平成 14 年度　卒業研究論文

Web-ORDB 空間検索インタラクション・モニタリング機能の実装とシステム性能の向上

2002 年 2 月 12 日提出

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）
S99442-M　　鷲見 明

目 次

第1章　まえがき

本研究では、Web-ORDB 空間検索においてシステム全体の性能低下を招くボトルネックとなる箇所を調べ、その箇所の性能向上を図ることを目的とする。本研究では、システムの性能評価の基準として処理速度に着目する。現在はハードウェアの性能が著しく向上し、メモリー消費量などのリソース面よりも処理速度の方が Web アプリケーションにおいては重要であると思われる。

クライアント・サーバ（C/S）型の Web アプリケーションシステムでは、クライアントの Web ブラウザ上から Web サーバへリクエストを送信する。リクエストを受信した Web サーバは要求された処理を行い、その結果をクライアントの Web ブラウザへ転送する。

ここで、クライアント側の立場で考えてみる。この場合、クライアントは WWW（World Wide Web）の利用者であるが、リクエストを送信してから要求した結果が即座に Web ブラウザに表示されることが望ましい。結果が表示されるまでに何分間も待たされるようではこのシステムの利用価値が半減してしまう。

いま、ある Web-ORDB 空間検索トランザクションを含む C/S 型システムが稼動している。このシステムは正常に動作しているが、検索結果がブラウザ上へ表示されるまで 1 分以上かかった。本研究では、実際にこのシステム全体のボトルネックを調べ、その箇所の性能向上を実現する。

第2章　システム概要

2.1　システム概要

システム名　：「対話型意思決定支援システム」
開 発 言 語　：HTML、Java 言語、C 言語（埋込み SQL）
概　　要　：（参考文献・資料[1] より一部引用）

島根県内の国道沿いの左右帯域幅 $x$km（$0.5 \leqq X \leqq 500$：500m 毎）内に含まれる選択された人口データに対する 1km×1km の基準地域メッシュの表示」という検索を Web 上で利用するという仕様のシステムの試作。詳細は参考文献・資料[1] 卒業論文を参照。

ユーザはブラウザ上で表示したい国道とその国道に対する左右帯域幅、表示したい人口データの種類を選択し、HTML フォー

ム（図 1）の送信ボタンを押す。選択したパラメータがサーバへ送信され、Java サーブレットへ渡される。サーブレットプログラム内では、必要なデータを得るために必要な SQL 文を埋め込んだ C 言語プログラム（UAP：**U**ser **A**pplication **P**rogram）を使用してデータベースにアクセスする。データベースシステム内で必要なデータを検索し、検索条件に合致するデータを UAP を通して Java サーブレットへ受け渡す。サーブレットプログラムは、受け取ったデータを元に画像を描画しブラウザへ送信する（図 2）。

