地図によるバス停時刻表検索システムのクラウドへの移行

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

s073083　三島　健太

平成 23 年 3 月 17 日

目 次

第1章　序論

1.1　研究目的

近年，インターネット上には様々なコンテンツや、サービスが多々存在している．そのため，インターネット上のデータ量や，アクセス数は膨大な量になっている．しかし，サイトの運営を行っている側の一般的なシステム構成では，システムに負荷がかかるとパフォーマンスが悪くなってしまう．その改善には，物理的にマシンの性能を上げるスケールアップという方法があるが限界がある．それに対して，並列にマシンを増やし，その数に比例して性能を上げるスケールアウトという方法が常套手段になってきた．

また，近年コンピュータのネットワーク化が進んでいき，ユーザが管理していた情報をインターネット上に集める「クラウド」と呼ばれる新しい利用形態が多くの企業で提供されている．そこで，先行研究である地図によるバス停時刻表検索システムをクラウド技術を用いている Google App Engine[1](以下，GAE) 上で実装を行うことにした．

1.2　研究概要

前述したシステムの GAE 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，検索構造などが異なるため，先行研究のプログラムをそのまま実装を行うことはできないので，プログラムの変更を行った．

プログラムの作成においては，Eclipse 3.4 (Ganymede)[2] を用いて作成を行い，Google Plugin for Eclipse を使用し，ウェブアプリケーションのプロジェクト作成，ローカルサーバでの実行，Google のインフラストラクチャへのデプロイを行った．

1.3　先行研究

先行研究 [3] はリレーショナルデータベースの PostgreSQL[4] と地理空間情報を扱うための PostGIS[5]，及び GoogleMaps[6] 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，バス時刻およびバス路線を検索し，結果を表示するシステムである．本研究では，このシステムを GAE 上で実装する．

図 1.1: 先行研究での実行例

第2章　クラウド

2.1　クラウドとは

クラウドとは，インターネットをベースとし，コンピュータの処理をネットワークを経由して利用する利用形態である．従来におけるコンピュータの利用はユーザが，コンピュータのハードウェア，ソフトウェア，データ等の保有，管理を行っていた．

しかし，近年ネットワークの高速化などから，サーバ等の物理マシンがどこにあっても気にならなくなったことから，最低限の接続環境 (パーソナルコンピュータ等のクライアント，インターネット環境など) ほど用意し，クラウドのサービスを利用したほど料金を支払う形態となっていて，ユーザの負担は軽減される．クラウド技術についてまとめると，主な利点は少なくとも以下の 2 つが挙げられる．

1. コストの削減

使用回数をベースにした Infrastructure as a Service(以下，IaaS) サービスを使用すれば，独自のサーバの投資額だけでなく，サーバの保守コストも削減が出来る．また，Software as a Service(SaaS) の使用により，ソフトウェアのインストールが不要となるためコストを節約できる．そして，データはクラウドに保存されるため，ユーザはインターネットに接続されている環境から，場所を問わずアプリケーションにアクセスが可能となる．

2. 機敏性

IaaS サービスの使用により，必要に応じてスケールアップやスケールダウンを行い，使用した分のみ使用料を支払うことが可能となっている．また，サービス提供の拡張が短時間で行うことも可能である．

2.2 Google App Engine

GAE とは，Google のインフラ上で，独自に開発した Web アプリケーションを動作させることができる Web アプリケーションホスティングサービスである．アプリケーションの構築や維持管理は容易に行え，トラフィックやデータストレージの増大に応じて容易なスケーリングが可能である．サーバの構築は必要なく，アプリケーションをアップロードを行うだけで利用できる．この GAE の特徴としては，以下が挙げられる．

- アプリ公開までの簡単さ
- スケールアウトのしやすさ
- データストアは Key-ValueStore である Bigtable
- ある程度の使用量までは無償

2.3 スケールアウト性能

GAE は高いスケールアウト性を持っている．一般的な Web システムにおいてアクセス数が増えると性能が悪化してしまうが，これはデータベースやアプリケーションサーバ，Web サーバやネットワークなど様々な原因により発生してしまう．性能を改善するためには，ボトルネックになりやすいデータベースサーバのサーバ機自体の入れ替えや強化に頼っていたがスケーラビリティに上限があった．

