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

計算機科学講座 田中研究室

S119325　三島　健太

目 次

第 1 章 序論 **3**
1.1 研究目的 3
1.2 研究概要 3
1.3 先行研究 4

第 2 章 Google App Engine **5**
2.1 Google App Engine とは 5
2.2 クラウドとは 6
2.3 スケールアウト性能 7

第 3 章 JTS **8**
3.1 JTS とは 8
3.2 JTS の特徴 8
3.2.1 Spatial Data Model 8
3.2.2 Binary Predicates 9
3.2.3 Spatial Analysis Methods 9

第 4 章 システム実装 **10**
4.1 システムの流れ 10
4.2 開発環境 11
4.3 位置情報取得方法 12
4.4 環境設定 13
4.5 格納データ 13
4.5.1 バス停位置エンティティ 14
4.5.2 バス路線エンティティ 16
4.5.2.1 全文検索エンジン Lucene を用いた検索用データの登録 16
4.5.2.2 行方向に展開した路線検索用データエンティティ 17
4.5.3 バス編成エンティティ 18
4.5.4 バス路線位置エンティティ 19
4.5.5 沿線検索範囲ポリゴンデータエンティティ 20
4.6 データのアップロード 21

4.6.1 エンティティの定義 21
4.6.2 データの追加 23
4.7 インデックスの登録 24
4.7.1 インデックスの登録方法 24
4.7.2 インデックスの削除方法 25
4.8 バス停，バス路線及び時刻表検索 27
4.8.1 空間検索を用いた周辺バス停検索 27
4.8.1.1 ユークリッド距離を用いた検索 28
4.8.1.2 円形の Buffer と Within メソッドの組み合わせによる検索 31
4.8.1.3 isWithinDistance メソッドによる検索 32
4.8.1.4 distance メソッドによる検索 33
4.8.1.5 バス停検索方法の比較と検証 34
4.8.2 路線検索 35
4.8.2.1 全文検索エンジン Lucene を用いた路線検索 . 36
4.8.2.2 行方向に展開した路線検索 39
4.8.2.3 路線検索手法の比較と検証 39
4.8.3 時刻表検索 40
4.8.4 路線座標検索 42
4.8.5 南北乗換えありの場合の路線・時刻表検索 43
4.9 結果表示 44
4.10 沿線検索 46
4.10.1 Google Local Search API 46
4.10.2 沿線検索方法 46

第 5 章 システム検証 **48**

第 6 章 終論 **49**

第 7 章 謝辞 **50**

第1章　序論

1.1　研究目的

近年，Google Maps[1] のような JavaScript を用いた Web アプリケーションの利用が拡大している．このようなサイトでは地図上に店舗情報や自社の所在地を表示し，住所を文字としてでなく地図上で視覚的に表示することで店舗などの位置を探しやすく，分かりやすくなった．しかし，現在位置周辺やバスの路線沿いにある店舗や観光情報の検索のような空間検索を行うサイトはまだ少なく，移動手段として徒歩，車，電車等を想定したものが多く，バスを想定したものはほぼ皆無である．そこで本研究では，高いスケールアウト性能を持つ Google App Engine (GAE)[2] を用いて島根県松江市のバス停・時刻表及び路線沿いの施設情報の検索を行うシステムの実装を行った．GAE 上で実装することについては，従来は開発者がサーバーの構築・維持・管理を行っていたが，GAE ではあらかじめ環境が用意されているので開発に集中できること，また，信頼性のある Google のインフラ技術，特に，サーバーの負荷分散，パフォーマンスの高さ，セキュリティ面などが一通り整備され，利用できるということが利点として挙げられる．

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，検索構造に留意し実装を行った．

プログラムの作成においては，Eclipse 3.5 (Galileo)[3] 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 [4] はリレーショナルデータベースの PostgreSQL[5] と地理空間情報を扱うための PostGIS[6]，及び GoogleMaps 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 JTS を用いて実装し，新たに沿線検索システムの機能を実装した．

図 1.1: 先行研究での実行例

第2章　Google App Engine

2.1　Google App Engineとは

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 [2]．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 の主な特徴としては，以下が挙げられる．

- アプリ公開までの簡単さ
- 自動スケーリングと負荷分散
- クエリ，並べ替え，トランザクションに対応した永続ストレージ
- ある程度の使用量までは無償
- 一般的なウェブ技術を完全にサポートする動的ウェブ処理

2.2　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 [7].

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．

2.3　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 [7].