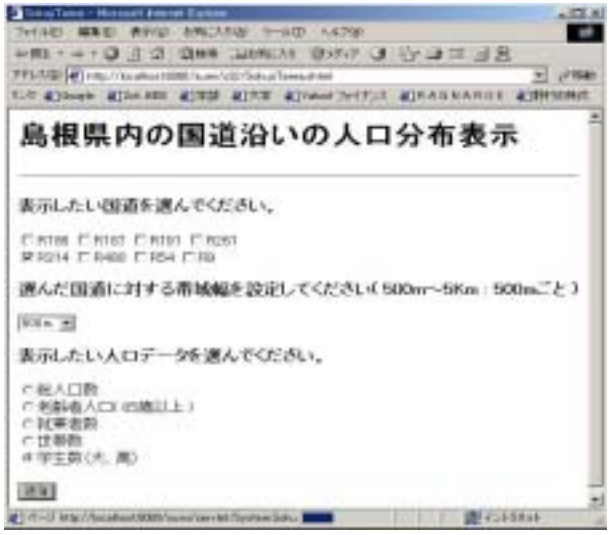


図 1

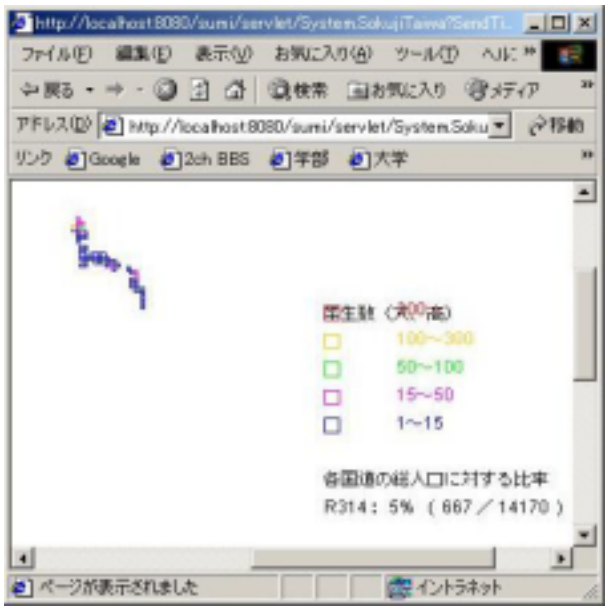


図 2

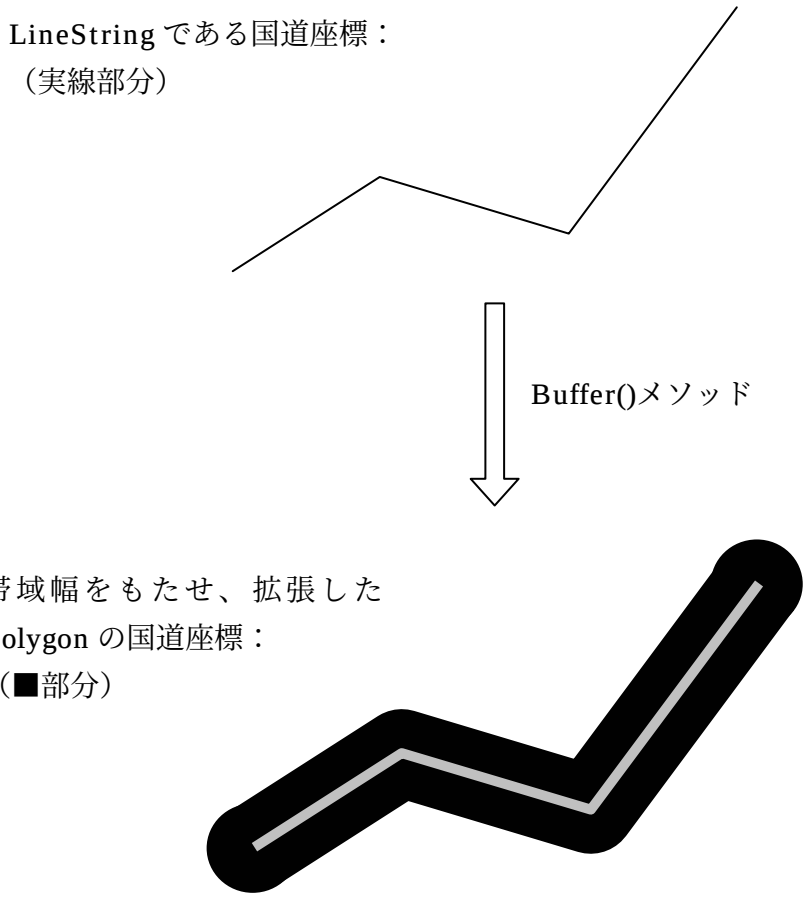
2.2　システム機能要素

本研究で DBMS および Java ライブラリで使用しているメソッドに buffer()と intersects()がある。これらは OpenGIS（OGC）[2] の仕様に準拠している。ここでは、これらのメソッドについて説明する。

2.2.1　buffer()メソッド

buffer()とは空間座標の拡張をおこなうメソッドで、Geometry クラスに含まれている。国道座標は LineString 型（折線）である。この国道の左右帯域幅 x キロメートル圏内の人口分布を検索するため、この座標（LineString）を拡張し、Polygon 型（多角形）の空間に変換する必要がある。

その図を以下に示す。

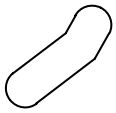


2.2.2　intersects()メソッド

intersects()は 2 つの空間座標の共通集合演算をおこなうメソッドである。

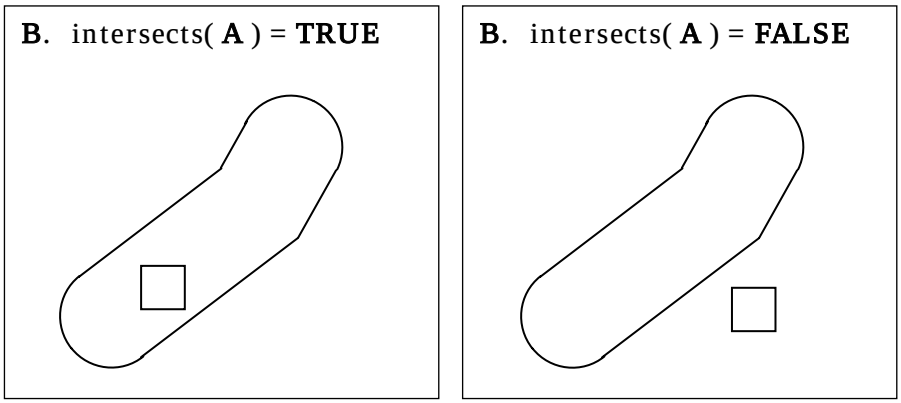
これは Geometry クラスのメソッドで、一方の座標がもう一方に含まれていれば TRUE、そうでなければ FALSE を返す。

検索ルーチンでは空間座標データによる検索を行っている。この検索の条件は、”人口メッシュが国道沿いの圏内に含まれているか” というものであり、intersects()メソッドを使用している。その例を以下に示す。



：拡張した国道座標の Polygon 図形（**A**）

：人口メッシュの Polygon 図形（**B**）　　とする。



2.3　システム内データ概要および DB 定義

本システムで使用しているデータは DBMS 内に格納している。

国道データと人口データの 2 つの表を作成し、それぞれ、表「KOKUDOU」、表「JINKOU」としている。各表の概要を以下に示す。

・ 表「KOKUDOU」

この表には次のような構成で、国道座標に関するデータを格納する。

表「KOKUDOU」の列名とデータ型

ID	NAME	GEO_CLMN
INTEGER	VARCHAR	GEOMETRY

ID　　：　国道データ ID（例：8）

NAME　　：　国道名（例：’R9’）

GEO_CLMN　：　国道座標（例：LineString(60421 -50000, ･･･)）

データ列 GEO_CLMN（GEOMETRY 型）に対して空間インデッ

クスを作成している。その他の列に対しては索引付けを行っていない。

・ 表「JINKOU」

この表には次のような構成で、人口データを格納する。

表「JINKOU」の列名とデータ型

MESH_CODE	JINKOU	OLDER	WORKER	FAMILY	STUDENT	GEO_CLMN
INTEGER	INTEGER	INTEGER	INTEGER	INTEGER	INTEGER	GEOMETRY

MESH_CODE ： 人口メッシュコード（例：51313688）
JINKOU ： 総人口数（例：24500）
OLDER ： 老齢者数
WORKER ： 就業者数
FAMILY ： 世帯数
STUDENT ： 学生数
GEO_CLMN ： 人口メッシュ座標（例：Polygon((68000 -38000,･･･))）

データ列 GEO_CLMN（GEOMETRY 型）に対して空間インデックスを作成している。その他の列に対しては索引付けを行っていない。

また、これらの表に対するアクセス権限について説明する。

使用している DBMS において、”root”と user01 の 2 つのユーザを用意している。

”root”はユーザ権限の全てを持つ。表・インデックスの作成、データ挿入・更新などはこのユーザで行う。

user01 は利用者権限のうち CONNECT 権限（DB 接続）と、この 2 つの表へのアクセス権限（SELECT 権限）をもつ。システム内のプログラムにより DB にアクセスするが、SELECT 文によるデータの抽出のみを行う。そのため、本システムのプログラム内ではこのユーザとパスワードを使用してデータへアクセスする。