しかし，GAE においては高負荷状況が一定時間続くと，自動的に新しい App Server が追加され負荷分散をし，負荷が低くなると削除される．そのため，アクセスが集中しても，パフォーマンスが悪くなることはなく，サーバの構築といった作業が不要となるため，アプリケーションのプログラミングを行い，GAE にデプロイするだけで，スケールするアプリケーションを開発して公開できることが大きな特徴である．

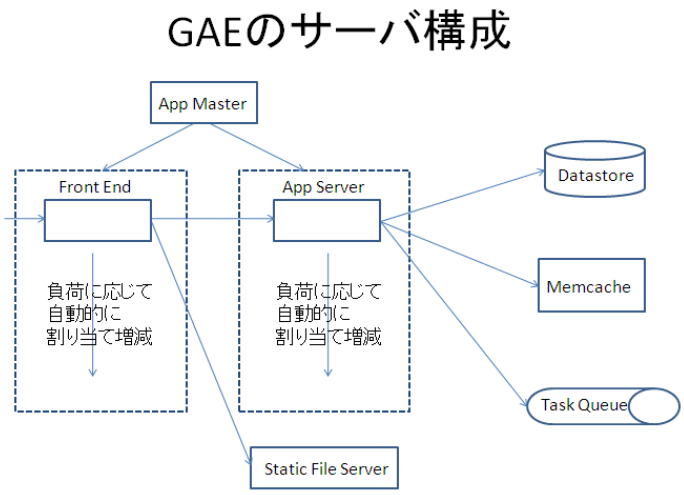

図 2.1: GAE のサーバー構成図

2.4 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 3.4 (Ganymede)[2] を用いて，開発言語は JAVA とした．GAE と Eclipse の連携を取るために Google App Engine SDK for Java[7] のパッケージの「appengine-java-sdk-1.4.0.zip」をダウンロードし，インストールを行った．そして，Eclipse の機能を用いて Google Plugin for Eclipse 3.4[8] をインストールした．

また，後に述べる全文検索を行うプログラムの動作のために，
`~\workspace\project\war\WEB-INF\lib`
に全文検索エンジン
「`lucene-core-2.9.1.zip`」，および
「`lucene-snowball-2.9.1.zip`」の追加，
CSV ファイルのアップロードのために
「`commons-fileupload-1.2.2.jar`」を追加した．

また，データの格納には AppEngine でデータストアの API 標準としてサポートされている Java Data Object(以下，JDO) を利用した．

第3章　システム移行

3.1　システムの流れ

使用方法は，先行研究と同様，地図上から出発地，目的地をクリックし，平日か休日かの選択 (①) をし，その情報を App Server に送信する (②)．App Server は受け取ったデータを基にクエリを発行し (③)，Datastore に接続 (④)．出発地周辺のバス停と目的地周辺のバス停を検索し (⑤)，検索結果のバス停から路線を検索する⑥．この路線を基に時刻表を検索し (⑦)，出発のバス停と目的のバス停の位置と時刻を取得し (⑧)，取得した情報をクライアントへ送信する (⑨)．クライアントはその情報から時刻とバス停名，および地図上に出発のバス停と目的のバス停のバス停位置，路線を表示する (⑩)．

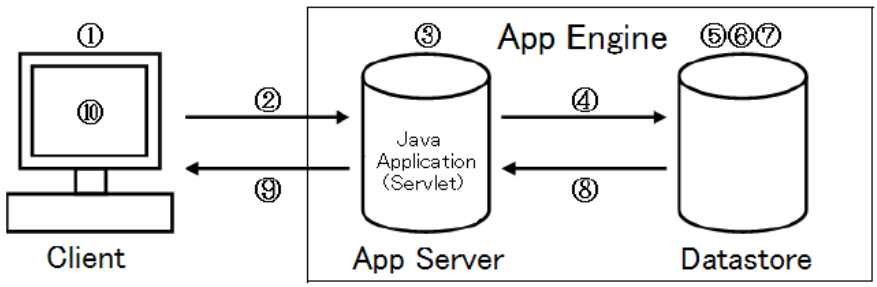

図 3.1: システムの流れ

3.2　位置情報取得方法

Google Maps API を用いてクリックした位置の情報を取得することで出発地と目的地の位置情報を取得することが出来る．以下に具体的な方法としてプログラムを示す.

