関西国際大学研究紀要　第14号，2013年，231－241

WebClass 履歴データ処理ツールの開発について
－ツールの開発とソートアルゴリズムの改良による処理効率の向上－

Development of the Tool which Processes the Log-data of WebClass

山　下　泰　生*
YasuoYamashita

抄録

Web-Class の経年履歴データに対する分析処理を行うためのデータを抽出するツールを開発した。データ構造の制約や開発に費やすことができる時間的な制約もあり，CSV ファイル編集ソフトである Cassava のマクロ機能を利用して開発を進めた。

開発の最終段階で，本番データを使用したテスト時にレコードのソート処理に問題があることが判明し，理論的な検討をした上で，その改良を行った。その結果，処理時間は短縮され，一定の効果は見られたが，稼働するマシン環境による効果改善の違いがあり，複数のマシン環境でのベンチマークテストを行った上で，処理速度の計測結果と合わせてその原因を考察した。その結果 Cassava のマクロ機能での限界が確認された。

Abstract

The tool which extracts the data of Web-Class was developed using the macro function of Cassava which is CSV file editing software. The improvement was performed after it became clear that there was a problem in the sorting process of a record at the time of the test in acting-before-the-audience data and carrying out theoretical examination in the final stage of development. The findings show that although processing time was shortened and the fixed effect was seen, there was a difference in the effect of improvement by the machine environment which works. After the benchmark test in three machine environments was done, the cause was considered together with the measuring result of processing speed.

＊関西国際大学共通教育機構

１．はじめに（経緯と目的）

2008年度よりコンピュータリテラシー演習という授業のクラスプレースメントに Web-Class のテスト機能を利用してきた。

これまで，プレースメントの結果に対する分析は，その妥当性についての検証を中心として，ポストテストの結果との比較も含めてテストスコアの変化に着目した分析を進めてきた。また，ポストテスト時に同時に行った受講生の主観的評価も併せて，プレースメントの結果は良好であり，経年でのスコア自身も上昇していることが確認された。

現時点で 5 年間分のプレースメントテストの結果データが蓄積されている。今後は，これまで蓄積されているデータを利用して，受講生の動向の変化やプレースメントテストで使用している個々の問題に対する分析を進めていくことを課題としている。

Web-Class は，総合的なＬＭＳ (Learning Management System) であり，教材管理，コース管理，成績管理をはじめとして豊富な機能を有している。しかしながら，プレースメントテストで利用している成績管理機能では，テストの実施，採点，集計などの処理は行うことができるが，解答の正誤情報をもとにした受講生の動向や出題問題の分析などまではサポートされていない。そのような分析をするためには，テストを実施して，正誤判定をした結果のオリジナルの履歴テータを Web-Class から取り出し，独自の分析を行う必要がある。また，Web-Class にはデータをダウンロードする機能はあるが，必要なデータを選択して切り出す機能はなく，各種の集計結果に限定された個別データか全データを一括でダウンロードするかのいずれかである。これまでのプレースメント処理で使用してきたデータも一括ダウンロード機能を利用したデータである。今後の分析処理においてもオリジナルの履歴データを使用するためには，全データの一括ダウンロード機能を利用するしかないが，この履歴データは，当該テストに係るすべてのデータが対象となっており，成績データや正誤判定情報のみでなく，教材コンテンツ（テスト問題）情報なども全て同時に出力される。そのため，470名程度の受講生のテスト 1 回分のデータで，30,000レコード以上のデータセットとなる。

Web-Class の一括ダウンロード機能で出力されるファイルの形式は，CSV 形式のデータであるため，これまでのプレースメント処理も全データをダウンロードして MS-Excel 上で必要なデータ部分の抽出や連結を手作業で処理をしていた。プレースメント処理では，必要となる部分は限られており，ある程度ルーチン化しているが，それでも全てのクラスのプレースメントの処理に数時間の時間を要していた。

今後，バックアップをとっている過去のデータも含めて，別の視点での分析を進めるにあたって，その分析に応じて必要となるデータを簡単に抽出する仕組みの構築が必要となる。そのために，Web-Class からダウンロードした履歴データから，必要なデータ群を抽出するツールの開発を行うこととした。