表作成およびデータ操作方法、詳細については 4.2 空間データの操作、参考文献[1] を参照。

2.4 システム動作環境

本研究における計算機システムの環境を述べる。

使用マシンの主な仕様を次に示す。

サーバ機：

CPU　：　Intel PentiumIII 733 MHz
メインメモリ　：　256 MB
LAN　：　100BASE-TX
OS　：　Microsoft Windows 2000 Server (SP3)
Web サーバ・サーブレットコンテナ：Jakarta-Tomcat 3.3.1
DBMS　：　HITACHI HiRDB/Single Server Version6
および、

クライアント機：

CPU　：AMD K6 3D　448 MHz
メインメモリ　：240 MB
LAN　：100BASE-TX
OS　：Microsoft Windows XP Professional Version 2002
Web ブラウザ　：Internet Explorer 6.0

第3章　Java について

3.1　Java サーブレットについて

Java サーブレットとは、サーバサイドで動く Java プログラムである。同じ Java プログラムで Java アプレットがあるが、これはクライアントサイドで実行する Java プログラムである。Java アプレットの場合は Java プログラム自体と必要なデータをクライアントにダウンロードする必要がある。しかし、サーブレットの場合は、Java プログラム自体をサーバ上で実行し、その結果を（多くの場合）HTML としてクライアントに送信するだけである。そのため、アプッレトと比較した場合、次のような長所がある。

- 重たいプログラム・データをクライアントにダウンロードする必要がない。
- サーバ側にサーブレットを実行する環境を整備すればよいので、クライアントに負荷がかからない（最低限 Web ブラウザが動作すればよい）。

また、サーブレットと似た技術に CGI がある。CGI に対するサーブレットの利

点を次に示す。（一部、[3] 書籍『コア・サーブレット&JSP』より引用）

- 効率的
  CGI の場合、HTTP リクエスト毎に新たなプロセスが生成される。それに対し、サーブレットの場合、Java 仮想マシンが唯一のプロセスとして常に動作しており、個々のリクエストはスレッドによって処理される。そのため、プロセス起動のためのオーバヘッドが生じない。また、処理終了後もメモリ上に存在するため、複数のリクエストから簡単に利用できる。

- 開発しやすさ
  サーブレットは HTML フォームデータの構文解析、HTTP ヘッダの読み取りと設定、クッキーの処理、セッションの管理、といった多くの基礎的機能を豊富に備えている。また、信頼性も高く再利用性にも富む。

- 強力
  複数サーブレットでのデータベース接続の共有など、資源を共有する最適化を容易に実装することができる。また、サーブレットは複数のリクエストにまたがって情報を保持するため、セッション管理や、前の処理結果のキャッシュなどの技術も簡単に実現できる。

- 可搬性
  サーブレットは Java の標準 API を使用するため、一般的にプラットホーム非依存である。

- 安全
  CGI は通常 OS のシェルから実行するため、特殊文字（¥n など）に対して細心の注意が必要である。また、配列に対するバッファオーバフローの攻撃対象になりうる。これに対し、サーブレットはこれらの問題点がない。

本研究ではサーブレット実行環境（サーブレットコンテナ）にフリーソフトウェアである Jakarta-Tomcat 3.3.1[4] を使用している。Tomcat は単体でも Web サーバの機能を備えている。また、IIS や Apach にサーブレットコンテナ単体の機能をリンクさせることも可能である。詳細は次の URL を参照。

The Apach Project（http://jakarta.apache.org/tomcat/）

3.2　JDBC について

JDBC とは、SQL 文を実行するための Java API で、後述の人口座標検索 Java プログラムで使用している。

JDBC には次の４つのタイプに分類される。

（参考文献[5]　より一部引用）

■ タイプ１：　JDBC-ODBC Bridge ドライバ

既存の ODBC ドライバを使ってデータベースへアクセスする方式。

■ タイプ２：　Native API partly ドライバ

データベースシステム固有のドライバソフトウェアを使用する。JDBC API をデータベース固有 API へ変換し、このドライバへ渡す。

■ タイプ３：　Net Protocol All-Java ドライバ

クライアントとデータベースの間に中継サーバを配置し、このサーバに Java で作成されたドライバを格納する。クライアントは実行時にこのドライバをダウンロードして使用する。クライアント側で実行した JDBC 処理は、サーバ上の中継プログラムがデータベース固有の命令に変換して実行される。

■ タイプ４：　Native Protocol All-Java ドライバ

データベース固有のドライバの機能を Java で作成し、JDBC ドライバ内部に取り込んだもの。

第4章　ORDB

4.1　ORDB とは

データベースシステムのひとつにリレーショナル・データベース（RDB）がある。オブジェクトリレーショナルデータベース（ORDB）は、RDB にデータとメソッドのカプセル化、型の継承関係といったオブジェクト指向の環境を提供するものである。ORDB では、このようなカプセル化された型を抽象データ型またはユーザ定義型という。

4.2　空間データの操作

4.2.1　空間データのフォーマット

空間データの表現方法として Well-Known Text（WKT）フォーマット形式がある。これは、OGC によって定義されている図形をテキストで表現する方法であり、空間検索システムや SQL 等のこれをサポートしているソフトウェア同士でデータの受け渡しを容易に行うことができる(参考資料[2])。本研究で使用している Java ライブラリやデータベースシステムでも使用している。以下に、WKT の例を挙げる。

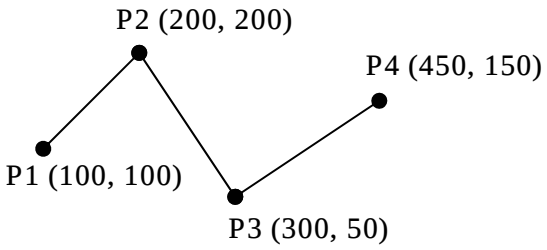
・ 点（Point）：

Point(100 200)

● (100, 200)