Google Maps API を用いた位置情報取得方法

```
function onLoad(){
  //地図作成
  map = new GMap2(document.getElementById('map'));
  map.addControl(new GScaleControl());
  map.setCenter(center, 13);
  //クリックした位置を取得
  GEvent.addListener(map, 'click', onsMapClick);
}
//クリック処理
function onsMapClick(overlay, point){
  x=point.x;  //x に緯度格納
  y=point.y;  //y に経度格納
}
```

ここでは，GEvent.addListener(map,'click',onsMapClick); によりクリックした位置の緯度・経度をそれぞれ point.x，point.y に格納する．その緯度・経度を x と y に格納することにより，クリックした位置から緯度・経度を取得することが出来る.

3.3 環境設定

作成したアプリケーションのアップロードのためには様々な設定がある．Eclipse を用いたアップロードには，Google アカウントのメールアドレスとパスワードを入力する．Eclipse は，アプリケーション ID とバージョン番号を appengine-web.xml ファイルから取得し，war ディレクトリのコンテンツのアップロードを行う．そのため，appengine-web.xml に設定を記述する必要がある．

```
1<?xml version="1.0" encoding="utf-8"?>
2<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
3    <application>application-id</application>
4    <version>1</version>
5
6    <!-- Configure java.util.logging -->
7    <system-properties>
8        <property name="java.util.logging.config.file" value="WEB-INF/logging.properties"/>
9    </system-properties>
10
11</appengine-web-app>
```

図 3.2: appengine-web.xml の記述

図 3.2 の 3 行目の＜ application ＞タグに作成したアプリケーションの ID を記述する．4 行目の＜ version ＞タグには，アプリケーションのバージョンを記述することもできる．

3.4 格納データ

格納データに関しては，先行研究と同様に松江市を走行している一畑バス [9]，松江市営バス [10] の 2 つのバス会社のバス停，路線，時刻表のデータを格納している．今回は平成 20 年 4 月 1 日のダイヤ改正に対応したデータを用いた．

3.4.1　バス停位置テーブル

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．バス停 ID は書くバス停毎にユニークに付けた ID を格納している．このユニークな ID は提供して頂いたデータに付けてあった ID とほぼ同じものを付けている．

次にバス停名は，提供して頂いたデータをそのまま使用させて頂いている．

そして，バス停の経度・緯度に関しては，提供して頂いたデータは，日本測地系で計測したデータを 1/1000 秒単位の 10 進整数で記述してあった（例えば経度が 133 度 12 分 34.567 秒であったとすると度と分を秒に直し，1000 倍したもので，つまり $(133 \times 3600 + 12 \times 60 + 34.567) \times 1000$ を計算した値）．そのため，Google Maps で扱えるようにするために世界測地系に変換する必要がある．

世界測地系への変換方法は，まず，3600000 で割り，度で表す．次に，その座標の整数部分から１次メッシュコードを導く．さらに，その座標の整数部分をひいたものからそれぞれ緯度を 12 倍，経度を 8 倍し，その整数部分を２次メッシュコードとする．最後に，2 次メッシュコードを導いた座標から整数部分を引いたものからそれぞれ緯度を 120 倍し，経度を 80 倍し，その整数部分を 3 次メッシュコードとする．導いた 3 次メッシュコードでは地図からの検索をする上で，緯度と経度の 1 度あたりの長さが異なっているため，円形で検索をすると誤差が出てしまう．この問題に対して，3 次メッシュがほぼ 1km ということから，緯度・経度をそれぞれ 1200 倍，800 倍することで，1 が 100m となり，この値を格納している．

