パスワード埋め込み画像を用いた
Drag & Drop ログインシステムの実装と評価

島根大学　総合理工学部　数理・情報システム学科

計算機科学講座　田中研究室

Ｓ０３３０８１　的場祥仁

平成２０年３月４日

目次

第１章　序論

近年、光ファイバーや ADSL などのブロードバンドが非常に発展してきており、インターネットが一般に普及した。セキュリティを考慮するため、ユーザ認証として ID とパスワード（ログイン情報）を用い、ユーザ毎のサービスを提供する Web サイトが少なくない。しかし、パスワード認証が増えるにつれて、ユーザのログイン情報の暗記・入力の負担も増加する。

長いパスワードを利用すれば当然高いセキュリティを実現可能である、しかし長いパスワードを設定すると、それを暗記する事は困難である。また、長いパスワードを設定したとしても、それをメモすると無意味なものになってしまう。

そこで本研究では、ログイン情報を埋め込んだログイン画像を用いる事により、ユーザの負担を軽減、さらによりセキュリティの向上を図るための手段の検討を行う。

先行研究において喜代吉容大[1]は 24bit の BMP(bitmap)画像へログイン情報を埋め込み、そのログイン画像を Web サーバへ送信する事で認証を行う先駆的なシステムを実装している。しかし、「24bit の BMP 画像はファイルサイズが大きく Web 上で利用するには向いていない」、「予め指定された画像の中からログイン画像とする画像を選択する」、「クライアントに不正にログインした人物が、手当たりしだいに Web サーバへ画像を送信するとログインされてしまう」などの問題が挙げられる。

本研究では上記のような問題を解決しつつ、より利便性やセキュリティ強度の高い、実際に使用が可能となるシステムの構築を目指す。

第 2 章　ログイン画像

本システムではログイン情報を埋め込んだログイン画像を用いる。ログイン画像を作成する際にステガノグラフィという技術を用いる。

2.1　ステガノグラフィ

ステガノグラフィ[2]とは、画像や動画、音声などのマルチメディアデータに、画質や音質にほとんど影響を与えずに秘密情報を埋め込む技術のことである。技術的には同一の電子透かしという技術が存在するが利用目的が異なっている。電子透かしは一般的に著作権情報の埋め込みに用いられている。

2.2　使用する画像の形式

先行研究においては 24bit BMP 画像へのステガノグラフィを行っていたが、Web 上で利用されること、ユーザの嗜好性を考慮することなどを考え、JPEG 画像を利用することにした。

現在、携帯電話のカメラ機能やデジタルカメラなどでは一般的に JPEG 画像が用いられている。このようなデバイスに対応できる JPEG 画像を利用することによって、よりユーザの嗜好性を考慮した独自のログイン画像を作成できると考えられる。例えば、デジタルカメラで撮影した自分の子供の画像をログイン画像として利用する事が可能となる。さらに JPEG 画像は圧縮されているため、BMP 画像よりも Web 上で利用するために向いている。

2.3　隠蔽アルゴリズム

JPEG 画像[3]へのステガノグラフィに関しては、利用用途は異なるが高木明[4]が実装を行っている。このプログラムを本システムでの利用が可能となるように改良を行った。

この隠蔽アルゴリズムでは、周波数領域における量子化誤差を利用する方法のひとつとして、量子化テーブルへの隠蔽を行う。この方法の特徴として、隠蔽可能なビット数は少ないが、画像が加工されたりしない限り必ず復号できるという点がある。本研究で埋め込むログイン情報は大きなデータではないため、隠蔽可能なビット数が少なくても問題はない。また抽出が簡単という点が挙げられる。ステガノグラフィの考え方としては、頑健性の低い隠蔽方法ということになるが、本研究のシステムではログイン処理時間がかからないという点で有用である。隠蔽アルゴリズムに概観を図 2.1 に示す。

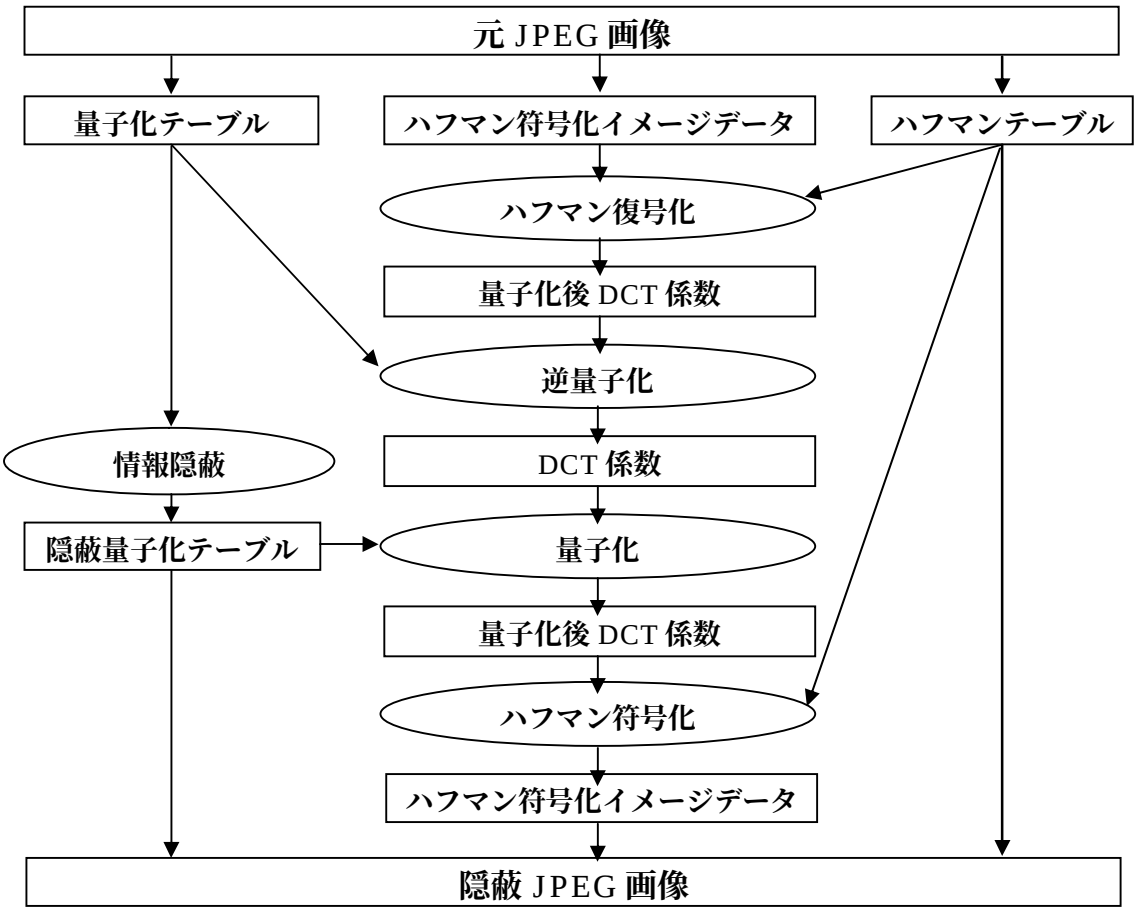

図 2.1　隠蔽アルゴリズムの概観

この隠蔽方法はスタンダード方式の 24bit フルカラーJPEG 形式にのみ対応している。今回は 24bit フルカラーJPEG 形式の画像を用いるという前提でシステムの実装を行った。よってそれ以外の画像を用いた場合の動作は保障できない。理想としては、24bit フルカラー JPEG 形式でない場合、隠蔽処理を行う前に 24bit フルカラーJPEG 形式に変換する処理を行うべきである。隠蔽には JPEG 圧縮されたイメージデータを量子化前の DCT 係数にまで復号を行う必要がある。

まず JPEG 画像の各セグメントより量子化テーブル、ハフマンテーブル、及びその他の復号に必要な情報を取り出す。そのあと JPEG の復号ステップのハフマン復号化、逆量子化を行う。これによってイメージデータは DCT 係数にまで変換される。

その後量子化テーブルに対してログイン情報の隠蔽を行う。隠蔽は量子化テーブルの各要素の最下位１ビットに対し、画素置換型ステガノグラフィと同様の方法を用いて隠蔽を行う。

隠蔽の例を示す。例えば”t”という文字を隠蔽する場合、”t”は ASCII コードで 0111 0100 であるから、これを量子化テーブルの最下位ビットと置き換える（図 2.2）

隠蔽前量子化テーブル（抜粋）

16	11	12	14	12	10	16	14	
0001 0000	0000 1011	0000 1100	0000 1110	0000 1100	0000 1010	0001 0000	0000 1110	
0	1	1	1	0	1	0	0	隠蔽文字 ”t”
0001 0000	0000 1011	0000 1101	0000 1111	0000 1100	0000 1011	0001 0000	0000 1110	
16	11	13	15	12	11	16	14	

隠蔽後量子化テーブル

図 2.2　隠蔽の様子

以上のように 1 文字隠蔽するために 8 個の量子化テーブルの要素が必要となる。量子化テーブルは 8x8 であるので 1 枚の量子化テーブルに 8 文字が隠蔽可能となる。通常のカラー画像の場合、量子化テーブルは輝度と色差の 2 枚が存在するため、合計 16 文字の隠蔽が可能となっている。本研究では ID とパスワードをそれぞれ 8 文字とし、隠蔽を行う。

量子化テーブルへの隠蔽後、ログイン情報が隠蔽されたテーブルを用いて JPEG 画像を再符号化する。つまり量子化、ハフマン符号化を行う。本来ここでハフマンテーブルを再設定する必要があるが、

・一般的にハフマンテーブルはデフォルトのものが使われることが多い

・データの特性がほとんど変わらないためハフマンテーブルに大きな変更は無い

の 2 点から本システムではハフマンテーブルは元の JPEG 画像のものをそのまま使用して符号化を行う。

ただ量子化テーブルに隠蔽するのではなく、イメージデータを DCT 係数にまで復号する理由は、量子化テーブルのみを変更すると実際に画像を表示した際、元の画像と異なったイメージで再生されるためである。実際にログイン情報隠蔽を行った画像の例を図 2.3 に示す。

隠蔽前

隠蔽後

図 2.3　元画像と隠蔽画像の比較

2.4　抽出アルゴリズム

本研究で採用したステガノグラフィはイメージ部分に手は加えるものの、隠蔽されたログイン情報自体は量子化テーブルに保存されている。そのため抽出の際にはイメージデータなどを必要とせず、量子化テーブルのみを解析する事でログイン情報を抽出する事が可能である。この点は前述した通り、処理時間の軽減を可能にしている。

JPEG のヘッダは<SOI>,<EOI>はマーカのみから構成され、それ以外のマーカは情報の長さとマーカ情報から構成されている。そこで DQT（量子化テーブル定義セグメント）以外のセグメントは、マーカの次に登場するそのセグメントの長さを取得し、その長さだけ読み飛ばす。このセグメントの長さはマーカを除く情報の部分の長さであり、長さ情報の部分も含むため取得された長さから-2 バイト数だけ読み飛ばす。

DQT セグメントが出現するとそれを保存する。量子化テーブルが 2 枚出現すると即座に解析が行われ量子化テーブルかログイン情報の抽出を行う。

抽出では隠蔽とは逆に最下位ビットを順に取り出しそれらを 8 個単位で結合していく事でログイン情報が取り出せる。

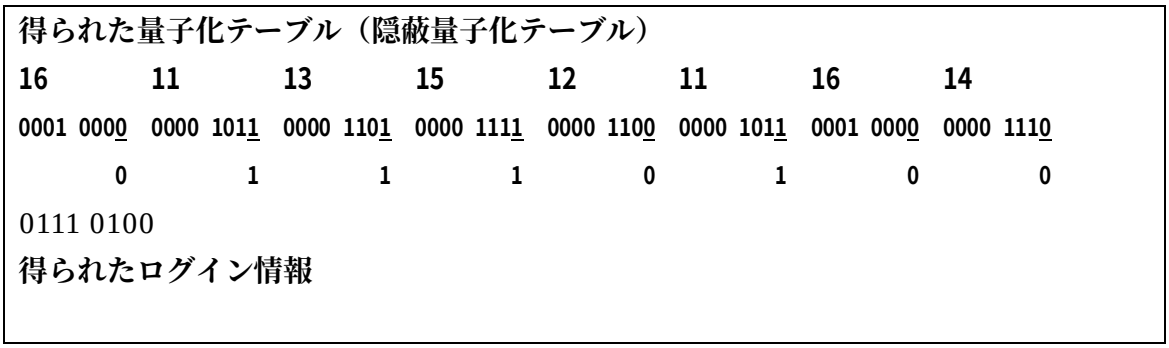

図 2.4　ログイン情報抽出の様子

得られた 0111 0100 は ASCII で”t”となる。これを量子化テーブル二つ分行い、ログイン情報を抽出する。

第 3 章　実装

3.1　システム概要

本研究のシステムではユーザはログイン情報を Web サーバに送信する事で簡単にログイン可能である（図 3.1）。またログイン情報は画像に埋め込まれているため、ユーザがログイン情報を知る必要がない。アカウント作成処理において、ユーザにはログイン情報を通知せずに、ログイン画像のみを発行して渡すものとする。

従来のパスワード認証と大きく異なる点として、外出先の PC からログインする場合は、ログイン画像を持ち歩く必要があるということである。近年フラッシュメモリなどで大容量のデータを容易に持ち運ぶことができるようになってきたため、この点は問題ない。

他人がログイン画像を見ても、一般的な普通の画像にしか見えず、それがログイン画像であると気づく可能性は低いと考えられる。しかし他人がログイン画像を用いて、不正アクセスを行う可能性がゼロであるとは言い切れない。例えばクライアント上の画像ファイルを手当たりしだいに Web サーバへ送信する事で不正にアクセスされてしまう可能性がある。この対策に関しては 4 章で述べる。

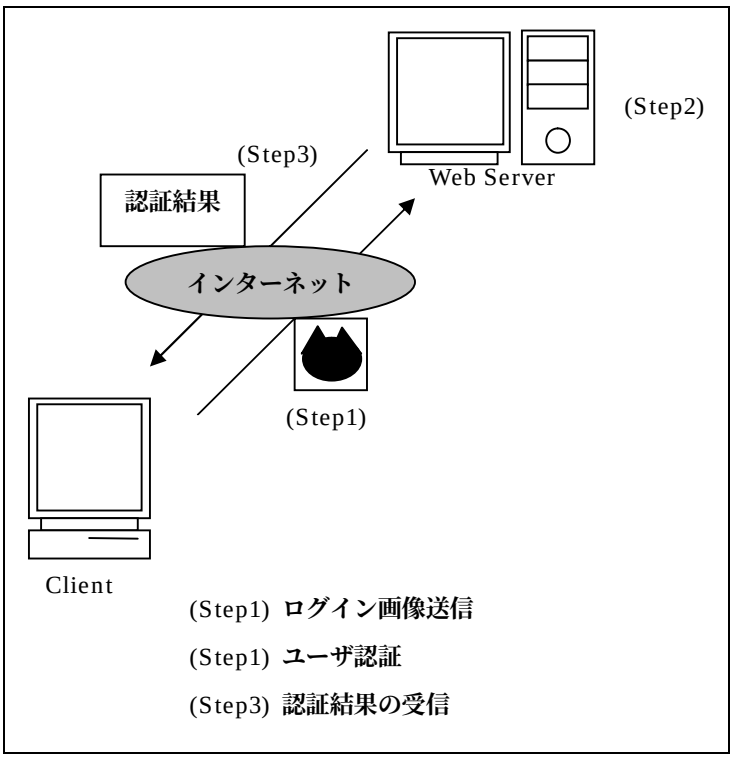

図 3.1　ログインシステム概要

3.2 システム構成

本研究におけるログインシステムは、「新規アカウント作成処理」と「ログイン処理」の 2 項目で構成される。それぞれの処理内容を示す（表 3.1）。

表 3.1 サーバ処理内容

処理	内容
新規アカウント作成処理	新規アカウント作成からクライアントにログイン画像をダウンロードさせるまでの一連の処理
ログイン処理	ログイン画像受信からユーザ認証結果をクライアントに返すまでの一連の処理

上記のログインシステムを作成するにあたって、Java Servlet（以下 Servlet）を用いた。Servlet とは、Web サーバ上で実行される Java プログラムの事である。類似するものとして CGI(Common Gateway Interface)が存在する。Servlet はサーバ上に常駐するが、アクセスが集中した際には CGI より処理が軽いという利点がある。

Servlet を利用するために、Jakarta Project の Tomcat[5]を使用した。通常ブラウザからのリクエストに対して Apache などが静的処理を行う。Tomcat と Apache を連携させる事により、Servlet へのリクエストを Tomcat 側へ送信し、Tomcat で動的処理を行わせる事が可能である。つまり Apche で静的処理、Tomcat で動的処理を行う。

ユーザのアカウント情報を管理する DBMS(Data Base Management System)として Hitachi の HiRDB を使用した。Servlet から DMBS へアクセスさせるために、開発に JDBC(Java Data Base Connectivity)を用いた、JDBC の詳細については後述する。今回用いたシステムの環境を表 3.2 に示す。ただし、第 5 章で後述するサーバ機 vajra に関しては OS に Windows 2003 Server R2 Standard Edition を用いている。

表 3.2 システムの環境

OS	Windows 2000 Server SP4
Web Server	Apache 2.2
Java Servlet Server	Tomcat 5.5
DBMS	HiRDB 7

3.3　インタフェース

本システムの処理はブラウザ上から行われる。新規アカウント作成処理のログイン画像作成時、ログイン処理の認証処理時に画像ファイルをクライアントから Web サーバへ送信する。先行研究においては、ブラウザ上からの一般的なファイル送信のフォーマット、つまり HTML の form 送信を用いている(図 3.3.1)

図 3.3.1　ブラウザ上からのファイル送信フォーマット

本研究では、より視覚的な操作が行えるようにブラウザ上へ画像ファイルの Drag & Drop を行う事でファイル送信が可能となるインタフェースを構築する。ブラウザ上に図 3.3.2 のような Drop 領域を作成し、その領域上へ画像ファイルを Drop することにより処理が行われる。

図 3.3.2　Drag & Drop によるファイル送信インタフェース

3.3.1 Java Applet

インタフェースの構築には、Java Applet（以下 Applet）を使用した。Applet とは、Web ページの HTML ソースコードから参照されるプログラムのことで、Web サーバからブラウザに動的ダウンロードされ、ブラウザの環境で実行される。

一般的に Applet が動作できる範囲はサンドボックスの中に限られており、今回行うようなクライアント側のローカルファイルへのアクセスは禁止されている。このセキュリティの制限を回避するために以下の 2 通りの方法が考えられる。

（1）Java のセキュリティポリシー[5]の設定変更
（2）Applet にデジタル署名[5][6]を付加する

Java のセキュリティポリシーの設定を変更することで、ローカルファイルへのアクセスが可能となる。しかし、他の悪意を持ったプログラムの実行も許可してしまう事になる。またクライアント側で個別に設定しなければならないため、システムの利用者が設定を行うこの方法は適していないと考えられる。よって今回は Applet にデジタル署名を付加する方法を取った。

3.3.2 署名付き Java Applet

Applet にデジタル署名を付加する場合、class ファイルなどの必要なファイルを JAR ファイルへ圧縮を行う必要がある。JAR ファイル化のツールは jdk に同梱されており、コマンドプロンプトから実行を行う。実行例を図 3.3.3 に示す。

```
C:¥Java¥Mnemonic >jar cvf MakeMnemonicApplet.jar *

マニフェストが追加されました。
drop.jpg を追加中です。(入 = 6858) (出 = 5836)(14% 収縮されました)
MakeMnemonicApplet$FileDropAcceptableLabel.class を追加中です。(入 = 2415) (出 = 1192)(50% 収縮されました)
MakeMnemonicApplet.class を追加中です。(入 = 3769) (出 = 2072)(45% 収縮されました)
MakeMnemonicApplet.java を追加中です。(入 = 6548) (出 = 2322)(64% 収縮されました)
Thumbs.db を追加中です。(入 = 5120) (出 = 2659)(48% 収縮されました)
```

図 3.3.3　JAR ファイル化の例

JAR ファイル化を行ったファイルにデジタル署名を付加するために、キーストアを作成する。キーストア作成に必要な keytool も jkd に同梱されている。キーストア作成の実行例を図 3.3.4 に示す。これにより「mykey」というキーストアが作成される。

```
C:¥>keytool -genkey
キーストアのパスワードを入力してください:  testkey
姓名を入力してください。
  [Unknown]:  matoba shoji
組織単位名を入力してください。
  [Unknown]:  shimane university
組織名を入力してください。
  [Unknown]:  lab tanaka
都市名または地域名を入力してください。
  [Unknown]:  matsue
州名または地方名を入力してください。
  [Unknown]:  nishikawatsu
この単位に該当する 2 文字の国番号を入力してください。
  [Unknown]:  JP
CN=matoba shoji, OU=shimane university, O=lab tanaka, L=matsue,
ST=nishikawatsu,
 C=JP でよろしいですか?
  [no]:  y

<mykey> の鍵パスワードを入力してください。
        (キーストアのパスワードと同じ場合は RETURN を押してください):
```

図 3.3.4　キーストア作成の例

先ほど作成した mykey というキーストアを用いてデジタル署名を付加する。実行例を図.3.3.5 に示す。これらの処理により、Applet 上に Drop された画像ファイルを Web サーバへ送信することが可能となる。

```
C:¥Java¥Mnemonic >jarsigner MakeMnemonicApplet.jar mykey
キーストアのパスワードを入力してください: testkey

警告: 署名者の証明書は 6 か月以内に期限切れになります。
```

図 3.3.5　デジタル署名付加の例

3.4　新規アカウント作成処理

新規アカウント作成処理は、ログイン画像作成処理とログイン画像ダウンロード(DL)処理の２つに大きく分類される。新規アカウント作成の Web サーバ上における処理の流れを図 3.4.1 に示す。

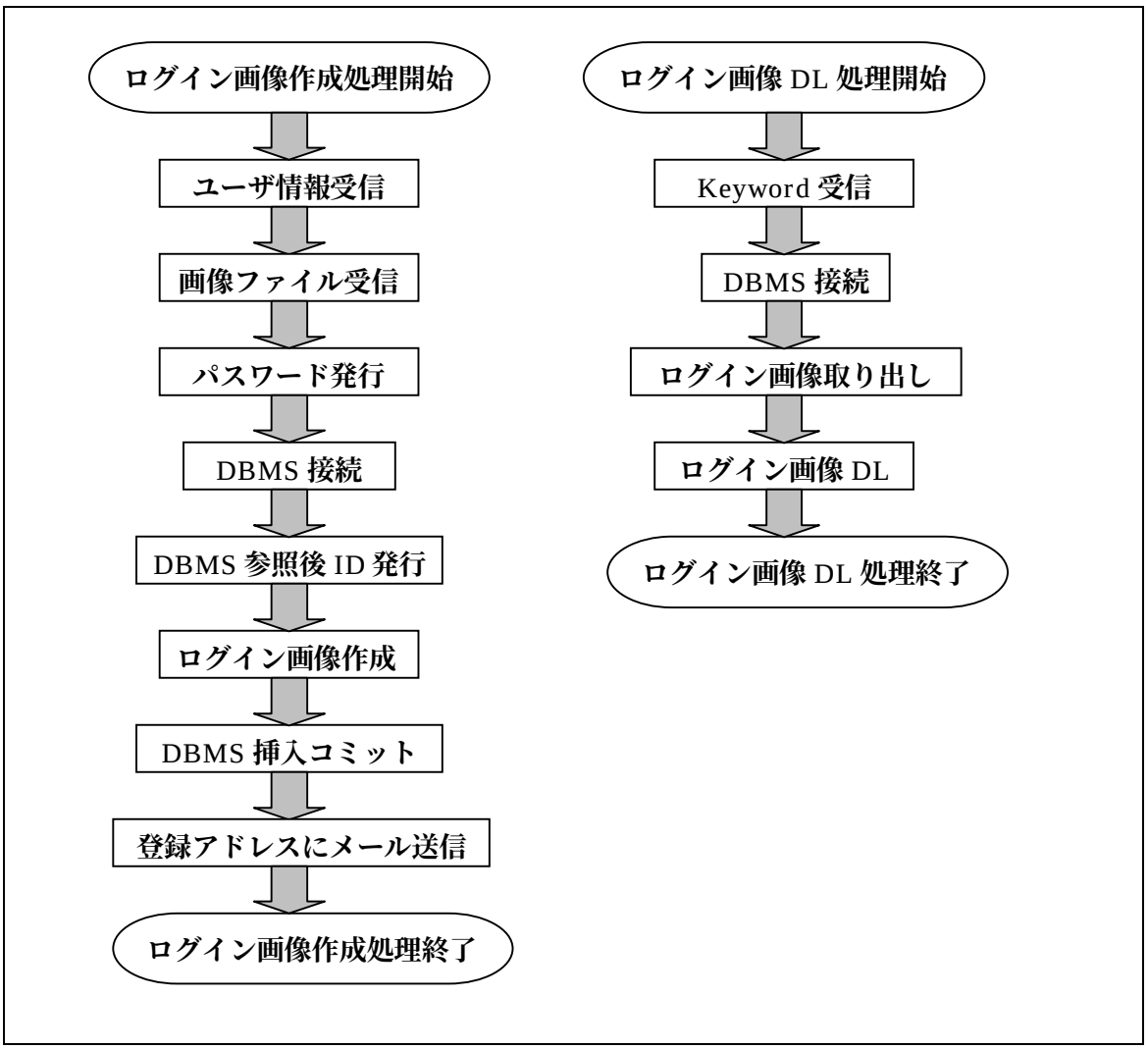

図 3.4.1　Web サーバ側の新規アカウント作成の流れ

まずクライアントからブラウザを起動し、フォームに必要事項を記入する（図 3.4.2）。続けて、ログイン画像とする画像を Drop する領域が表示されるため、画像を Drop する。Drop された画像は Web サーバへ送信され、ログイン画像の作成・DBMS への登録が行われる。Web サーバで処理が終了するとフォームに入力したメールアドレス宛に URL が記載されたメールが送信される。その URL へアクセスし、初めに登録した keyword を入力する事でログイン画像がブラウザ上に表示され、DL が可能となる。