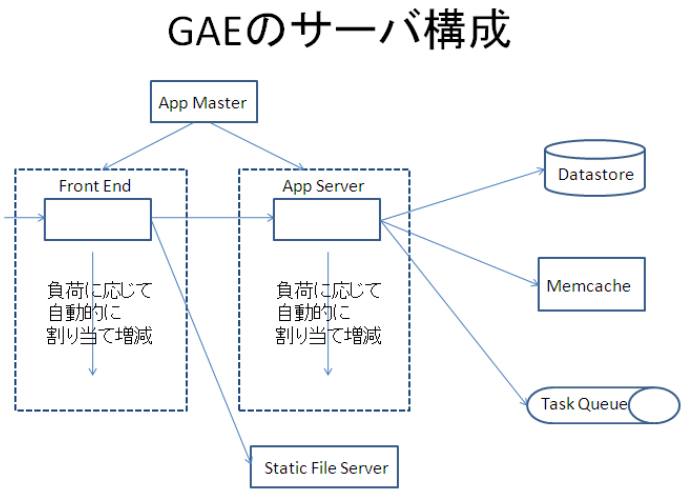

図 2.1: GAE のサーバー構成図

第3章　JTS

3.1　JTS とは

JTS とは Java Topology Suite の略であり，カナダの Vivid Solutions 社によって開発された 2 次元空間アルゴリズムの完全な一貫性を持った Java API である．JTS はオープンソースであり，自由に改良，用途に合わせた利用を行うことが許されている [8].

3.2　JTS の特徴

Java 言語で構成された空間情報システム向けの API であり，高性能，高品質，信頼性の高い機能やアルゴリズムを多々採用している．

3.2.1　Spatial Data Model

Spatial Data Model という，JTS で定義されている図形のモデルがある．今回実装に用いたモデルは，以下のようなモデルが定義されている．

表 3.1: JTS でサポートされている Spatial Data Model

図形名	内容	適用
Point	点	バス停の座標
LineString	折れ線	バスの走行順路

3.2.2 Binary Predicates

Binary Predicates は，図形に対して問いかけを行い，その問いかけの条件に合う部分が存在するかを調べ，存在すれば True，存在しなければ False の結果を返す関数である．今回の実装に用いたものは以下のような述部が定義されている．

表 3.2: JTS でサポートされている Binary Predicates

述部名	機能	適用
Within	一方の図形が他方の図形の内部にあるか	バス停検索，沿線検索
LineString	折れ線	バスの走行順路

3.2.3 Spatial Analysis Methods

Spatial Analysis Methods は，Spatial Operation を実行するメソッドである．Spatial Operation とは，ある図形に対して問いかけを行い，その条件に一致する部分が存在するかどうかを調べ，結果を Geometry 型で抽出する操作のことである．今回の実装に用いたものは以下のようなメソッドが定義されている．

表 3.3: JTS でサポートされている Spatial Analysis Methods

メソッド名	機能	適用
Buffer	図形の指定した範囲内の領域を返す	バス停検索，沿線検索

第4章　システム実装

4.1　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)．

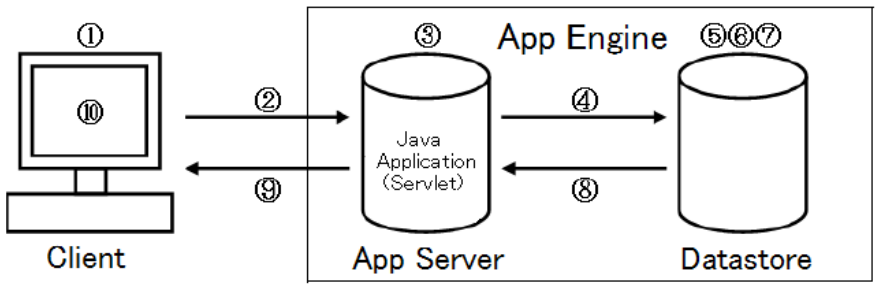

図 4.1: システムの流れ

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 3.5 (Galileo)[3] 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[9] のパッケージの「appengine-java-sdk-1.5.5.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 3.5[10] 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，`JTS` の機能を使用するために
「`jts-1.8.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．

4.3　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.

4.4 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 4.2: appengine-web.xml の記述

図 4.2 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．

4.5 格納データ

今回使用したデータに関しては，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．

4.5.1　バス停位置エンティティ

各バス停の情報を格納したエンティティであり，バス停検索を行う際にこのエンティティを利用する．

このバス停位置エンティティには，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 4.1: バス停位置エンティティ

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 4.1 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 562 カ所である．

4.5.2　バス路線エンティティ

バス路線エンティティは，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．

4.5.2.1　全文検索エンジン Lucene を用いた検索用データの登録

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 4.2: 全文検索エンジン Lucene を用いた検索用バス路線エンティティ

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 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']
:	:	:	:
18740102	松江駅－大海崎（フリー） －八束町中央（復路）	11272 10802 … 11220	[u'10242',u'11220',…,u'11802']

表 4.2 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，リレーショナルデータベースでは，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．