表 3.1: バス停位置テーブル

バス停名 STRING 型	バス停 ID STRING 型	緯度 Double 型	経度 Double 型	X 座標 Double 型	Y 座標 Double 型
上乃木	10013	35.447	133.065	42536.937	106452.624
:	:	:	:	:	:
島根大学前	10732	35.484	133.068	42581.547	106455.013
:	:	:	:	:	:
田和山史跡公園	14814	35.438	133.052	42525.800	106442.130

表 3.1 にあるように，ユニークに付けたバス停 ID，バス会社名及びバス停名は STRING 型，表示に使用する座標である緯度，経度は Double 型で定義し，格納した．また，先行研究においては，バス停の検索に使用する X，Y 座標を GEOMETRY 型で定義していたが，Bigtable ではサポートされていないため，座標の数値を Double 型で定義した．

格納したバス停の数は，松江市営バスが 562 カ所，一畑バスが 684 カ所である．

3.4.2 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．具体的にはユニークに付けた路線 ID，路線名，バス会社，通過するバス停 ID の列で構成されており，通過するバス停 ID の列を検索することで，路線の検索を行っている．路線の検索において，通過するバス停 ID の列はスペースで区切った文字列として，格納している．

表 3.2: バス停路線テーブル

路線 ID STRING 型	路線名 STRING 型	通過するバス停 ID STRING 型	検索用バス停 ID LIST 型
10020101	県合同庁舎－川津 竪町～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']
:	:	:	:
16030101	レイクライン (嫁ヶ島) 松江駅止	11227 10374 … 11220	[u'10832', u'12741',…,u'14194']

表 3.2 にあるように，路線 ID, 路線名及び通過する路線 ID は STRING 型で定義し，格納した．また，先行研究では，SQL 文の発行において，LIKE を用いた部分一致検索を用いていたが，Bigtable には対応していないため，検索用に新たにリストを作成し，検索を行えるようにした．格納したバス路線数は，松江市営バスが 128 路線，一畑バスが 117 路線である．

3.4.3 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．このデータに関しては，提供していただいたデータの状態を基に作成を行った．

表 3.3: バス編成テーブル

路線 ID STRING 型	備考 STRING 型	バス ID INT 型	休日 STRING 型	時刻 1 STRING 型	…	時刻 132 STRING 型
10020101	平日	1	0	7:30:00	⋯	0:00:00
:	:	:	:	:	:	:
10460101	平日	1	0	10:20:00	⋯	0:00:00
10460101	平日	2	0	11:20:00	⋯	0:00:00
10460101	平日	3	0	12:20:00	⋯	0:00:00
10460101	休日	1	1	10:20:00	⋯	0:00:00
:	:	:	:	:	:	:
18740102	平日	2	1	16:30:00	⋯	0:00:00

表 3.3 にあるように，路線 ID，備考及び休日は STRING 型で定義し，バス ID は INT 型で定義し，格納した．先行研究においては，時刻は TIME 型を用いていたが，Bigtable には対応していないため，STRING 型で定義し，格納した．格納した時刻表の数は，松江市営バスが 911 個，一畑バスは 378 個である．

3.4.4 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 3.4: バス路線位置テーブル

路線 ID STRING 型	路線座標 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)
:	:
16030101	LINESTRING(35.4642 133.0624,35.4699 133.0652,……,35.4641 133.0631)

表 3.4 にあるように，路線 ID は STRING 型で定義し，格納した．また，先行研究では，バス路線のバス停全てを GEOMETRY 型の LINESTRING で定義していたが，Bigtable には対応していないため，TEXT 型で定義を行った．STRING 型でなく TEXT 型としたのは,「STRING 型は 500 文字まで」という制限があるためである．

これらの表をまとめると以下の表 3.5 のようになる．

表 3.5: テーブル一覧

テーブル名	内容
バス停位置	バス停の座標を格納した表.
バス路線	バス路線の停車バス停を格納した表.
バス編成	バス停の時刻表を格納した表.
バス路線位置	バス路線の路線座標を格納した表.

