バス停沿線探索システムの
Windows Azure へのポーティング

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

S113101　徐　碧晨

目次

第 1 章 序論.....3
1.1 研究目的.....3
1.2 研究概要.....4
1.3 先行研究.....5
第 2 章 Windows Azure.....6
2.1 Windows Azure とは.....6
2.2 Windows Azure のサービス[12].....7
2.3 Windows Azure と Google App Engine の簡単な比較.....8
第 3 章 LINQ.....9
3.1 LINQ とは.....9
3.2 LINQ4j とは.....9
第 4 章 システム実装.....9
4.1 システムの流れ.....9
4.2 開発環境.....10
4.3 Windows Azure ストレージの設定.....11
4.4 格納データ.....11
4.4.1 バス位置テーブル.....12
4.4.2 バス路線テーブル.....12
4.4.3 バス編成テーブル 13
4.4.4 バス路線位置テーブル.....13
4.5 データのアップロード.....14
4.5.1 テーブルの定義.....14
4.5.2 テーブルにデータを追加する.....17
4.5.3 LINQ4j によるデータの取得.....18
4.6 バス停、バス路線及び時刻表検索.....20
第 5 章 今後の課題.....21
謝辞.....22

第 1 章　序論

1.1　研究目的

近年扱うデータ量が急激に増え、集中的なアクセスによるアクセス不可能が発生する可能性がある。例えば、東日本大震災のときのアクセスの集中などによる高負荷にある Web サイトについて Windows Azure を活用した。またはどんなに優れた Web サイトでも、アクセス集中によって Web サイトがなかなか表示されない、クリックしても反応がないとなれば、ユーザーの満足度が下がることは避けられない。サービスの成長や繁忙期のアクセス増加に柔軟に対応し、応答の遅延を改善しつつ、さらにコストを最適化するにはどうしたらよいのだろうか。年 2 回のイベントに合わせ負荷ピークが来る「コミケ」Web サイトなど、クラウドプラットフォーム「Microsoft Azure」の活用事例がある。[1]

多くの企業がクラウドの導入を進めている現在, Windows Azure はそうした企業のビジネス成長を加速させるプラットフォームとして、日々その存在感を増している[2]。

本研究では、先行研究の上に Google App Engine で実装したバス停沿線探索システムを Windows Azure への移行を行い、簡単な比較を行う。

1.2　研究概要

システムの Windows Azure 上での実装において、データベースでのデータの取り扱い, Windows Azure と Google App Enging の簡単な比較をおこなった。プログラムの作成においては, Eclipse 4.3(Kepler)[3] を用いて作成を行い, Azure Plugin for Eclipse with Java (Microsoft Open Technologies 提供)[4]を使用し, Windows Azure ストレージの作成, Apache Tomcat アプリケーションサーバーでの実行を行った。

システムの実行のためには, Java Development Kit (JDK) をインストールし, Azure サブスクリプションに Azure ストレージ アカウントを作成する必要がある。その後、開発システムが Azure Storage SDK for Java リポジトリに示されている最小要件と依存関係を満たしていることを確認する必要がある。リポジトリからシステムに Azure Storage Libraries for Java をダウンロードしてインストールし、システムの作成を行った[5]。