さらに，抽出ツールの開発過程で作り上げたいくつかのパーツを利用して，データ抽出ツールだけでなく，今後の分析処理につなげるようなデータに変換するツールや，これまで手作業で行ってきたプレースメント処理を必要クラス数の入力だけで自動的に行うルーツも同時に開発した。

開発およびその実行には，Cassava というＣＳＶファイル編集ソフトのマクロ機能を利用した。開発作業のテスト段階でレコードソートにおける処理効率上の問題点が顕著になり，ツールの改

良を行った。その結果，件数が異なる複数のデータによる改良前後の処理時間の計測により改良効果は確認できたが，同時に稼働環境による差異も確認できた。

本稿では，開発したツールの概要，および，その改良に伴う複数のマシン環境での稼働結果の違いに対して，ベンチマークテストの結果を踏まえた上での改良内容に対して考察をする。

2．ツールの開発

2.1 開発環境

Web-Class の履歴データは CSV 形式のデータであるため，ツールの開発は，当初，MS-Excel の VBA での開発を想定していた。しかし，開発用の PC に導入されていた MS-Excel のバージョンが2003で Web-Class の履歴データを取り込んだ際，列数の制限でデータの全てをシート上に展開することができなかった。つまり，部分的にデータの欠落が発生したということである。これまでのプレースメント処理では欠落部分は使用していなかったため問題はなかったが，データを抽出するツールの開発という点では問題である。

そこで，マクロ機能を持った CSV ファイルの編集用ソフトである Cassava を利用することにした。Cassava の特徴は，Windows 上であればプラットフォームを選ばす，マシンにインストールする必要もない。したがって，Cassava の実行ファイルとマクロプログラムファイルと対象データが揃っている環境であれば稼働可能である。また，CSV ファイルの編集ソフトであるため，複数のシートを同時に処理することはできないが，シートのサイズに制約はないことが Cassava を選択した大きな要因である。

Cassava のマクロ機能におけるプログラミング言語としての構文構造は，C 言語や JavaScript 言語と類似しており，プログラム上にユーザ独自の関数を組み込むことも可能である（ただし，再帰呼び出しには未対応と思われる）。そのマクロ機能を利用して Web-Class の履歴データを処理する独自のツールを開発した。

2.2 開発方式

Cassava のマクロプログラムの実行は，インタプリタ形式で稼働するため，部分的な開発が可能な環境であった。また，開発作業は，筆者単独で，休日などの空き時間を利用して進めてきたこともあり，プロトタイプ方式の開発手法で進めてきた。

ウォーターフォール方式の開発は，作業工程をたてて段階的に開発を進める方式であり，作業計画に対する進捗管理が容易である。そのため，プロジェクトでの開発作業に適している方式であるといえる。しかし，今回は，単独の開発であり，マクロプログラムを部分的にパーツとして組み立てて，結合していく方式で開発を進めたため，プロトタイプ方式による開発が適していると判断した。しかしながら，そのために正確なドキュメンテーションが残されていない点は問題であり，開発期間も結果的に 4 か月ほど要している。

2.3 Web-Class 履歴データのデータ構造

Web-Class の一括ダウンロードで出力される履歴データの構造については，マニュアル等での記載はなかったため，ツールの開発に先立ち，実際に出力されたデータの内部構造を解析する必

表2-1　WebClass の履歴データ構造

データ群	選択対象
①[基本情報] 作成日時分、問題名、実施日、コース名	－
②[問題情報] 問題データ（問題、解答選択肢、回答方式、正答、配点、・・）	－
③[成績一覧] （コース、氏名、ユーザID、解答日、解答時刻、**得点**、得点率、偏差値）	◎
④[分析結果] （対象人数、平均値、最大値、最小値、中央値、標準偏差） （問題別：正答率、最小解答時間、平均回答時間、最大解答時間）	◎
⑤[ユーザ毎の解答リスト] （コース、ユーザ名、ユーザＩＤ、解答時刻、問題別解答値）	◎
⑥[ユーザ毎の解答時間リスト(単位:秒)] （コース、ユーザ名、ユーザＩＤ、解答時刻、問題別解答時間）	◎
⑦[解答数リスト] （問題別解答数）	－
⑧[解答の正否リスト] （ユーザ毎の問題別正否（○、×）データ）	◎
⑨[個別全解答データ履歴]	－