※なお，一畑バスの場合，テーブル名の後に数字の「11」がつき，松江市営バスの場合はテーブル名の後に数字の「13」がつく．また，今回はダイヤ改正に対応させたテーブル名を一畑バスの場合は「テーブル名 11_20」，松江市営バスの場合も同様に「テーブル名 13_20」とした．

3.5　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．JavaSDK には，データのモデリング，保持のため JDO および Java Persistence API(以下，JPA) というインスタンスが実装されている．

3.5.1　エンティティの定義

クラスのインスタンスのエンティティとしてデータストアへの保存は，JDO では，クラスのアノテーションを使用している．以下にバス停のデータクラスを示す．

GAE では，JDO をサポートしていて，JDO を使用することでオブジェクトの永続化を行う．クラスは POJO で書き，アノテーションを付与することで永続化を行う．

```java
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

import com.google.appengine.api.datastore.Key;

@PersistenceCapable(identityType=IdentityType.APPLICATION)
public class bus_stop_loc_13_20wgs{
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key key;

    @Persistent
    private String id;

    @Persistent
    private String bus_stop_name;

    @Persistent
    private double point_x;

    @Persistent
    private double point_y;

    @Persistent
    private String com;

    @Persistent
    private double geo_x;

    @Persistent
    private double geo_y;
}
```

図 3.3: エンティティの定義

「@ PersistentCapable」アノテーションは永続化対象のクラスであることを宣言する.

「@ Persistent」アノテーションは永続化対象のフィールドであることを宣言する.

「@ PrimaryKey」アノテーションはオブジェクトを一意に判断するためのフィールドであることを宣言する.

3.5.2　データの追加

PersistenceManager を使用して行う．以下にバス停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．

```
40                  if (contentType.contains("text")
41                          || itemStream.getName().endsWith(".csv")) {
42                      resp.setContentType("text/html");
43                      BufferedReader buffer = new BufferedReader(
44                              new InputStreamReader(inputStream, "UTF-8"));
45                      String line = null;
46                      int i = 0;
47                      while ((line = buffer.readLine()) != null) {
48                          String[] split = line.split(",",-1);
49
50                              // csvからデータの取り出し
51                              String id = split[0].trim();
52                              String bus_stop_name = split[1].trim();
53                              double point_x = Double.parseDouble(split[2].trim());
54                              double point_y = Double.parseDouble(split[3].trim());
55                              String com = split[4].trim();
56                              double geo_x = Double.parseDouble(split[5].trim());
57                              double geo_y = Double.parseDouble(split[6].trim());
58                              // 登録するモデル
59                              bus_stop_loc_13_20wgs per = new bus_stop_loc_13_20wgs(id,
60                                      bus_stop_name,point_x,point_y,com,geo_x,geo_y);
61                              PersistenceManager pm=pmf.getPersistenceManager();
62                              if (pm != null) {
63                                  try{
64                                      pm.makePersistent(per);
65                                  }finally{
66                                      pm.close();
67                                  }
68                              }
69                              i++;
70                      }
71                      out.print( i + "件登録しました。");
72                  }
73
```

図 3.4: データのアップロード

プログラムの手順を示す．

1. String 型の配列「split」に CSV ファイルのデータを一行ずつ取得．
2. trim メソッドを用いて，各パラメータを取り出す．
3. インスタンス化し，データストアに格納 (格納は close メソッドのタイミング)．

3.6 Bigtableにおける検索

Bigtableは，リレーショナルデータベースではなく，分散Key-Value Store(以下，KVS) の 1 つである．KVS は,「Key(キー)」と「Value(値)」のペアからなるデータストアである．リレーショナルデータベースでは，パラメータの値に基づいて検索を行っていたが，Bigtable においては Key の値を基に検索を行う．また，SQL 文のシンタックスも異なる．下記に出発バス停の検索を行う SQL 文 (クエリ) の例を挙げる．

a) リレーショナルデータベースでの SQL 文

```
SELECT 時刻，バス ID FROM バス編成 WHERE 路線 ID=’17720101’
AND day=’0’ AND 時刻 ＞＝’10:59:00’;
```

b)Bigtable でのクエリ

```
SELECT FROM バス編成 WHERE id=’17720101’ && day=’0’
```

上記にあるように a) では，SELECT と FROM の間取得するパラメータを書いてしていたが，b) ではなくなっていたり，a) では，WHERE 以下に複数の条件を指定する際には「AND」と書いてあるが，b) では,「&&」になっているなどシンタックスが異なる．先行研究を GAE 上で再現するためにはプログラム内の全てのクエリの修正を行う必要があった．