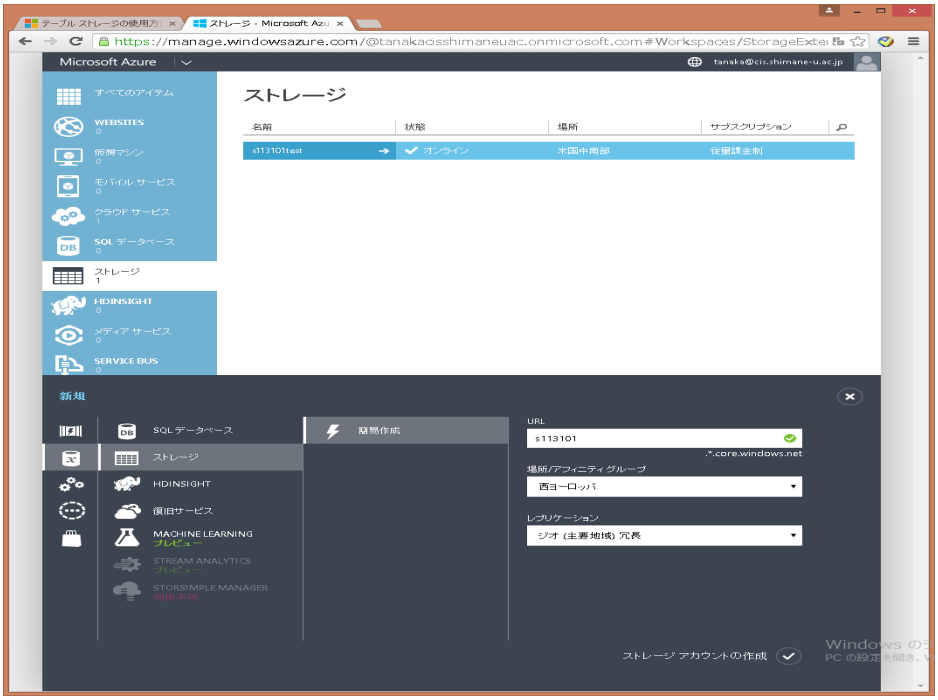

図 1.1Widows Azure 管理ポータル

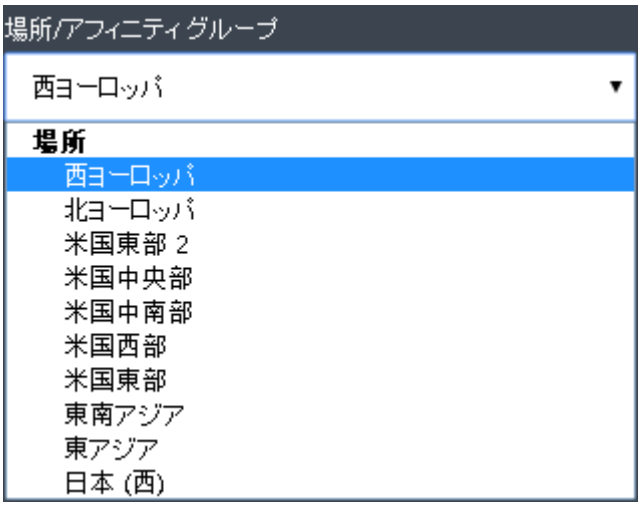

図 1.2Windows Azure データセンターの分布

1.3 先行研究

先行研究 [6] はリレーショナルデータベースの PostgreSQL[7] と地理空間情 報を扱うための PostGIS[8], GoogleMaps 及び JTS を用いて、現在位置情報、目的地情報、時刻等を元に近隣バス停、バス時刻、バス路線及び沿線情報を検索し、結果を表示するシステムである。

図 1.3 先行研究の例

第 2 章　Windows Azure

2.1　Windows Azure とは

Microsoft が提供する PaaS 型のクラウドサービス。2010 年 1 月から商用サービスが開始した[9]. Windows Azure は, Windows Server や SQL Server 相当の機能に加え、外部アプリケーションとの連携など多くの機能をもつクラウドサービスである. Windows Azure の特徴として、以下のように挙げられる[10]。

1. 分散データストア
2. Key/Value Store
3. トランザクション（制限あり）
4. 追記型（LSM-tree）
5. 複数ストレージへの書き込みでバックアッ プを作成している
6. データセンターを跨ったリプリケーション をしている
7. 複数エンティティのトランザクション処理
8. 高いコンカレンシー
9. 書き込みは DISK と同期、replication 先も同様
10. 高いスケーラビリティ
11. 高いレイテンシー
12. 制限された Query

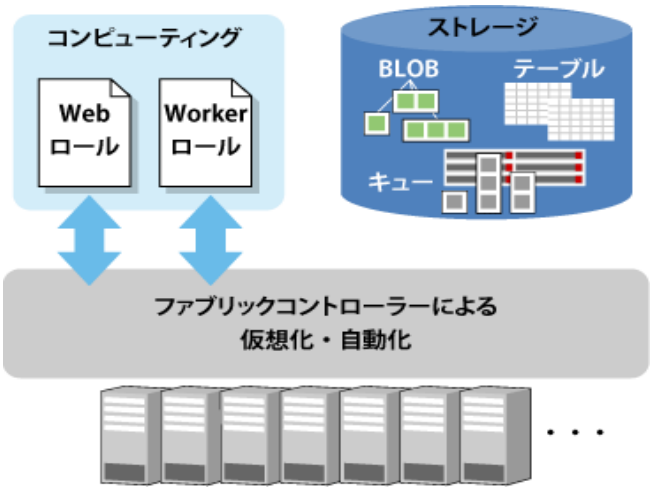

図 2.1.Azure の内部構造図(コンピューティングとストレージ)[11]