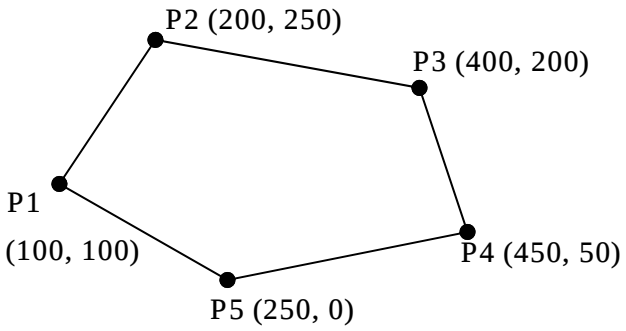
・ 折れ線（LineString）：

LineString(100 200, 200 200, 300 50, 450 150)



・ 多角形（Polygon）：

Polygon((100 100, 200 250, 400 200, 450 50, 250 0, 100 100))



4.2.2　表作成

空間データを格納するためには、抽象データ型の列を含む表を作成する。定義系 SQL である CREATE TABLE 文を実行する。

・ 国道座標を格納する表『KOKUDOU』の作成

この表には、データ ID、国道名、国道座標を格納する。

```
CREATE TABLE   KOKUDOU (
   ID          INTEGER,
   NAME        VARCHAR(100),
   GEO_CLMN    GEOMETRY
);
```

・ 人口データを格納する表『JINKOU』の作成

この表には、区画コード、総人口数、老齢者数、就業者数、世帯数、学生数、区画座標を格納する。

```
CREATE TABLE   JINKOU (
   MESH_CODE   INTEGER,
   JINKOU      INTEGER,
   OLDER       INTEGER,
   WORKER      INTEGER,
   FAMILY      INTEGER,
   STUDENT     INTEGER,
   GEO_CLMN    GEOMETRY
);
```

ここで、座標を格納する列の型名に GEOMETRY 型を指定している。GEOMETRY 型とは HiRDB 空間検索プラグイン(HiRDB Spatial Search Plug-in Version 3)によって定義されている抽象定義型で、点(Point)、折れ線(LineString)、多角形(Polygon)といった空間データを格納することができる。

4.2.3 空間座標データ格納方法

空間座標データは抽象定義型である GEOMETRY 型の列に格納するので、データの格納にはコンストラクタ関数を使用する。

・ 国道座標データ格納例

```
INSERT INTO KOKUDOU ( ID, NAME, GEO_CLMN)
   VALUES( 8, 'R9', GeomFromText('LINESTRING(60421 -5
          0000,60570 -50386,61473 -51420,61643 -5170
          6, ･･･ , 186815 -171489,186916 -171608)',9,
          1));
```

表「KOKUDOU」

ID	NAME	GEO_CLMN
8	R9	LineString(60421 -50000, ･･･)

・ 人口データ格納例

```
INSERT INTO JINCOU(MESH_CODE,JINKOU,OLDER,WORKER,FAMI
                   LY,STUDENT,GEO_CLMN)
    VALUES( 51313688, 8,     5,     0,     0,     0,G
            eomFromText('POLYGON((68000 -38000,68000
            -39000,69000 -39000,69000 -38000))',1,
            1));
```

表「JINKOU」

MESH_CODE	･･･	GEO_CLMN
51313688	･･･	Polygon((68000 -38000, ･･･))

GeomFromText 関数が GEOMETRY 型のコンストラクタ関数である。この関数を使用すると WKT 形式のあらゆる空間データを GEOMETRY 型データ列に格納することができる。この関数についての詳細は HiRDB マニュアル「HiRDB Spatial Search Plug-in Version 3 解説・手引・文法・操作書」を参照のこと。

4.2.4 空間インデックス作成方法

空間座標データを操作するには索引付けを行う必要がある。HiRDB Version6 における空間インデックス作成例を以下に示す。

```
国道座標データへの空間インデックス作成
> CREATE INDEX  GEOIDX
   using type spatial on KOKUDOU ( GEO_CLMN )
   in ( RLOB1 )
   PLUGIN 'EXTENT=100,-174810,186916,300;
            DEPTH=1;PRECISION=1';


人口座標データへの空間インデックス作成
> CREATE INDEX  GEOIDXJ2
   using type spatial on JINKOU ( GEO_CLMN )
   in ( RLOB2 )
   PLUGIN 'EXTENT=54000,-280000,191000,-38000;
            DEPTH=4;PRECISION=1';
```

ここでは国道座標データへの空間インデックス作成について説明する。CREATE INDEX 文がインデックスを作成する SQL 文である。ここではGEOIDX という名前で、表『KOKUDOU』の GEO_CLMN 列（Geometry 型）への空間インデックスを作成している。HiRDB では検索に Quad-Tree を使用している。この例では、深さ１（DEPTH=1）の整数座標系（PRECISION=1）としている。また、これらの SQL 文の作成には「HiRDB Index Tool」を使用した。

「HiRDB Index Tool」とは HiRDB Spatial Search Plug-in によって提供されている ”空間インデックス SQL 作成ユーティリティ” で、空間データのインデックスを自動的に作成することができる。

これらに関する詳細は HiRDB マニュアル「HiRDB Spatial Search Plug-in Version 3 解説・手引・文法・操作書」[6] を参照のこと。

4.2.5　空間座標データ検索方法

空間座標の検索も一般的な SELECT 文を使用して行う。座標データを抽出したい場合、AsText()関数を使用すると、空間座標データを WKT として返す。また、WHERE 句に条件を指定することも可能である。以下にその例を示す。

・ 国道座標を WKT 形式で検索する例

国道名が‘R54’である国道の座標を抽出する

```
> SELECT AsText(GEO_CLMN,1) FROM KOKUDOU
    WHERE NAME='R54' WITHOUT LOCK NOWAIT;

linestring(140000 -111658,139458 -112439, ･･･
                              ,152496 -168087,152419 -168393)
```

この例では、国道座標データは GEO_CLMN 列に格納されている。AsText 関数によって、国道名が'R54'と一致するレコードの国道座標データを WKT 形式で出力している。