3.7 検索方法

このシステムの検索手順は以下のように行う．

1. 出発地と目的地で共に半径 500m 以内にバス停があるか判定する．なければ終了．
2. 受け取った出発位置からまず，半径 100m 以内の周辺のバス停を検索する．なければ，半径 200m 以内の周辺のバス停を検索する．これを 100m 毎に 500m まで行い，１件も見つからなければ，終了する．
3. 目的位置からも出発位置と同様にまず，半径 100m 以内の周辺のバス停を検索する．なければ，半径 200m 以内の周辺のバス停を検索する．これを 100m 毎に 500m まで行い，１件も見つからなければ，終了する．
4. 周辺のバス停が見つかった場合，そのバス停と出発のバス停との路線を検索する．路線があれば，時刻表を検索し，なければ，次の目的位置周辺のバス停を検索する．それでもない場合は 2. へ戻る．

5. 路線があった場合，この路線の何番目のバス停かを取得する．

6. 「何番目のバス停か」と「日時」を基に時刻を検索する．

7. 路線 ID，何番目のバス停かを基に路線座標検索を行う．

3.7.1 空間検索を用いた周辺バス停検索

先行研究では，周辺バス停位置の検索には，取得した位置情報と各バス停の位置情報を PostgreSQL の空間情報の拡張である PostGIS を利用して DBMS で検索を行っていた．まず，取得した位置情報を変換して，各バス停の位置情報を用いた検索をできるようにする．実際に PostgreSQL へ発行する SQL 文は以下のようなものである．

```
SELECT バス停名 FROM バス停位置 WHERE ST __ DWITHIN(座標,
GeomFromText('POINT(緯度 経度)',SRID[4326]), 半径 [500m]) = TRUE
```

※"バス停名"は列名，"バス停位置"はテーブル名，"WHERE"以下が検索条件である．また，"GeomFromText()"は PostGIS 独自のもので空間検索を行う際 () 内に，Geometry，SRID，半径などの条件を指定する．"POINT"も PostGIS 独自のものであり，円形を表し，() 内には緯度・経度を指定する．

しかし，Bigtable においては PostGIS のような空間検索をサポートしていないため，このような検索を行えない．そのため，座標を数値として格納しているので，以下のようにして検索を行う．