図 3.4.2　ログイン画像作成処理（フォーム入力）

3.4.1　Applet と Servlet 間の Stream 通信

ブラウザ上の Applet に Drop された画像を Web サーバへ送信するため、今回は Stream 通信を行った。先行研究においては、ブラウザ上からファイルの送信を行い、結果をブラウザ上で受け取っていたが、Applet-Servlet 間で通信を行った場合は、処理結果をブラウザで受け取る事ができず、Applet 上で受け取らなければならない。つまり処理結果をブラウザが描画する事ができず、基本的に Applet 上に描画しなければならない。Applet で受け取った処理結果をブラウザ上に描画する方法も存在する、これに関しては後のログイン処理の節で述べる。Stream 通信を行うための必要部分のソースコードを一部抜粋する(図 3.4.3、図 3.4.4)。また、複数回通信を繰り返す事により複数の画像をアップロード、文字列のやり取りなども可能である。

注意点としては、Applet から Servlet の一方的な通信では正しく処理が行われない点である。必ず相互間で通信する必要がある。

なお、Applet から Servlet に画像ファイルを送信する方法として、Jakarta Commons の httpclient を用いて multipart post の処理を実装する事も可能である。ただし、これらを用いる場合、Applet の JAR ファイル内に追加 API を同梱する必要があり、ファイルサイズが大きくなってしまう。

```
URL url = new URL("http://rena.cis.shimane-u.ac.jp/s033081/servlet/StreamServlet");
URLConnection uc = url.openConnection();
//ＵＲＬ接続に書きこみを行うための設定
uc.setDoOutput(true);
//キャッシュ内のデータを無視するための設定
uc.setUseCaches(false);
//Content-type の指定を要求プロパティに設定
uc.setRequestProperty("Content-type","application/octet-stream");

//送信ストリームの準備
DataOutputStream out = new DataOutputStream(uc.getOutputStream());
dout.write(buf,0,(int)FileSize);
dout.flush();
dout.close();

/************************/
/** サーブレット処理中 **/
/************************/

//受信ストリームの準備
DataInputStream din = new DataInputStream(uc.getInputStream());

//受信したストリームからデータ読み出し
String receive = din.readUTF();

//受信ストリームを破棄
din.close();
```

図 3.4.3　Applet-Servlet **間の** Stream **通信**(Applet **側)**

```
//アプレットからのストリームを受信する
DataInputStream din = new DataInputStream(req.getInputStream());

//ファイルオブジェクト、ファイルストリーム作成
File fp = new File("C:¥¥LGW¥¥Data¥¥login.jpg");
String fname = fp.getAbsolutePath();
FileOutputStream fout = new FileOutputStream(fp);

//読み込みと書き込み
int c;
int i=0;
while ((c = (din.read())) != -1) {
        fout.write(c);
        i++;
}

//ストリームを閉じる
din.close();
fout.close();
```

図 3.4.4　Applet-Servlet 間の Stream 通信(Servlet 側)

3.4.2　ログイン画像作成

前述した隠蔽アルゴリズム、抽出アルゴリズムを実装したプログラムをそれぞれ、「jpegstetgano.exe」、「rejpegstegano.exe」とする。これらのプログラムは C 言語で作成されており、exe ファイルとして実行可能な状態となっている。

パスワードが乱数により生成され、一度 DBMS へアクセスを行い、登録されている ID の情報を検索した後、昇順に ID の発行を行う。例えば DBMS に ID が「00000002」まで登録されている場合、「00000003」が発行される。

隠蔽プログラム jpegstegano.exe を Servlet 上から実行する方法として Java.Runtime.exec メソッドを用いる。これは Java プログラムから外部プログラムを実行し、結果を得るメソッドである。外部プログラムとして jpegstegano.exe を実行すると、ログイン画像が新たに生成される。Tomcat 内のディレクトリにファイルを生成する場所は、Tomcat のセキュリティ設定ファイル（Catalina.policy）を設定する必要がある。今回は Tomcat のディレクトリ外にファイルを生成したため、設定は不要であった。参考までに設

定例を図 3.4.5 に示す。また Java.Runtime.exec メソッドの実行例を図 3.4.6 に示す。

```
grant codeBase "file:${catalina.home}/LGW/*" {
    permission java.lang.RuntimePermission "java.lang.Runtime.*", "write";
    permission java.io.FilePermission "java.io.*", "write";
};
```

図 3.4.5　Catalina.policy の設定例

```
//実行コマンドを String 型に用意
com = new String("C:/LGW/Prog/jpegstegano.exe " + "C:/LGW/Images/01.jpg " +
"C:/LGW/Images/fout.jpg " +"maketest");
//Runtime オブジェクト取得
Runtime rt = Runtime.getRuntime();
try {
  //コマンド実行後、終了まで待機
  Process pr = rt.exec(com);
  rt.waitFor();
}
//エラー処理
catch (Exception e) {
  System.exit(1);
}
```

図 3.4.6　Java.Runtime.exec メソッドの実行例

3.4.3　トランザクションの衝突

図 3.4.1 に示したログイン画像作成処理において、トランザクションの衝突が発生する可能性がある。ID は先ほど述べたように、一度 DBMS へアクセスを行い DBMS に登録されている ID の MAX 値を取得し、「MAX 値+1」として昇順に新しい ID が発行される。図 3.4.7 に示す例のように、ID が発行されてから、DBMS へ挿入コミットが行われる前に他のトランザクションで同じ ID が発行されてしまう可能性がある。

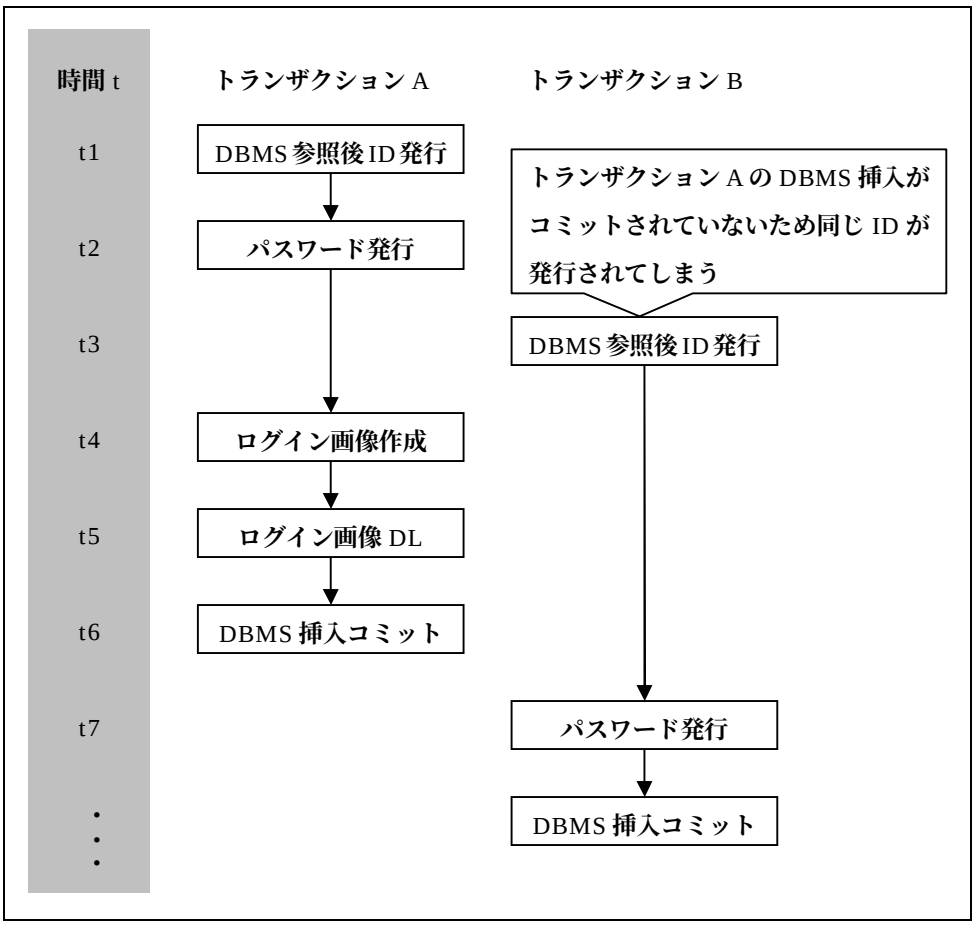

図 3.4.7　トランザクション衝突の例

この解決策として、プログラム側で制御を行う方法として以下が挙げられる。

（1） java の synchronized **宣言を用いる方法**
（2） Java.util.concurrent.atomic **パッケージを用いる方法**

synchronized 宣言を用いる方法では、Java プログラム内の method の前に synchronized 修飾子を付加することにより、実行開始時にロックを取得し、スレッドを順番に実行する。

Java.util.concurrent.atomic パッケージ（以下 concurrent）を用いる方法では、synchronized のように共有資源に対してロックをかける必要がないため、synchronized を使用するより性能が良く、デッドロックなどの問題が起こりえないプログラムを書く事ができる。この方法では、全てのスレッドが固まらずに、結果が正しく導かれるという性質を持っている。このパッケージを用いたプログラムの具体例を示す（図 3.4.8）

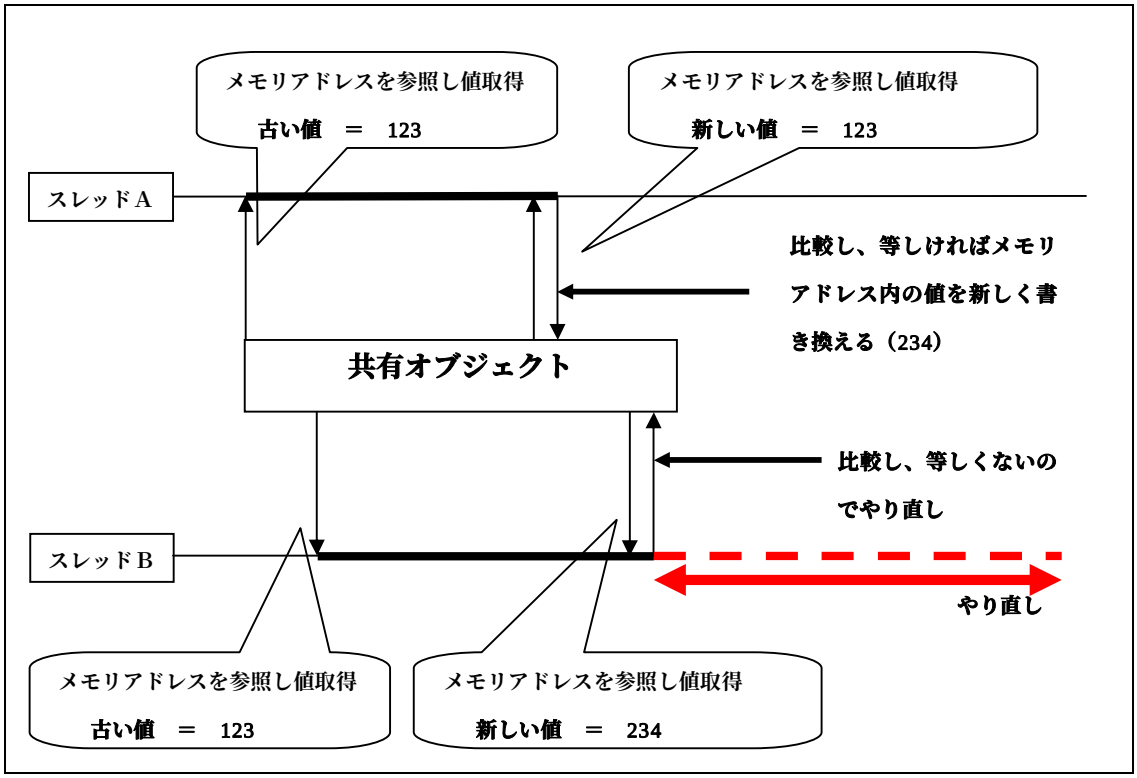

図 3.4.8　Java.util.concurrent.atomic パッケージを用いたプログラムの具体例

上記の図の例のように、最初に取得したメモリ内の値と、終了時に取得したメモリ内の値が等しければメモリの値を書き換え、異なっていれば処理そのものをやり直すという動作を行う。一般的に concurrent の方が性能は良いとされているが、本システムの場合では synchronized 宣言を用いた方が、時間がかからない結果を得られると考えられる（図 3.4.9）

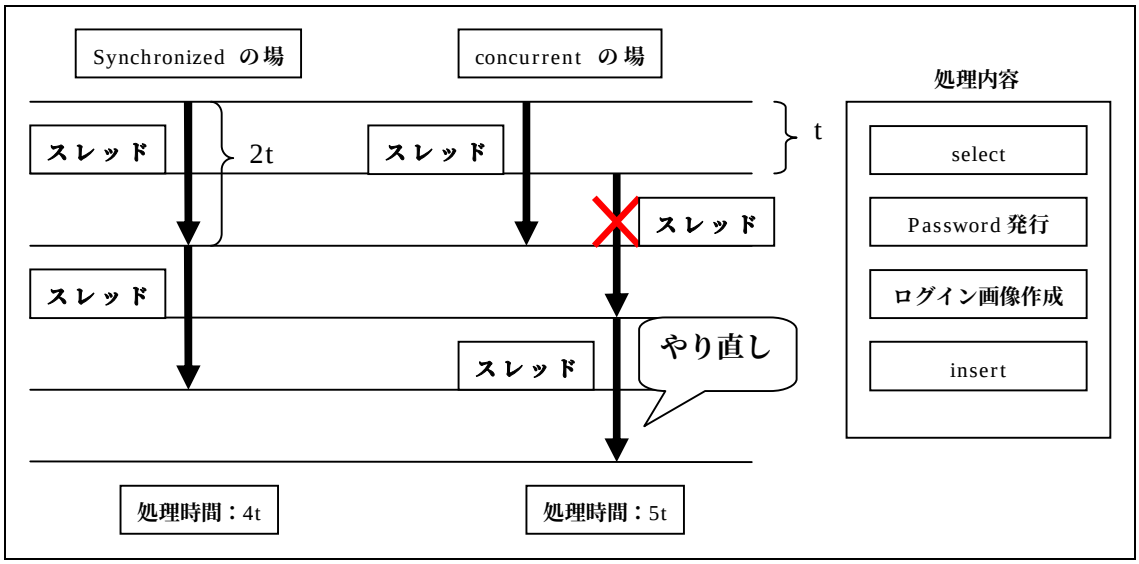

図 3.4.9　同期処理を取る方法による速度比較

１つのアクセスに対して、処理時間が 2t とする。Synchronized を用いた場合は、スレッドが順番に実行され、処理時間は 4t となり正しい結果が得られる。

concurrent を用いた場合を考えてみる、ここでの共有オブジェクトは、表内の ID の MAX の値である。上記の場合、１つ目の処理が実行されている途中(時間 t の際)に２つ目の処理の実行が始まったとする。この場合２つ目の処理の終了時に ID の MAX の値は開始時と異なるものとなり、必ず処理のやり直しが発生する。結果処理時間は 5t となり、遅くなると考えられる。concurrent を用いて、速度が向上する場合というのは、例えば select のみが行われる場合､表の一部分に対して update などが大量に行われる場合であると考えられる。これに関しては実装してみて検証が必要となるが、今回は実装行わないものとした。

3.5　ログイン処理

本システムでは、Drop 領域にログイン画像を Drop するだけで、ログイン処理が自動的に行われる。ログイン画像は 3.4.1 節で述べた Applet と Servlet 間の通信を利用し、Web サーバへ送信される。送信されたログイン画像を用いて、Web サーバ側でログイン処理が開始される（図 3.5.1）

ログイン情報の抽出は、先ほどの新規アカウント作成時と同様に Java.Runtime.exec メソッドを用いて、抽出プログラム「rejpegstegano.exe」を実行する。さらに DMBS へアクセスを行い、登録されているパスワードと、ログイン画像から抽出した情報が一致するかどうか判定を行う。その処理結果を Applet 側へ返す。

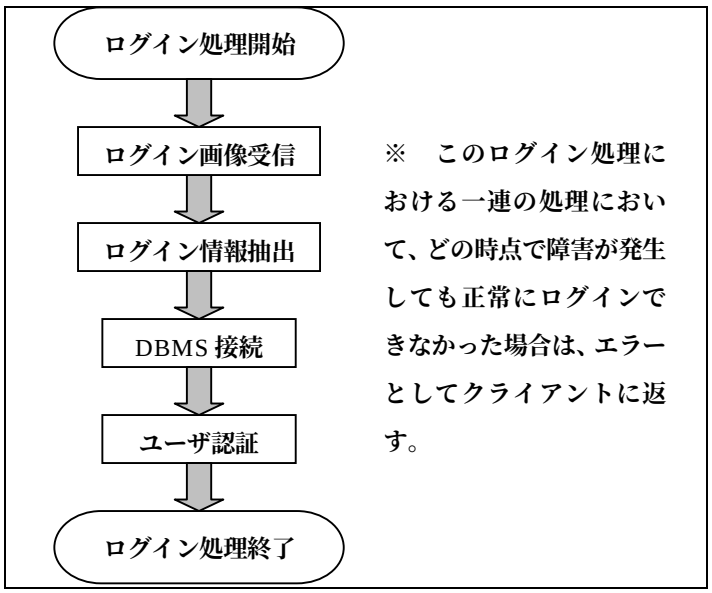

図 3.5.1　ログイン処理の流れ

3.5.1 Applet と JavaScript の連携

3.4.1 節で述べたように、Applet と Servlet で通信を行った場合、処理結果は Applet 上で受け取られ、ブラウザがそれを受け取り表示することは基本的にはできない。Applet で受け取った処理結果をブラウザ上に表示させる方法として JavaScript との連携[7]が考えられる。ただしこの方法を用いると、Applet が表示されている Web ページが遷移してしまうことに注意すべきである。

Java プログラムから JavaScript を呼び出すためには、Java の Plug-in である JSOject を利用する。JSObject クラスは jdk¥jre¥lib¥plugin.jar ファイル内に格納されている。Java のプログラム内で import netscape.javascript.*;とすることで利用が可能となる。具体的な方法について以下に述べる。

（1） Applet を表示される HTML ソースコード内に JavaScript を記述。
（2） <applet>タグ内に MAYSCRIPT (JavaScript の実行を許可する)と記述を行う。
（3） Applet から JSObject を利用して JavaScript を Call する。

図 3.5.2 から図 3.5.4 に実際の連携を行った例を示す。この例の場合、HTML ソースコード内に記述されている JavaScript の「f」という関数を呼び出している。Servlet から受け取ったログイン結果が error1 または error2 でなければ、登録者の名前を表示する。

```
<script language="JavaScript">
<!--
function f(data){
        document.open();
        if( (data != "error1") && (data != "error2") ){
                document.write("こんにちは",data,"さん");
        }else{
                document.write("認証に失敗しました");
        }
        document.close();
}
// -->
</script>
```

図 3.5.2　JavaScript ソースコード(HTML ソースコード内)

```
<applet
code="StreamApplet"
archive=http://rena.cis.shimane-u.ac.jp/~s033081/Applet/StreamApplet.jar
width="200"height="200" MAYSCRIPT>
</applet>
```

図 3.5.3　JavaScript の実行許可(HTML ソースコード内)

```
import netscape.javascript.*;

//JavaScript へ処理を渡す
public void JavaScript(String data) {
        JSObject win = JSObject.getWindow(this);
        JSObject doc = (JSObject) win.getMember("document");
        JSObject loc = (JSObject) doc.getMember("location");
        Object[] args = new Object[3];
        args[0] = new String(data);
        String s = (String) loc.getMember("href");    // document.location.href
        win.call("f",args );                                    // Call f() in HTML page
}
```

図 3.5.4　Applet から JavaScript の呼び出し

3.6　Java Database Connectivity

本システムで DBMS へアクセスすることは先ほど述べたとおりである。Java プログラムから DBMS にアクセスする際に Java Database Connectivity（以下 JDBC）API を用いる。実際に利用する場合は、DBMS に対応した JDBC Driver が必要となる。今回は、使用した DBMS である「HiRDB」付属の HiRDB JDBC Driver を使用した。

3.6.1　Driver Manager と Data Source

Servlet から DBMS へアクセスする際に JDBC を利用することに関しては前述した通りである。JDBC を利用する方法として２つの方法が存在する（表 3.6）

表 3.6　JDBC の利用方法

名称	説明
Driver Manager	個々の Java プログラムで JDBC を用いて、DBMS に接続する方法
Data Source[7]	Data Source を用いて DBMS に接続する方法

従来、利用されてきたのは Driver Manager を用いる方法である。しかし JDBC 2.0 標準拡張機能 API で、Data Source インタフェースが追加されたことにより、Data Source オブジェクトの利用が推奨されている。どちらも DBMS へアクセスするという点では同様であるが、DBMS のアカウント情報の記述の場所、接続速度、移植性などに大きな差が見られる。

Driver Manager は DBMS への接続を個々の Servlet で行うが、Data Source の場合、Tomcat で接続を行い、その接続を他の Servlet に使わせることも可能である。さらにコネクションプールや分散トランザクションを実装できる。

また Driver Manager では Java のソースコード内に DBMS のアカウント情報などを記述するため、DBMS が変更された場合はソースコードの変更を行う必要がある。Data Souce を利用した場合は、Tomcat の設定ファイルでアカウント情報を定義するため、ソースコードの変更が必要なく、移植の際に便利である。それぞれの接続の違いを図 3.6.1、Data Source を利用する際に必要となる Tomcat の設定を図 3.6.2 に示す。Tomcat のバージョンによって、設定方法が大きく違うため、tomcat の配布サイトで確認を行うこと。