4.5.2.2　行方向に展開した路線検索用データエンティティ

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，乗車バス停，降車バス停で構成されており，乗車・降車バス停検索することで，路線の検索を行っている．そのため，各路線から考えられる全ての乗車，降車バス停の組み合わせを登録する．

表 4.3: 行方向に展開した路線検索用データエンティティ

路線 ID STRING 型	路線名 STRING 型	通過するバス停 ID STRING 型	乗車バス停 ID STRING 型	降車バス停 ID STRING 型
10020101	県合同庁舎－川津 竪町～大橋	11612 11463 12552… 10270	11612	11463
10020101	県合同庁舎－川津 竪町～大橋	11612 11463 12552… 10270	11612	11552
:	:	:	:	:
18740102	松江駅－大海崎（フリー） －八束町中央（復路）	11272 … 10842 11022 10542	10842	10542
18740102	松江駅－大海崎（フリー） －八束町中央（復路）	11272 … 10842 11022 10542	11022	10542

表 4.3 にあるように，登録したデータはすべて STRING 型で定義し，格納した．格納したデータ数は 86913 件である．

4.5.3 バス編成エンティティ

バス編成エンティティは，各バス停の時刻表を格納している表である．バス停位置エンティティよりバス停 ID を検索し，その ID より，路線エンティティ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 はバス路線エンティティの路線 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 4.4: バス編成エンティティ

路線 ID STRING 型	備考 STRING 型	バス ID INT 型	休日 STRING 型	時刻 1 INT 型	…	時刻 64 INT 型	時刻表 STRING 型
10020101	平日	1	0	730	…	0	7:30/7:32/…/8:10
:	:	:	:	:	:	:	:
10460101	平日	1	0	1020	…	0	10:20/10:21/…/11:01
10460101	平日	2	0	1120	…	0	11:20/11:21/…/12:01
10460101	平日	3	0	1220	…	0	12:20/12:21/…/13:01
10460101	休日	1	1	1020	…	0	10:20/10:21/…/11:01
:	:	:	:	:	:	:	:
18740102	平日	2	0	1930	…	0	19:30/19:31/…/20:19

表 4.4 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，INT 型で定義し，格納した．そのため「7:30」という時刻は「: (コロン)」を取り除いた「730」という数値にとしてデータを作成した．また，時刻表プロパティを作成し，検索した際に該当する時刻表を取得できるようにした．格納した時刻表の数は 911 件である．

4.5.4　バス路線位置エンティティ

バス路線位置エンティティは，乗車部分の路線座標を抜き出すためのエンティティであり，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: バス路線位置エンティティ

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)
:	:
18740102	LINESTRING(35.49446521 133.1760623,……, 35.46417419 133.0631905)

表 4.5 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．

4.5.5　沿線検索範囲ポリゴンデータエンティティ

沿線検索における範囲を，路線を毎回 Buffer メソッドを用いて拡張を行うと処理速度が低下してしまう．そのため，あらかじめ路線を拡張してポリゴンデータを作成しておく．ポリゴンデータの作成には，JTSTestBuilder.bat を用いた．

表 4.6: 沿線検索範囲ポリゴンデータエンティティ

路線 ID STRING 型	沿線検索範囲ポリゴンデータ TEXT 型
10020101	POLYGON((35.43356501990566 133.07846995380473,……))
:	:
13270101	POLYGON((35.448421011579164 133.12684877589163,……))
:	:
18740102	POLYGON((35.45506546005474 133.09193823566295,……))

表 4.6 にあるように，路線 ID は STRING 型で，ポリゴンデータは TEXT 型で登録した．

これらの表をまとめると以下の表 4.7 のようになる．

表 4.7: エンティティ一覧

エンティティ名	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.
沿線検索範囲ポリゴンデータ	沿線検索範囲のポリゴンを格納した表.

4.6　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．

4.6.1　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 4.3: 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.

4.6.2　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 4.4: 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)．

4.7　インデックスの登録

リレーショナルデータベースでの SQL 文（時刻表検索）

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’;

上記のような SQL 文では，時刻に対して範囲検索を行っている．しかし，Bigtable における検索では何も設定しなければ複数の条件と範囲検索を同時に行うことはできない．本研究では，そのような複雑な検索を行うためにコンポジットインデックスを用いた．

4.7.1　インデックスの登録方法

~\workspace\project\war\WEB-INF\lib

以下に xml ファイル「datastore-indexes.xml」がある．

このファイルに

```
<datastore-index kind="bus_org_13_23" ancestor="false">
        <property name="id" direction="asc" />
<property name="weekday" direction="asc" />
        <property name="time1" direction="asc" />
 </datastore-index>
```