・国道沿いの人口を検索する例

国道座標を拡張（buffer()）し、その範囲内（intersects()）の人口データを抽出する。

```
> SELECT jinkou FROM JINKOU
    WHERE IntersectsIn(GEO_CLMN, RegionBuffer(:R_GEOM, 500)
IS TRUE WITHOUT LOCK NOWAIT;
```

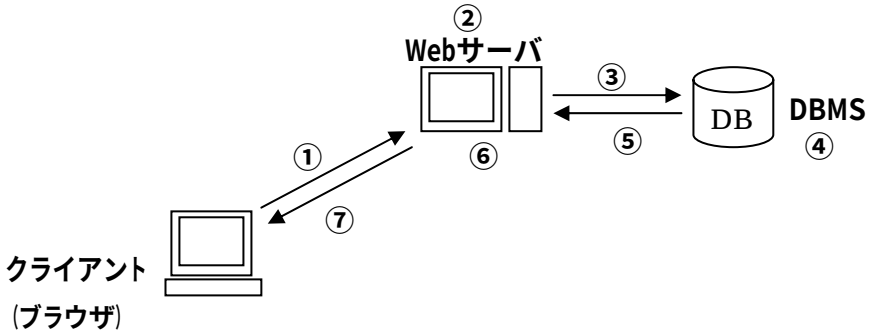
GEO_CLMN 列に人口メッシュ（Polygon）、変数:R_GEOM は国道座標（LineString）が格納されているものとする。

RegionBuffer()関数（OGC[2] の buffer()に相当）により、国道座標データを左右帯域幅 500 で拡張する（Polygon に変換される。）

この拡張した国道座標と人口メッシュ（GEO_CLMN）の共通集合演算をおこない、国道沿いの圏内に含まれていればその人口数を抽出する。

第5章　計測

システム全体のデータ・処理の流れについて説明する。



① クライアントの Web ブラウザ上から Web サーバにリクエストを送信する。
② Web サーバがリクエストを受信すると、同サーバ上でサーブレットに処理が渡される。
③ サーブレットは必要なデータ（座標、人口など）を得るためにデータベース（DBMS）へ接続する。
④ DBMS 内部で必要なデータを検索する。
⑤ 検索したデータをサーブレットへ渡す。
⑥ サーブレットは受け取ったデータをもとに、人口分布画像を作成する。
⑦ 検索結果である人口分布画像を Web ブラウザへ送信する。

①と⑦は通信 Web ブラウザ・サーバ間の通信である。Web サーバではその他のコンテンツも提供しているが、特に通信速度においての問題は発生していない。そのため、この箇所がボトルネックとることは考えにくい。

②と⑥では同じサーブレットが処理を行っている。ここで行っていることは、リクエストに含まれるパラメータ（国道、人口の種類、帯域幅）を C 言語埋込み SQL プログラム（UAP）に渡す。UAP 内で国道および人口を検索する SQL 文（SELECT 文）を作成し、DB にアクセスして実行する。ここでの主な処理は画像を描画することであるが、この箇所がボトルネックとなる可能性はある。

④の箇所では、主に２つのデータの検索を行っている。ひとつは、指定した国道の空間データ（LineString）の検索である。この検索は、国道名もとに格納された空間データを AsText 関数で WKT 形式で返す、というものである。もうひとつは、人口の検索である。ひとつ目で検索した国道座標（LineString）に Buffer 関数を使用し帯域幅をもたせ Polygon に変換し、その Polygon と全ての人口メッシュ（Polygon）との共通集合演算（intersection()）を行う。その結果 TRUE となる人

口データを返す、というものである。今回の人口データのレコード数は 3894 である。ここでの処理は DBMS に依存しているが、処理内容が複雑であるため、ボトルネックとなる可能性はある。

以上のことから、システム全体の処理を眺めてみたところ、②～⑥間にボトルネックが存在することが予想できる。そこで、本研究では①～⑥間の各箇所において計測を行うことにする。

5.1 計測方法

①～⑥で実際に測定した箇所について、具体的に説明する。

計測した箇所は

1. Web ブラウザ→Web サーバ間の通信時間（①の部分）
2. 国道検索 SQL の発行時間（③～⑤の部分）
3. 〃 実行時間（ 〃 ）
4. 人口検索 SQL の発行時間（ 〃 ）
5. 〃 実行時間（ 〃 ）
6. サーブレット全体の処理時間（②～⑥の部分）

の計６箇所である。

5.1.1 Web ブラウザから Web サーバまでの通信時間

まず、①でクライアントからリクエストを送信してから Web サーバで受信するまでの時間を計測する。リクエストの送信は、HTML を用いた送信フォームを使用している。ここでの送信時に、送信ボタンが押されたときの時刻を送信データに含めて送る。<FORM>エレメント（フォーム名：form1）の中に次のコードを記述する。

```
          :
          :
01:     <INPUT type="hidden" name="SendTime">
02:     <INPUT type="hidden" name="TZOffset">
03:     <SCRIPT language="JavaScript">
04:       <!—
05:         function nowTimeSet(){
06:            now = new Date;
07:            form1.SendTime.value = now.getTime();
08:            form1.TZOffset.value = now.getTimezoneOffs
    et();
09:         }
10:       -->
11:     </SCRIPT>
          :
          :
12:     <INPUT onClick="return nowTimeSet(form1)" type="
    submit" value="送信">
          :
```

01:と 02:は送信時刻とその時差を格納するフィールドである。これらは type=”hidden”とすることで非表示としている。03:～11:は 01: 02:で用意したフィールド SendTime、TZOffset にそれぞれ時刻と時差を入力するための関数 nowTimeSet()（Java Script で記述）である。12:が送信ボタンの INPUT タグであるが、クリック時（onClick）に nowTimeSet()関数を呼び出し、時刻を入力後、リクエストを送信する。