なお速度差に関しては、5 章の実験で実際に速度を計測し、比較を行う。

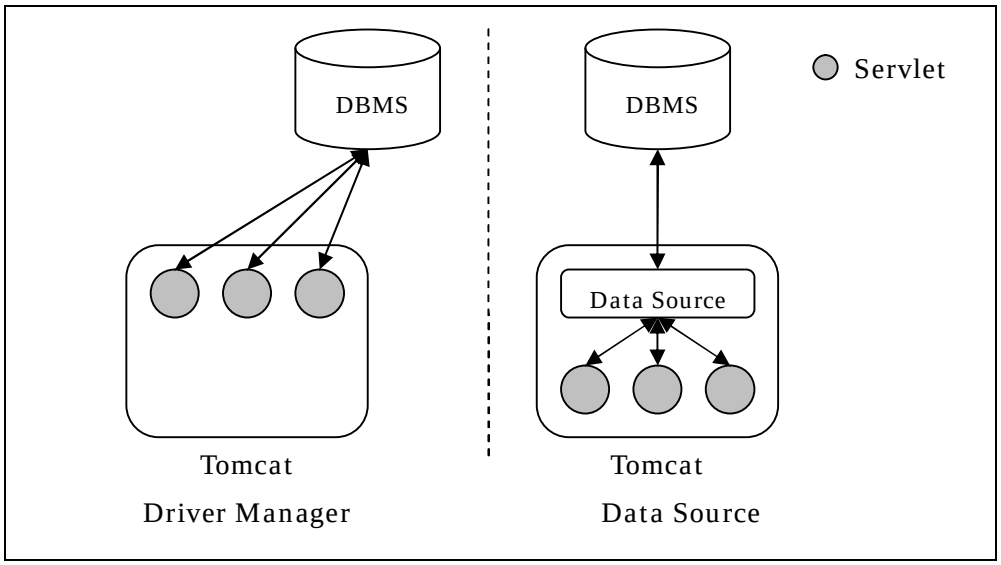

図 3.6.1　Driver Manager と Data Source の比較

```
<Context path="" docBase="C:¥Web¥users¥s033081¥Servlet">
  <Resource name="jdbc/datasource" auth="Container"  type="javax.sql.DataSource"
  driverClassName="JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver"
  url="jdbc:hitachi:PrdbDrive://DBID=22200,DBHOST=192.168.1.102,DB=HiRDB"
  username="S033081" password="S033081"
   maxActive="50" maxIdle="200" maxWait="1000"/>
</Context>
```

図 3.6.2　CATALINA_HOME/conf/Catalina/localhost/s033081.xml の設定

※　上記ファイル名の s033081 箇所はコンテキスト名

3.7　BLOB 型データの格納

本研究のシステムでは、作成したログイン画像をアカウント情報と共に DBMS へ格納を行い、管理を行っている。表の定義を図 3.7.1 に示す。

ID	NAME	PASSWORD	MAIL	KEYWORD	IMAGE
00000000	MATOBA	MATOPASS	xxx@xxx	MATOKEY	
00000001	⋮	⋮	⋮	⋮	⋮
⋮					

CHAR 型　　BLOB 型

図 3.7.1　表定義

3.7.1　DBMS の設定

上記の表定義に示したように、ID,NAME,PASSWORD,MAIL,KEYWORD 列は CHAR 型として、IMAGE 列が BLOB(Binary Large Object)型[9]として定義されている。BLOB 型の列を含んだ表を作成する場合は、DBMS で設定を行う必要がある。BLOG 型を場合は、ユーザ LOB 用 RD エリアにデータを格納する。これは文書、画像、音声などの長大な可変長データを格納する領域のことである。図 3.7.2 に挙げた SQL 文の例のように、ユーザ LOB 用 RD エリアの指定を行い、BLOB 列を作成する。例では RLOB811 という領域に BLOB 列を定義している。ユーザ LOB 用 RD エリアの作成に伴い、グローバルバッファの割り当てを行う必要がある。グローバルバッファとは、ディスク上の RD エリアに格納されているデータを入出力するためのバッファのことである。さらに RD エリア作成時に、セグメント数を増加させる設定を行う事で、より大きなデータを格納することが可能となる。

設定の詳細方法に関しては、HiRDB Version7 スタートアップガイドの

P169~P171,P178,P181~183,P204~P209 を参照されたし。[10]

```
create table IMAGE_LGA (ID char(8),NAME char(16),PASSWORD char(8)
,MAIL char(40),KEYWORD char(8), IMAGE BLOB(100K) in RLOB811);
```

図 3.7.2　**表作成** SQL **文**

3.7.2　画像データ挿入方法

画像データを BLOB **列に挿入するためにはパラメータマーカである「？パラメータ」を用いる。**Java **プログラム上での具体的な使用例を示す（図** 3.7.3)。

```
//File オブジェクトを作成し画像のパスを指定
File Ifile = new File(FilePath);

//入力ストリームのオブジェクトを作成
fin = new FileInputStream(Ifile);

// Statement を作成
stmt = con.prepareStatement("INSERT INTO s033081.IMAGE_LGA(ID, NAME,
PASSWORD, MAIL, KEYWORD, IMAGE) VALUES('"+id+"', '"+name+"', '"+pass+"',
'"+mail+"' , '"+key+"', ?)");

stmt.setBinaryStream(1,fin2,(int)Ifile.length());
stmt.executeUpdate();
```

図 3.7.3　?**パラメータの使用例**

第 4 章　ニーモニック認証

序論にて「クライアントに不正にログインした人物が、手当たりしだいに Web サーバへ画像を送信するとログインされてしまう」という先行研究での問題点を挙げた。この問題を「ニーモニック認証[11]」という技術を用いることより、解決することが可能である。

4.1　ニーモニック認証

ニーモニック認証とは従来のパスワード認証とは異なり、本人の経験上の記憶により、セキュリティ認証を実現するテクノロジーのことである。本人の記憶という、第三者が知りえない情報を用いる認証技術であるため、他人が不正に認証を行うことがほぼ不可能であり、容易に高いセキュリティを実現できる。

4.1.1　既存ソフトウェア紹介

ニーモニック認証を用いた既存のソフトウェアの具体的な例を示す（図 4.1.1）

図 4.1.1　ニーモニック認証を用いた既存ソフトウェアの例

図は、株式会社ニーモニックセキュリティ[11]のニーモニックガードというソフトウェアの認証実行画面である。認証は予め登録されたパスシンボルを複数個選択することで行われる。図の 64 個のシンボルの中に、例えば自分の子供、飼っていたペットなどの画像を含ませておき、そのシンボルを全て選択することで認証される。

この方法では、本人以外は知りえない情報を利用するため、高いセキュリティを実現することが可能となる。ただし、認証シンボル以外の囮画像を準備するという点で多少手間がかかる。

4.1.2　応用方法

先ほど述べた技術を本システムに応用する案として、複数枚のログイン画像を準備し、それらを Web サーバへ送信することが考えられる。ログイン画像が 1 枚だけの場合、手当たりしだいに画像を送信することで不正認証を行われる可能性が非常に高いが、この方法を用いることにより、ほぼ不正認証は防げると考えられる。セキュリティの強度に関しては後ほど検証を行う。今回はログイン画像を 3 枚として、試験的に実装を行う。実際のシステムではログイン画像の枚数はユーザが決定する形とする。

4.2　システムの変更点

3 章で実装を行ったシステムに、ニーモニック認証を組み込むために、いくつかの箇所に変更を行う。以下にそれぞれについて示す。

4.2.1　インタフェースの変更

ログイン画像が 1 枚の場合は、新規アカウント作成処理、ログイン処理の共に、Drop 領域に画像が Drop されると自動的に一連の処理を行っていた。ログイン画像を複数枚利用するにあたって、Drop された画像の枚数を Applet 上へ表示、処理開始ボタンの追加を行うためプログラムを修正した。(図 4.2.1)。

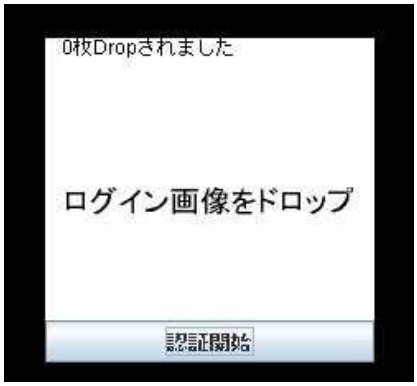

図 4.2.1　インタフェースの変更

図はログイン処理を行う画面であるが、ユーザが登録した枚数のログイン画像を全て Drop し、認証開始ボタンを押すことで認証が開始される。つまり本人以外には、ログイン画像を 3 枚以内で何枚 Drop すれば良いかがわからないことになる。この事から、ログイン画像の枚数が増えるにつれて、さらにセキュリティの向上が見込める。

4.2.2　表定義の変更

ログイン画像を複数枚用いるため、それぞれの画像へパスワードを埋め込む必要がある。それに伴い、DBMS に登録されている表の定義の変更を行った（図 4.2.2）

ID	NAME	PASS1	PASS2	PASS3	MAIL	KEYWORD
00000000	MATOBA	MPASS1	MPASS2	MPASS3	xxx@xxx	MATOKEY
00000001	⋮	⋮	⋮	⋮	⋮	⋮
⋮						

CHAR 型

図 4.2.2　表定義の変更

今回はログイン画像が 3 枚であるという前提で表を定義したため、このような形になった。しかし実際のシステムでは、より多くの枚数に対応した表の定義を考える必要がある。

なお、変更前はログイン画像データを表に格納していたが、変更後は格納を行っていない。容量的な問題はあるが、画像を表に格納した方が、ログイン画像の再発行処理時に容易だと考えられる。

4.3　セキュリティ強度

ニーモニック認証を用いた本システムで、どの程度のセキュリティ強度が実現できるのかについて検証を行う。

本研究のシステムでは、ログイン画像が複数枚 Drop される際の順番を考慮している。順番を考慮するか否かによって、大きくセキュリティ強度が変化する。例えば、ログイン画像 3 枚、囮画像 7 枚の計 10 枚の画像が 1 つのディレクトリで管理されているとする。Drop する画像が重複しても構わないとすると、「$10^3 = 1000$ 通り」の組み合わせが考えられる。Drop される順番を考慮する（ログイン画像を決められた順に Drop する）と、正解の組み

合わせは 1 通りしか存在しないが、順番を考慮しない（ログイン画像が Drop されていれば順番は問わない）場合は、「3! = 6 通り」の正解が存在することになる。

ここで、ログイン画像が m 枚、ダミー画像が n 枚の場合を考えてみる。重複を許すので、画像の組み合わせの数は、 $(m+n)^m$ 通り存在する。順番を考慮する場合は正解の組み合わせは 1 通り、順番を考慮しない場合の正解の組み合わせは m!通りである。当然、順番を考慮する場合の方がセキュリティは高くなる。今回のシステムは順番を考慮する場合を考え、実装を行った。

4.3.1 他の認証システムとの比較

本システムを実用するにあたって、どの程度のセキュリティ強度を実現できればよいのかについて、他の認証システムと比較を行いながら検証する。

まず一般的なパスワード認証（a~z,A~Z,0~9 計 62 文字からなる 8 桁）を用いた場合、「62^8 = 約218兆通り(10^{14}程度)」の組み合わせが存在する。これと同程度の組み合わせ(10^{14}程度)を実現するためには図 4.3.1 に示す程度の枚数は必要となる。

ログイン画像：8 枚　囮画像：49 枚	111 兆通り
ログイン画像：9 枚　囮画像：27 枚	165 兆通り

図 4.3.1　一般的なパスワード認証と同程度のセキュリティ実現

例として挙げた既存ソフトウェアと異なり、画像を Drag & Drop する本システムでは、ログイン画像の枚数が増えることはあまり好ましくない。

ここで銀行等の認証システムについて考えてみる。一般的に銀行等の認証システムとして、0~9 までの数字を用いて 4 桁の組み合わせで認証を行っている。この組み合わせは高々 1 万通り (10^4) である。しかし銀行等のシステムの場合、一定時間内に認証に 3 度失敗すると、利用が停止されるという特性を持っている。これを本システムでも組み込む事により、画像の枚数が少ない場合でも、実用に耐えうるセキュリティ強度の実現が可能である。これらを考慮すると

ログイン画像：3 枚　囮画像：19 枚　$22^3 = 10648$ 通り

であれば良いと考えられる。これまで述べてきたことは、ログイン画像と囮画像が全て同じディレクトリで管理されているという前提である。実際に複数枚あるログイン画像を別々のディレクトリで管理することにより、組み合わせの計算はより複雑になり、かつ高いセキュリティが実現可能になると考えられる。また本システムは使用するユーザによっ

てセキュリティの差がかなり出てしまうのも事実である。上記に述べた画像の枚数は、あくまで最低限のセキュリティを確保するための指標である。有効な利用方法に関しては今後さらに検証を進める必要がある。

第 5 章　性能評価実験

5.1　概要

本研究で実装したシステムを実用するにあたって、Web サーバからのレスポンスタイム等は重要である。そこで様々な状況を想定し、アクセスが集中した際にどの程度 Web サーバが負荷に耐えうるか、レスポンスタイムの計測、Web サーバ側での処理時間の詳細を計測、ボトルネックとなっている部分の特定などを通し、性能評価実験を行った。

5.1.1　性能評価用プログラムの実装

本実験において、速度の計測を行うために、性能評価用 Applet を作成した（図 5.1.1）Applet で実装した理由は、可能な限り本システムと同様な環境で測定を行うためである。

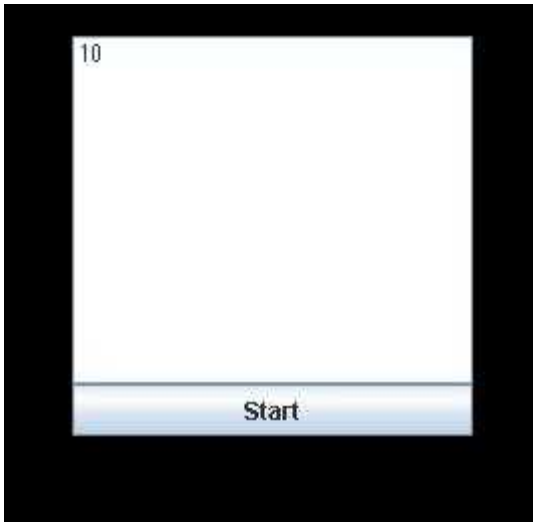

図 5.1.1　性能評価用 Applet

Applet 上にテキストフィールドとボタンを配置し、テキストフィールド上に同時アクセス数を入力することで、入力された数字だけスレッドを生成し、Web サーバへ同時アクセスを行う。Web サーバに配置してある Servlet は実際のシステムに用いるものをそのまま使用する。

基本的に Applet と Servlet の通信部分は Stream 通信を用いているが、新規アカウント作成における最初のステップ（フォームに名前等の入力を行う処理）においては、Web サーバへ Post 送信を行っている。Post 送信を Java プログラム上から実現するために、Jakarta Commons の HttpClient を用いた。HttpClient を用いることにより、簡単に Post 送信を実装可能である。つまり、実際のシステムではフォームからの Post 送信、本実験で

は HttpClient を用いて Post 送信を行っている。他の処理部分に関しては同様の処理を行っている。ただし、画像ファイルの Drag & Drop を行う部分に関しては、自動 3 枚の画像を送信している。

5.1.2 サーバ機の組み合わせ

本実験では、合計３台のサーバ機を用いて、その組み合わせごとの計測を行う。今回用いたサーバ機のマシンスペック等の実験環境を表 5.1.1 に示す。

表 5.1.1 実験環境

サーバ名	Web サーバ	DB サーバ	CPU	メモリ
rena	○	×	Celeron 2.4GHz	512MB
db	×	○	Celeron 2.8GHz	512MB
vajra	○	○	Dual Core Xeon 1.86GHz	1GB
LAN			100BASE-T(Sun MicroSystems Type PCC FT4)	

上記表において、サーバ機 rena は Web サーバとして稼動するが、DB サーバとしては稼動しないことを表している。サーバ機 vajra のみ Web サーバ、DB サーバの両方として稼動する。このサーバ機の組み合わせは次のように考えられ、それぞれの組み合わせについて条件 A～D と定義する。（表 5.1.2）

表 5.1.2 サーバ機の組み合わせ

	DB サーバ db	DB サーバ vajra
Web サーバ rena	条件 A	条件 B
Web サーバ vajra	条件 C	条件 D

5.1.3 計測の詳細

本研究のシステムでは、新規アカウント作成処理で 3 個、ログイン処理で 2 個の Servlet を用いて処理を行っている。計測の基本としては、それぞれの Servlet に対してのレスポンスタイムを計測する。必要に応じて Servlet 内の処理時間の内訳の計測を行う。それぞれの Servlet の処理の概要を表 5.1 に示す。以降、実験結果等は、これらの Servlet 名で表すものとする。

表 5.1.3　Servlet の処理内容

	Servlet 名	処理内容
新規アカウント作成処理	Servlet1	フォームデータ受信、クライアントへ Applet を返す
	Servlet2	画像アップロード
	Servlet3	ログイン画像作成、DBMS 接続、ユーザ情報検索(SELECT)、ユーザ情報登録(INSERT)、メール送信
ログイン処理	Servlet4	ログイン画像アップロード、抽出処理
	Servlet5	DBMS 接続、認証処理(SELECT)

新規アカウント作成処理に関しては、3 章の図 3.4.1 で示した、ログイン画像作成処理のみについて計測を行った。計測方法の概要の例を図 5.1.2 に示す。下図は新規アカウント作成処理の例である。Applet から同時アクセス数を入力し、その数だけ Thread を生成、新規アカウント作成処理で用いている Servlet1～Servlet3 へアクセスし、それぞれに対してレスポンスタイムの計測を行う。

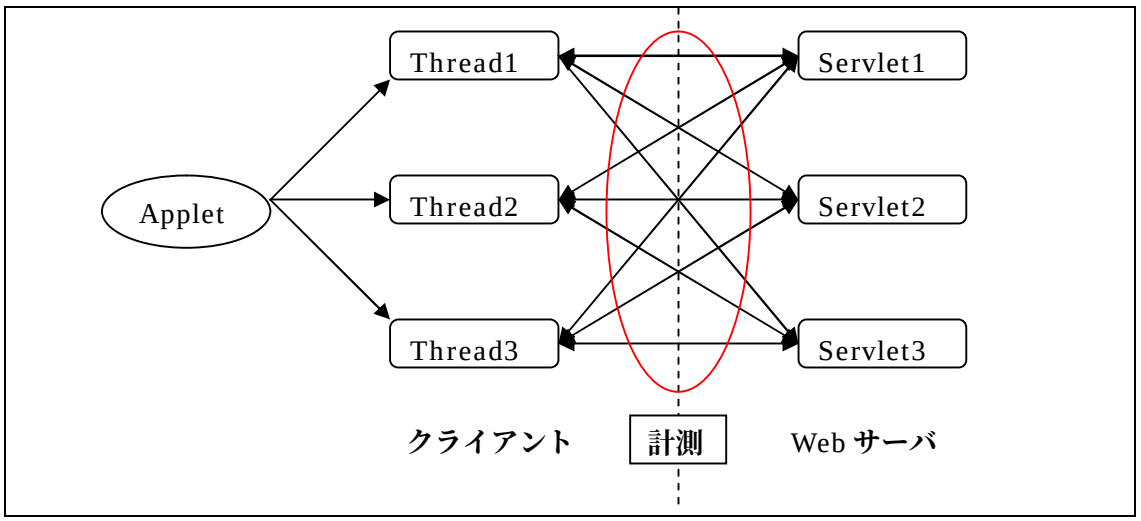

図 5.1.2　計測方法概要

本実験において大きく分類された３つの項目（サーバ機条件・JDBC 使用方法・変化させる項目）を組み合わせて、必要な部分に対して計測を行う。計測条件の一覧と、今回行った計測条件の組み合わせを図 5.1.3 に示す。

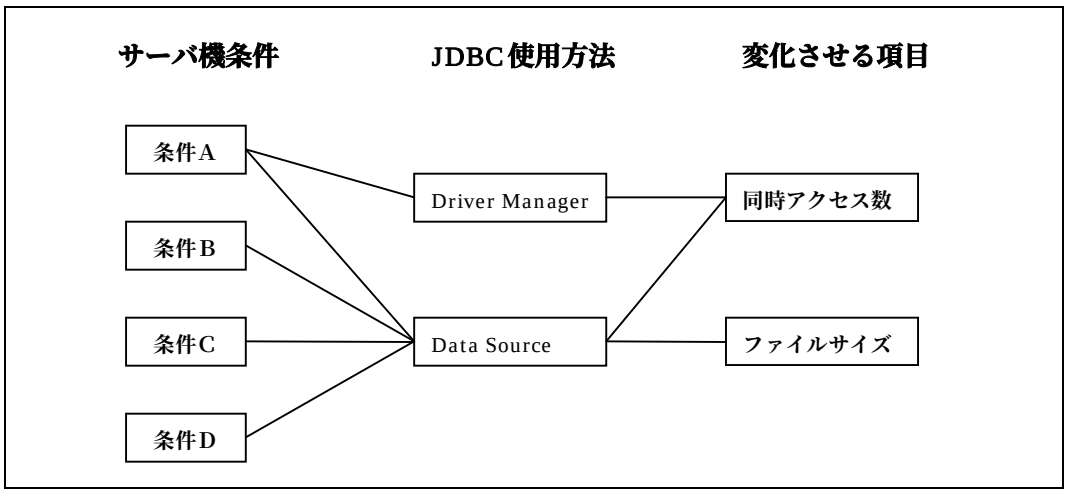

図 5.1.3　計測条件の組み合わせ

まず Driver Manager と Data Source の性能の比較を行うため、サーバ機の条件Aの場合に関して、同時アクセス数を変化させた場合について計測し、比較を行った。その比較結果より以降は Data Source を用いた場合のみについて他の条件での計測を行った。同時アクセス数を変化させる場合、及びファイルサイズを変化させる場合の条件を表 5.1.4 に示す。

表 5.1.4　同時アクセス数とファイルサイズ変化の条件

	同時アクセス数変化	ファイルサイズ変化
同時アクセス数	10～100 で変化	10 で固定
ファイルサイズ	10KB で固定	10KB～600KB で変化

5.2　実験結果

以下の実験結果のグラフにプロットされたデータは、５回実行を行った結果の平均とする。画像のアップロード処理(Servlet2 及び Servlet4)に関しては画像 3 枚分が実行されるため、同時アクセス数×3 回の実行が行われている。全体のレスポンスタイムは、新規アカウント作成処理は Servlet1～Servlet3、ログイン処理は Servlet4～Servlet5 のレスポンスタイムの合計である。

なお今回はホットスタート時の計測とし、初回起動時のコールドスタートの場合は考えないものとする。

5.2.1　JDBC の利用方法による比較

3.6.1 節で述べた通り、JDBC の利用方法には Driver Manager と Data Source の 2 種類の方法が存在する。これらの両方を用いて、レスポンスタイムにどのような影響が出るか、

DBMS へ接続を行っている Servlet3 と Servlet5 に関して計測と比較を行った。実験条件を表 5.2.1、実験結果を図 5.2.1～図 5.2.2 に示す。