と，記述する．カインド名「bus_org_13_23」の「id」,「weekday」,「time1」プロパティに対し昇順のインデックスを定義した．

この定義を平日と休日の時刻 1〜64，合計 128 のエンティティに対して定義した．

4.7.2　インデックスの削除方法

Python で開発している場合は appcfg.py vacuum_ indexes があるためデータストア上のインデックスの削除が行えるが，2013 年 1 月現在 Java 版の SDK にはツールを用いた削除はできない．そのため，インデックスの削除のみ Python を用いた．

1. Python 及び Google App Engine SDK for Python をインストール
2. Eclipse のプロジェクト内にアプリケーションディレクトリを作成し，app.yaml と vacuum_ indexes.bat を作成
3. vacuum_ indexes.bat を実行すると不要なインデックスの削除が行える

- app.yaml
  - application にはアプリケーション ID を記述
  - version は空のアプリをデプロイしてしまうことを避けるため，被らないバージョンにしておく
  - handlers は不要だが必須項目なので，適当な形に設定しておく

[app.yaml]

```
application: ApplicationID
version: 9999
runtime: python
api_version: 1

handlers:
- url: /
  script: dummy
```

- vacuum_ indexes.bat
  - eclipse 上からバッチファイルを実行するとカレントディレクトリが eclipse になってしまうため，以下のような記述でバッチファイルのあるディレクトリに移動

[vacuum_ indexes.bat]

```
@echo off
cd /D %~dp0
appcfg.py vacuum_indexes .
pause
```

以上の設定ができたら eclipse から vacuum_ indexes.bat をダブルクリックすることで，インデックスの削除が可能になる．バッチファイルを実行するといきなりインデックスの削除が行われるのではなく，サーバとの差分のインデックスが順番に列挙され，それぞれ削除するかどうか問われるので，(N/y/a) で答えて削除を行う．y を選択することで不要なインデックスを削除できる．

4.8　バス停，バス路線及び時刻表検索

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．

4.8.1　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．

4.8.1.1　ユークリッド距離を用いた検索

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．

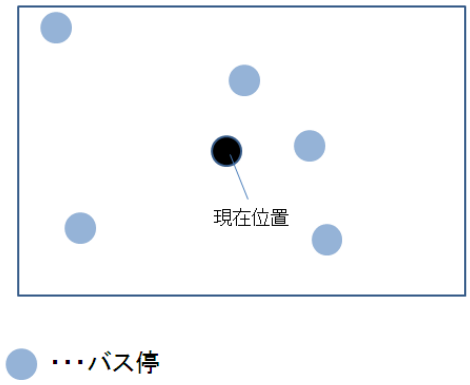

図 4.5: 周辺バス停の検索

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．

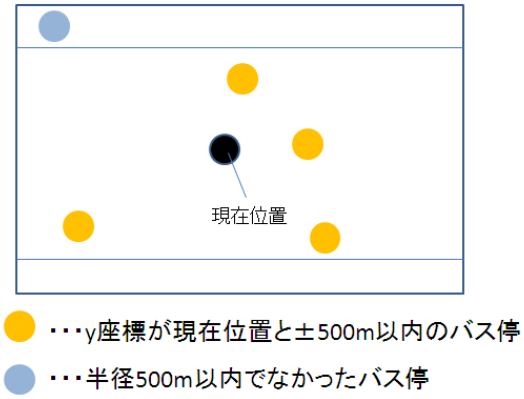

図 4.6: 周辺バス停の検索

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.

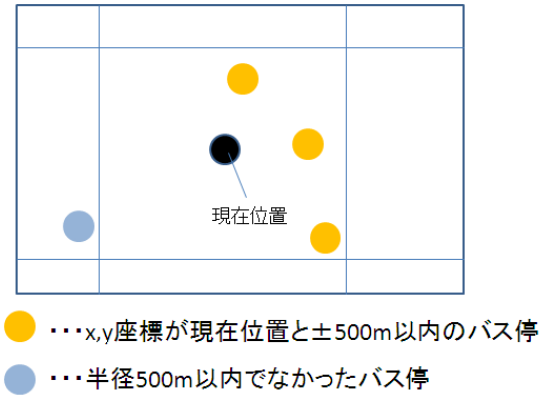

図 4.7: 周辺バス停の検索

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.

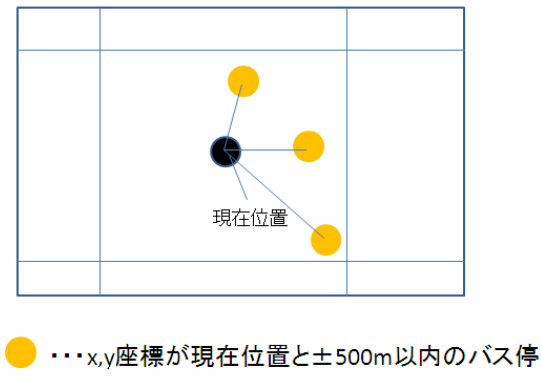

図 4.8: 周辺バス停の検索

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.