送信されたデータは Web サーバが受信し、サーブレットが呼び出される。このサーブレットでは次のようにして送信データを受信する。

```
          :
          :
13:    java.util.Date rsvTime = new Date(); // 時刻を取得
          :
          :
14:    Long sendTime = Long.valueOf(req.getParameter("Send
   Time")); // クライアントの送信時刻(GMT,ミリ秒)
15:    Long ctTZ = Long.decode(req.getParameter("TZOffset
   "));
16:    long eTime1 = (rsvTime.getTime()) + (rsvTime.getTim
   ezoneOffset()*6000) - (sendTime.longValue() + ctTZ.long
   Value()*6000); // eTime1: クライアント→Web サーバの通信時間
17:
          :
```

13:はサーブレット実行開始時に時刻を記録しておく。14: 15:で HTML フォームからのフィールド名 SendTime および TSOffset のデータを取得する。16:がサーブレット実行開始時刻からフォームデータの送信時刻の差をとることで求める。getParameter()メソッドを使用して HTML フォームからデータを取得しているが、文字列として取得する。差をとるためには、long 型などの数値データに変換する必要がある。ここでは、longValue()メソッドを使用して変換をおこなっている。

5.1.2 SQL による検索時間の計測

次に、SQL による検索時間を計測する（③～⑤の区間における計測）。国道検索および人口検索は C 言語による埋込み SQL プログラムを使用している。HiRDB 内部による計測は困難であるため、C 言語プログラムによる測定を行う。

C 言語プログラム内では、標準ライブラリの clock()関数を使用し必要箇所にて開始時刻・終了時刻を取得し、その差により実行時間を求める。一般的によく用いられる方法であるが、以下にその例を挙げる。

```
          ⋮
18:    clock_t start, finish;  /* 開始・終了時刻の格納 */
19:    double   etime;    /* 所要時間の格納      */
          ⋮
20:    start = clock();  /* 開始時刻 */
21:    /*
         計測対象のコード
       */
22:    finish = clock();  /* 終了時刻 */
23:    etime=(double)(finish-start)/CLOCKS_PER_SEC; /* 実
   行時間 */
          ⋮
```

18: 19:は時刻と計測時間を格納するための変数の宣言である。21:の部分に計測対象のコードを記述する。この場合計測対象とは、SQL 文を作成する箇所と、DB へ接続し SQL によってデータを取得する箇所である。20: 22:により開始・終了時刻を取得し 23:によって実行時間を計算する。

5.1.3 サーブレット全体の実行時間

最後に、②～④の区間おけるサーブレット全体の実行時間の計測である。サーブレットは Java 言語を使用しているので、Date クラスの Date()メソッドにより時刻を取得し、上記 C 言語の場合と同様に求める。以下はその記述例である。

```
          ⋮
24:    java.util.Date start = new java.util.Date();   // 開
   始時刻取得
25:    /*
         計測対象のコード
       */
26:    java.util.Date finish = new java.util.Date();  // 終
   了時刻取得
27:    long eTime = finish.getTime() - start.getTime();  //
    実行時間
          ⋮
```

この場合、計測対象はサーブレットプログラム全体であるので、リクエス

トを受信しサーブレットが呼び出された時点から、結果画像をブラウザへ返す直前までである。

5.2　計測

計測ルーチンを実装後の実行例とその結果を次に示す。

検索条件

国道　　：R314
帯域幅　：500
人口　　：学生

検索結果

クライアント→Web サーバの通信時間	：	0.03 sec
国道検索 SQL 発行時間	：	0.04 sec
国道検索 SQL 実行時間	：	0.04 sec
人口検索 SQL 発行時間	：	0.04 sec
人口検索 SQL 実行時間	：	**62.62** sec
サーブレット全体処理時間	：	63.10 sec

測定の結果、人口検索 SQL の実行がシステム全体の処理時間のほぼ全体を占めており、ボトルネックとなっていることがわかった。

尚、SQL 発行時間は SQL 文を動的に作成し、実行の準備にかかる時間である。SQL 実行時間は SQL 文（SELECT 文）を実行しデータを抽出するのにかかった時間である。

人口検索はデータベース内のストアドファンクション、Intersects()および

buffer()を使用している。このストアドファンクションは HiRDB Spatial Search Plug-in Version 3 によって提供されており、空間データの共通集合演算と空間データの拡張を行う関数である。本研究では、この共通集合演算部分をデータベース外部の Java ルーチンで行い、システム全体の高速化を試みる。

第6章　Java による外部検索ルーチン実装

6.1　JTS ライブラリ

Java 言語を使用しプログラムを作成するにあたり、使用したライブラリについて説明する。

これらのメソッドについてのアルゴリズム、それにかかる計算量などを調べる必要がある。しかし、まだ現時点においてこれらの詳細を把握できていない部分も多く、調査中である。

6.1.1　JST とは

JTS (JTS：Java Topology Suite) は、2 次元空間アルゴリズムを実装した Java API 群である。OpenGIS Consortium Simple Fertures Spacification for SQL (OGC SFS) [2] の定義に従い作成され、主に GIS 関連のソフトウェアでの使用を目的としている。

JTS についての詳細は以下を参照のこと。

JTS : Vivid Solutions 社（http://www.vividsolutions.com/）

6.1.2　JTS の使用例

JTS の buffer()メソッドと intersects()メソッドを使用し検索ルーチンを作成した。いずれのメソッドも Geometry クラスに含まれている。以下に使用例を示す。

```
          ⋮
01:  Geometry rGeom = WKTRdr.read(rGeomWkt);  //国道座標を格
     納
02:  Geometry rGeomBuf = rGeom.buffer(bufN);  //バッファ幅分
     座標を追加
          ⋮

03:  if(pGeom.intersects(rGeomBuf)){
04:      /*
               条件を満たした場合、
                人口データを描画ルーチンに出力
          */
05:  }
          ⋮
```