2.2 Windows Azure のサービス[12]

- コンピューティング系のサービス
- ネットワーク系のサービス
- Web 系のサービス
- モバイル系のサービス
- データと分析・解析系のサービス
- ストレージとバックアップ系のサービス
- 自動化とスケジューラー系のサービス
- ID 認証とアクセスの管理サービス
- CDN とメディア系のサービス
- ハイブリッド統合のサービス

2.3 Windows Azure と Google App Engine の簡単な比較

Google App Engine の Bigtable と Azure Table の基本の比較基本構成について, Azure Table は, Bigtable と GFS の組み合わせに類似している。[10]

(1) GFS ではレプリカ間の緩やかな整合性が許可され、すべてのレプリカのビット単位の同一性を保証して いないが, Azure Table はこれを保証する。

(2) Bigtable は tablet の書き込みの コミット ログを 2 つ の GFS ファイルに並行して書き込むことで GFS の 障害を回避. Azure Table では Stream Layer の Journaling で耐障害性を担保する。

トランザクションについて、どちらも , Entity Group Transaction と呼ばれるトランザクションをサポートする. GAE Datastore 同一 Entity Group に属する複数の Entity をトランザクション処理可能である。 Azure Table は 同一 Table, PartitionKey のデータは、トランザクション処理可能である。

データの配置について、配置のモデルが重要分散ストレージにどのようにデータを配置するか。分散トランザクションにしないため、同じロケーションであることが必要である. Bigtable エンティティ作成時に親となるエンティティを決めて、Entity Group を作成する。同一 Entity Group は、同じロケーションに配置される. Azure Table エンティティは作成時に Table, PartitionKey で指定する。両者が同じエンティティは、同じロケーションに置かれる。[10]

表 2.1Windows Azure とGoogle App Engine の簡単な比較

グランドサービス	Windows Azure	Google App Engine
サービス型	PaaS	PaaS
利用者が管理する単位	アプリケーション	アプリケーション
利用できる言語	.NET(C#,Visual Basic など), JAVA,PHP,Ruby,Pythom など	Python,JAVA,PHP,Go
開発ツール	Visual Studio,Eclipse など	Eclipse,Maven、Git など
利用できるストレージサービス	Blob,Table,Queue	データストアなど

第 3 章　LINQ

3.1　LINQ とは

LINQ とは「統合言語クエリ」で、言語（C#や Visual Basic など）のコード内に記述できるクエリとなる[13]。このクエリは SQL 文に似た構文により記述し、データベースをはじめ、さまざまな対象から、データの検索や集計、取得などが可能である。

3.2　LINQ4j とは

LINQ4j とは JAVA への LINQ のポート(A port of LINQ to Java)のことである[14]。本研究で作成したプログラムは JAVA を用いるため、LINQ4j を採用した。

第 4 章　システム実装

4.1　システムの流れ

はじめに、開発言語の選択と開発環境の実装を行った. Google App Engine を用いた先行研究のバス停沿線探索システムを Windows Azure にポーティングするため、言語を JAVA を選択し、開発環境 Eclipse, Tomcat の実装を行った。次に, Azure のストレージを作成し、データのアップロード[3]を行った. Google App Engine の BigTable と違い、アップロードされたデータはデータベース上の直接の確認ができないめ、LINQ を採用し, Azure Table 上のデータを確認するためのプログラムを作成した。LINQ とは「統合言語クエリ」で、言語（C#や Visual Basic など）のコード内に記述できるクエリとなる[4]。このクエリは SQL 文に似た構文により記述し、データベースをはじめ、さまざまな対象から、データの検索や集計、取得などが可能である。

さらに、半径 500 メートル以内や出発地、目的地の ID などの条件を加えてバス停の探索を行いたいため, LINQ を用いて、探索条件付きのプログラムを作成した。最後に、地図上の表示を行い、バス停沿線探索を行う。