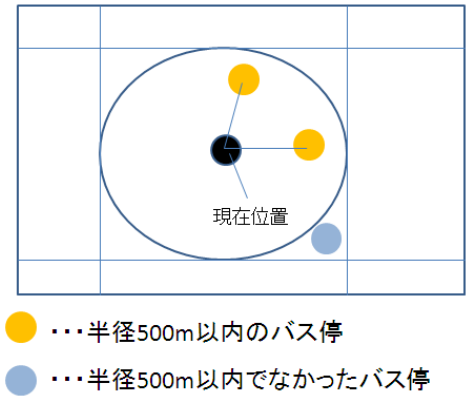

図 4.9: 周辺バス停の検索

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，上記の手順である程度選別を行い，近隣バス停の検索を行う方法を考案した.

4.8.1.2　円形の Buffer と Within メソッドの組み合わせによる検索

1. 出発地点に対して円形の Buffer を発生させ，面を生成する
2. Within メソッドを用いて面の範囲内にあるバス停を検索する

図 4.10: 円形の Buffer と Within メソッドの組み合わせによるバス停検索

4.8.1.3 isWithinDistance メソッドによる検索

1. isWithinDistance メソッドを用いて，出発地点とバス停との距離を計測し，その距離が指定した距離以内かどうかを調べる

図 4.11: isWithinDistance メソッドによるバス停検索

4.8.1.4 distance メソッドによる検索

1. distance メソッドを用いて出発地点とバス停との距離を計測

2. 計測された距離が指定した距離以内かどうかをコード上で判定

図 4.12: distance メソッドによるバス停検索

4.8.1.5　バス停検索方法の比較と検証

ここまでバス停の検索方法について考案した．

これらについて同じ条件で検索を行い，処理速度について計測を行った．縦軸は処理にかかった時間 (単位:ms) を，横軸は松江駅周辺や島根大学周辺など検索のパターンを表している．

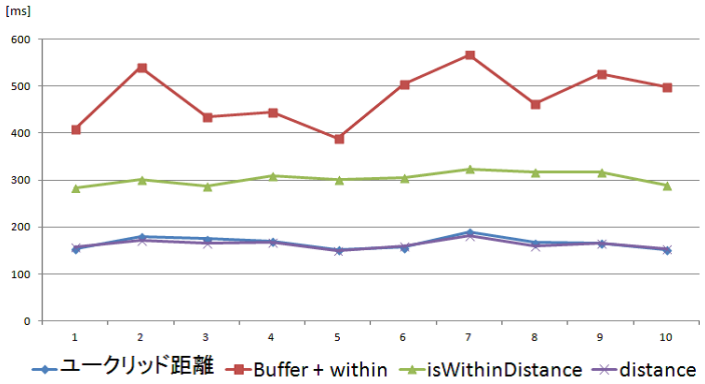

図 4.13: バス停検索処理速度計測結果グラフ

図 4.13 のグラフから，以下のような考察を行った．

1. ユークリッド距離を用いた検索
    - 処理時間は安定している
    - 毎回距離を計算している
2. 円形の Buffer と Within メソッドの組み合わせによる検索
    - Buffer 処理に時間がかかり安定しない
3. isWithinDistance メソッドによる検索
    - 処理時間は安定し高速
4. distance メソッドによる検索
    - 処理時間は安定していて最も高速