要があった。

その結果，Web-Class の履歴データの構造は，表2-1に示すように 9 つのデータ群の集合体となっており，各データ群固有のキーワードが，データ群の先頭に挿入されていることが確認できた。そのキーワードを識別することにより，必要とするデータ群の抽出が可能であることを利用して抽出ツールの開発を進めた。

2.4　開発ツールとシステム構成

まずは，表2-1における 9 つのデータ群の中から今後の分析処理に必要な抽出対象データ群として，③［成績一覧］，④［分析結果］，⑤［ユーザ毎の解答リスト］，⑥［ユーザ毎の解答時間リスト（単位：秒)］，⑧［解答の正否リスト］の 5 つを対象として，抽出するデータ群を実行時に番号で指定することで抽出する機能とした。

また，部品を作り上げていく方式で開発を進めたため，他のツールの開発も容易となり，別ツールとして，③［成績一覧］，⑥［ユーザ毎の解答時間リスト（単位：秒)］，⑧［解答の正否リスト］の 3 群のデータに対して，ユーザ（受講者）をキーに連結するデータ変換ツールも開発した。それにより，ユーザ毎の問題別解答時間と正誤関係の分析のためのデータ作成が可能となった。

さらに，学科毎の開講クラス数を入力することで，クラスプレースメントを自動的に処理するツールも開発した。これまでのプレースメント処理では，③［成績一覧］データ群の「得点」のデータのみを利用して処理をしていた。そのため，データ切り出し操作自身は比較的単純であったが，データ量が膨大であるためプレースメント作業全体としては，かなりの時間を費やしていた。プレースメント作業の効率向上のため，「得点」データをもとにクラス設定をするプレースメントのツールも開発した。

最終的には，以下に示す 3 つのツールと 5 つの部品の計 8 ツールを開発した。

ⅰ）「必要データ群抽出ツール」（data-ext）
ⅱ）「分析用データ変換ツール」（data-conv）
ⅲ）「クラスプレースメントツール」（placement）
ⅳ）その他のツール（部品）

・最下位ポジショニングテスト用ツール，・レコードソート開発チェック用ツール
・解答正誤リスト部分行・列入れ替えツール，・学科別順成績一覧作成ツール
・クラス設定ツール

部品であっても Cassava 側ではインタプリタ形式で実行するため，マクロプログラムとして単独の稼働が可能である。

マクロプログラムの実行時は，まず Cassava 起動後に対象となる Web-Class の履歴データ（CSV ファイル）を開き，その後，処理を行うマクロプログラムを選択する。Cassava 上でマクロプログラムが実行されると，開いているＣＳＶファイルに対して直接処理がなされる。処理終了後のデータに対してそれぞれのツール自身にファイル保存機能は組み込んでいないため，処理終了後，Cassava 本体のファイル保存機能を利用して実行結果を保存する必要がある。Cassava を中心とした，開発ツールのシステム構成図を図2-1に示す。

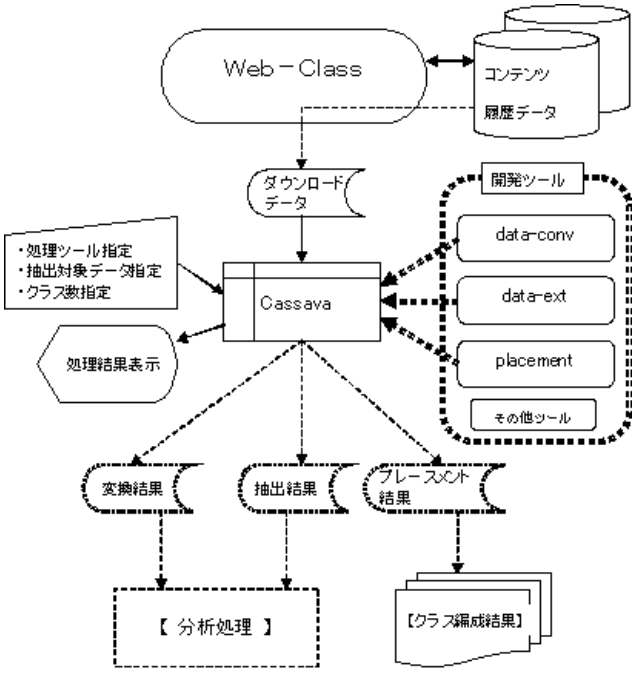

図2-1　システム構成図

3．開発ツールの問題点とツールの改良

3.1　開発ツールの問題点

本ツールの開発を進める上で，Cassava のマクロ機能としての問題点がいくつかあった。一例をあげると，レコード処理に関して，複数レコードを同時に削除する関数やレコードソート機能の関数がサポートされておらず，その機能は独自にマクロプログラムに組み込む必要があった。そのため，レコード削除に関しては，１行削除の関数をループ処理で実行するしか方法はなく，レコードソートに関しても，アルゴリズムレベルでの独自の組み込みが必要であった。