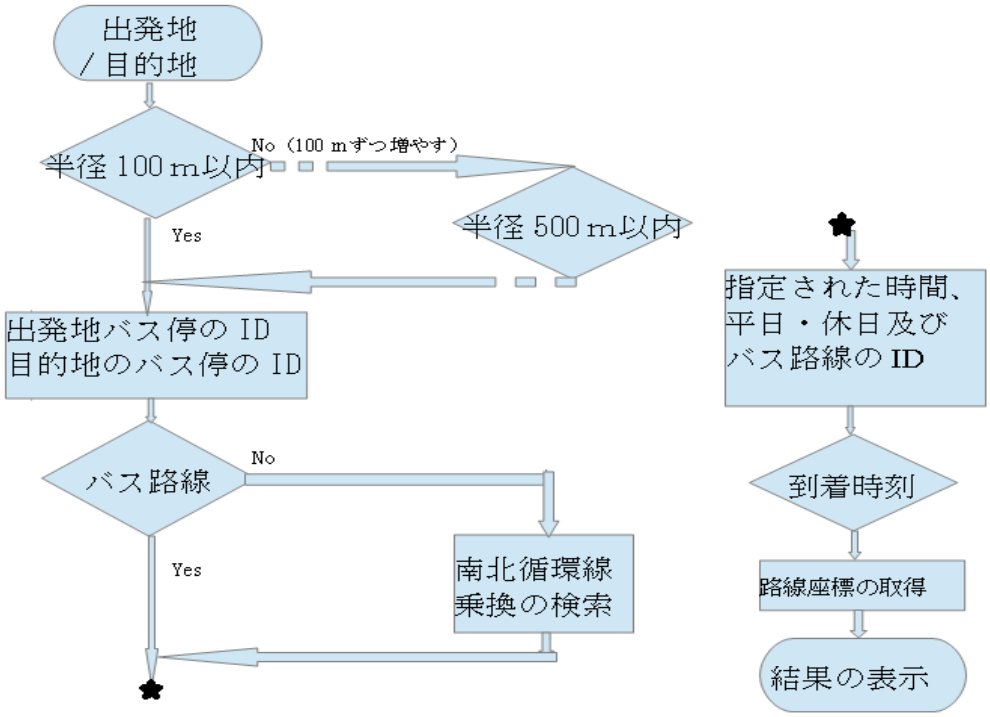

図 4.1　システムの流れ

4.2 開発環境

本研究を行うために開発環境を整える必要があった。今回は 1.2 研究概要 でも述べた通り，Eclipse 4.3(Kepler)[3] を用いて作成を行い，Azure Plugin for Eclipse with Java (Microsoft Open Technologies 提供)[4]を使用し，Windows Azure ストレージの作成，Apache Tomcat アプリケーションサーバーでの実行を行った。

システムの実行のためには，Java Development Kit (JDK) をインストールし，Azure サブスクリプションに Azure ストレージ アカウントを作成する必要がある。その後、開発システムが，Azure Storage SDK for Java リポジトリに示されている最小要件と依存関係を満たしていることを確認する必要がある。リポジトリからシステムに Azure Storage Libraries for Java をダウンロードしてインストールを行った[5]。

4.3 Windows Azure ストレージの設定

Windows Azure ストレージサービスを利用するために，Windows Azure 管理ポータルでの作成が必要である。

図 4.2　Widnows Azure ストレージアカウントを作成する画面

```
34    public String classpath=null;
35    public static final String storageConnectionString =
36            "DefaultEndpointsProtocol=http;" +
37            "AccountName=s113101test;" +
38            "AccountKey=ZquylsGe0UQRqk1F/OVcgPkXLSsRP9soF5GfBQiRmgPI+clPoi765K/y9HQo2QrRu/GDQr0fxLvXDr17Gh89oQ==";
39
```

図 4.3　作成した Widnows Azure ストレージサービスに接続するためのプログラムの一部

4.4 格納データ

先行研究[6]を参考として、島根県松江市を走行している松江市営バス [11] のバス停, 路線, 時刻表のデータを格納している。今回は平成 23 年 4 月 1 日のダイヤ改正に対応したデータを作成し, 使用した。以下は Windows Azure table にアップロードしたデータである。

表 4.1 利用するテーブルデータの一覧

テーブル	対応するデータ
バス編成テーブル	bus_org_*.csv （計179KB)
バス停路線テーブル	bus_route_13_23.csv(46KB), bus_route_13_23wgs.csv(120KB)
バス路線位置テーブル	bus_route_polygon_13_23.csv(565KB)
バス停位置テーブル	bus_stop_loc_13_23.csv(31KB)

4.4.1 バス位置テーブル

各バス停の情報を格納したテーブルであり, バス停検索を行う際にこのテーブルを利用する。このバス停位置テーブルには、バス停 ID、バス停名、バス停の緯度・経度のデータを格納している。