以上より，distance メソッドを用いて距離を計測し，コード上で判定する方法でバス停検索を実装した．

4.8.2　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エンティティから検索を行う．バス停位置エンティティから取得したバス停 ID を基にバス路線エンティティから，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 ‘ %10731%11220% ’;
```

※”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．

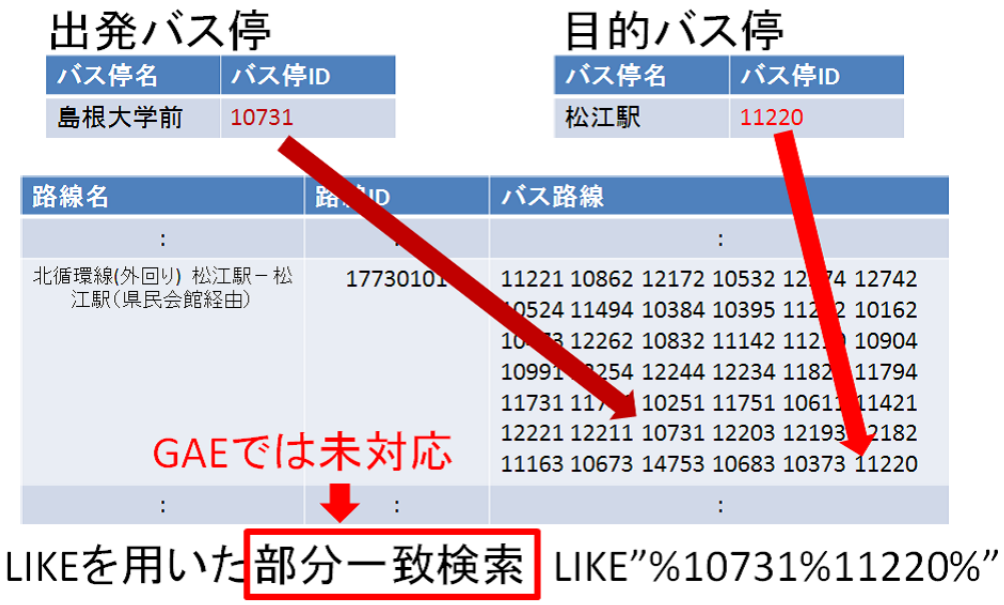

図 4.14: バス路線検索

そのため，全文検索エンジン Lucene を用いた検索方法と，通過するバス停 ID 列を行方向に展開し検索を行うプログラムを考案した．

4.8.2.1　全文検索エンジン Lucene を用いた路線検索

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．その手順は以下のようになっている．

1. バス路線プロパティに対して形態素解析を行い，バスが通過する順番に並んでいるバス ID をひとつの単語として分割する．

図 4.15: 形態素解析

2. 解析したデータを加え，データストアにアップロードをする

com	fts	id	route_name	route_table
松江市営バス	["12254", "10832", "10524", "12742", "11220", "11221", "12172", "10673", "12211", "10991", "11731", "12193", "11751", "10395", "12234", "10384", "10731", "11794", "10611", "12274", "12203", "11421", "10904", "10162", "11210", "11824", "11142", "11...	17750101	北循環線(外回り) 松江駅－松江駅(淞北台経由)	11221 10862 12172 10532 12274 12742 10524 11494 10384 10395 11232 10162 10473 12262 10832 11142 11210 10904 10991 12254 12244 12234 11824 11794 11731 11741 10251 11751 10603 10611 11421 12221 12211 10731 12203 12193 12182 11163 10673 14753 10683 1...
松江市営バス	["12254", "10832", "10524", "12742", "11220", "11221", "12172", "10673", "12211", "10991", "11731", "12193", "11751", "10395", "12234", "10384", "10731", "11794", "10611", "12274", "12203", "11421", "10904", "10162", "11210", "11824", "11142", "11...	17730101	北循環線(外回り) 松江駅－松江駅(県民会館経由)	11221 10862 12172 10532 12274 12742 10524 11494 10384 10395 11232 10162 10473 12262 10832 11142 11210 10904 10991 12254 12244 12234 11824 11794 11731 11741 10251 11751 10611 11421 12221 12211 10731 12203 12193 12182 11163 10673 14753 10683 10373 1...
松江市営バス	["10832", "12741", "11220", "11221", "11752", "10272", "12171", "12212", "10991", "12194", "12233", "10395", "10523", "10674", "11732", "10383", "10732", "10612", "11793", "12273", "10262", "11422", "10041", "12204", "12253", "10904", "10161", "11...	17720101	北循環線(内回り) 松江駅－松江駅(淞北台経由)	11221 10374 10684 14754 10674 11164 12181 12194 12204 10272 10732 10262 12212 12222 11422 10612 10603 11752 10252 11742 11732 11793 11823 12233 12243 12253 10473 12262 10832 11142 11210 10904 10991 10161 11231 10383 10395 11232 10523 12741 12273 1...
松江市営バス	["10832", "12741", "11220", "11221", "11752", "10272", "12171", "12212", "10991", "12194", "12233", "10395", "10523", "10674", "11732", "10383", "10732", "10612", "11793", "12273", "10262", "11422", "10041", "12204", "12253", "10904", "10161", "11...	17700101	北循環線(内回り) 松江駅－松江駅(県民会館経由)	11221 10374 10684 14754 10674 11164 12181 12194 12204 10272 10732 10262 12212 12222 11422 10612 11752 10252 11742 11732 11793 11823 12233 12243 12253 10473 12262 10832 11142 11210 10904 10991 10161 11231 10383 10395 11232 10523 12741 12273 10531 1...

バス会社	検索用バス停ID列	路線ID	路線名	バス路線
STRING	LIST	STRING	STRING	STRING

図 4.16: 解析データをデータストアにアップロード

以上のようにあらかじめデータの登録を行っておく．

次に路線検索について全文検索エンジン Lucene を用いる．出発バス停と目的バス停が fts プロパティに同時に存在するものを検索する．

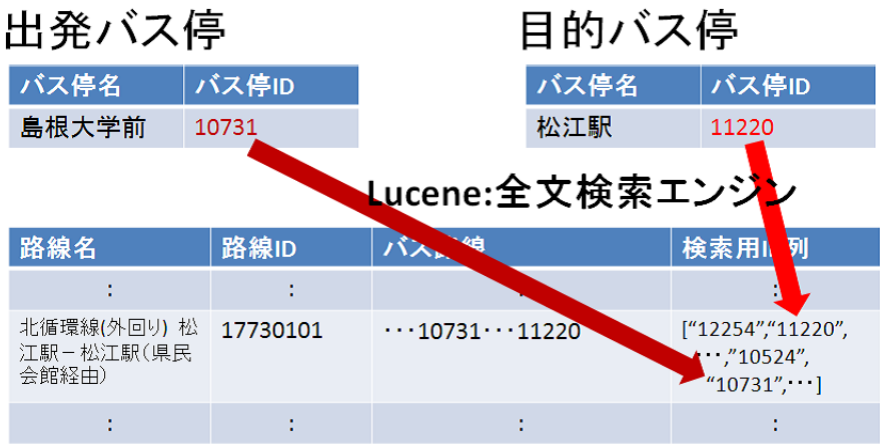

図 4.17: Lucene を用いた路線検索

出発バス停及び目的バス停を同時に含む路線が見つかれば，バスが通過する順番に並べてあるバス路線プロパティを取得する．そして，indexOf メソッドを用いて出発バス停が目的バス停よりも前にあるば該当するバス路線のデータを取得する．

4.8.2.2　行方向に展開した路線検索

行方向に展開した路線検索では，クエリはとても簡単になる．