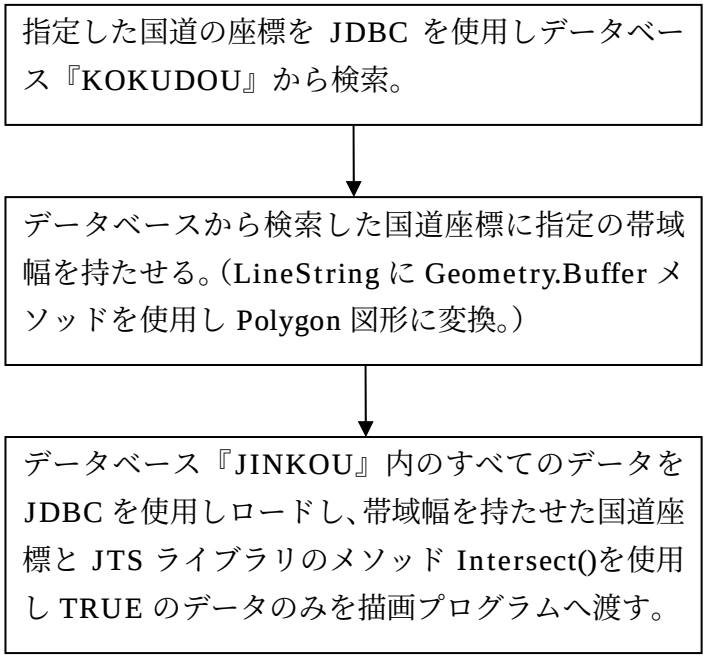
変数 pGeom は人口メッシュ座標（Polygon）、bufN は左右帯域幅の数値が格納されている。また、WKTRdr は JTS の WKTReader クラスのインスタンスで、WKT 形式のデータを Geometry に変換するためのものである。

01:では Geometry 型の変数 rGeom に国道座標（LineString）を格納している。WKTRdr.read()は WKT 形式のデータ(rGeomWkt)を Geometry 型の変数に格納するメソッドである。

02:で buffer()メソッドを使用した空間座標の拡張を行っている。この場合引数 bufN を左右帯域幅とし、国道座標（LineString）を bufN 分拡張した Polygon 座標が変数 rGeomBuf に格納される。

03:の pGeom.intersects(rGeomBuf)によって pGeom と rGeomBuf の共通集合演算を行い、TRUE であれば検索条件に一致するデータとして、描画ルーチンに出力する。

6.2　検索ルーチン実装



検索ルーチンを上記のような流れで作成した。また、Java 言語で開発し、空間データに関するクラス（Geometry など）は JTS のライブラリを使用している。（使用例は 6.1.2 JTS 使用例 を参照。）

検索ルーチンの内容および計測箇所については、埋込み SQL 使用時と同様になるように作成してある。

第7章　考察

7.1　結果比較

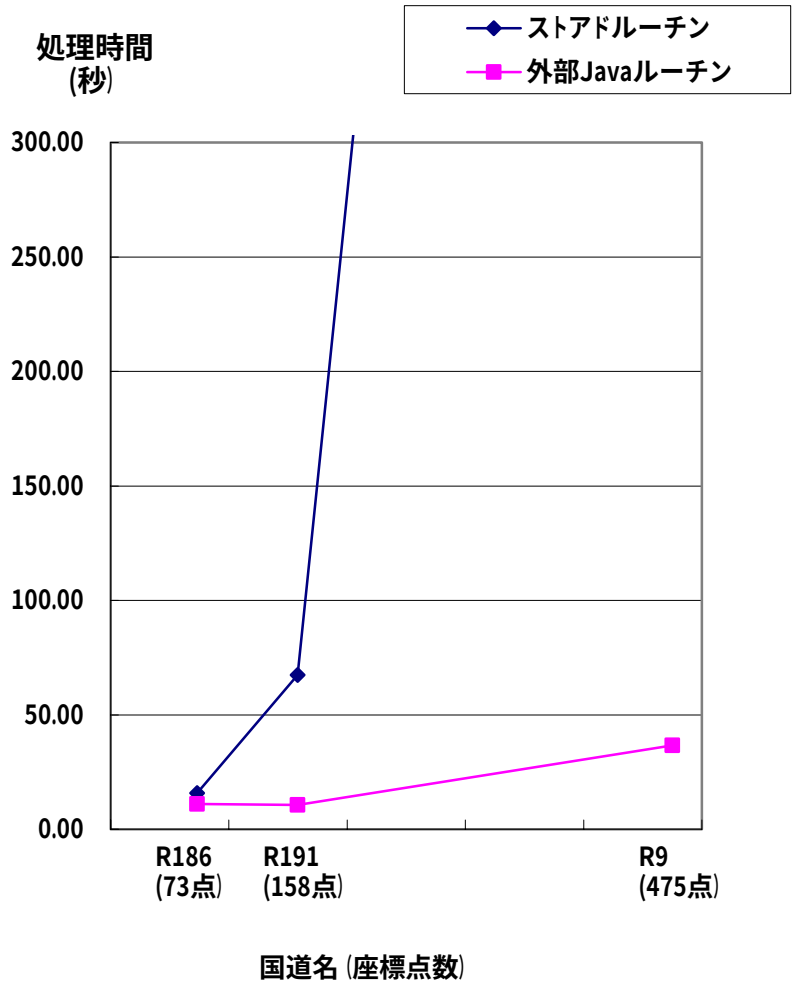
ボトルネックとなっていた人口検索部分を、HiRDB のストアドルーチンを使用した場合と外部 Java プログラムを使用した場合の比較を行う。格納されている国道座標データのうち大中小のデータに対し、帯域幅を 500 とし、老齢者人口検索行った。使用している全ての国道データのサイズは、この３つデータのいずれかとほぼ同じであるため、これら大中小３つのデータで試行する。結果を次に示す。

各国道データ座標点数

国道名	座標点数
R186	73
R191	158
R9	475

検索処理速度（時間）比較

国道	ストアドルーチン	外部 Java ルーチン
R186	15.78 秒	11.01 秒
R191	67.47 秒	10.66 秒
R9	1656.78 秒	36.59 秒



ここで言うストアドルーチンとは、データベース内部の HiRDB 空間検索プラグイン[6] が提供する機能よる共通集合演算を含む DB 内部ルーチンのことである。外部 Java ルーチンとは第 6 章で作成した共通集合演算をおこなうプログラムのことである。尚、グラフの横軸は国道名であるが、座標点数を基に間隔を設けてプロットしている。

