javaスレッドを実行する場合には，GC中にプログラム処理が中断し性能が低下する。W-Sunでは特にその影響が大きい。この問題点を改善する方法としては，単一JVM上で複数Javaスレッドを起動するのではなく，複数のJVMを起動し，それらが独立してJavaスレッドを実行することが考えられる。複数JVMを起動することにより，一方のJVMがGCを実行中にも，残りのJVMが通常のプログラム処理を継続実行できるため，システム全体としてプログラム処理が停止しないようにできる。複数JVMの効果を調査するため，実験的に評価プログラムを複数JVM上で実行した。実験結果より，JVMの複数化により10%程度性能改善できる見こみを得た。

(3)処理性能比③(W-SunとL-Sunの性能差)

W-SunとL-Sunの処理性能比は1:0.8である。W-SunおよびL-SunではヒープメモリのJ使用状態が同じでありverboseオプションでは原因を特定できない。そこで，先に述べたバストレーサにより原因を特定した。図4は，評価プログラムのト

ランザクション処理部分中の経過時間と単位時間(100,000 サイクル)あたりのバスリクエスト数との関係を示す。

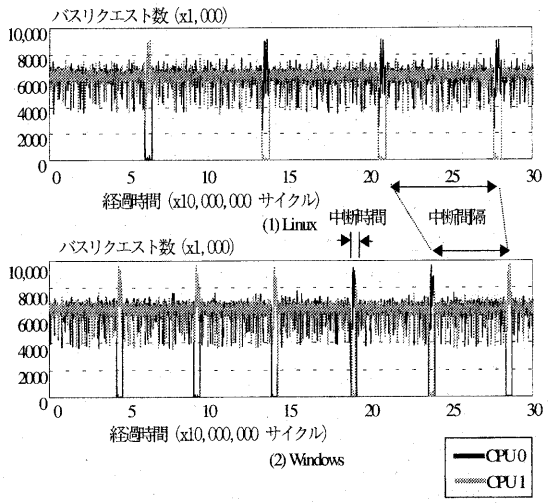

図4: トランザクション処理部における単位時間あたりのバスリクエスト数

図4に示すように，プログラム処理はGC処理により一定間隔で複数回中断する。GCは，新規オブジェクトに割り当てるメモリサイズが一定値を越えると発生する。W-SunおよびL-Sun上で同じ評価プログラムを実行すれば，GC間で新規オブジェクトに割り当てるメモリサイズも同じであるから，GCも同じ間隔で発生し，プログラム処理の中断間隔も同じはずである。ところが，L-SunはW-Sunに比べると，単位時間あたりのプログラム中断回数が少ないことがわかった。表2に図4から算出したプログラム処理の中断時間と中断間隔を示す。

表2：プログラム処理の中断時間と中断間隔

ガベージコレクション	L-Sun	W-Sun
中断時間[ミリ秒]	68.7	55.1
中断間隔[ミリ秒]	1010	659

表2に示すように，L-sunでは，W-Sunに比べてプログラム処理の中断間隔が長い（W-Sunの1.53倍)。プログラム処理の中断間隔におけるプログラム処理量は，L-SunおよびW-Sunとも同じであるから，L-Sunは単位時間あたりのプログラム処理能力が低く，その結果プログラム全体の性能が低くなると考えられる。

L-Sunにおける単位時間あたりのプログラム処理量が少ない原因を付きとめるため，バストレーサで取得したCPUバスリクエストの内訳を調査した。バスリクエストを解析した結果，L-SunではW-Sunに比べメモリライト要求が90倍発生していることが判明した。メモリライト要求は，CPUキャッシュの内容を全て無効にしてメモリにデータを書きこむ処理である。そのため，後続のバスリクエストは処理を進めることができずに，プログラム全体の処理時間が延びると考えられる。

(4) 処理性能比④(L-IBMとL-Sunの性能差)

L-IBMとL-Sunの処理性能比は，1:0.52であり，W-IBMとW-Sunの処理性能比よりも差がある。これは，(2)で述べたGC方式の差に加えて，(3)で述べた要因が含まれるためと考えられる。

表3に処理性能比の原因をまとめる。上述の解析結果から処理性能比①②の原因は特定できた。ところが③及び④に関して，なぜL-Sunではメモリライト要求が多発するのか，またGC処理とメモリライト要求の関連はあるのかという点については明らかでない。これらの不明点を明らかにするため，次章ではGC内部処理方式を解析する。

表3：処理性能比の原因

#	原因
①	マルチプロセッサOSの差
②	ガベージコレクション方式の差
③	メモリライト要求発生頻度差による単位時間あたりのプログラム処理量の差+①
④	メモリライト要求発生頻度差による単位時間あたりのプログラム処理量の差+②

4. GC内部処理方式の解析

4.1 GC内部処理方式

SunのJVMが採用する世代別GC方式では，各マイナーコレクションにおいて，Eden領域の新規オブジェクトのうちプログラムからアクセス可能なオブジェクトをSurvivor領域に再配置する[8]。Survivor領域に再配置されたオブジェクトは数マイナーコレクションの後，Perm領域に再配置される。再配置はメモリコピー処理により行なわれ，長時間生存したオブジェクトはEden→Survivor→Permへと順次コピーされる。