Cassava の標準関数に Sort 関数はあるが，マクロプログラム上でその関数が呼ばれた時は直接データソートを実行するのではなく Cassava 本体のソートメニューが選択された状態となり，ソート条件を指定するダイアログボックスが表示される。そのため，Sort 関数呼び出し時のオペランドとして，ソートキー，ソート範囲，並べ替え順などのソート条件をマクロプログラム上で直接指定することができない仕様となっていた。

レコードソート処理に Sort 関数を利用することができないため，指定された 2 つのセル同士のデータを交換する Swap 関数を使って，レコードソートを処理する独自のソート関数を開発してそれを部品としてプログラム上で利用した。

初期開発段階において，27名分のテスト用データを使用した主要ツールの全てのテストが終了した後，過去の実データ（468名分）で実行したところ，特に「分析用データ変換ツール」の処理時間が，テスト用データでは 3 分程度で終了していたところが15分以上の時間を要した。この原因が，データ件数の差だけでなく，独自開発のソート関数にあることは，ある程度の見当はついていた。それは，ソート機能を使用していない「必要データ群抽出ツール」では実データでもデータ件数の差以上に処理時間の顕著な差はなかったからである。

独自のソート関数を開発する際，Swap 関数の利用によるオーバーヘッドは気にはなっていたが，当初は，あまり処理効率を意識はしていなかったため，ソートアルゴリズムとして，最もシンプルなバブルソートのアルゴリズムで組み上げていた。

しかしながら，ソートの対象データがレコードであるため，バブルソートの場合，レコード数だけでなく 1 レコード内のフィールド数（この場合列数）によって Swap 関数の使用回数が極端に異なってくる。その影響が開発当初の想定以上であったと考えられる。

同じソート関数を使っている「クラスプレースメントツール」では，異なるソートキーで 4 回同じ関数を使用しているが，レコード内のフィールド数は最大10ということもあり，極端に処理時間が長くなることはなかった。

それに対して，「分析用データ変換ツール」は，複数のデータ群を連結して，その全体をソート対象としているため，レコード内のフィールド数は42となり，極端に Swap 関数によるオーバーヘッドの影響を受けることになったと想定される。

3.2　ツールの改良

Cassava にレコードの交換関数がサポートされていない以上，Swap 関数のオーバーヘッドは避けられないが，ソートの処理効率を向上させることで，全体的な処理効率をあげることは可能である。基本的な考え方は，レコード交換処理回数を減らすことで，Swap 関数自身の使用回数を減らす方法である。

ソートアルゴリズムには，ヒープソート，クイックソートなど効率のよいアルゴリズムが存在する。しかしながら，改良するための作業の効率を考え，データ全体がテーブル構造であることを利用して，簡易なインデックスソートのプログラム構造へ改良することとした。その改良方式は，バブルソートでのソートキーの判定結果に対して，その都度レコード交換を行うのではなく，判定対象となる全レコードのキー判定が終了するまで，交換候補レコードのキーインデックスを保持し，各レコードのキー判定処理終了後にキーインデックスが更新されている場合（交換が必要である場合）にのみ該当レコードの交換処理を行う方式である。

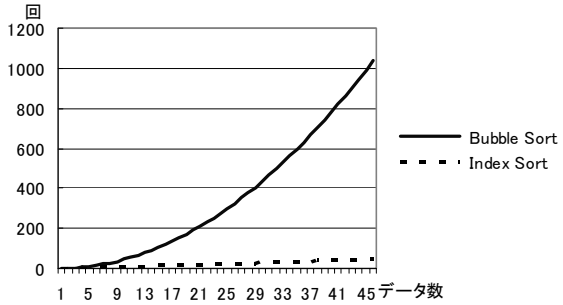

図3-1　バブルソートとインデックスソートのデータ交換処理回数比較

この改良方式では，レコード数を n としたレコード交換処理回数に着目した場合，理論値としてのレコード最大交換回数を改良の前後で比較すると，バブルソート（改良前）では n(n-1)/2回で，オーダーとして $O(n^2)$ となり，改良版インデックスソートでは，(n-1) 回で，オーダーは $O(n)$ となる。