表 5.2.1　実験条件 1

サーバ機条件	条件A
JDBC 使用方法	Driver Manager,Data Source
変化させる項目	同時アクセス数

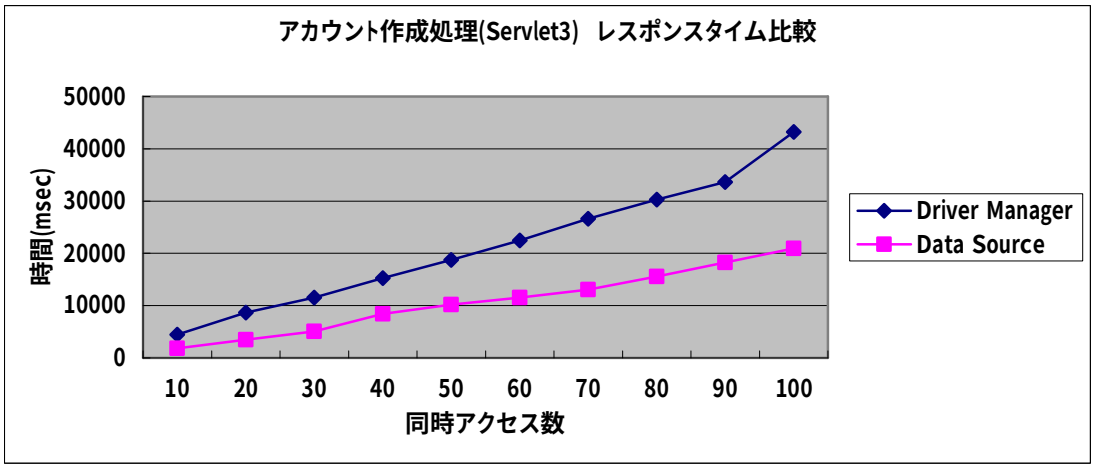

図 5.2.1　JDBC 利用方法によるレスポンスタイム比較（アカウント作成処理）

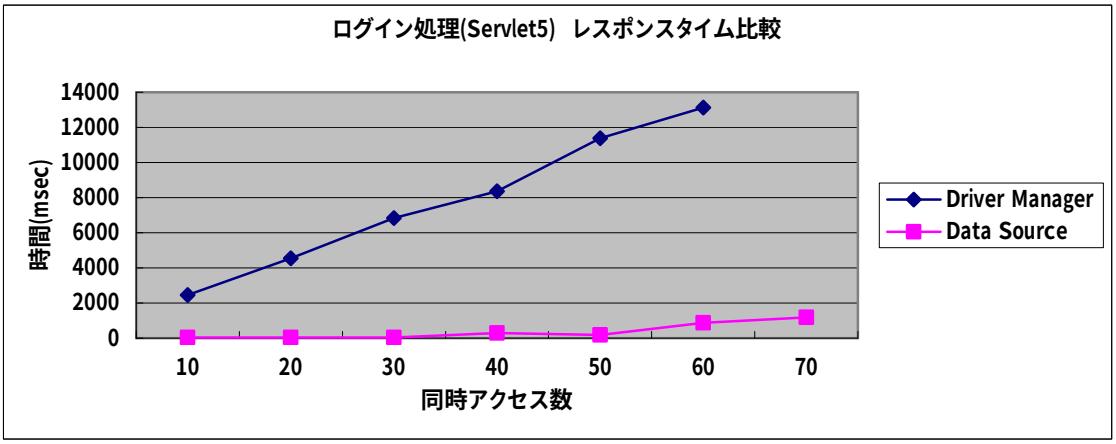

図 5.2.2　JDBC 利用方法によるレスポンスタイム比較（ログイン処理）

計測の結果を見ると、Data Source を用いた場合の方が圧倒的に早いことがわかる。Data Source は初回のコールドスタート時は Driver Manager よりも DBMS に接続するのに時間がかかるが、ホットスタート時はほぼ一瞬で接続を完了する。よって Sun が推奨するとおり、Data Source を使用する方が適切である。

なお、ログイン処理において同時アクセス数を増加させた場合、リクエストの失敗が見

られた。503 エラーが発生していたことより、Web サーバの過負荷によるものと考えられる。この結果から、以後の測定は Data Source を用いて行うものとする。

5.2.2　Web サーバの性能による比較

使用する Web サーバの性能によって、どの程度のレスポンスタイムの差が見られるかについて比較を行う。サーバ機の組み合わせとしては、条件Ａと条件Ｃの比較を行うのが適当である。条件Ｂと条件Ｄの比較は、条件Ｄの際は Web サーバと DB サーバが一機のサーバ機内に共存してしまう事から測定の条件としては適さない。実験条件を表 5.2.2、実験結果を図 5.2.3～図 5.2.6 に示す。

表 5.2.2　実験条件２

サーバ機条件	条件A、条件C
JDBC 使用方法	Data Source
変化させる項目	同時アクセス数、ファイルサイズ

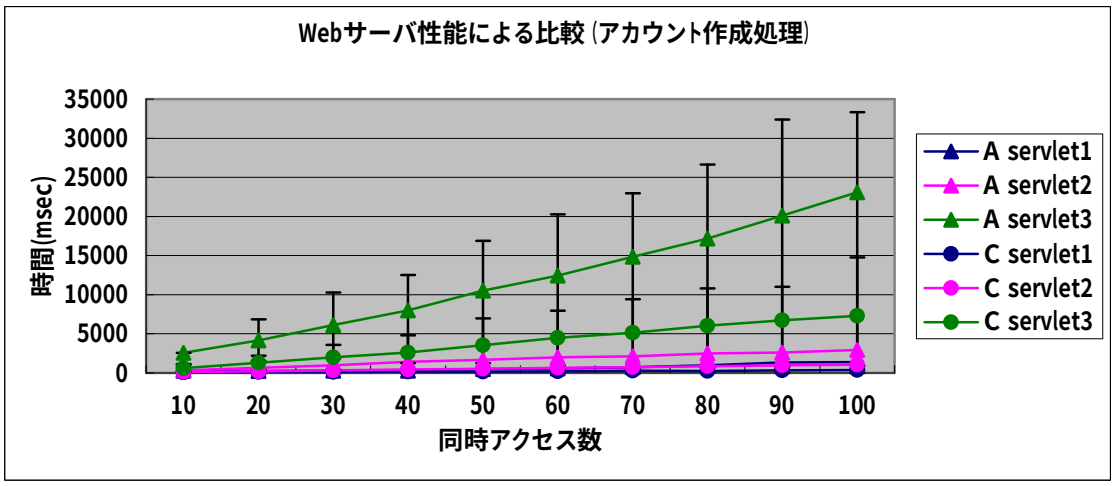

図 5.2.3　Web サーバ性能による比較（アカウント作成処理）

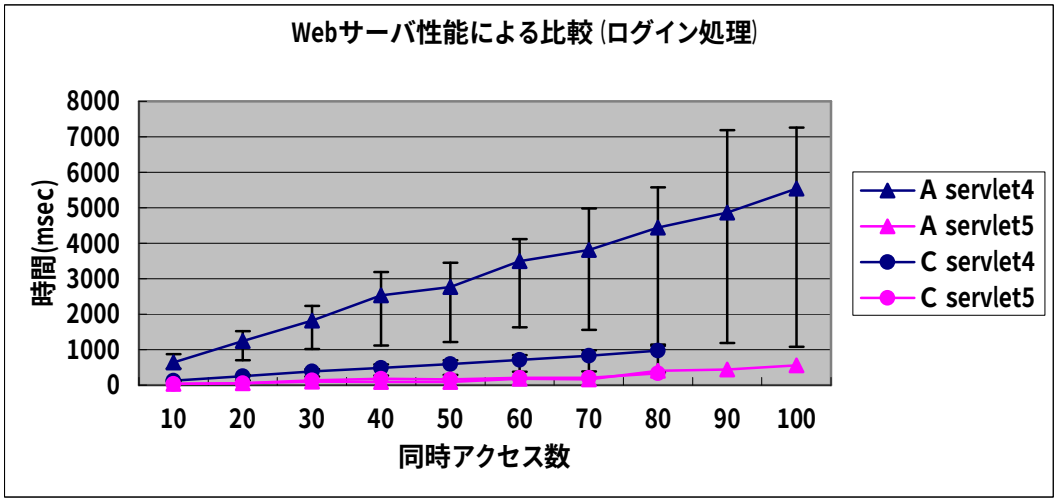

図 5.2.4　Web サーバ性能による比較（ログイン処理）

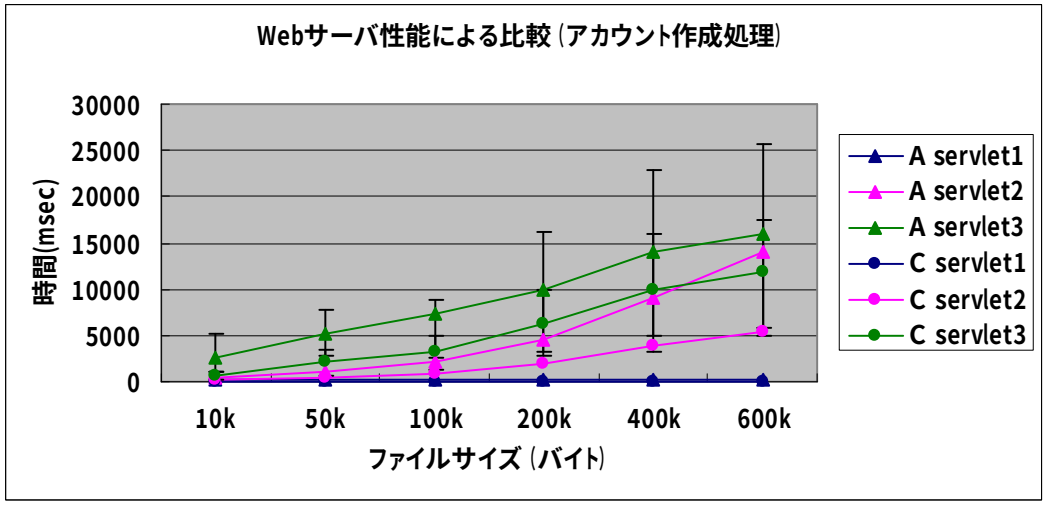

図 5.2.5　Web サーバ性能による比較（アカウント作成処理）

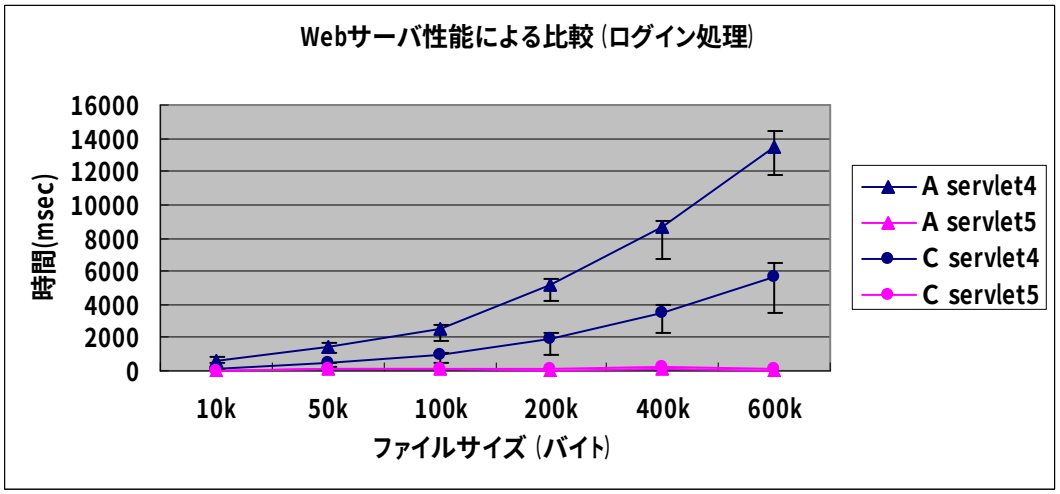

図 5.2.6　Web サーバ性能による比較（ログイン処理）

基本的にグラフは右上がりの比例関係になっている。ただしファイルサイズを変化させた場合の、図 5.2.5、図 5.2.6 では、画像のアップロードを行う Servlet（Servlet2,Servlet4）のグラフが指数的に増加する傾向にある。これは他の条件で計測を行った場合も共通である。全体として条件Ａの場合より、条件Ｃの方が格段に高速である事が見て取れる。図 5.2.4 のログイン処理においてデータが 80 までの場合しか存在しない。この理由は同時アクセス数を増やすと HiRDB の接続最大ユーザ数をオーバーするエラーが発生するためである。本実験において、HiRDB の設定で最大接続ユーザ数を 50 に設定している。ラインセンスの関係上、これ以上の値を設定できないため、今回はこのまま使用した。

5.2.3　DB サーバの性能による比較

使用する Web サーバの性能によって、どの程度のレスポンスタイムの差が見られるかについて比較を行う。サーバ機の組み合わせとしては、条件Ａと条件 B の比較を行うのが適当である。5.2.2.節の実験と同様に条件Ｃと条件Ｄの場合では、条件Ｄの際に Web サーバと DB サーバが一機のサーバ機内に共存してしまうことから条件として適さない。実験条件を表 5.2.3、実験結果を図 5.2.7～図 5.2.10 に示す。

表 5.2.3　実験条件３

サーバ機条件	条件Ａ、条件Ｂ
JDBC 使用方法	Data Source
変化させる項目	同時アクセス数、ファイルサイズ

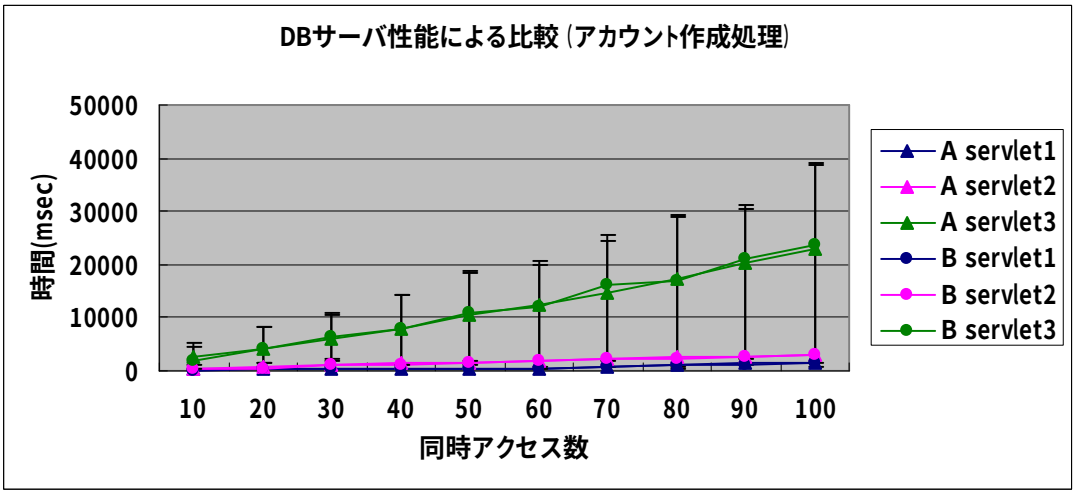

図 5.2.7　DB サーバ性能による比較（アカウント作成処理）

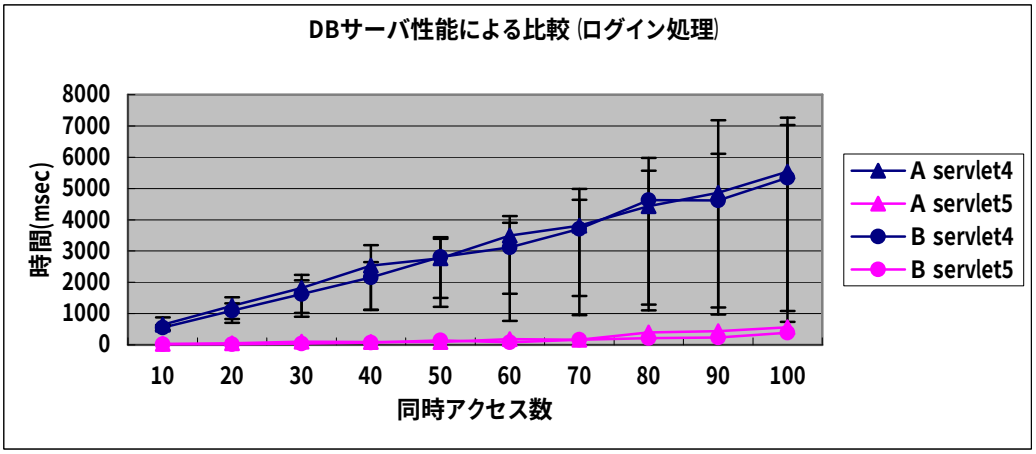

図 5.2.8　DB サーバ性能による比較（ログイン処理）

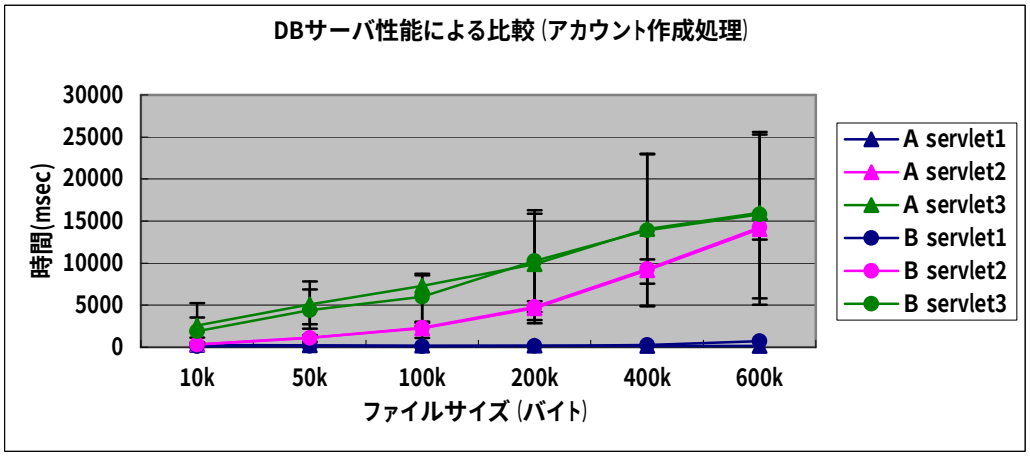

図 5.2.9　DB サーバ性能による比較（アカウント作成処理）

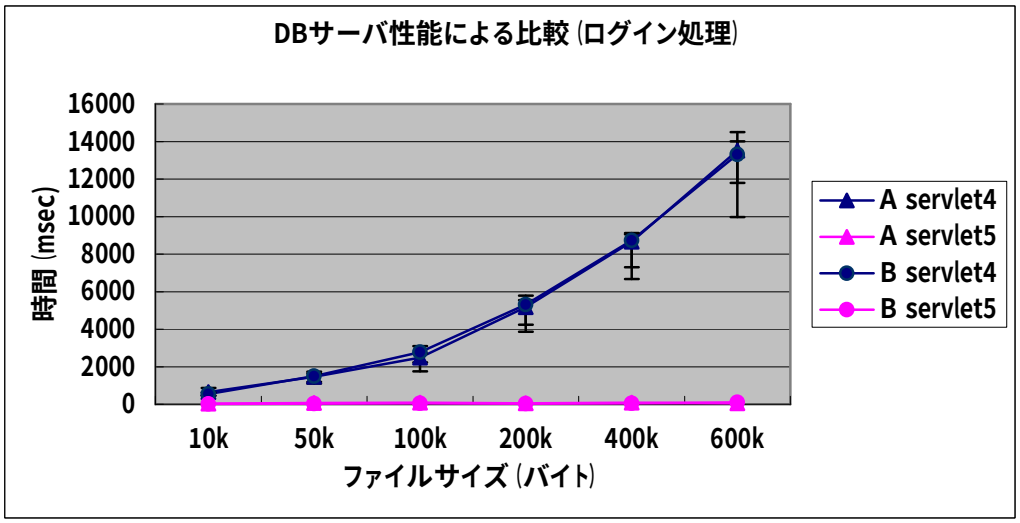

図 5.2.10　DB サーバ性能による比較（ログイン処理）

性能の高い DB サーバを使用しても、DBMS に接続を行っている Servlet3 と Servlet5 はそれほどクライアント-Web サーバ間のレスポンスタイムに差は見られない。この差は十分誤差の範囲内だと考えられる。ここで、Web サーバ側で計測を行った Servlet3 における DBMS へ接続を行っている部分の処理時間を図 5.2.11 に示す。

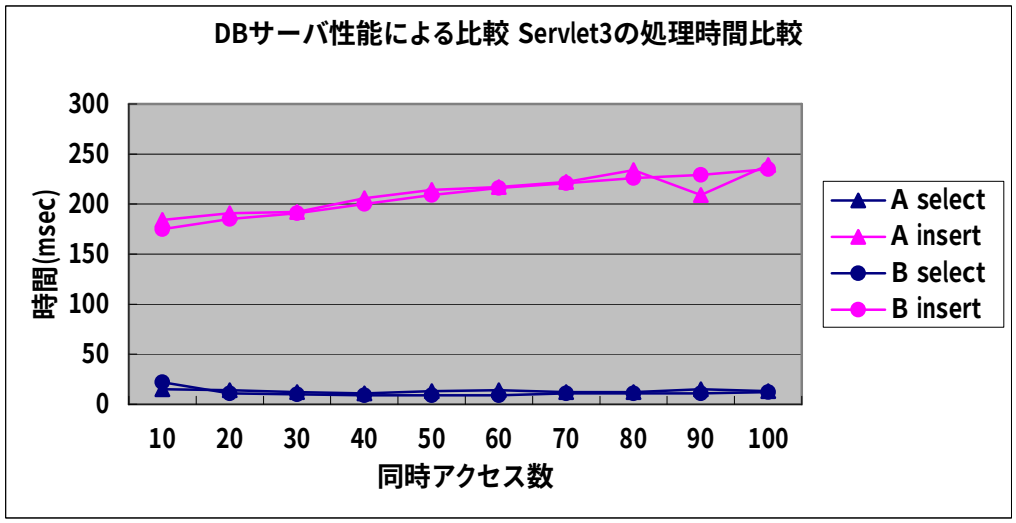

図 5.2.11　DB サーバ性能による比較　Servlet3 の処理時間比較

上記図を見ると、Web サーバ-DB サーバ間のレスポンスタイムにはほとんど差が見られない。先ほど述べたように、クライアント-Web サーバ間のレスポンスタイムにも差があまりないため、DB サーバの性能は本システムの速度にほとんど影響を与えないと考えられる。

5.2.4　DB サーバと Web サーバ共存時の比較

サーバ機の組み合わせが条件Dである場合は、サーバ機 vajra 内に Web サーバと DB サーバの両方が存在する事になる。DB サーバを別体型とした場合との速度等の差が出るか比較を行う。本来であれば比較には、サーバ機 vajra と全く同一性能のサーバ機を用意する必要がある。しかし 5.2.3 節の実験において、DB サーバの性能は本システムの速度にほとんど影響を与えない事がわかっている。よって、条件 C と条件Dの場合の比較を行えば良いことになる。実験条件を表 5.2.4、実験結果を図 5.2.12～図 5.2.15 に示す。