まず，前提として，以下のようにバス停の座標が格納されていて，現在地を図のように設定したとする．

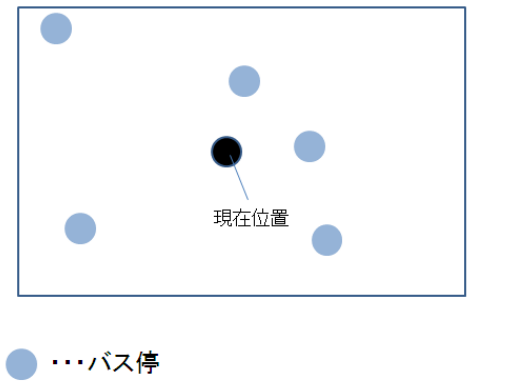

図 3.5: 周辺バス停の検索

1. バス停の Y 座標が，現在地の Y 座標と 500M 以内のものを検索する．

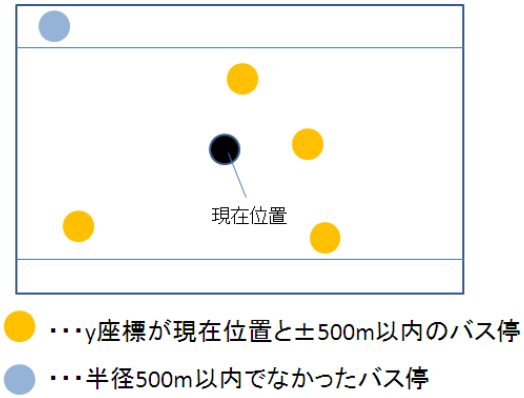

図 3.6: 周辺バス停の検索

2. 1 の条件に合ったもので X 座標も同様に 500M 以内のものを検索する．

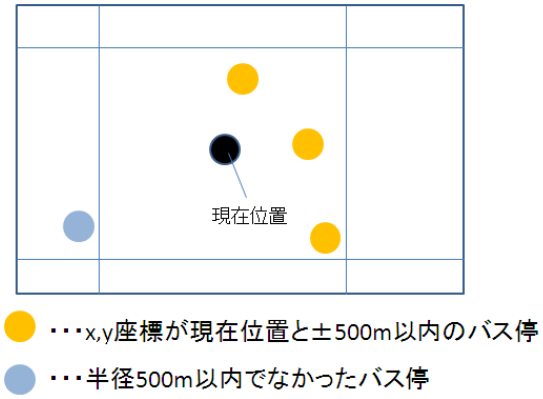

図 3.7: 周辺バス停の検索

3. X，Y 座標ともに 500M 以内であるバス停のみに対して，現在位置との距離を取る．

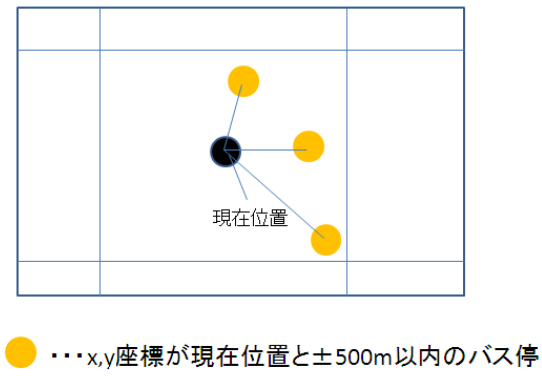

図 3.8: 周辺バス停の検索

4. 距離が 500M 以内であれば，そのバス停は 500M 以内にあるとみなす．

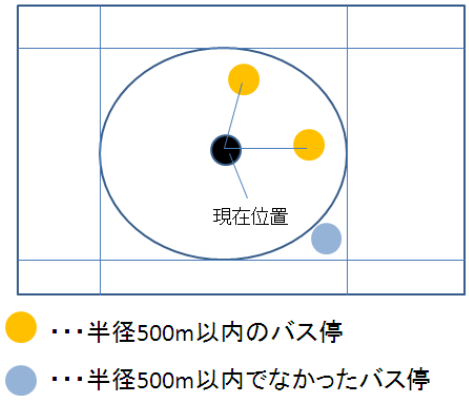

図 3.9: 周辺バス停の検索

このように，最初から全てのバス停 (1264 カ所) と距離を取れば，著しく処理速度が低下してしまうので，ある程度選別を行い，距離を取るようにし，近隣バス停の検索を行った．

3.7.2 路線検索

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，先行研究で実際に発行する SQL 文は，以下のようなものである．

```
SELECT 路線 ID FROM バス路線 WHERE ルート一覧 LIKE ‘ %11220%10732% ’;
```

※”路線 ID”は列名．”バス路線”は表名．”WHERE”以下が検索の条件，”LIKE”は部分検索を行うもので”%”には任意の 0 文字以上の文字列が挿入される．

しかし，Bigtable では”LIKE”を用いた部分一致検索を行うことはできない．そのため，全文検索を行うプログラムを作成した．

全文検索のプログラムは，あらかじめ，バス路線のデータのアップロードの際に，バス路線テーブルを細かく分割した「fts」というものを作成し，この「fts」の中に出発・目的地のバス停 ID がともに存在すれば，バス路線 ID を返すものとなっている．

3.7.3 時刻表検索

先行研究での時刻表検索における SQL 文は以下のようなシンタックスとなっている．

```
SELECT 時刻，バス ID FROM バス編成 WHERE 路線 ID=’17720101’
 AND day=’0’ AND 時刻 ＞＝’10:59:00’;
```