表 4.2 バス位置テーブルの一部

バス停名	バス停 ID	緯度	経度	X 座標	Y 座標
STRING 型	STRING 型	Double 型	Double 型	Double 型	Double 型
上乃木	10013	35.45	133.07	42536.94	106452.62
:	:	:	:	:	:
島根大学前	10732	35.48	133.07	42581.55	106455.01

4.4.2 バス路線テーブル

バス路線テーブルは、時刻表を検索する場合に必要となる、バスの路線の情報を格納した表である。出発バス停と目的バス停を使った検索（両方のバス停を通る路線の検索）を行うためのデータを格納してある。格納したバス路線数は 147 路線である。

表 4.3 バス路線テーブルの一部

路線 ID	路線名	通過するバス停 ID	検索用バス停 ID
STRING 型	STRING 型	STRING 型	LIST 型
10020101	県合同庁舎－川津 堅町～大橋	11612 11463 … 10270	[u'10894', u'10424',…, u'10434']
:	:	:	:
1327010	竹矢－東高校 界橋・駅・くにびき	10822 10722 … 14700	[u'10792', u'10174', …,u'14700']

4.4.3 バス編成テーブル

バス編成テーブルは、各バス停の時刻表を格納している表である。バス停位置テーブルよりバス停 ID を検索し、その ID より、路線テーブルからどの路線の何番目に通るバス停 ID かを検索した後で、時刻表を検索する。具体的には、路線 ID、時刻、バス ID、備考の列で構成されている。 路線 ID はバス路線テーブルの路線 ID と同じもの（その時刻の路線名の ID）が格納されており、バス ID については同じ路線でも何本もバスがあるので何本目のバスなのかを決定するために用いる。時刻については通過するバ ス停と同形式で時刻が格納されている。そのため、何番目に通過するバス停 かがわかれば，時刻を特定することが出来る。

表 4.4 バス編成テーブル

路線 ID	備考	バス ID	休日	時刻1	…	時刻 64
STRING 型	STRING 型	INT 型	STRING 型	STRING 型	…	STRING 型
10020101	平日	1	0	730	…	0
:	:	:	:	:	…	:
10460101	休日	1	1	1020	…	0

4.4.4 バス路線位置テーブル

バス路線位置テーブルは、乗車部分の路線座標を抜き出すためのテーブルであり、先に検索された路線 ID と何番目のバス停かを元に検索を行う。

表 4.5 バス路線位置テーブル

路線 ID	路線座標
STRING 型	TEXT 型
10020101	LINESTRING(35.4477 133.0852,35.4453 133.0846,……,35.4854 133.0709)
:	:
13270101	LINESTRING(35.4387 133.1203,35.4409 133.1185,……,35.4842 133.0783)

4.5 データのアップロード

前述したデータをひとつひとつアップロードするには膨大な時間がかかる．そこで、先行研究[6]と同じ方法を採用し、各々のデータを CSV ファイルにまとめ，ファイルをアップロードするプログラムを作成した．さらに、データのアップロードを確認するために，LINQ4j を用いて、データ全体を確認するためのプログラムを作成した。

4.5.1 テーブルの定義

4.4Windows Azure ストレージの設定で述べたように, WindowsAzure ストレージにアクセスするために、アクセスキーなどの必要である. AccountName と AccountKey の値には、管理ポータルに表示されるストレージアカウントの名前とプライマリアクセス キーを使用する。

```
1  package bus;
2
3
4  import java.io.BufferedReader;
5  import java.io.FileInputStream;
6  import java.io.IOException;
7  import java.io.InputStreamReader;
8  import java.io.PrintWriter;
9
10 import javax.servlet.ServletException;
11 import javax.servlet.annotation.WebServlet;
12 import javax.servlet.http.HttpServlet;
13 import javax.servlet.http.HttpServletRequest;
14 import javax.servlet.http.HttpServletResponse;
15
16
17 //Include the following imports to use table APIs
18 import com.microsoft.azure.storage.*;
19 import com.microsoft.azure.storage.table.*;
20
21 /**
22  * Servlet implementation class UploadDataset
23  */
24 @WebServlet("/UploadDataset")
25 public class UploadDataset extends HttpServlet {
26     private static final long serialVersionUID = 1L;
27
28     /**
29      * @see HttpServlet#HttpServlet()
30      */
31     public String classpath=null;
32     public static final String storageConnectionString =
33             "DefaultEndpointsProtocol=http;" +
34             "AccountName=s113101test;" +
35             "AccountKey=ZquylsGe0UQRqk1F/OVcgPkXLSsRP9soF5GfBQiRmgPI+clPoi76
36
37     public UploadDataset() {
38         super();
39         // TODO Auto-generated constructor stub
40     }
```