表 5.2.4　実験条件 4

サーバ機条件	条件C、条件D
JDBC 使用方法	Data Source
変化させる項目	同時アクセス数、ファイルサイズ

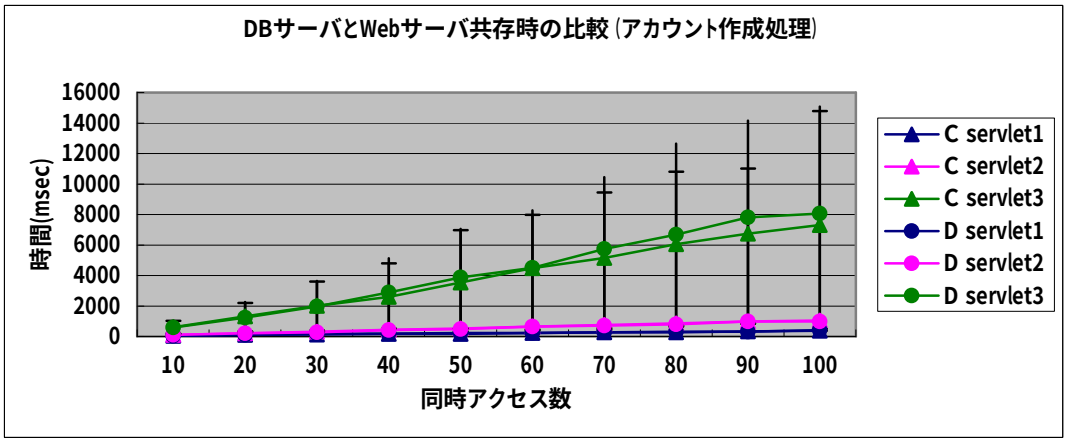

図 5.2.12　DB サーバと Web サーバ共存時の比較（アカウント作成処理）

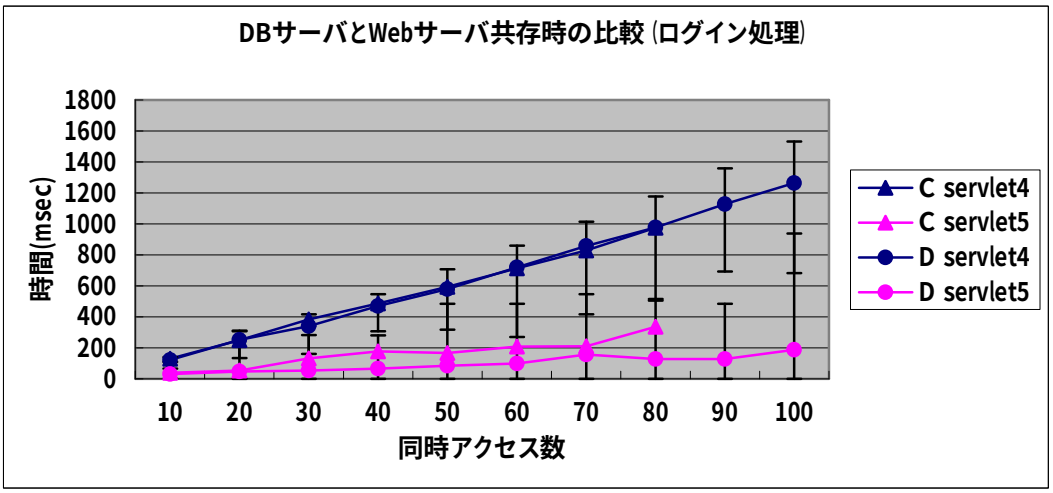

図 5.2.13　DB サーバと Web サーバ共存時の比較（ログイン処理）

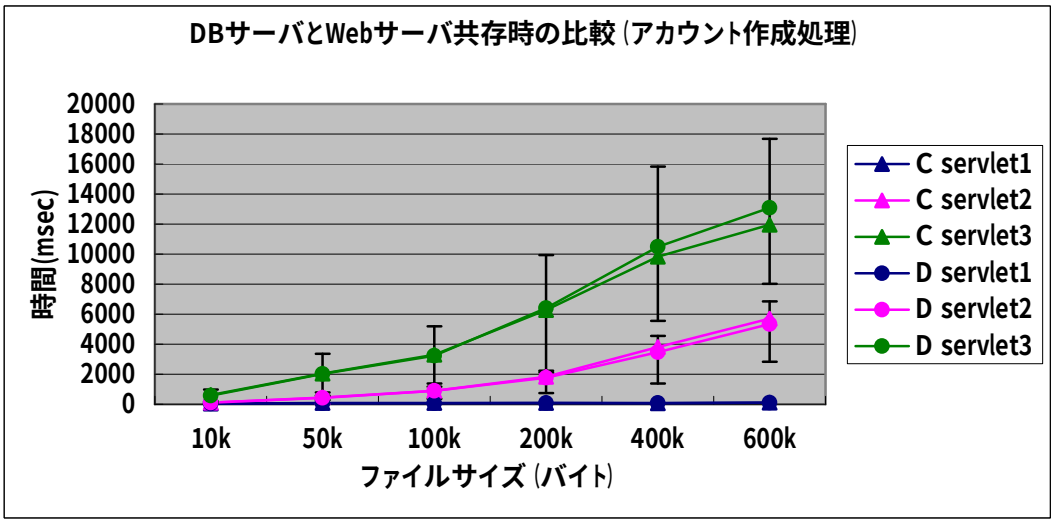

図 5.2.14　DB サーバと Web サーバ共存時の比較（アカウント作成処理）

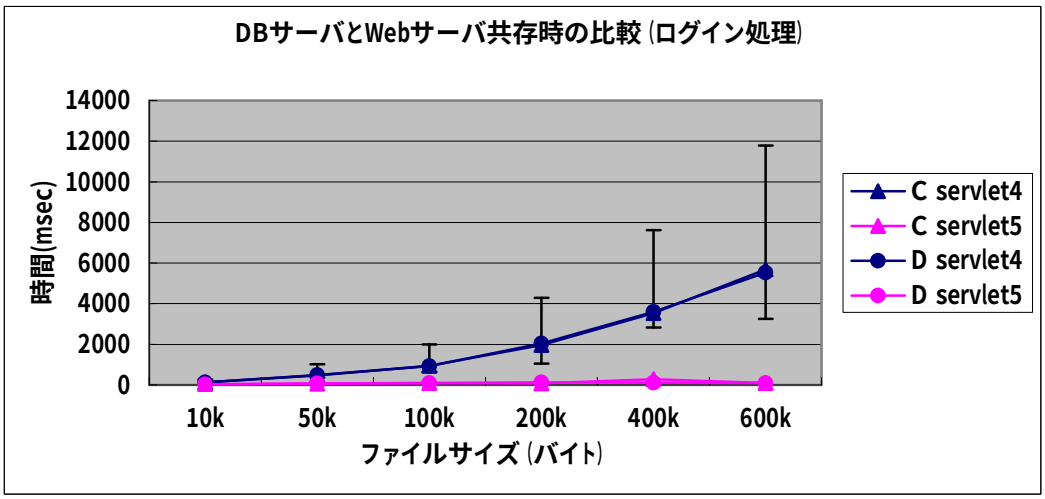

図 5.2.15　DB サーバと Web サーバ共存時の比較（ログイン処理）

結果を見てみると、さほど条件Cと条件Dで速度差は見られない。ただしアカウント作成処理時の Servlet3 において、同時アクセス数、またはファイルサイズが大きくなるにつれて条件Cの方が速い結果を得られている。これは Servlet3 において共存している DBMS へのアクセスが発生し、サーバ機 vajra に通常よりも大きい負荷がかかっているためと予想される。

実際にシステムが稼動する予定の環境である条件Dについて見てみると、負荷がログイン処理よりも格段に大きいと予想されるアカウント作成時に、同時アクセス数が 100 の時に全体のレスポンスタイム（Servlet1～Servlet3 の和）は 10 秒以下である。ファイルサイズを増加させた場合では 600KB 時に全体のレスポンスタイムは 20 秒程度と長い。本実験

では、スレッド生成により、ほぼ同時のアクセスを行った。しかし実際に利用される場合は完全に画像 Drop のタイミング重複することはあまりなく、負荷はもう少し分散されると考えられる。よって実際の稼動時にはもう少し高い耐久性が実現できる可能性がある。

実際にログイン画像として用いる画像であるが、ネットワークを通じて画像を送信する事を考えれば、アカウント作成処理時の全体のレスポンスタイムが 10 秒以下となる 200KB 程度の画像が適当であると考えられる。実際のシステム上では、ファイルサイズが上限を超えている場合は圧縮をかけるプログラムを通す、といった方法も考えられる。

ログイン処理に関しては負荷がそこまで大きな処理を行っていないため、現状でも問題ないと考えられる。

5.2.5 考察

前節で画像ファイルサイズは 200KB 程度が適当であると考えられると示した。そこで、追加実験として画像ファイルサイズを 200KB として同時アクセス数を増加させていった場合、どの程度の負荷まで耐え得るのか、またその時のレスポンスタイムに注目した。結果を表 5.2.5 に示す。

表 5.2.5　追加実験結果

ファイルサイズ	同時アクセス数	Servlet3	Servlet4
100KB	50	約 16 秒	約 4.3 秒
200KB	50	約 38 秒	約 9 秒

比較の参考のためにファイルサイズが 100KB の場合も計測を行った。Servlet3 と Servlet4 はそれぞれ、新規アカウント作成処理、ログイン処理で最も処理時間全体で占めている割合が大きい Servlet である。200KB で同時アクセス数を 50 より増加させた場合、Web サーバの過負荷により、正常に新規アカウント作成処理が実行できなかった。そのため今回は同時アクセス数を 50 で統一している。

Servlet3 のレスポンスタイムは登録メールアドレスへメール送信が完了するまでの時間、すなわち 38 秒という時間はメールが到着するまでの大よその最低時間を表している。よって利用上さほど大きな問題にはならないと考えられる。

Servlet4 にレスポンスタイム、すなわち画像が Drop されてから、次の画像が Drop 可能となるまでの時間であるが、ユーザが次の画像をディレクトリから探す時間を考慮すると、こちらもそれほど問題にならないと考えられる。

よってこれらより表 5.1.1 に示すような環境を用いても、画像ファイルサイズが 200KB 程度、同時アクセス数が 50 程度の小規模な利用であれば、十分本研究のシステムは実用に耐え得る範囲内であると考えられる。

第 6 章　結論

6.1　まとめ

本研究では先行研究のシステムを、安全かつ実用に向けた改良を行うことができた。従来、HTML の送信フォームからログイン画像を送信していたことを考えると、Drag & Drop の組み込みにより、ユーザの負担を減らす事ができた。また JPEG 画像への対応により、使用する画像ファイルのサイズが小さくなり、ユーザがより使用しやすくなった。

ニーモニック認証を実装することで、先行研究での問題点を解決することもでき、より高いセキュリティを実現可能とするシステムの雛形を実装できたと言える。

性能評価実験を行う事により、画像ファイルサイズが 200KB 程度、同時アクセス数が 50 程度という制限があるものの十分実用が可能である事を示すことができた。

6.2　今後の課題

ニーモニック認証を実装したことにより、多少利便性が低くなっているのも事実である。24bit のフルカラーJPEG 画像にしか対応していないなど、まだまだ条件的に限られたシステムである。例えば他のステガノグラフィのアルゴリズムの実装により、他の画像形式にも対応させる事が可能になれば、ユーザの選択肢が増加する。

また前述した通り、今回はログイン画像を 3 枚として試験的に実装を行ったが、理想としてはユーザがログイン画像の枚数を決定することが出来ればそれが最適である。

謝辞

本研究を進めるにあたって、最後まで熱心なご指導をいただきました田中章司郎教授には、心より御礼申し上げます。

また、同じ研究室の方々をはじめとして、他の研究室の方々、また他の多くの御助言をいただいた方々には厚く御礼申し上げます。なお、本論文、本研究で作成したプログラム及びデータ、並びに発表資料等の全ての知的財産権を、本研究の指導教官である田中章司郎教授に譲渡致します。

文献

[1]喜代吉　容大「パスワード埋込み画像によるログインシステムの実装と評価」
島根大学大学院　総合理工学研究科　修士論文 , 2004

[2]河口英二, 野田秀樹, 新見道治「画像を用いたステガノグラフィ」, 情報処理 2003 vol.44 No.3

[3]小野文孝,渡辺裕「国際標準画像符号化の基礎技術」,コロナ社，1998

[4]高木　明「ステガノグラフィを用いた文書流出のリアルタイム検出」
島根大学大学院　総合理工学研究科　修士論文 ,2003

[5] スコット・オークス「Java セキュリティ」,オライリー・ジャパン ，2001

[6]RSA 署名付き証明書を使用したアプレットの署名方法，http://sdc.sun.co.jp/java/docs/j2se/1.4/ja/docs/ja/guide/plugin/developer_guide/rsa_signing.html

[7]デイビッド・フラナガン：「JavaScript」,オライリー・ジャパン,2000

[8]Data source,
http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/jdbc/getstart/datasource.html

[9] ジム・メルトン「SQL：1999」 ，ピアソン・エデュケーション，2003

[10]秋本芳伸，岡田泰子，青島純一「HiRDB Version7 スタートアップガイド」，翔泳社，2003

[11]株式会社ニーモニックセキュリティ，http://www.mneme.co.jp/

付録　実験データ

Ⅰ．性能評価実験で得られた実験結果データ（レスポンスタイム）を以下に示す。

表 9.1　JDBC 使用方法の違いによるレスポンスタイム比較（Servlet3）

	10	20	30	40	50	60	70	80	90	100
Driver Manager	4431	8667	11523	15284	18760	22473	26596	30309	33663	43228
Data Source	1855	3500	5067	8411	10191	11519	13050	15578	18273	20933

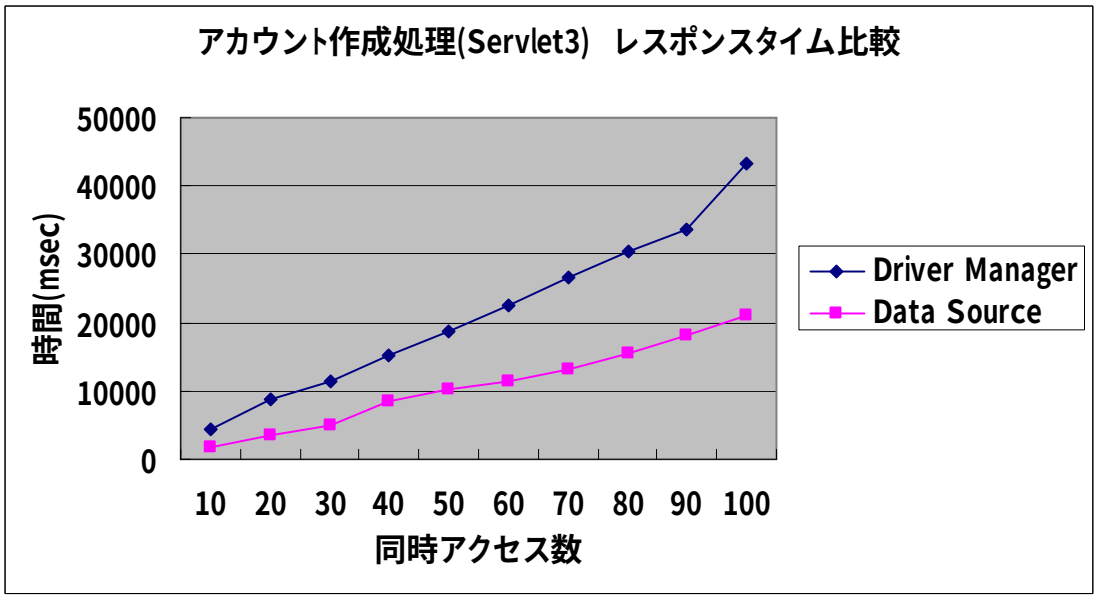

図 9.1　JDBC 使用方法の違いによるレスポンスタイム比較（Servlet3）

表 9.2　JDBC 使用方法の違いによるレスポンスタイム比較（Servlet5）

	10	20	30	40	50	60	70
Driver Manager	2463	4548	6833	8358	11383	13120	
Data Source	33	32	43	289	179	878	1187

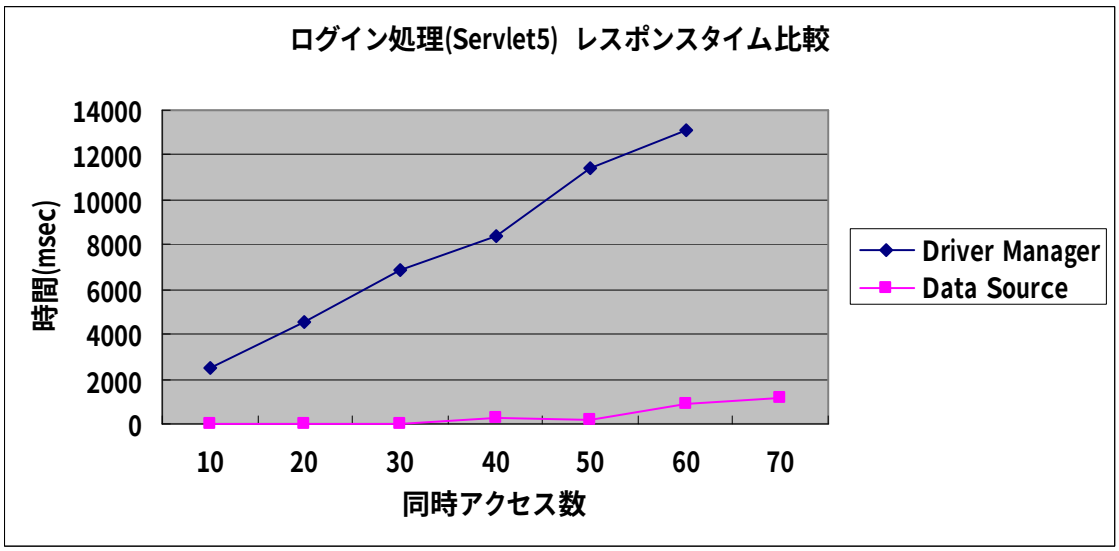

図 9.2　JDBC 使用方法の違いによるレスポンスタイム比較（Servlet5）

表 9.3　Web サーバの性能による比較（アカウント作成処理、同時アクセス数変化）

	A servlet1	A servlet2	A servlet3	C servlet1	C servlet2	C servlet3
10	279	331.96	2564	46	109.8	612
20	241	661.8	4181	114	203.1	1312
30	274	968.2	6104	128	316.6	2015
40	269	1410.1	8006	173	434.2	2596
50	479	1690.8	10531	174	527.2	3539
60	562	1981.3	12458	222	654.7	4487
70	749	2123	14833	270	748.3	5149
80	997	2503.7	17191	264	867.7	6056
90	1365	2613.8	20121	326	985.3	6749
100	1403	2947.2	23091	387	1040.8	7303

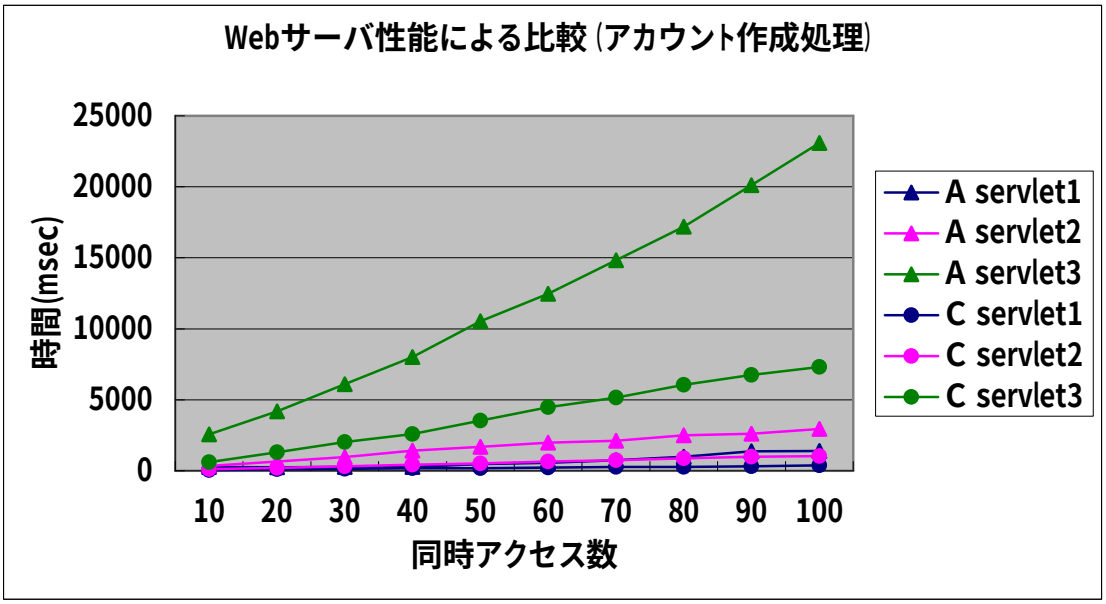

図 9.3　Web サーバの性能による比較（アカウント作成処理、同時アクセス数変化）

表 9.4　Web サーバの性能による比較（ログイン処理、同時アクセス数変化）

	A servlet4	A servlet5	C servlet4	C servlet5
10	639	32	130	39
20	1243	55	250	54
30	1820	101	383	131
40	2534	93	487	178
50	2770	94.4	594	166
60	3492	181	714	208
70	3810	159	828	210
80	4438	401	976	336
90	4863	438		
100	5540	560		

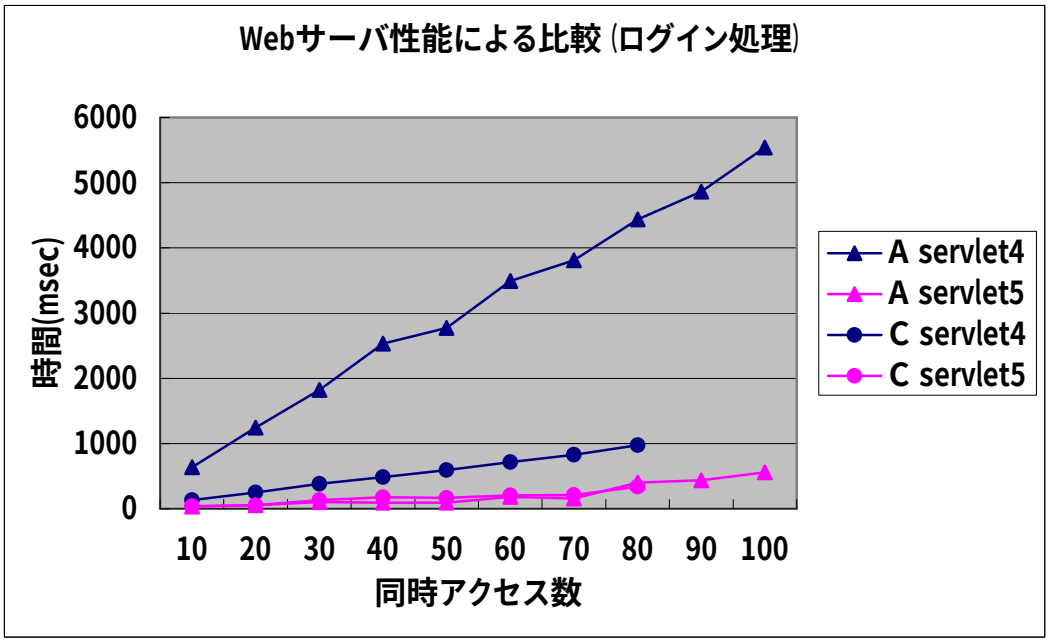

図 9.4　Web サーバの性能による比較（ログイン処理、同時アクセス数変化）

表 9.5　Web サーバの性能による比較（アカウント作成処理、ファイルサイズ変化）

	A servlet1	A servlet2	A servlet3	C servlet1	C servlet2	C servlet3
10k	279	331.9	2564	46	109.8	612
50k	227	1156.9	5073	83	451	2051
100k	201	2244.4	7301	88	894	3297
200k	196	4627.2	9888	81	1836.8	6269
400k	161	9157	14050	71	3800	9824
600k	160	14050	16001	103	9824	11964

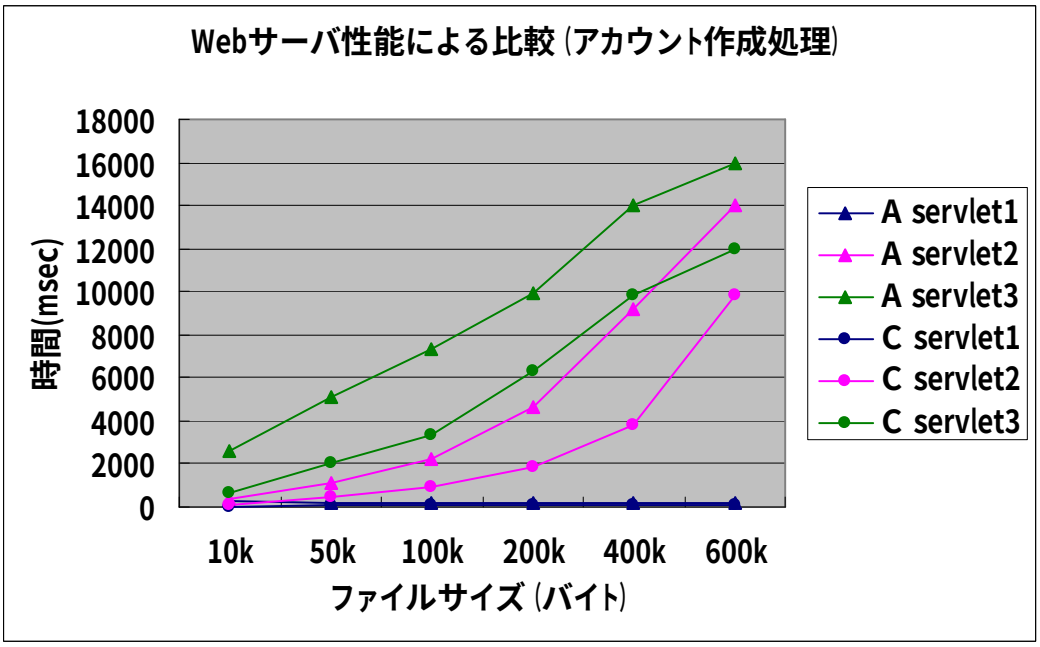

図 9.5　Web サーバの性能による比較（アカウント作成処理、ファイルサイズ変化）

表 9.6　Web サーバの性能による比較（ログイン処理、ファイルサイズ変化）

	A servlet4	A servlet5	C servlet4	C servlet5
10k	639	32	130	39
50k	1456	66	472	64.6
100k	2481	77	941	90
200k	5177	54	1964	78
400k	8670	85	3536	278
600k	13512	45	5685	100