4.2 評価モデルの提案と性能評価

(1)オブジェクト生存率の定義

マイナーコレクションは新規オブジェクトに割り当てたメモリサイズが一定値を越えると発

生する。そこで，Javaの実行時オプションであるverboseオプションにより，評価プログラムにおけるメモリ使用状況を調べた。その結果，各マイナーコレクションは新規オブジェクトに2MBのメモリを割り当てると発生し，常に1151KBのメモリが回収されることが分かった。マイナーコレクション毎に，回収されずに生き残るメモリが一定の割合で増加するため，この割合をオブジェクト生存率と定義する。

$$\text{オブジェクト生存率} = \frac{\text{生存オブジェクトが割り当てられたメモリサイズ}}{\text{マイナーコレクションの対象となるメモリサイズ}}$$

(2)オブジェクト生存率に基づく評価モデル提案

SPEC JBB2000のオブジェクト生存率は0.437であり，ほとんどのオブジェクトが生成後すぐに捨てられるという世代別GCの前提とは異なることが判明した。従来のJavaは，サーバ単体で比較的小規模なプログラムを実行することを想定していたため，オブジェクト生存率が低い。ところがインターネットの普及によりJavaがより大規模なWebシステムで使用されるようになってきている。Webシステムで稼動するトランザクション処理は，複数のサーバに要求を出してその応答を待つステートフルモデルであるため本質的にオブジェクトの生存期間が延びる傾向にある。

そこで，OSの差異による処理性能比をオブジェクト生存率と性能の関係にマッピングする評価モデルを提案する。まず，オブジェクト生存率と性能の関係を調査するため，生存期間の長いオブジェクトと生存期間の短いオブジェクトの比率を変更可能な評価プログラムを作成し，L-SunおよびW-Sun上でプログラムの処理時間を計測した。図5に実行結果を示す。

オブジェクト生存率が1の場合，マイナーコレクションにおいてごみ集めの対象となるオブジェクトがないことを示す。図5より，オブジェクト生存率が低い場合は，OSの差異によらずプログラム処理時間は同じである。オブジェクト生存率が高くなると，処理時間が長くなり，OSの差異による性能差も1.5倍以上になる。評価プログラムから算出したオブジェクト生存率0.437のポイントでは，L-Sun/W-Sunの処理時間比が1.37でありSPEC JBB2000の中断間隔比にほぼ一致し，評価モデルが有効であることを確認した。

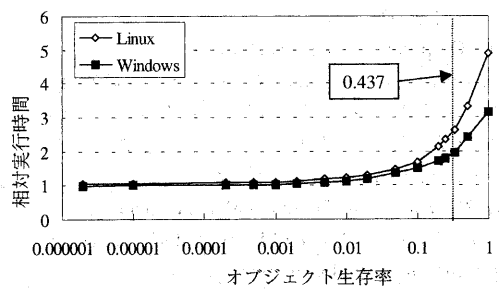

図5： オブジェクト生存率と相対実行時間の関係

(3)評価モデルによる処理性能比の原因特定

オブジェクト生存率が高い場合のOSの差異による性能差の原因を特定するため，再度バストレーサを用いて，バスリクエストの内訳を調査した。図6に，L-Sunにおいてオブジェクト生存率を変化させた場合のプログラム中断時間と中断間隔を示す。また，表4にメモリライト要求数とメモリライト要求により書きこまれたメモリサイズを示す。

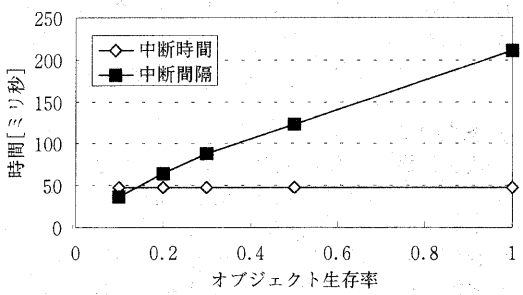

図6: オブジェクト生存率とプログラム中断間隔の関係

表4：ｵﾌﾞｼﾞｪｸﾄ生存率とﾒﾓﾘﾗｲﾄ要求の関係

ｵﾌﾞｼﾞｪｸﾄ生存率	ﾒﾓﾘﾗｲﾄ要求数	ｱｸｾｽﾒﾓﾘｻｲｽﾞ
0.1	6478	207KB
0.2	13082	418KB
0.3	21591	690KB
0.5	32513	1024KB
1.0	65025	2048KB