バブルソートとインデックスソートとを比較したデータ件数による最大交換回数に対する理論値としての違いを図3-1に示す。

図3-1により，バブルソートの場合，データ件数の増加にともない急激にレコード交換処理回数が増加することがわかる。理論値としては，データ件数が100件の場合，バブルソートでの最大交換処理回数は4,950回であるのに対し，インデックスソートでは99回である。ツール開発のテストで使用したデータのレコード件数は27件であり，件数による影響は表面化されなかったことになる。さらに，実際の Swap 関数の使用は，レコード交換回数にフィールド数を乗じた回数となるため，レコード交換処理回数の減少だけでもかなりの効果が見込めることになる。

4．ツール改良の検証結果

前述の改良に対する根拠はあくまでも理論値によるものであり，改良効果については，実際の処理時間を計測して検証する必要がある。

Cassava の大きな特徴の一つとして，プラットフォームを選ばないという点をあげている。たしかに，開発したツールも含めて，テスト段階でも異なる Windows マシン上で問題なく稼働したが，体感的に処理時間に差を感じていた。そこで，いくつかの異なるマシン環境での，ツール改良効果に関する検証も併せて行った。

4.1　改良効果の計測

効果測定に使用するデータは，ツール開発のテストで使用したデータ（D 1）と2011年度のプレースメント時（プリテスト）のデータ（D 2）と授業終了時のポストデータ（D 3）の 3 種である。各データの受講者数は，D 1 が27人分，D 2 が296人分，D 3 が468人分のデータである。

表4-1　効果測定で使用したデータ

No	概要	受講者数
D1	テストで使用したデータ	27
D2	2011年度のプレースメント時（プリテスト）のデータ	296
D3	2011年度の授業終了時のポストデータ	468

（表4-1）

また，ツール改良の効果計測にあたって，対象とするマシン環境を表4-2に示す 3 機種とした。同系統のＣＰＵの機種を選択することにより，異なる Windows 機種間でも，ベンチマーク値の相対比較が可能となるようにした。

表4-2　効果計測用マシン環境

マシン環境	Windows	CPU（クロック周波数）	RAM
m1	WindowsXP	Intel CoreDuo（1.06GHz）	1.5Gbyte
m2	WindowsVista	Intel Core2　（ 1.2GHz）	2.0Gbyte
m3	Windows7	Intel Corei7　（ 3.4GHz）	12.0Gbyte

表4-1に示す各々のデータに対して，表4-3のマシン環境ごとに「分析用データ変換ツール」を実行し，特にソート処理に関して，バブルソートと改良版インデックスソートの各々の処理時間を計測して比較検討をした。

しかし，Cassava に処理時間を計測する関数はサポートされていないため，マクロプログラム上で改良対象のソート関数を使用する前後に，システム時刻を取得する関数（GetMinutes, GetSeconds）を使用してソート処理の前後の時刻を取得し，その差で所要時間を算定した。計測処理を複数回実行して平均時間を算出したが，マシン時刻を取得する関数のデータ精度が秒単位までであるため，小数点以下一桁目を切り捨てた数値を計測結果とした。

各データ別のマシン環境のごとのソート処理時間計測結果を表4-3に示す。

表4-3　ソート処理時間計測結果　（Sec）

	Machine：		m−1		m−2		m−3	
Data		n	Bubble Sort	Index Sort	Bubble Sort	Index Sort	Bubble Sort	Index Sort
テスト用データ	（D1）	27	9.0	2.0	13.0	2.0	5.0	1.0
2011年ポストデータ	（D2）	296	143.0	5.0	167.0	6.0	106.0	3.0
2011年プリデータ	（D3）	468	558.0	9.0	640.0	9.0	467.0	4.0