図 4.4 Windows Azure ストレージに接続するためのプログラムの一部

テーブルの作成を行うには CloudTableClient オブジェクトを作成し、これを使用して新しい CloudTable オブジェクトを作成する. CloudTableClient オブジェクトを使用すると、テーブルとエンティティの 参照オブジェクトを取得できる。この CloudTable オブジェクトは、″busorg1323，busroute1323，busroute1323wgs，busroutepolygon1323，busstoploc1323wgs″ という5つのテーブルを表す。

```
87      public void createTables(){
88
89          String tables[]={
90                  "busorg1323",
91                  "busroute1323",
92                  "busroute1323wgs",
93                  "busroutepolygon1323",
94                  "busstoploc1323wgs"};
95          try
96          {
97              // Retrieve storage account from connection-string.
98              CloudStorageAccount storageAccount =
99                         CloudStorageAccount.parse(storageConnectionString);
100             CloudTableClient tableClient = storageAccount.createCloudTableClient();
101             CloudTable cloudTable;
102
103             for(int i=0;i<tables.length;i++){
104                 try {
105                     cloudTable = new CloudTable(tables[i],tableClient);
106                     cloudTable.createIfNotExists();
107                 } catch (Exception e){
108                     // Output the stack trace.
109                     e.printStackTrace();
110                 }
111             }
112
113         }
114         catch (Exception e)
115         {
116             // Output the stack trace.
117             e.printStackTrace();
118         }
119
120
121             //Create tables online
122
123     }
```

図 4.5　テープルの定義のプログラムの一部

エンティティは，TableEntity を実装するカスタム クラスを使用して Java オブジェクトにマップされます。コードがシンプルになるように，TableServiceEntity クラスでは TableEntity を実装し、リフレクションを使用することで、プロパティを、それらのプロパティの名前が付いた getter および setter メソッドにマップしている。エンティティをテーブルに追加するには、最初に、エンティティのプロパティを定義するクラスを作成する。id を行キーとして，route_name をパーティション キーとしてそれぞれ使用するエンティティ クラスを定義する。エンティティのパーティション キーと行キーの組み合わせで、テーブル内のエンティティを一意に識別する。同じパーティション キーを持つエンティティは、異なるパーティション キーを持つエンティティよりも迅速に照会できる。

```
import com.microsoft.azure.storage.table.TableServiceEntity;

 public class busroute1323 extends TableServiceEntity implements Comparable
     public busroute1323(){}
     public String id; ///バスID

     public String route_name; //路線名

     public String com; ///バス会社名

     public String route_table; //路線

    public String getId(){
        return id;
    }

    public void setId(String id){
        this.id=id;
    }

    public String getRoute_name(){
        return route_name;
    }

    public void setRoute_name(String route_name){
        this.route_name=route_name;
    }

    public String getCom(){
        return com;
    }

    public void setCom(String com){
        this.com=com;
    }

    public String getRoute_table(){
        return route_table;
    }

    public void setRoute_table(String route_table){
        this.route_table=route_table;
    }

     public busroute1323(String id,String route_name, String com, String ro
         this.partitionKey = id;
         this.rowKey = route_name;
             this.id=id;
             this.route_name=route_name;
             this.com=com;
             this.route_table=route_table;
```

図 4.6 テーブル毎の定義のプログラムの一部

4.5.2 テーブルにデータを追加する

エンティティに関連するテーブル操作には TableOperation オブジェクトが必要である。このオブジェクトを使用して、エンティティに対して実行する操作を定義する。定義した操作は、CloudTable オブジェクトを使用して実行できる。

格納用に CustomerEntity クラスの新しいインスタンスを作成する。次に, TableOperation.insertOrReplace を呼び出し、テーブルへのエンティティの挿入用に TableOperation オブジェクトを作成し、そのオブジェクトを新しい CustomerEntity に関連付けている。最後に、CloudTable オブジェクトの execute メソッドを呼び出し、"busroute1323" テーブルと新しい TableOperation を指定している。それにより、新しいユーザー エンティティを "busroute1323" テーブルに挿入する 要求がストレージサービスに送信されるようになっている。