図6に示すようにプログラム処理の中断時間は，一定である。一方，プログラムの中断間隔はオブジェクト生存率が高い程長くなる。また，表4に示すようにオブジェクト生存率が高くなる程メモリライト要求数も増加し，メモリライト要求により書きこまれたデータ量はマイナーコレクションにより増加するメモリサイズにほぼ一致する。アドレス解析により，JVMがGCにおけるSurvivor領域→Perm領域へのコピー処理を通常

のプログラム処理と並行して行なっていることがわかった。L-Sun では，コピー処理にメモリライト要求とデータ読み出し要求を使用する。一方，W-Sun では，メモリライト要求に代わり，読み出し＆無効化要求とライトバック要求を使用する。Pentium アーキテクチャでは，メモリライト要求の発生により，後続バスリクエストの発行が抑制されるため[6]，結果としてプログラム処理全体が遅延する。一方，W-Sun で使用するバスリクエストは後続バスリクエストをブロックしないため，L-Sun よりも性能低下の度合いは小さくなる。

以上のように，オブジェクト生存率に基づく評価モデルを構築し，評価モデルをバストレーサにより解析することで，W-Sun と L-Sun の処理性能比の原因が GC 内部処理方式におけるコピー処理方法の差異であることを明らかにした。性能を改善するためには，プログラム処理と並行して行うコピー処理を削減するように GC 内部処理方式を改造するのが最善であるが，オブジェクトの生存率が低くなるようにソースコードレベルでのチューニングを行なうことでも対応可能である。

5. おわりに

機能仕様のみならず性能目標をも満足するアプリケーションサーバを短期間に開発するため，マルチプロセッサ環境の Java 仮想マシン(JVM)上で動作するアプリケーションサーバの性能特性を解析した。アプリケーションサーバは同一ハードウェアであっても搭載する OS や JVM の差異により性能が異なる。そのため，選択するプラットフォームによっては性能目標を達成できず，プログラム短期開発を阻害する要因になる。

本稿では OS および JVM の差異がアプリケーションサーバ性能に及ぼす影響を調査した。さらに，JVM の内部処理方式を解析するための評価モデルを構築し，評価モデルをバストレーサにより解析することで性能低下要因を特定した。評価では，マルチプロセッササーバに，OS として Linux あるいは Windows2000 を搭載し，その上に Sun 社と IBM 社が開発した JVM を搭載して，評価プログラムを実行した。

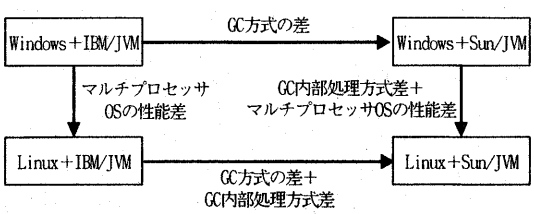

図7：性能支配要因まとめ

解析結果から，OS と JVM の組合せにより処理性能が最大 48%も異なり，各処理性能比の原因は図 7 に示す理由であることを特定した。マルチプロセッサ対応 OS の性能差は，Linux カーネルのバージョンアップにより改善可能である。また GC 方式の差も，マルチプロセッサ上で複数 JVM を起動することにより 10%程度吸収可能である。GC 内部処理方式の性能差を改善するにはベンダが提供する JVM を改造するのが最善ではあるが，プログラマによるソースコードレベルでのチューニングでも対処可能である。

上述のように解析結果は OS や JVM のバージョンによって変わりうる。しかしながら，プログラマが処理系により性能が異なることを意識してアプリケーションサーバのプラットフォームを選択することで，短期間で性能目標をクリアするアプリケーションサーバを開発できる。

謝辞

本研究を進めるにあたり有益な助言を頂きました日立製作所中央研究所　樋口達雄主任研究員に深く感謝いたします。

参考文献

[1] Standard Performance Evaluation Corporation: *SPEC JBB2000 Benchmark Documentation*: http://www.spec.org/osg/jbb2000/

[2] Transaction Processing Performance Council: TPC BENCHMARK C Standard Specification Revision 5.0 (2001)

[3] Sun Microsystems: *Java 2 Platform Standard Edition*: http://java.sun.com/j2se/1.3/

[4] International Business Machines Corporation: http://www.106.ibm.com/developerworks/java/

[5] 保田　他 2: 大容量サイクルベースバストレーサを用いた Web コンピューティングアプリケーションの特性解析: 第 64 回情報処理学会全国大会(2002)

[6] Intel Corporation: *Pentium Pro Family Developers Manual Volume 1, Specification* (1996)

[7] 平井　他 2：Web Server における Linux2.4 のスケーラビリティ-ボトルネック解析と改善：Linux Conference 2001(2001)

[8] Sun Microsystems: Tuning Garbage Collection with the 1.3.1 Java Virtual Machine: *White paper*: http://java.sun.com/docs/hotspot/gc/