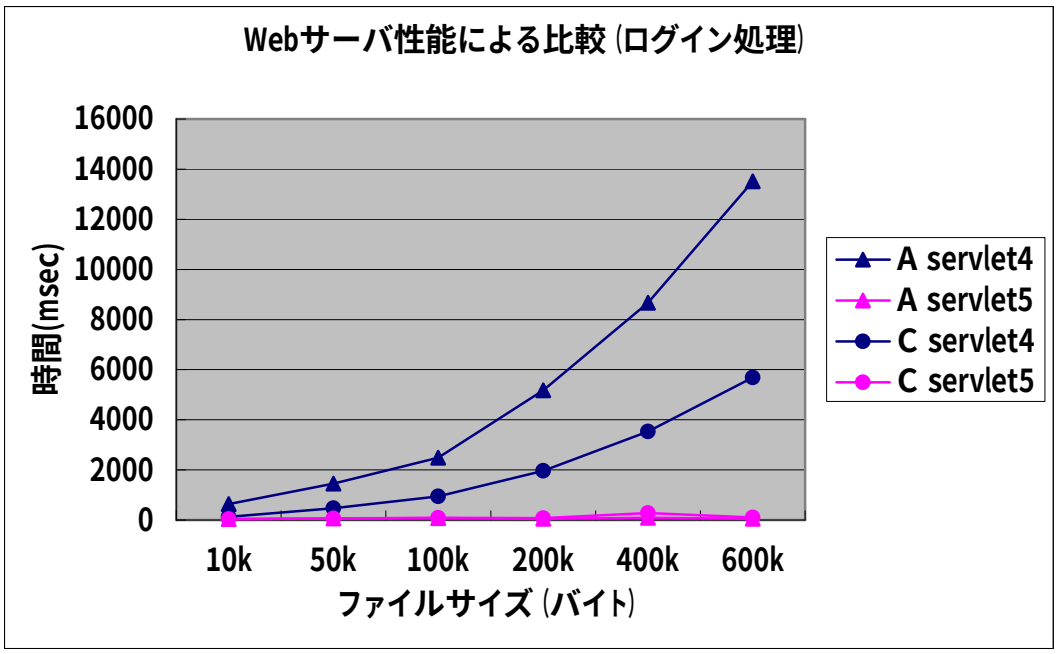

図 9.6　Web サーバの性能による比較（ログイン処理、ファイルサイズ変化）

表 9.7　DB サーバの性能による比較（アカウント作成処理、同時アクセス数変化）

	A servlet1	A servlet2	A servlet3	B servlet1	B servlet2	B servlet3
10	279	331.96	2564	103	288.8	1910
20	241	661.8	4181	208	541.2	4027
30	274	968.2	6104	233	950.3	6379
40	269	1410.1	8006	338	1221.3	7904
50	479	1690.8	10531	364	1543	11014
60	562	1981.3	12458	480	1886.6	11945
70	749	2123	14833	809	2133.9	16085
80	997	2503.7	17191	1055	2316.2	16786
90	1365	2613.8	20121	1166	2764	21112
100	1403	2947.2	23091	1359	3072.7	23735

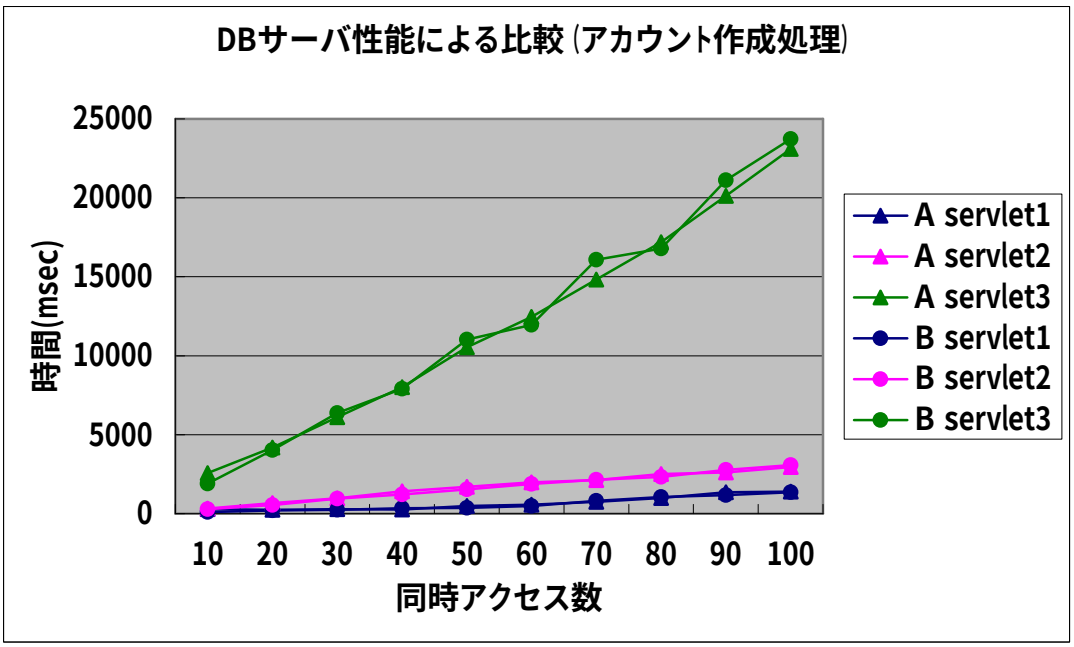

図 9.7　DB サーバの性能による比較（アカウント作成処理、同時アクセス数変化）

表 9.8　DB サーバの性能による比較（ログイン処理、同時アクセス数変化）

	A servlet4	A servlet5	B servlet4	B servlet5
10	639	32	552	18
20	1243	55	1097	16
30	1820	101	1629	38
40	2534	93	2156	66.7
50	2770	94.4	2807	145
60	3492	181	3112	85.5
70	3810	159	3704	168
80	4438	401	4624	213
90	4863	438	4621	237
100	5540	560	5345	384

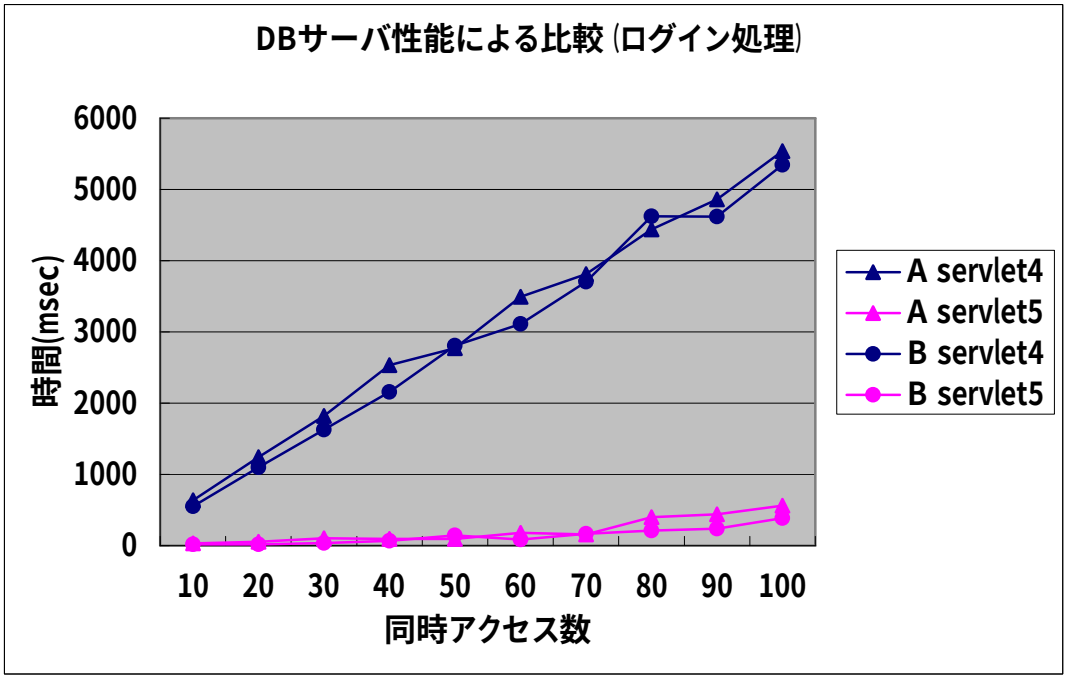

図 9.8　DB サーバの性能による比較（ログイン処理、同時アクセス数変化）

表 9.9　DB サーバの性能による比較（アカウント作成処理、ファイルサイズ変化）

	A servlet1	A servlet2	A servlet3	B servlet1	B servlet2	B servlet3
10k	279	331.9	2564	103	288.8	1910
50k	227	1156.9	5073	139	1097.9	4421
100k	201	2244.4	7301	130	2314.7	6018
200k	196	4627.2	9888	152	4780	10253
400k	161	9157	14050	259	9310.5	13884
600k	160	14050	16001	706	14242	15780

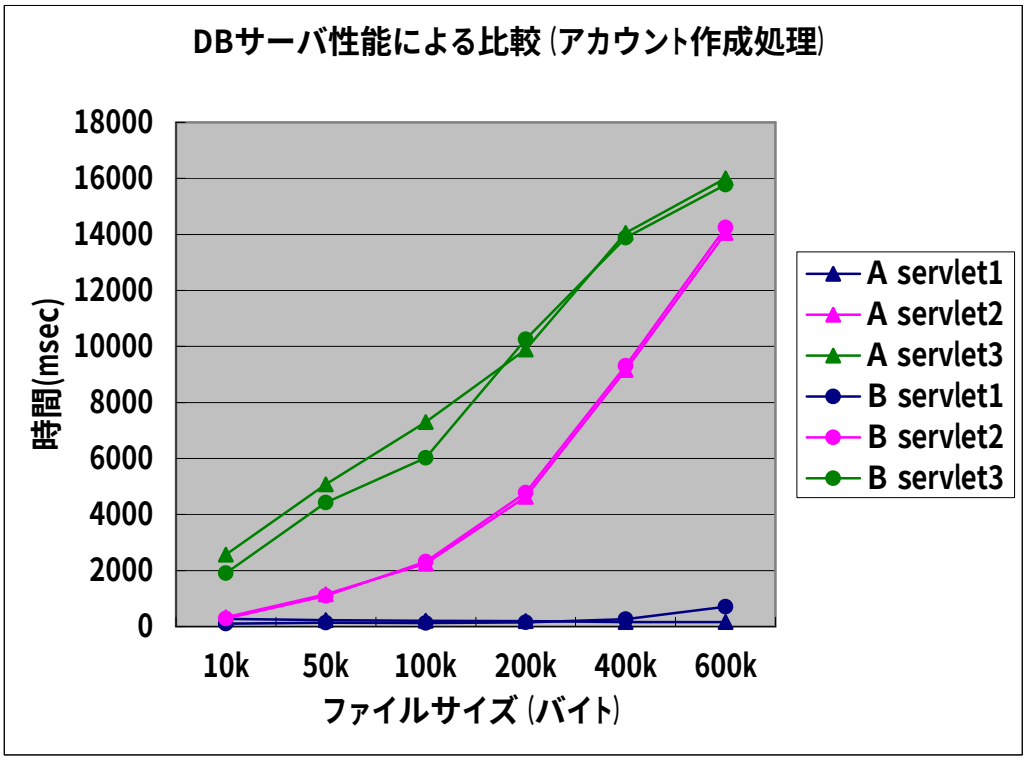

図 9.9　DB サーバの性能による比較（アカウント作成処理、ファイルサイズ変化）

表 9.10　DB サーバの性能による比較（ログイン処理、ファイルサイズ変化）

	A servlet4	A servlet5	B servlet4	B servlet5
10k	639	32	552	18
50k	1456	66	1490	37
100k	2481	77	2773	57
200k	5177	54	5316	27
400k	8670	85	8724	46
600k	13512	45	13313	94.7

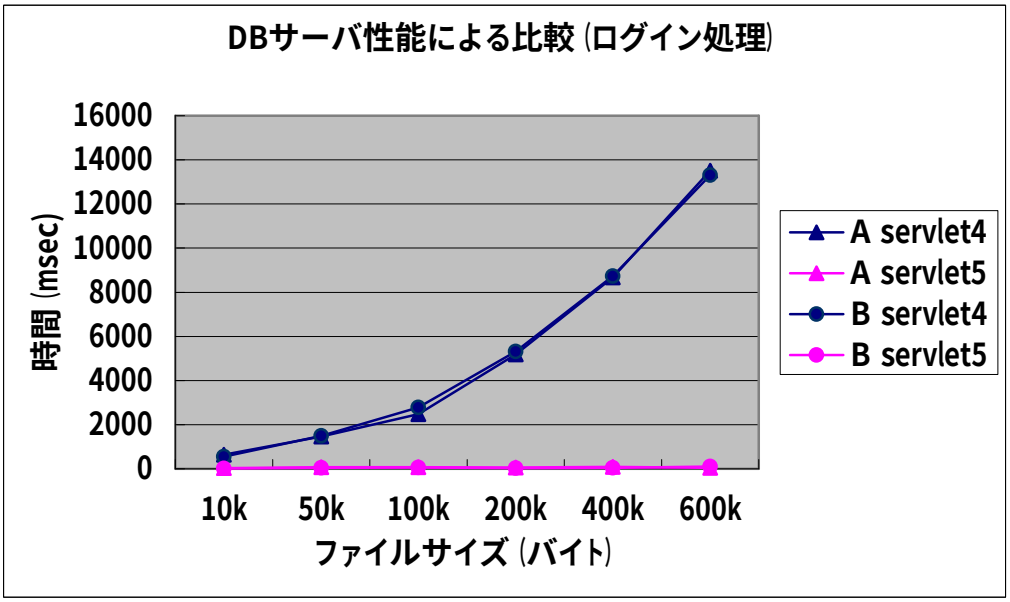

図 9.10　DB サーバの性能による比較（ログイン処理、ファイルサイズ変化）

表 9.11　DB サーバと Web サーバ共存時の比較（アカウント作成処理、同時アクセス数変化）

	C servlet1	C servlet2	C servlet3	D servlet1	D servlet2	D servlet3
10	46	109.8	612	36	99.2	581
20	114	203.1	1312	98	206.5	1234
30	128	316.6	2015	179	277.3	1963
40	173	434.2	2596	181	409.4	2887
50	174	527.2	3539	228	485.5	3866
60	222	654.7	4487	230	652.8	4507
70	270	748.3	5149	274	713.4	5742
80	264	867.7	6056	319	798.7	6674
90	326	985.3	6749	316	988.27	7807
100	387	1040.8	7303	422	981.65	8053

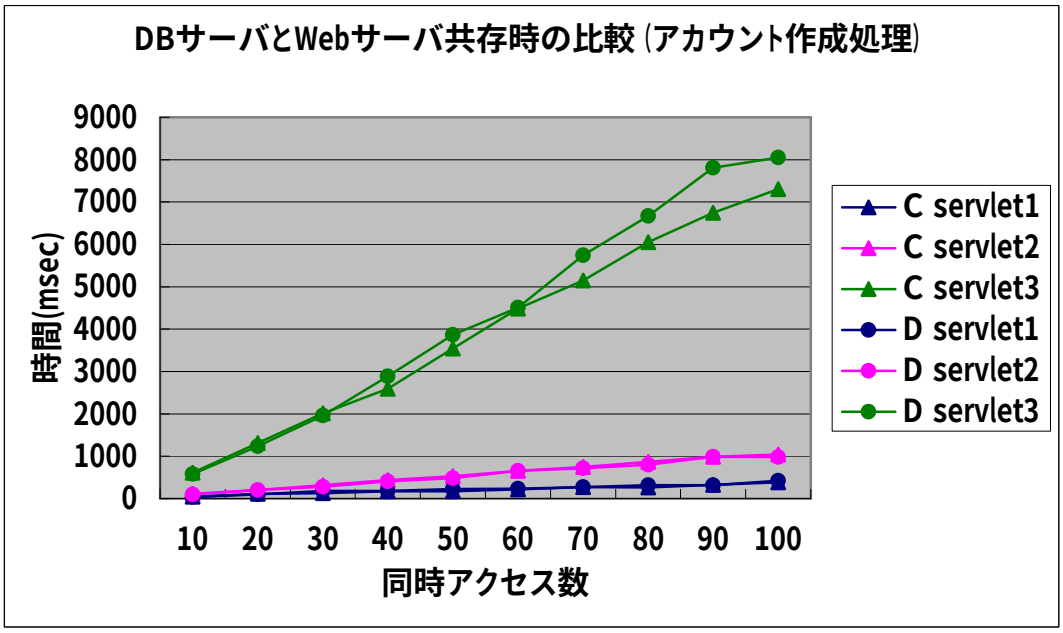

図 9.11　DB サーバと web サーバ共存時の比較（アカウント作成、同時アクセス数変化）

表 9.12　DB サーバと Web サーバ共存時の比較（ログイン処理、同時アクセス数変化）

	C servlet4	C servlet5	D servlet4	D servlet5
10	130	39	125	31
20	250	54	251	47
30	383	131	341	54
40	487	178	470	67
50	594	166	580	85
60	714	208	720	99
70	828	210	857	156
80	976	336	978	128
90			1128	128
100			1264	188

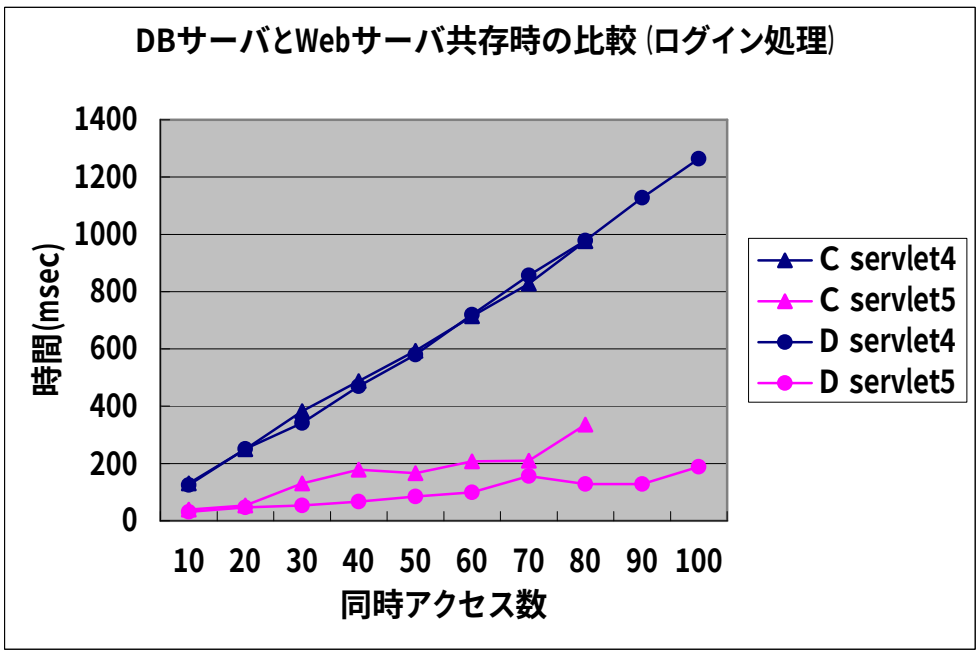

図 9.12　DB サーバと Web サーバ共存時の比較（ログイン処理、同時アクセス数変化）

表 9.13　DB サーバと Web サーバ共存時の比較（アカウント作成処理、ファイルサイズ変化）

	C servlet1	C servlet2	C servlet3	D servlet1	D servlet2	D servlet3
10k	46	109.8	612	36	99.2	581
50k	83	451	2051	70	434	2015
100k	88	894	3297	59	896.9	3229
200k	81	1836.8	6269	83	1759.6	6407
400k	71	3800	9824	82	3466.7	10500
600k	103	5695.9	11964	97	5331.3	13077

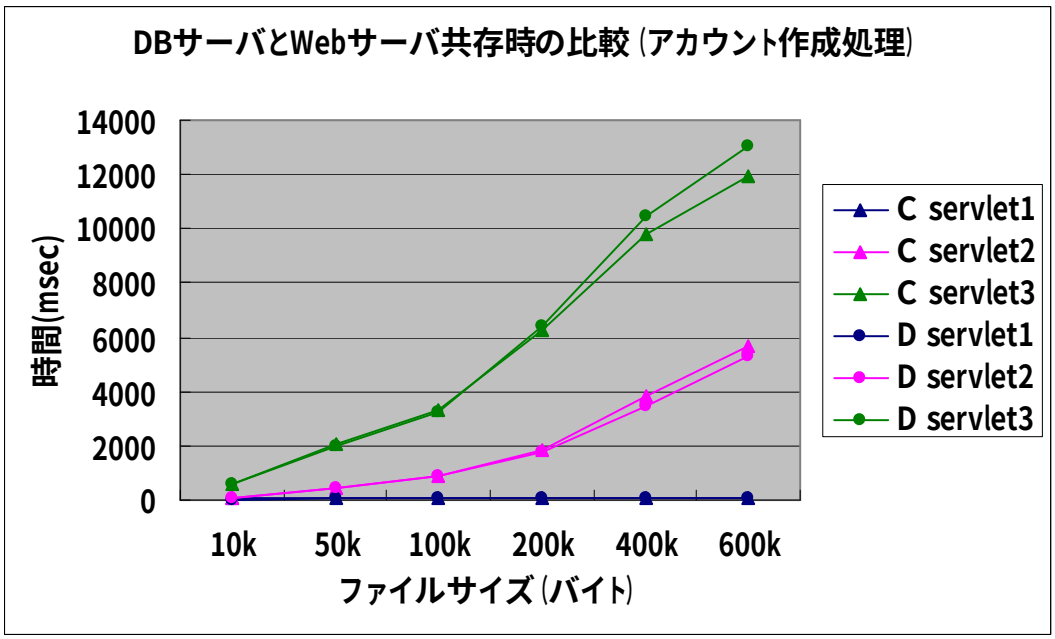

図 9.13　DB サーバと Web サーバ共存時の比較（アカウント作成処理、ファイルサイズ変化）

表 9.14　DB サーバと Web サーバ共存時の比較（ログイン処理、ファイルサイズ変化）

	C servlet4	C servlet5	D servlet4	D servlet5
10k	130	39	125	31
50k	472	64.6	483	73
100k	941	90	927	102
200k	1964	78	2035	120
400k	3536	278	3606	107
600k	5685	100	5535	99

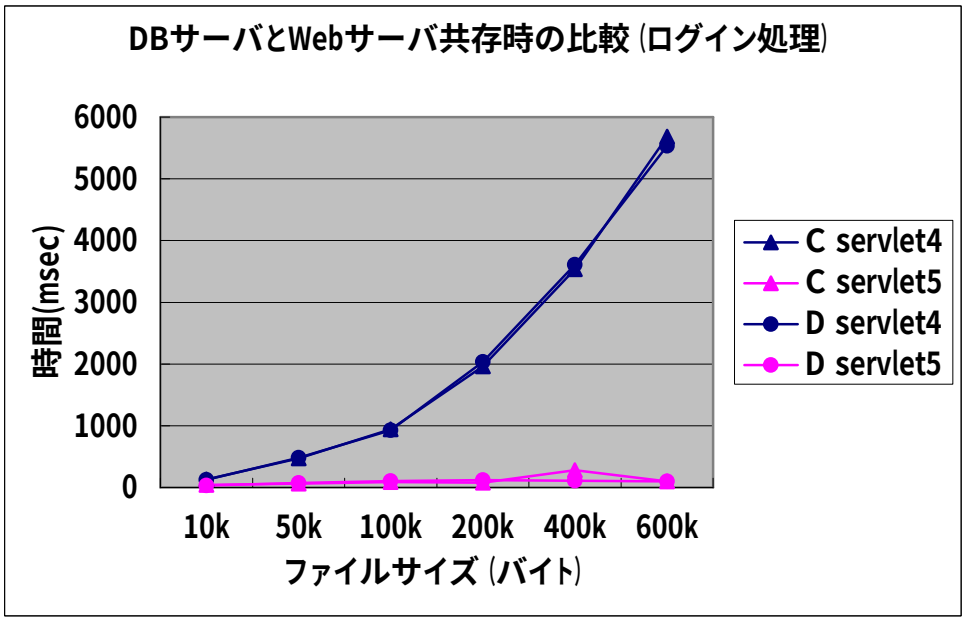

図 9.14　DB サーバと Web サーバ共存時の比較（ログイン処理、ファイルサイズ変化）

II．性能評価実験で得られた実験結果データ（Servlet 処理時間の内訳）を以下に示す。
なお、グラフ中に Servlet 名の記述がないものは、アカウント作成処理では Servlet3、ログイン処理では Servlet5 の処理を表す。