以下にバス停路線のデータのアップロードを行うプログラムの一部を示す。

```
279 public void uploadDataroute1323() throws NumberFormatException, IOException{
280
281 InputStreamReader isr = new InputStreamReader(new FileInputStream(classpath+"WEB-INF/classes/bus_route
282     BufferedReader br = new BufferedReader(isr);
283             String line = null;
284             while ((line = br.readLine()) != null) {
285                     String[] split = line.split(",",-1);
286
287                      System.out.println(line);
288                      String id = split[0].trim();
289                      String route_name = split[1].trim();
290                      String com = split[2].trim();
291                      String route_table = split[3].trim();
292                      String fts = split[3].trim();
293
294
295                     try
296                     {
297                         // Retrieve storage account from connection-string.
298                         CloudStorageAccount storageAccount =
299                            CloudStorageAccount.parse(storageConnectionString);
300
301                         // Create the table client.
302                         CloudTableClient tableClient = storageAccount.createCloudTableClient();
303
304                         // Create a cloud table object for the table.
305                         CloudTable cloudTable = tableClient.getTableReference("busroute1323");
306
307                         // Create a new customer entity.
308                         busroute1323 busroute1323 = new busroute1323(id,route_name,com,route_table);
309
310                         // Create an operation to add the new customer to the table.
311                         TableOperation insertCustomer1 = TableOperation.insertOrReplace(busroute1323);
312
313                         // Submit the operation to the table service.
314                         cloudTable.execute(insertCustomer1);
315                     }
316                     catch (Exception e)
317                     {
318                         // Output the stack trace.
319                         e.printStackTrace();
320                     }
```

図 4.7 データをテーブルにアップロードプログラムの一部

4.5.3　LINQ4j によるデータの取得

3.2LINQ4j で述べたように、データのアップロードを確認するため、LINQ4j を用いた。テーブルに対してパーティション内のエンティティを照会する場合は, TableQuery を使用できる. TableQuery.from を呼び出し、特定のテーブルに対するクエリを作成し、指定した型の結果が返るようにする. TableQuery.generateFilterCondition はクエリのフィルターを作成するためのヘルパー メソッドである. TableQuery.from メソッドによって返された参照の where を呼び出し、フィルターをクエリに適用する。

```
        \
            // Define constants for filters.
            final String PARTITION_KEY = "PartitionKey";

            // Retrieve storage account from connection-string.
            CloudStorageAccount storageAccount =
               CloudStorageAccount.parse(storageConnectionString);

            // Create the table client.
            CloudTableClient tableClient = storageAccount.createCloudTableClient();

           // Create a cloud table object for the table.
           CloudTable cloudTable = tableClient.getTableReference("busroute1323");
           //System.out.println(cloudTable.getUri());
            String partitionFilter = TableQuery.generateFilterCondition(
                    PARTITION_KEY,
               QueryComparisons.NOT_EQUAL,
               "0");


           TableQuery<busroute1323> partitionQuery =
               TableQuery.from(busroute1323.class)
               .where(partitionFilter);

          busroute1323 aa=new busroute1323();


           List<busroute1323> list =  new ArrayList<>();

           int count=0;

            for (busroute1323 entity : cloudTable.execute(partitionQuery)) {
                list.add(entity);
                /*
                System.out.println(entity.getPartitionKey() +
                    " " + entity.getRowKey() +
                    "\t" + entity.getCom() +
                    "\t" + entity.getWeekday());

               */
              count++;
           }


            List<String> nameList= Linq4j.asEnumerable(list).where(new Predicate1<busroute1323>
                public boolean apply(busroute1323 arg0) {

                    return arg0.id!="0";
                }
            }).select(new Function1<busroute1323, String>() {

                public String apply(busroute1323 arg0) {

                    return arg0.getRoute table();
```