表4-3より，明らかな改良効果が確認できた。当然のことではあるが，データ件数（レコード件数）が多いほど，その効果は顕著に表れている。

4.2　稼働環境の相違による改良効果の差

ソート処理時間の計測結果から改善効果は確認できたが，その結果をマシン環境の違いによる効果の差を照らし合わせて検証することにした。それは，表4-2に示すマシン環境として一般的なスペック順としては低い方から m1 < m2 < m3 であるが，実際の計測結果がマシンスペックの高低関係とは異なっている部分があったからである。

表4-3の計測結果をもとにして，マシン環境毎のデータ件数別処理時間としてバブルソートとインデックスソート各々の結果を図4-1に整理をした。

図4-1では，バブルソート（左グラフ），インデックスソート（右グラフ）各々の横軸が処理時間であるが，データのオーダーが異なるため目盛りのスケールは異なっている。しかし，マシン種別でデータ件数毎の処理時間の大きさをみると，特に m1と m2の環境では，もう少し差が顕著であってもよいどころか，所要時間が逆転をしている状況も見受けられる。

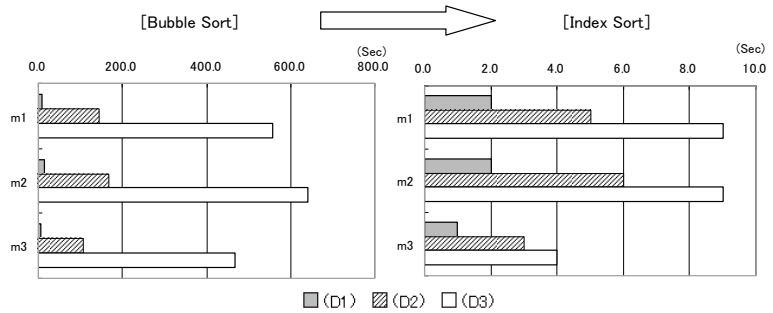

図4-1　効果計測結果

このような状況の原因として想定される点は 2 つある。一つは，ＯＳ自身のオーバーヘッドであり，もう一つは，メインメモリとレジスタ間の内部バスのデータ転送スピードである。特に後者は，ハードウェアにも大きく依存するため，機種固有の条件ともなりうる。

その点を確認するために，3 つのマシン環境について，ベンチマークテストを行った。プロセッサーの系統が異なる場合は，ベンチマークによる比較はあまり意味があるとはいえないが，同一のＣＰＵファミリーのプロセッサーの環境であるため，結果に対する相対的な比較はベンチマークの結果より可能である。

ベンチマークテストには，CrystalMark 2004R3 というベンチマークソフトを利用した。CrystalMark 2004R3 は，詳細なシステム情報収集機能を備えた総合ベンチマークソフトである。

3 つのマシン環境それぞれで，ベンチマークテストを実行した。その結果を表4-4に示す。今回のベンチマークの場合，ソート処理の時間に関する検証であるため，ベンチマークの結果の中で，浮動小数点演算性能，グラフィックス性能，ビデオチップ性能関係の数値は省略している。

また，常識的なことではあるが，ベンチマークテスト実行結果の数値は，それぞれのソフト固有の値であり，一般的な基準値や単位があるわけではない。したがって，ベンチマークテスト結果の数値に関する絶対的な比較は意味を持たず，相対的な評価に意味がある。そのため，異なるベンチマークソフトを使用した比較検証も無意味であり，同一ソフトでの実行結果に対する相対的な比較検証が基本となる。

表4-4おいて，「ALU」はＣＰＵ性能に関するベンチマークで，その中で「ALU(QS)」は内部でクイックソートを実行した結果のベンチマークである。「MEM」はメモリ性能に関するベンチマークで，「MEM(R/W)」はメモリのＩ／Ｏ，「MEM(Cache)」はキャシュメモリ性能である。「HDD」はハードディスク性能のベンチマークである。そして，最上段の「Total」は，省略され

表4-4　ベンチマークテスト結果

B.M.	m1	m2	m3
・Total	33,740	33,755	217,197
・ALU	8,277	10,636	73,981
ALU （QS）	2,280	2,774	22,438
・MEM	9,877	5,352	48,474
MEM （R/W）	1,083	862	8,896
MEM （Cache）	1,102	1,520	12,621
・HDD	3,557	2,368	12,559