```
Where 出発バス停='10731' && 目的バス停='11220'
```

上記のように出発バス停，目的バス停プロパティを指定するだけで，条件に合う路線を検索する．

4.8.2.3　路線検索手法の比較と検証

「全文検索エンジン Lucene を用いた路線検索」,「行方向に展開した路線検索」の二つの路線検索方法を考案し，実装した．検索の条件を同様にし，処理時間の検証を行った．

- Lucene を用いた検索　　：　1500ms〜3000ms
- 行方向に展開した検索　：　250ms〜1000ms

行方向に展開した検索の方が，Lucene を用いた検索の場合の 2〜3 割程度の処理時間となった．Lucene を用いた検索では 128 件のデータから検索をしているが，行方向に展開した路線検索では 89613 件のデータから検索を行っている．この結果から，Bigtable では単純なクエリが高速に処理されるということが挙げられる．本システムでは，行方向に展開した路線検索の方法で実装を行った．

4.8.3　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’;
```

路線 ID の指定と平日・休日の指定，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約がある．

「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」

この制約のため，

- 路線 ID の指定
- 平日休日の指定
- 時刻の指定（範囲検索）

上記の条件を同一のクエリで指定できないということになる．この制約を解決するため，データストアに登録したデータにインデックスを作成した．(4.6 インデックスの登録) コンポジットインデックスを検索の条件に必要なプロパティに作成することで，複数のプロパティ値を組み合わせても，時刻の範囲検索が可能となった．

図 4.18: データストアのインデックスの状態

制約の解決をし，実際の検索の例を挙げる．

条件

1. 北循環線外回り（県民会館経由）　ID:17730101
2. 休日に運行している
3. 17:00 以降に発車

この条件をクエリにすると以下のようになる．

```
Where ID=’17730101’ && 休日=’1’ && 時刻 33 > 1700
```

複数の条件と時刻プロパティに対して範囲検索の組み合わせのクエリだが，コンポジットインデックスを作成しているため，このようなクエリを実行できる．クエリを実行すると条件に合うデータを取得する．図 4.19 のように複

図 4.19: 時刻表検索

数の結果が見つかった場合は早く出発するものを取得する．図 4.19 の場合はバス ID プロパティが 11 で，時刻 33 プロパティが 1804 の時刻表を取得する．検索する時刻は指定した時間から，2 時間以内に出発するものを検索する．

時刻表データは表 4.8 のようにデータが登録されている．

表 4.8: バス編成エンティティ

路線 ID STRING 型	備考 STRING 型	バス ID INT 型	休日 STRING 型	時刻 1 INT 型	…	時刻 64 INT 型	時刻表 STRING 型
10020101	平日	1	0	730	…	0	7:30/7:32/…/8:10
:	:	:	:	:	:	:	:
10460101	平日	1	0	1020	…	0	10:20/10:21/…/11:01
10460101	平日	2	0	1120	…	0	11:20/11:21/…/12:01
10460101	平日	3	0	1220	…	0	12:20/12:21/…/13:01
10460101	休日	1	1	1020	…	0	10:20/10:21/…/11:01
:	:	:	:	:	:	:	:
18740102	平日	2	0	1930	…	0	19:30/19:31/…/20:19

条件に合う時刻の取得は,「/」で時刻を区切られた時刻表プロパティを取得し，split メソッドにより「/」を基準に分割し，出発，目的バス停の該当する時刻を取得する．複数の時刻表が見つかった場合は，早く到着する順に 5 件ほど取得する．

4.8.4 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.