図 4.8　アップロードを行ったデータ全体の確認プログラムの一部

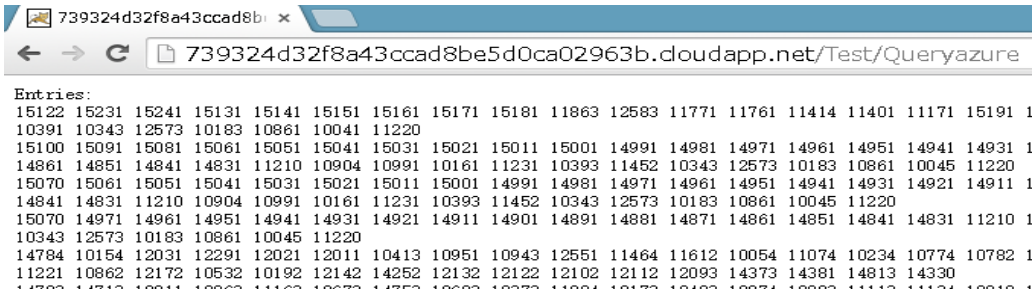

図 4.9　確認した結果

4.6　バス停、バス路線及び時刻表検索

路線検索は，周辺バス停の検索により，出発地のバス停と目的地のバス停 をバス停位置テーブルから検索を行う。バス停位置テーブルから取得したバス停 ID を基にバス路線テーブルから，路線 ID と何番目に通るバス停かを検索する。この検索結果を基にバス編成表から同じ位置にある時刻を検索する。この方法によって時刻を取得し、先行研究を参考し、作成した LINQ4j を採用するプログラムは，以下のようなになる。

```
180   List<String> start= Linq4j.asEnumerable(list).
181           where(new Predicate1<busstoploc1323wgs>() {
182       public boolean apply(busstoploc1323wgs arg0) {
183
184            return arg0.point_x>=(start_xWGS-r_length)
185                   &&arg0.point_x<=(start_xWGS+r_length);
186       }
187   }).select(new Function1<busstoploc1323wgs, String>() {
188
189       public String apply(busstoploc1323wgs arg0) {
190
191           return Double.toString(arg0.point_x);
192       }
193   }).toList();
194   for(int i=0;i<start.size();i++){
195       out.println(start.get(i));
196   }
```

図 4.10　条件にしたがう探索プログラムの一部

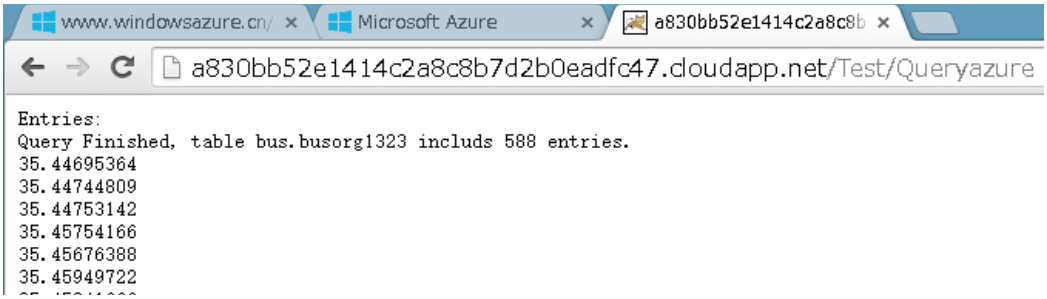

図 4.11　実行結果

第 5 章　今後の課題

本研究で作成しているシステムは、地図上で位置を指定し、指定したところの位置情報を探索する予定である。システム未完成のために、第4章のシステム実装は今後の課題となる。または、システムの実装にしたがって、GPSの導入などの工夫をし、よりユニークになると考える。さらに、Windows 8.1タッチパネル対応アプリケーションの作成し、便利性や探索における精度、速度の向上を望む。

謝辞

本研究にあたり，最後まで熱心な御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、田中究室の皆様にも、数々の御協力と御助言を頂きましたこと、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，並びに関連する発表資料などの全ての知的財産権を本研究の指導教員である田中章司郎教授に譲渡致します。

参考・引用文献

[1]http://web-tan.forum.impressrd.jp/e/2014/07/07/17691
[2]http://www.atmarkit.co.jp/ait/articles/1401/10/news010.html
[3]http://www.eclipse.org/
[4]http://azure.microsoft.com/ja-jp/documentation/articles/java-download-windows/
[5]http://azure.microsoft.com/ja-jp/documentation/articles/storage-java-how-to-use-table-storage/
[6]三島　健太,「JTSを用いた沿線検索システムの Google App Engine 上での実装」,島根大学大学院 総合理工学研究科数理・情報システム学専攻 修士論文,2012
[7]http://www.postgresql.org/
[8]http://postgis.refractions.net/
[9]http://www.atmarkit.co.jp/ait/articles/1401/10/news010.html
[10]http://www.slideshare.net/takekazuomi/nosql-bigtable-and-azure-table
[11] http://codezine.jp/ad/cloud_leader/azure
[12]http://www.atmarkit.co.jp/ait/articles/1408/18/news116.html
[13] http://www.atmarkit.co.jp/ait/articles/0803/25/news150.html
[14]https://github.com/julianhyde/linq4j