7.2 性能評価

JTS ライブラリを用いた Java プログラムの方がいずれの場合も高速に検索できるようであるが、結果比較より考察すると次のようなことがわかる。

ストアドルーチンの場合、座標点数の増加に対して処理時間も大幅に増加する。外部 Java ルーチンの場合は、座標点数によって段階的に処理時間が増加する。グラフから両者とも指数的な増加傾向がみられるが、外部 Java ルーチンのほうがかなり緩やかである。

まず、R186 と R191 について比較してみる。座標点数をみると R191 は R186 のおよそ 2 倍である。処理時間はストアドルーチンの場合、R186 が 15.78 秒、R191 が 67.47 秒である。座標点数が 2 倍になると、処理時間がおよそ 4 倍になっている。それに対し Java ルーチンの場合、R186 と R191 では処理時間はおよそ 11 秒で、ほぼ同じである。

次に、R186、R191 に加え、R9 についても比較する。R9 の座標点数は 475 点で、R186 のおよそ 6.5 倍、R191 のおよそ 3 倍である。ストアドルーチンの場合、R9 は検索するのに 15 分以上かかる。Java ルーチンの場合は 36.59 秒で、ストアドルーチンよりはるかに高速である。

座標点数が少ない（50 点程度）であれば、両者とも処理性能はほぼ同等ではあるが、いずれの場合も Java ルーチンを使用した場合の方が高速に処理できることがわかった。

ストアドルーチンは HiRDB のプラグインによって提供されているファンクションを使用しているため、内部の仕様を知ることはできない。Java ルーチンでは、オープンソースである JTS ライブラリを使用しているため、仕様が公開されている。ただし、現段階でこの仕様に関しては詳細を十分には把握できていない。そのため、内部仕様について比較すること困難である。（いずれのルーチンも外部仕様は OGC の仕様に基づいて作成されている。）

第8章　まとめ

本研究の目的である「Web-ORDB 空間検索においてシステム全体の性能低下を招くボトルネックとなる箇所を調べ、その箇所の性能向上を図ること」は実現できた。しかし、未だ課題は残っている。本章ではそれらを含め、本研究により得られたことについてまとめる。

まず、ボトルネックとなる箇所の調査を行った。これはシステム性能向上のために重要な作業である。このシステムは比較的小規模ではあるが、いくつかのシステムを組み合わせにより作成されている。数秒の処理を 1 秒で処理できるようにするより、数分～数十分の処理を数秒で処理できるようにするほうが明らかに効果的である。本研究では処理時間に注目し、ボトルネックとなり得る箇所の実行時間を計測するという手法を用いた。それにより、ボトルネックとなる箇所を限定することができた。これに関して今後の課題としては、計測の詳細化である。その方法として、パケット単位のモニタリングが挙げられる。Web-ORDB の場合、データがパケットとして流れてゆく。このパケット自体を監視することで、データの内容・種類・場所・時間・方向を細かく調べることができる。さらにエージェント化することで、より柔軟なモニタリングが可能だと思われる。

次にシステム性能の向上の実現である。結果的には高速化を実現できた。しかし、まだ課題は多い。特に R9 の場合は 30 秒以上かかっており、これでは実用的とは言い難い。また、高速化のために試すべき方法が他にも考えられる。DBMS における方法として 2 点挙げる。

1 つ目は、空間データの格納方法の変更である。いま空間データが LineString または Polygon で格納されている。例えば、これを Point の配列として格納する、等。

2 つ目は、DBMS 自体のシステムチューニングである。DBMS の設定の最適化を行うことで、高速化を図ることができる。

その他、データの格納を DBMS ではなく、外部データとして保存する方法がある。外部データとして格納することで直接データにアクセスすることがきるので、DBMS における処理時間を短縮することができる。ただし、データ記録形式・管理方法・場所などを決定する必要がある。本研究では、データを DBMS に格納する方法をとった。DBMS を使用した理由として次のようなことが挙げられる。

- 関連データを一元的に管理できる。（管理機能が豊富）
- アクセス権を明確にでき、セキュリティーに強い。
- データの操作が容易である。

DBMS に関して、現在 ORDBMS において Java ストアドプロシージャが使用可能である。本研究で使用した HiRDB/Single Server V 6.0（現時点で最新版）では、Java 言語によるストアドプロシージャをサポートしているが、抽象定義型を含むものは使用不可能であったので、試すことができなかった。これをサポートする ORDBMS であれば、本研究で開発した外部 Java ルーチンを Java ストアドプロシージャとして使用することも検討できる。

謝　辞

本研究において最後まで熱心な御指導、助言をして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、本研究室所属の貫目さん、辰巳さん、高木さん、中村さん、喜代吉君、長田君、堀君、田辺さん、垰田さんには、本研究に関して数々の御協力と御助言をいただきました。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、ならびに関連する発表資料等の著作権を、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。

参考文献・資料・URL　（参照順）

[1] 2001 年度 島根大学 総合理工学部 数理・情報システム学科 情報分野
卒業論文「対話型意思決定システムの試作　－国道と人口メッシュを例として－」、荒木俊太郎

[2] OpenGIS Simple Features Specification for SQL Revision1.1
OpenGIS Consortium、1999 年
(http://www.opengis.org/techno/specs/99-049.pdf)

[3] 書籍『コア・サーブレット＆JSP Java サーバ技術による Web 開発』（2001 年）
Marty Hall 著、岩谷博 訳、ソフトバンク パブリッシング、2001 年

[4] Jakarta-Tomcat：The Apach Project（http://jakarta.apache.org/tomcat/）

[5] 書籍『実践 JDBC　Java データベースプログラミング術』
菊田英明 著、オーム社開発局、1998 年

[6] HiRDB マニュアル「HiRDB Spatial Search Plug-in Version 3 解説・手引・文法・操作書」、（株）日和 出版センター、2001 年

[7] JTS：Vivid Solutions 社（http://www.vividsolutions.com/projects/jts.html）

[8] 2001 年度 島根大学 総合理工学部 数理・情報システム学科 情報分野
卒業論文「地理情報システムのための JAVA ライブラリ JTS の実装と評価」、森岡慶彦