表 9.15　処理時間内訳（条件A、アカウント作成処理、同時アクセス数変化）

	TempFile Read	password 発行	DB 接続準備	select	隠蔽	insert	servlet1(フォームデータ処理)	servlet2 (画像アップロード)
10	68	0	32.5	15	185	184	43	211
20	53.1	0	0	14	228	191	17	152
30	4	0	0	12	226	192	4	81.4
40	41	0	0	11	236	206	3	89.6
50	111	0	0	13	236	214	3	136
60	7.8	0	0	14	243	217	5.3	47
70	7.4	0	0	12	242	222	5.9	54.5
80	12.3	0	0	12	241	234	5.3	67.2
90	116	0	0	15	245	209	6.1	142
100	151	0	0	13	246	239	6	130

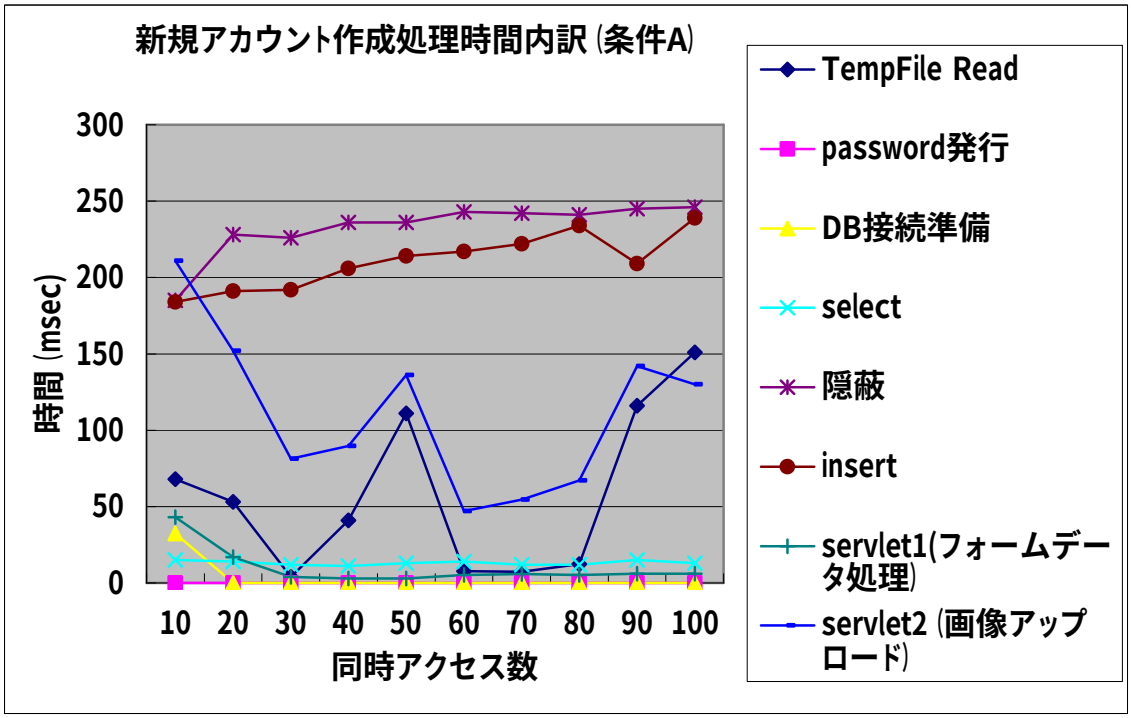

図 9.15　処理時間内訳（条件A、アカウント作成処理、同時アクセス数変化）

表 9.16　処理時間内訳（条件A、ログイン処理、同時アクセス数変化）

	TempFile Read	DB 接続準備	select	servlet4 (ログイン画像アップロード,抽出処理)
10	1	0	23	558
20	3.6	0	28	1103
30	5.3	0	27	1507
40	3.5	1	29	1904
50	2	0	28	2135
60	7.2	0	33	2634
70	5.1	0	36	2835
80	2.9	0	30	3036
90	3.2	0	36.5	3383
100	5.3	0	32	3613

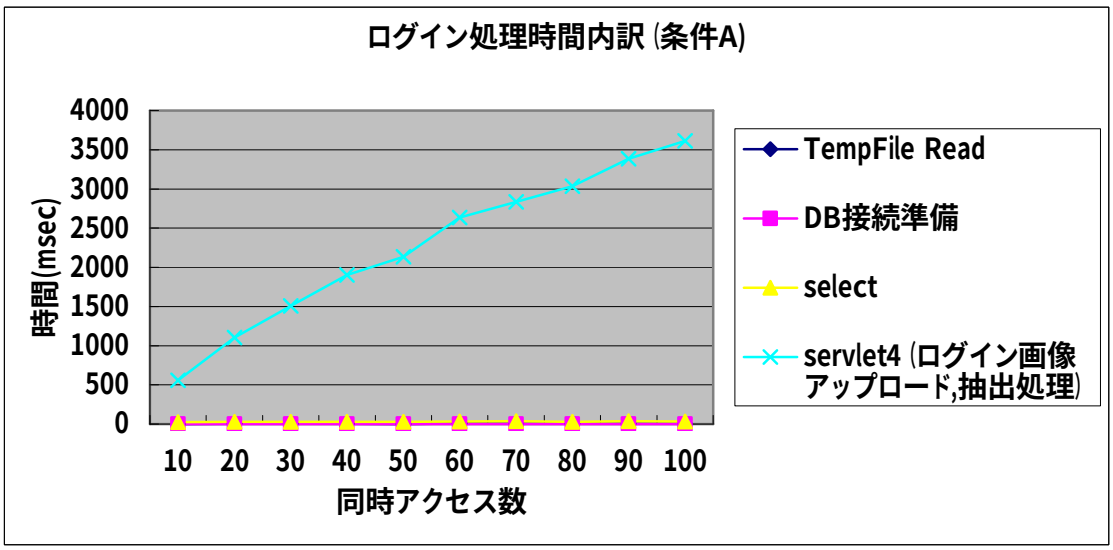

表 9.16　処理時間内訳（条件A、ログイン処理、同時アクセス数変化）

表 9.17　処理時間内訳（条件A、アカウント作成処理、ファイルサイズ変化）

	TempFile Read	password 発行	DB 接続準備	select	隠蔽	insert	servlet1(フォームデータ処理)	servlet2 (画像アップロード)
10K	68	0	32.5	15	185	184	3	211
50K	2	1	19	15	660	209	2	805
100K	19	0	0	11	976	208	5.6	1907
200K	27	0	0	12	1692	208	7.6	4419
400K	4	0	0	11	2500	213	2	8932
600K	79.3	0	0	12	2800	211	3	14010

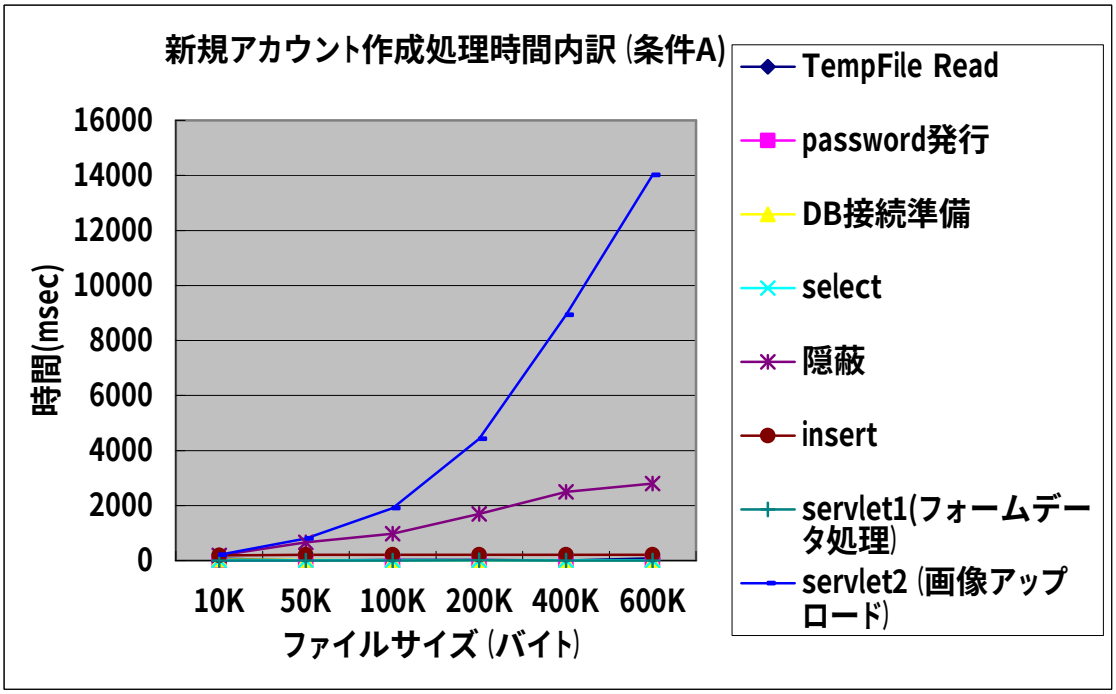

図 9.17　処理時間内訳（条件A、アカウント作成処理、ファイルサイズ変化）

表 9.18　処理時間内訳（条件A、ログイン処理、ファイルサイズ変化）

	TempFile Read	DB 接続準備	select	servlet4 (ログイン画像アップロード, 抽出処理)
10K	1	0	23	558
50K	25	0	17	1302
100K	3	0	23	2284
200K	6.6	0	32	4930
400K	11	0	22	8542
600K	5	0	20	13319

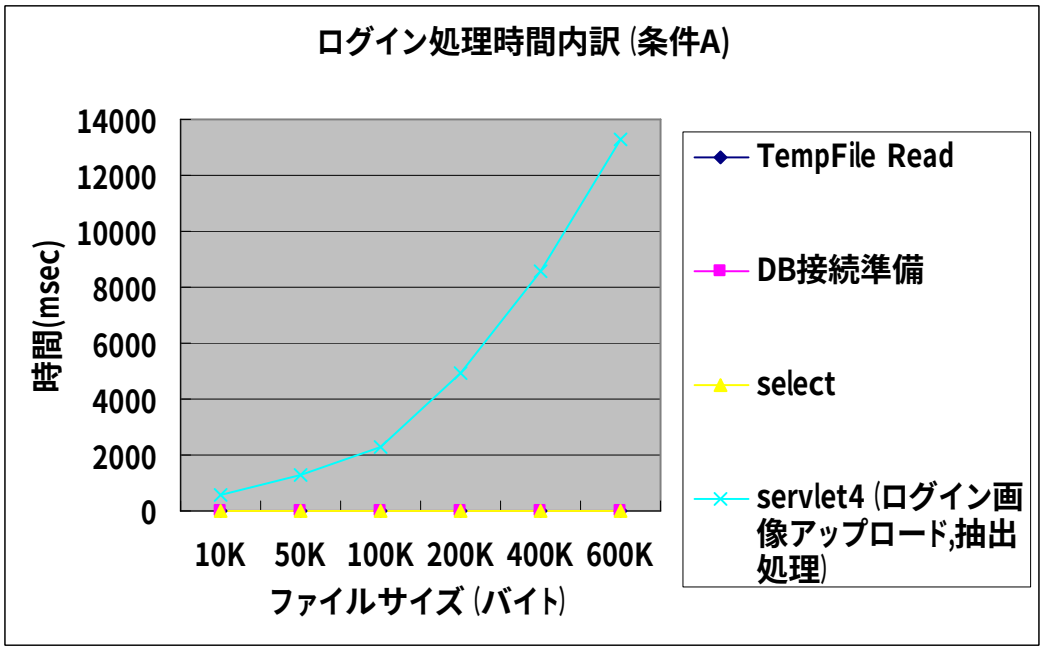

図 9.18　処理時間内訳（条件A、ログイン処理、ファイルサイズ変化）

表 9.19　処理時間内訳（条件Ｂ、アカウント作成処理、同時アクセス数変化）

	TempFile Read	password 発行	DB 接続準備	select	隠蔽	insert	servlet1(フォームデータ処理)	servlet2 (画像アップロード)
10	22	0	0	22	152	175	16	150
20	5.1	0	0	11	213	185	13.4	87.3
30	7.9	0	0	10	248	191	7	65
40	18.3	1.1	0	9	218	200	3	204
50	4	0	0	9	260	209	12	57.5
60	9.5	0	0	9	223	216	13	59
70	18	0	0	11	278	221	34.4	120
80	5.5	0	0	11	236	226	22	70.9
90	107	0	0	11	287	229	14	165
100	4.9	0	0	12	284	235	14.9	47.5

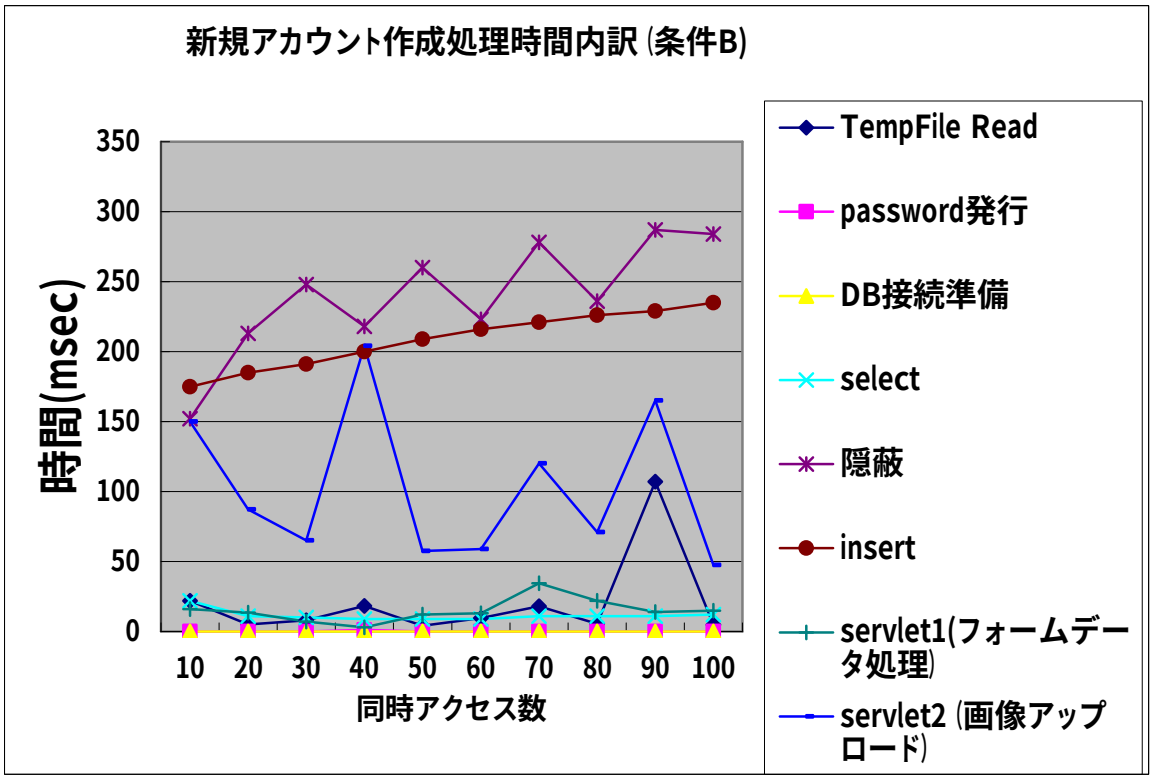

図 9.19　処理時間内訳（条件Ｂ、アカウント作成処理、同時アクセス数変化）

表 9.20　処理時間内訳（条件Ｂ、ログイン処理、同時アクセス数変化）

	TempFile Read	DB 接続準備	select	servlet4 (ログイン画像アップロード,抽出処理)
10	3	0	9	491
20	3	0	7	984
30	5.9	1	12	1342
40	12	0	20	1404
50	74	1	27	2074
60	4.9	0	19	1854
70	27.5	0.4	36	2300
80	6.42	0	27	2817
90	54.3	0.7	46.7	2649
100	36.9	0	35.2	2874

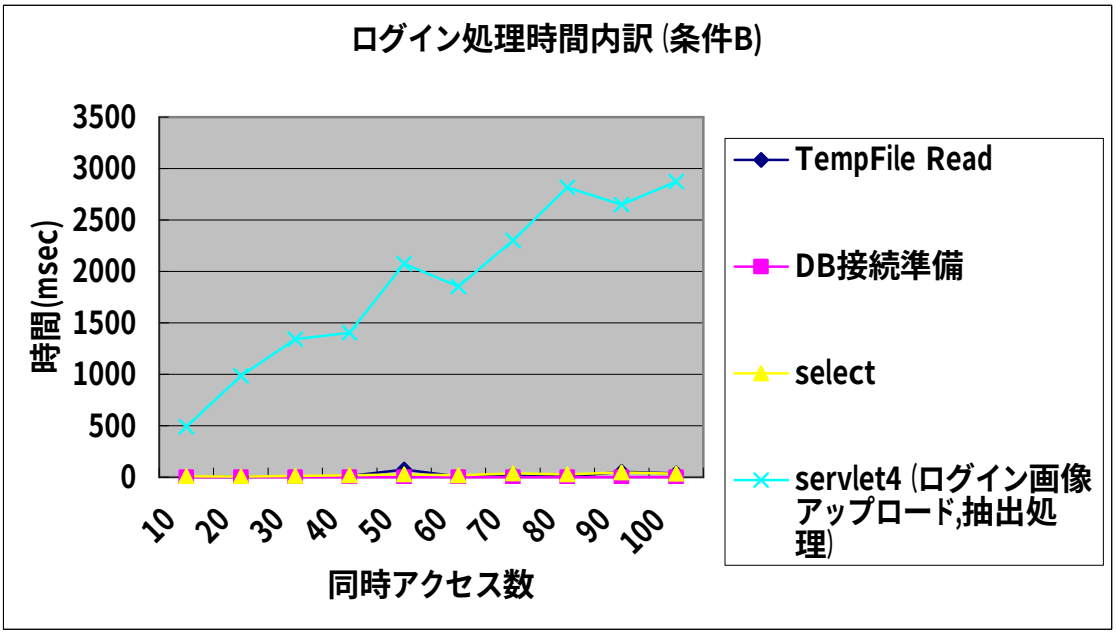

図 9.20　処理時間内訳（条件Ｂ、ログイン処理、同時アクセス数変化）

表 9.21　処理時間内訳（条件Ｂ、新規アカウント作成処理、ファイルサイズ変化）

	TempFile Read	password 発行	DB 接続準備	select	隠蔽	insert	servlet1(フォームデータ処理)	servlet2 (画像アップロード)
10K	22	0	0	22	152	175	16	150
50K	93.9	0	0	8.605	195	175	2	693
100K	3	0	0	8	1029	199	3	2091
200K	7.2	0	1	8	1873	198	4	4573
400K	49	0	0	8	2416	175	13	9102
600K	63.7	0	0	9	2819	179	3	14023

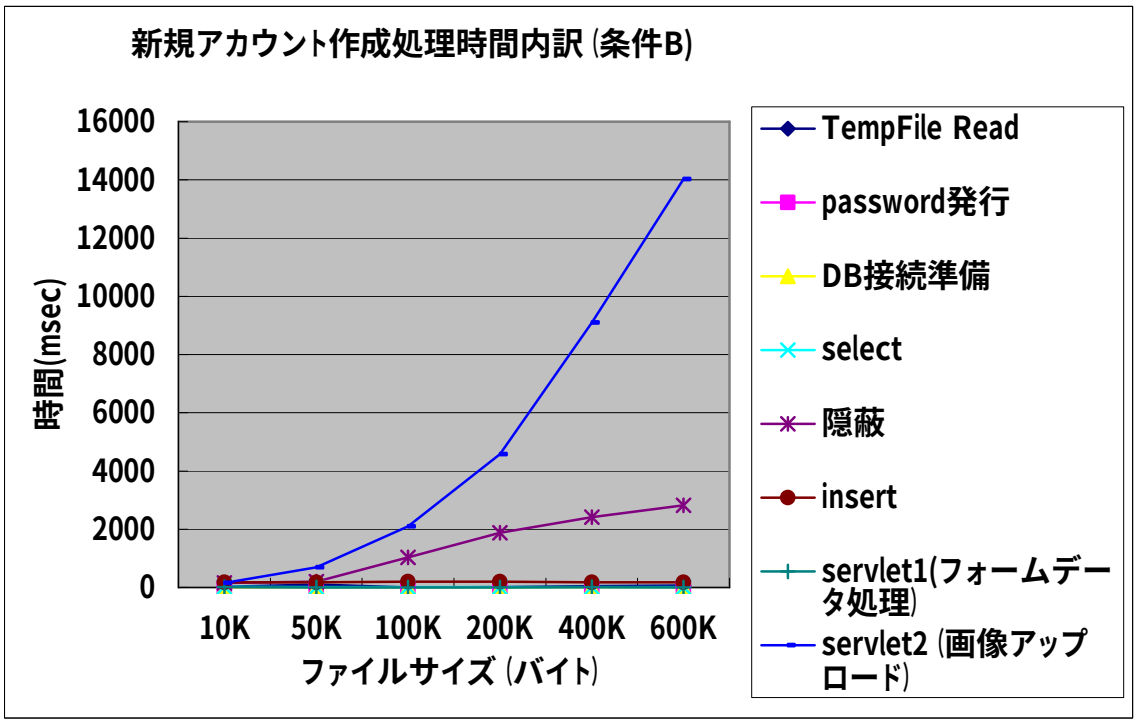

図 9.21　処理時間内訳（条件Ｂ、新規アカウント作成処理、ファイルサイズ変化）

表 9.22　処理時間内訳（条件Ｂ、ログイン処理、ファイルサイズ変化）

	TempFile Read	DB 接続準備	select	servlet4 (ログイン画像アップロード,抽出処理)
10K	3	0	9	491
50K	8.4	0	18	1334
100K	7.2	0	27	2538
200K	1	0	11	5070
400K	0	0	15	8464
600K	21.2	0	24	13036

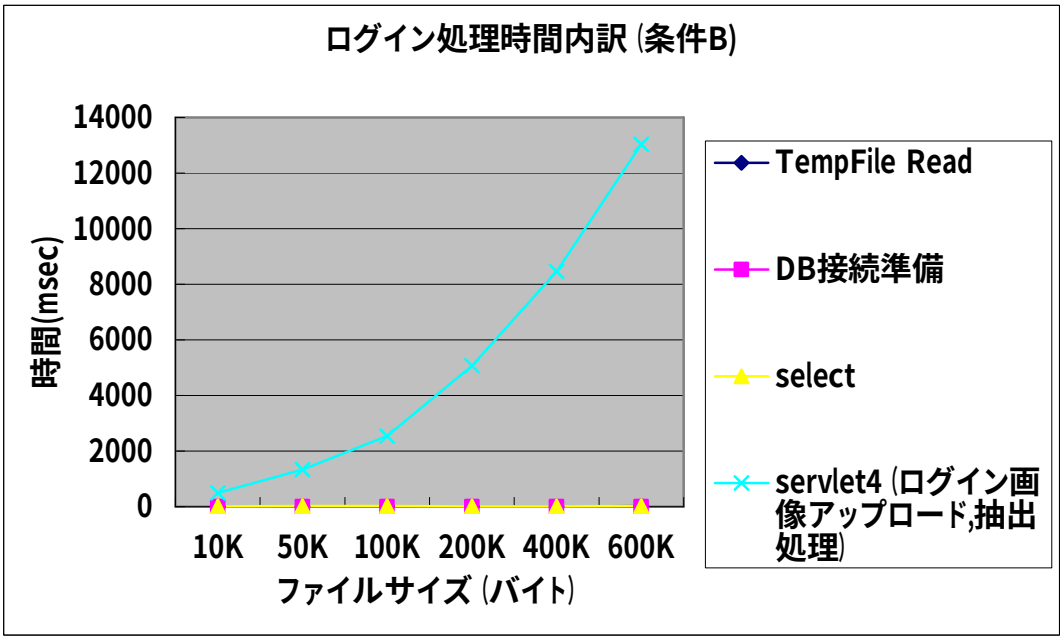

図 9.22　処理時間内訳（条件Ｂ、ログイン処理、ファイルサイズ変化）

表 9.23　処理時間内訳（条件C、アカウント作成処理、同時アクセス数変化）

	TempFile Read	password 発行	DB 接続準備	select	隠蔽	insert	servlet1(フォームデータ処理)	servlet2 (画像アップロード)
10	6	0	0	7.46	59	62	1	71
20	22	0	0	5	57	64	15	139
30	70	0	0	6.6	72	65	9.6	239
40	93	0	0	7	65	68	18	367
50	120	0	0	7	61	78	30	423
60	86	0	0	7	72.1	82	35	562
70	159	0	0	6	66	84	44	656
80	266	0.4	0	7	64.9	87	82	772
90	242	0.8	0	8	69.5	92	133	889
100	230	5.7	0	8	68	84	77	868