ている浮動小数点演算やグラフィック性能等のベンチマークも含めた総合ベンチマークの値である。各々のベンチマーク値は，数値の大きい方がより高性能であることを意味する。

m1と m2のベンチマーク値を比較すると，ＣＰＵ性能は相対的に m2の方が高くなっているが，メモリ性能は逆に m1の方が高くなっている。また，m2のメモリ性能の項目中でも，キャッシュメモリのベンチマークは m1より高くなっているが，Ｉ／Ｏ性能は逆に低くなっている。

また，m3のベンチマークは，m1，m2と比較するとかなり高い値となっているが，図４－１より，実際の処理時間の効率は，バブルソートの場合もインデックスソートの場合もベンチマーク比ほどの差があるような状況とはなっていない。

５．考察

ツールの改良について，ソートアルゴリズムの簡易な変更により，その処理時間は，かなり短縮されたことが確認でき，その点で一定の改良効果はあったと判断できる。

しかしながら，異なるマシン環境での処理時間の計測結果とベンチマークの結果から内部バスのデータ転送速度に影響を受けている可能性が高いことが予測される。

仮に Cassava 実行時におけるデータメモリ空間上の配置が表形式データの列方向にアドレッシングされているとすれば，フィールド数が多いレコードに対するセルデータの交換処理時のアドレス演算とメモリ－レジスタ間のデータロードとストア処理に影響が出てくることが想定される。また，CPU のページング効率に影響されているとすれば，CPU アーキュテキュチャーに依存することも考えられる。

この点を詳細に確かめるためには，Cassava の実行ファイルの機械語プログラムを解析するしかないが，仮にそれが判明したとしても，エンドユーザサイドでは，それ以上の対応は不可能である。

ソートアルゴリズム自身をヒープソートやクイックソートに変更することによるさらなる効率化も考えられるが，今回の検証結果より Cassava のマクロ機能を利用する以上，その制約があることは否定できない。まず，独自関数の再帰呼び出しがサポートされていない場合は，クイックソートの実装は困難である。Cassava のマクロでは，関数呼び出し時の引数のスタッキング機能がないようであるので，再帰呼び出しはサポートされていないと思われる。そうであれば，クイックソートへの改良は困難である。また，レコード単位の交換処理を直接行う関数がないために，どうしても項目間でのデータ交換を行う swap 関数を利用せざるを得ず，そのために，ソートアルゴリズムを変更しても，肝心なレコード交換に負荷がかかるため効率化には限界があると考えられる。

６．おわりに

Web-Class の履歴データから必要なデータを抽出するためのツールを開発したが，部品を組み上げるプロトタイプ方式で開発を進めた。そのため，結果的にデータ抽出ツールを含め 3 つのツールを開発した。

そのツールの中に，手作業で行ってきたプレースメントを学科別のクラス数を入力するだけで，

自動的に行うツールがあるが，これまで 2 〜 3 時間かけて行ってきたプレースメントの作業が，10分程度で完了するくらいに効率があがった。

ツールの開発過程で，レコードソート処理効率に問題があったため，その改良をおこなった。処理時間の短縮という点では，大きな効果が確認できたが稼働環境による違いも否定できない。そのため，異なる環境のベンチマークテストの結果も踏まえて検証を試みた。その結果，内部バスのデータ転送スピードに影響されている可能性があり，ツール自身の効率化には限界があることが確認できた。

しかしながら，今後，プレースメントテスト結果のデータを使った分析研究を進めるにあたり，対象データの変換処理が自動化された点は，大きな改善であるといえる。

【参考文献】

1）二村良彦著：「プログラミング技法」オーム社，1984.10
2）浪平博人著：「学習コンピュータ・アルゴリズム」技術評論社　1989.5
3）Brian W. Kernighan, P. J. Plauger: “SOFTWARE TOOLS”, ADDISON-WESLEY, 105-134, 1976
4）Wllis HOROWITZ, Sareaj SAHNI : “FUNDAMENTALS OF DATA STRUCTURES”, PITMAN, 335-377, 1976
5）Robert R. AMOLD, Harold C. Hill, Aylmer V. Nichols : “MODERN DATA PROCESSING THIRD EDITION”, JOHN WILEY & SONS, 257-259, 1978

【参考 URL】いずれも2012年 6 月時点

あすかぜ・ねっと：http://www.asukaze.net/soft/cassava/
Cassava マクロリファレンス：http://www.asukaze.net/soft/cassava/help/macroref.html
CrystalMark 2004R3：http://crystalmark.info/software/CrystalMark/