時刻について範囲検索を用いている．しかし，Bigtable には以下のようにクエリに制限がある．

「同一のクエリに，あるプロパティを対象に不等号を用いて範囲検索を行うと，ほかのプロパティの条件指定が含められない」

本システムでは，インデックスの作成をしていないため，時刻の範囲検索行おうとすると，ID や平日・休日の指定が行えない．そこで，本システムでは，ID と平日・休日の指定をする．例として，以下のように指定したとすると

```
SELECT FROM バス編成 WHERE id=’17720101’ && day=’0’
```

以下のような結果が得られる．

bus_id	com	day	id	remarks	time1
1	松江市営バス	0	17720101	平日	9:15:00
3	松江市営バス	0	17720101	平日	12:45:00
4	松江市営バス	0	17720101	平日	14:30:00
5	松江市営バス	0	17720101	平日	16:15:00
2	松江市営バス	0	17720101	平日	11:00:00

図 3.10: 出発バス停時刻表検索例

検索結果から該当するバス ID，時刻を取得する．目的地のバス停には，バス ID も条件に追加して検索を行う．

以下は，目的地のバス停時刻検索のクエリ

SELECT FROM バス編成 WHERE id='17720101' && day='0' && bus __ id='2'

このクエリから検索結果を以下のように得る．

bus_id	com	day	id	remarks	time1
2	松江市営バス	0	17720101	平日	11:00:00

図 3.11: 目的バス停時刻表検索例

この結果から該当する時刻を取得する．

3.7.4 路線座標検索

路線座標の検索には，バス路線テーブルから取得した路線 ID と何番目に通るバス停かを基に，出発バス停から目的バス停までの路線座標を検索する．

実際に先行研究で発行している SQL 文は，以下のようなものである．

SELECT AsText(路線座標) FROM バス路線位置 WHERE 路線 ID = 10020101

※"路線座標"は列名，"バス路線位置"は表名，"WHERE"以下が検索の条件である．また，AsText( ) はカプセル化された GEOMETRY 型データを WKT 文字列で返す．

しかし，PostgreSQL での路線座標の格納は，GEOMETRY 型の LINESTRING 型で定義していたが，BigTable では路線座標は TEXT 型で定義している．そのため，AsText( ) は使用せずに路線座標を取得するようにした．

3.8　結果表示

結果の表示には先行研究と同様に，JavaScript や GoogleAPI などを用いている．

図 3.12: 実行結果

第4章　終論

本研究では，先行研究の地図によるバス停時刻表検索システムをクラウド技術を用いている GAE への移行を行った．クラウド技術の GAE 上で実装を行うことで，高スケール性のシステムが無償で作成できたことは，企業などが新たな事業などに携わるときなどにも有益なことである．

また，今回作成したシステムでは，路線座標以外にはインデックスを作成しないで実装したため，高スケールなシステムだが検索などの処理速度が低下してしまった．それゆえに，今後の課題としては，シングルプロパティインデックス等のインデックスを用いた検索構造を用いて処理速度の向上を計ることである．

また，データを最新のものに更新し，沿線情報の検索機能などの追加をし，ユーザにとって有益なシステムにすることが今後の課題である．

そして，近年，携帯端末の発展している．その高機能な長所を生かすために，携帯端末にも対応するようにしていくことも必要である．

第5章　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，同じ研究室の韋于思さん，李昊さん，小室翼さん，橋本沙優さんには本研究に関して，数々の御協力と御助言を頂きました．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，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．

参考文献

[1] http://code.google.com/intl/ja/appengine/

[2] http://www.eclipse.org/

[3] 宇垣 貴裕,「空間データベースを用いた沿線検索システム構築の試み」, 島根大学 総合理工学部 数理・情報システム学科 卒業論文,2009

[4] http://www.postgresql.org/

[5] http://postgis.refractions.net/

[6] http://maps.google.co.jp/

[7] http://code.google.com/intl/ja/appengine/downloads.html

[8] http://code.google.com/intl/ja/eclipse/docs/download.html

[9] http://ichibata.co.jp/bus/

[10] http://www.matsue-bus.jp/