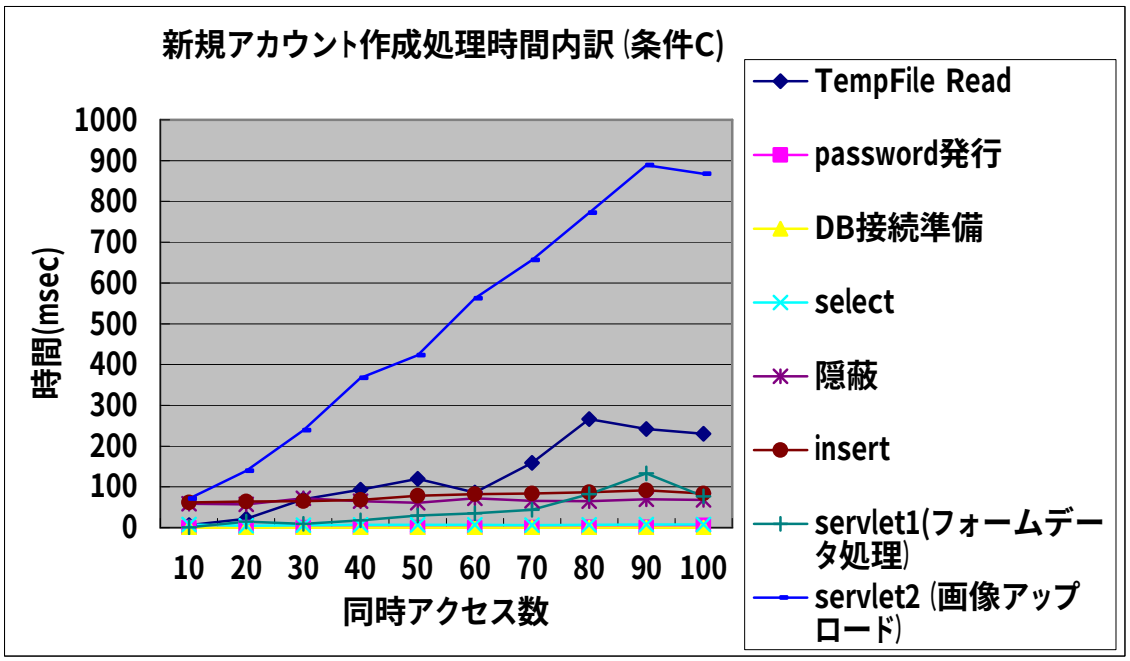

図 9.23　処理時間内訳（条件C、アカウント作成処理、同時アクセス数変化）

表 9.24　処理時間内訳（条件C、ログイン処理、同時アクセス数変化）

	TempFile Read	DB 接続準備	select	servlet4 (ログイン画像アップロード,抽出処理)
10	1	0	29	101
20	6	0	31	207
30	20	0	86	336
40	28	0	109	420
50	29	0	100	529
60	49	0	123	655
70	61	0	116	760
80	61	24	201	913
90				
100				

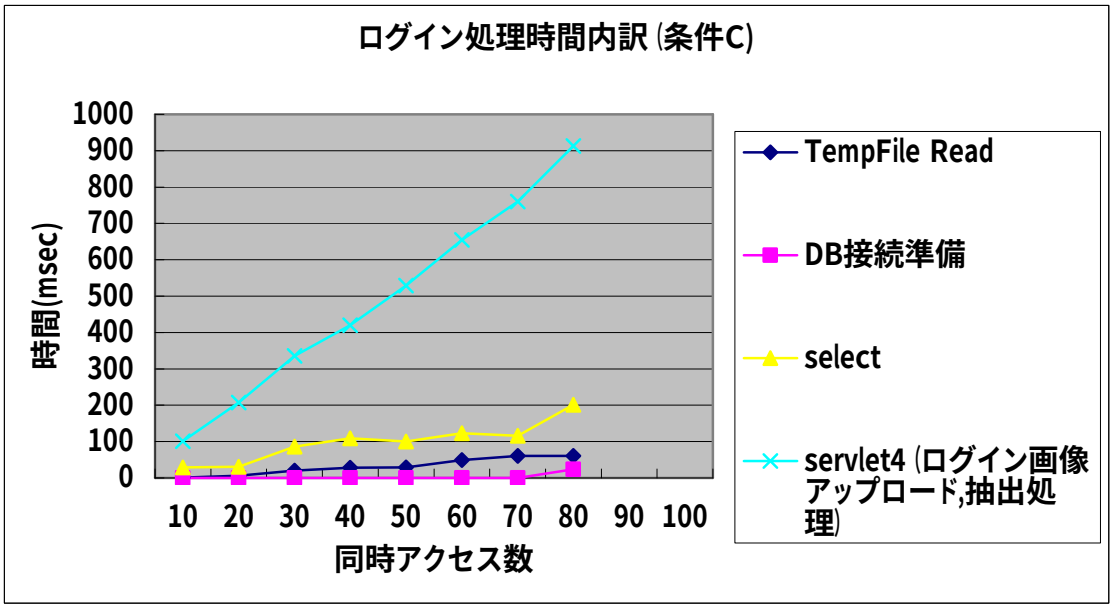

図 9.24　処理時間内訳（条件C、ログイン処理、同時アクセス数変化）

表 9.25　処理時間内訳（条件C、新規アカウント作成処理、ファイルサイズ変化）

	TempFile Read	password 発行	DB 接続準備	select	隠蔽	insert	servlet1(フォームデータ処理)	servlet2 (画像アップロード)
10K	6	0	0	7	46	59	1	71
50K	6.9	0	0	13	328	58	1	338
100K	2	0	0	14	593	61	0	793
200K	8.2	0	0	32	1184	62	1	1724
400K	7.2	0	0	21	1821	60	1	3684
600K	6.7	0	0	12	2510	60	0	5556

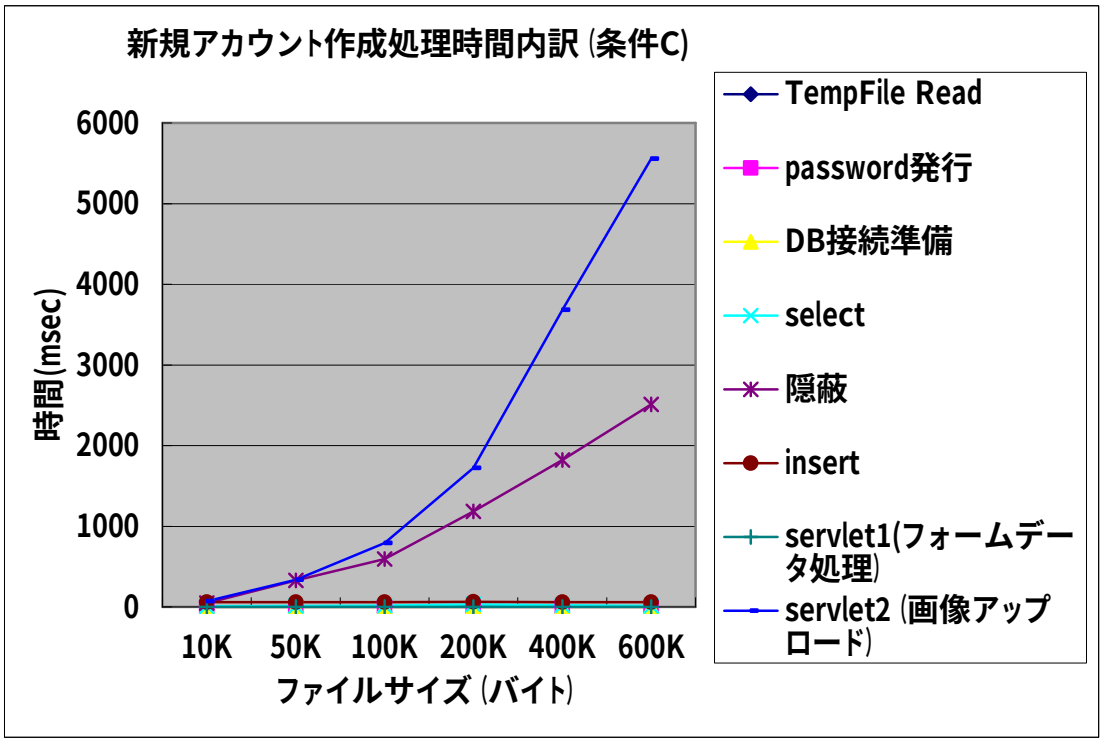

図 9.25　処理時間内訳（条件C、新規アカウント作成処理、ファイルサイズ変化）

表 9.26　処理時間内訳（条件C、ログイン処理、ファイルサイズ変化）

	TempFile Read	DB 接続準備	select	servlet4 (ログイン画像アップロード,抽出処理)
10K	1	0	29	101
50K	1	0	43	407
100K	1	0	63	849
200K	0	0	48	1895
400K	23.5	69.4	153	3368
600K	4	0	74	5532

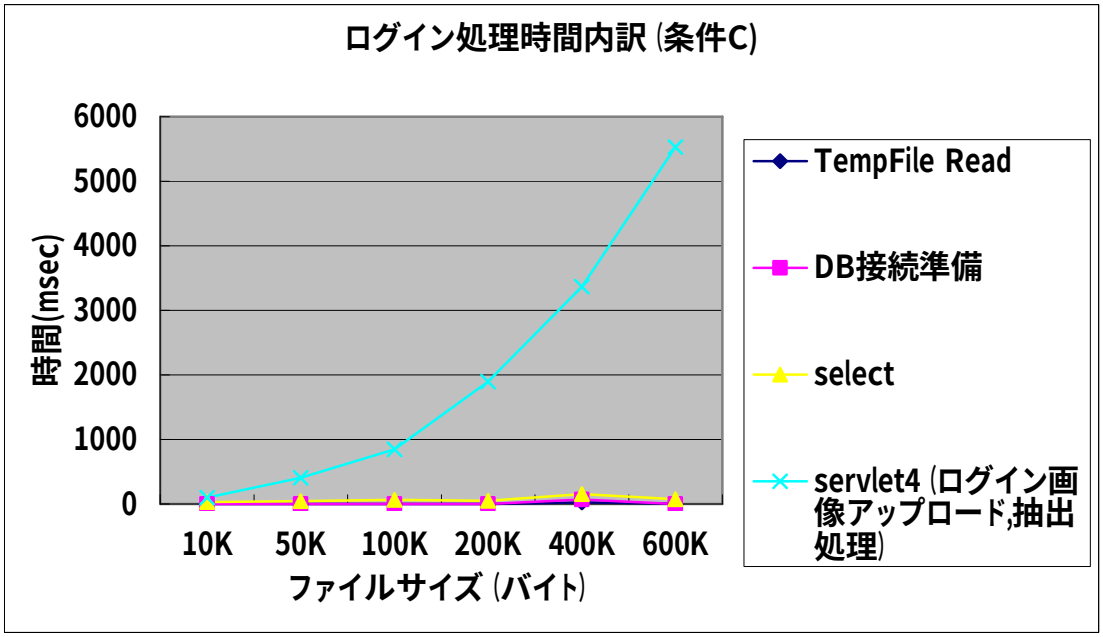

図 9.26　処理時間内訳（条件C、ログイン処理、ファイルサイズ変化）

表 9.27　処理時間内訳（条件D、新規アカウント作成処理、同時アクセス数変化）

	TempFile Read	password 発行	DB 接続準備	select	隠蔽	insert	servlet1(フォームデータ処理)	servlet2 (画像アップロード)
10	3	0	0	3	52	56	0	67
20	32	1.1	0	4	59	65	3	140
30	48	0	0	3.7	60	74	8.7	223
40	113	2	0	2	65	85	12	335
50	87	0	0	2.4	62	94	25	394
60	109	1	0	2	68	94	21	582
70	113	0	0	2.7	63	102	40	632
80	193	0	0	2.9	62	106	72	720
90	205	3.3	0	3	64.5	108	79	820
100	217	3.1	0	3.6	69	101	105	862

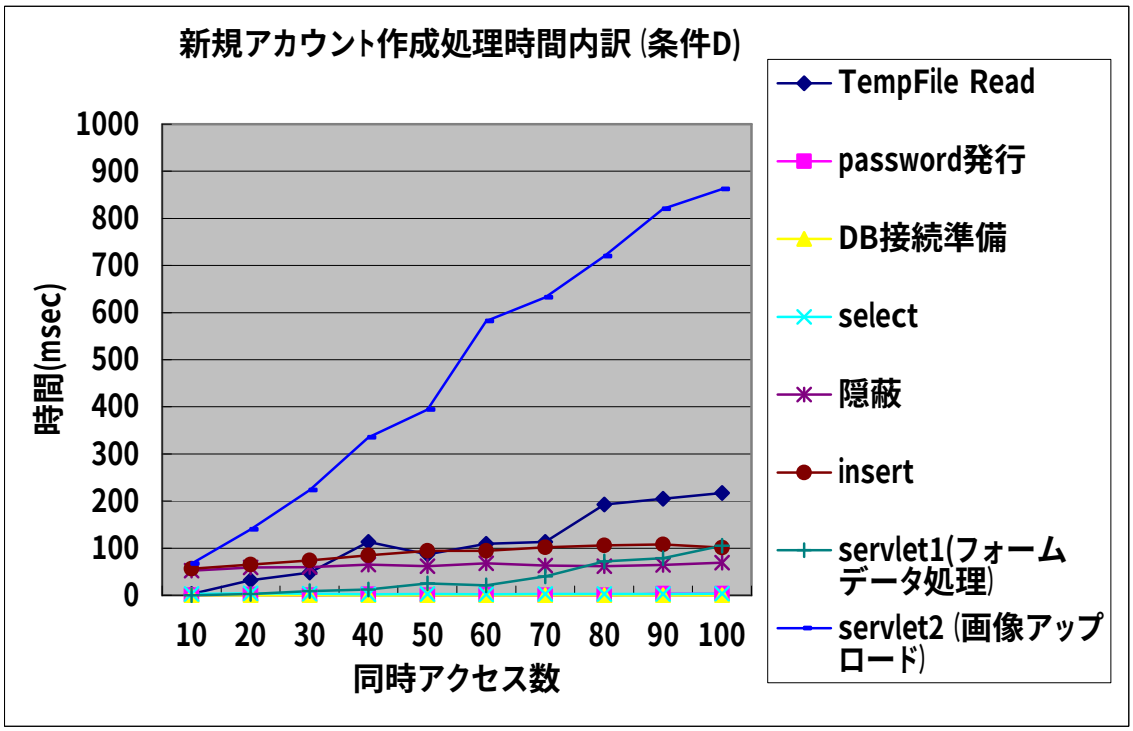

図 9.27　処理時間内訳（条件D、新規アカウント作成処理、同時アクセス数変化）

表 9.28　処理時間内訳（条件D、ログイン処理、同時アクセス数変化）

	TempFile Read	DB 接続準備	select	servlet4 (ログイン画像アップロード,抽出処理)
10	2	0	15	102
20	9	0	16	195
30	17	0	13	274
40	28	0	17	423
50	38	0	18	510
60	50	0	22	658
70	62	2	23	764
80	68	0	23	903
90	64	0	27	1063
100	89	0	37	1192

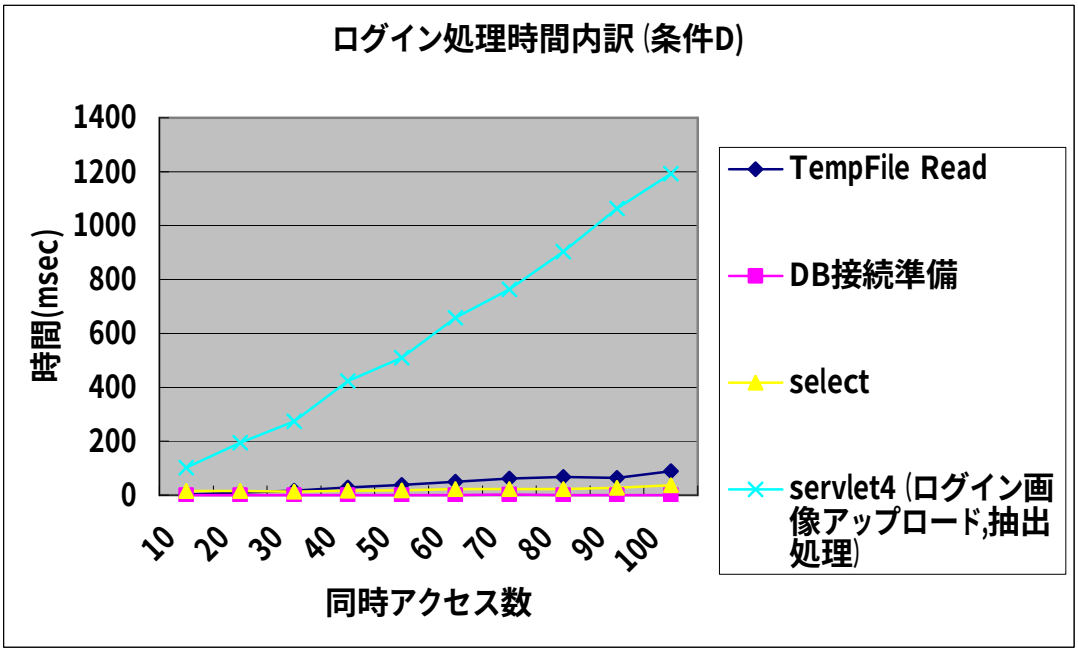

図 9.28　処理時間内訳（条件D、ログイン処理、同時アクセス数変化）

表 9.29　処理時間内訳（条件D、新規アカウント作成処理、ファイルサイズ変化）

	TempFile Read	password 発行	DB 接続準備	select	隠蔽	insert	servlet1(フォームデータ処理)	servlet2 (画像アップロード)
10K	3	0	0	3	52	56	0	67
50K	12	0	0	12	328	58	6.2	344
100K	3.4	0	0	40	564	58	7	801
200K	3	0	0	50	1181	58	2	1439
400K	46.9	0	0	24	2025	62	3	3372
600K	3.8	1	0	20	2583	64	1	5111

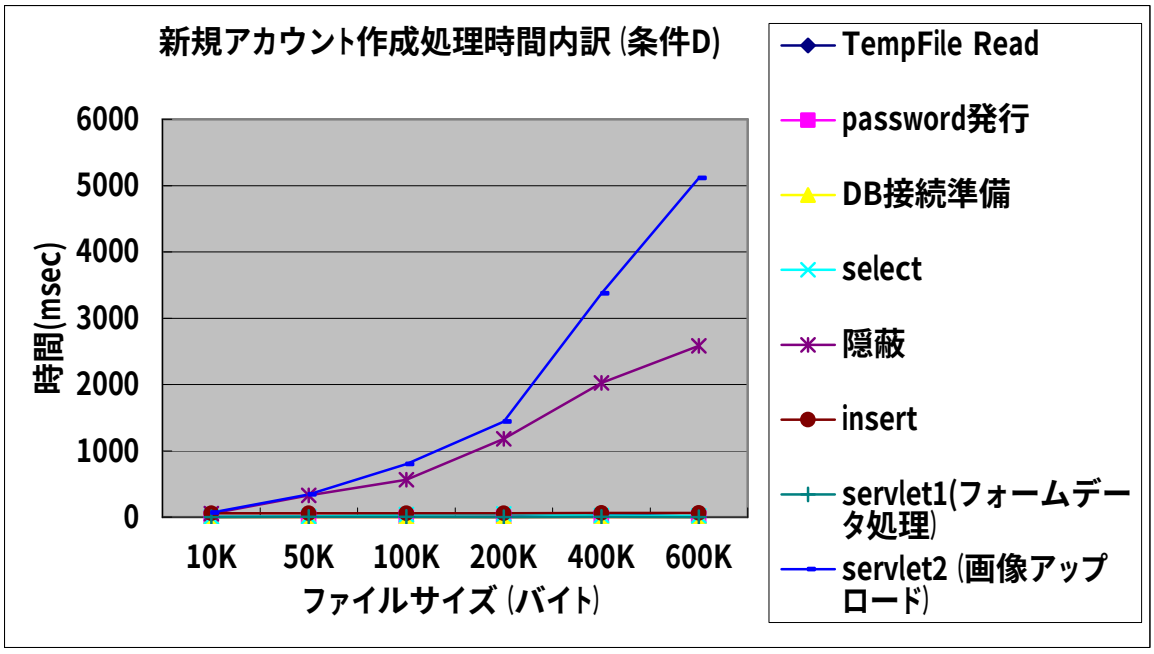

図 9.29　処理時間内訳（条件D、新規アカウント作成処理、ファイルサイズ変化）

表 9.30　処理時間内訳（条件D、ログイン処理、ファイルサイズ変化）

	TempFile Read	DB 接続準備	select	servlet4 (ログイン画像アップロード,抽出処理)
10K	2	0	15	102
50K	4.1	0	50	410
100K	1	0	70	833
200K	0	0	93	1920
400K	0	0	77	3488
600K	0	0	72	5414

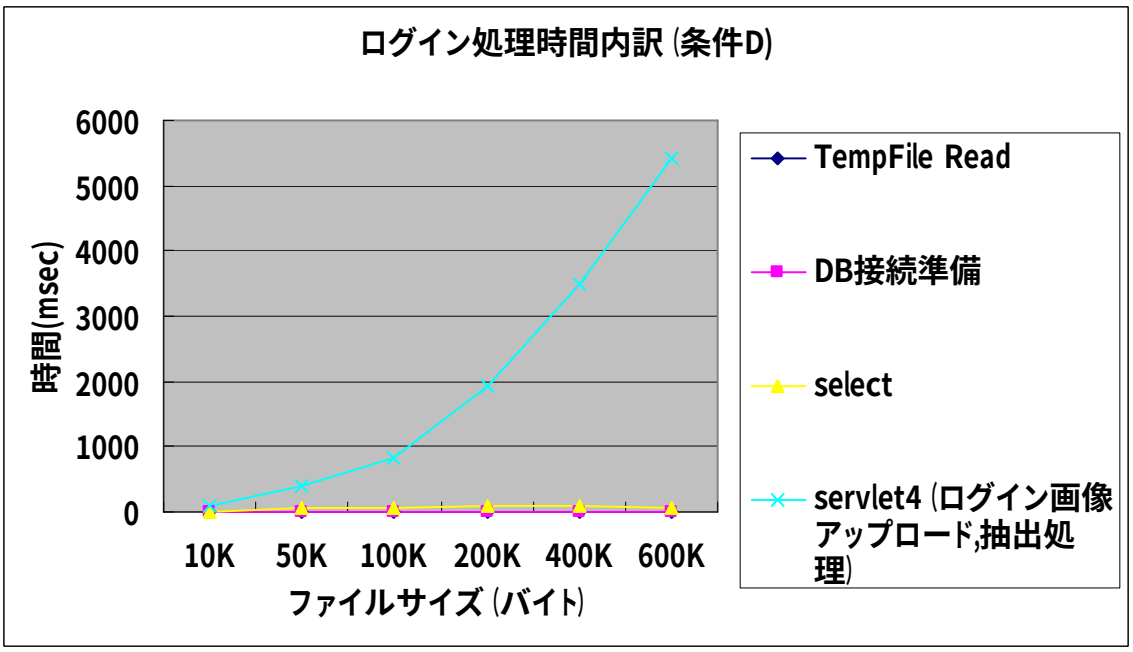

図 9.30　処理時間内訳（条件D、ログイン処理、ファイルサイズ変化）