4.8.5　南北乗換えありの場合の路線・時刻表検索

乗換えありの場合の検索方法は，まず，乗換えなしのバス停，バス路線検索を行う．ここで，路線が出発バス停から目的バス停への直通のバス路線がない場合に乗換えありの検索を行う．その検索方法 (図 4.20) は，まず出発バス停と乗換え地点の松江駅までの乗換えなしの検索を行い，次に松江駅から目的バス停までの乗換えなしの検索を行うというものである．経由駅として，松江駅，川津，松江しんじ湖温泉駅，県民会館前を検索する．

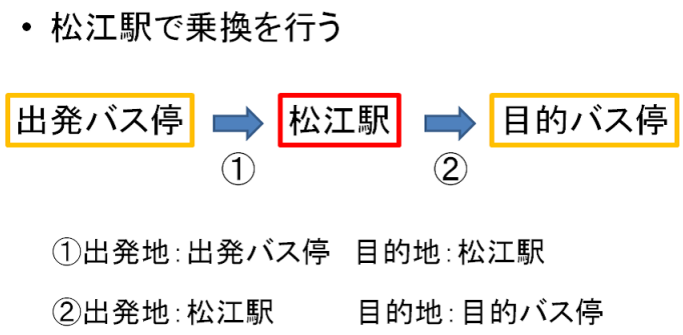

図 4.20: 乗換えありの時刻表検索のイメージ

4.9　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．URL は http://matsue-bus-su.appspot.com/busindex.html

図 4.21: 実行結果 1

図 4.22: 実行結果 2

時刻表の検索結果は早く到着する順番に上位 5 件の路線を表示した．上位 5 件の表示を行うため，ページトップに戻るボタン，リロードするためのボタンを設置し，利便性を図った．

4.10　沿線検索

出発地，目的地からバス停，バス路線，時刻表の検索を実装した．その検索結果を有効に活用するために，路線の近隣の店舗，施設情報の検索機能を実装した．

4.10.1　Google Local Search API

Google Local Search API[12] は，Google マップの結果をウェブサイトや，アプリケーションに埋め込むための Javascript インターフェースを備えている．また得られた結果から Google+[13] のページで，詳しい情報や口コミを閲覧できる．

4.10.2　沿線検索方法

検索結果欄から興味のある検索ワードを入力し，沿線検索チェックボックスをクリックすると，バス路線の結果表示をする地図に沿線検索範囲を緑色のポリゴンで表示し，バス路線近隣の沿線検索結果（初期検索ワードは「観光」）がマーカーで地図上に表示される．検索ワードを基に Google Local Search API を用いて，近隣の施設の検索を行い，条件にあうデータから緯度，経度情報を取得し，あらかじめ登録しておいた検索範囲ポリゴンデータの内部にあるかを,「geometry.poly.containsLocation (LatLng,Poly)」メソッドを用いて判定する．内部にあった場合，マーカーを地図上に描画する．バス路線から 1km 以内にある沿線情報を検索する．出発地，目的地に関係なくバス路線全体の沿線情報の検索を行う．マーカーをクリックすると，詳細な情報が吹き出しで表示され，タイトルをクリックで該当する Google+のページを表示する．

図 4.23: バス路線近隣の沿線情報検索

実行結果は図 4.23 のように表示される．初期検索ワードは観光となっているが，ATM，カフェなど別のワードを入力し，再度チェックボックスを入れると，新たな結果が表示される．同時にバス路線位置表示も行うことで，視覚的に位置関係が分かるようになる．

第5章　システム検証

本研究では，Google App Engine 上にバス停，バス路線時刻表検索システム及びその結果を用いた沿線検索機能を実装した．しかし，結果が表示されるまでに時間がかかるため，検索に要した処理時間を計測し，検証を行った．本システムへのアクセス集中時と非集中時では処理時間が大幅に異なるため別々に検証を行った．

表 5.1: 検索に要した処理時間

	非集中時	集中時
出発バス停検索	2108ms	71ms
目的バス停及び路線検索	1069ms	732ms
出発バス停時刻表検索	2296ms	210ms
目的バス停時刻表検索	172ms	104ms
路線座標検索	88ms	45ms
沿線検索範囲ポリゴンデータ検索	107ms	52ms

表 5.1 より，アクセス集中時，非集中時における処理時間を比較すると，GAE の自動スケーリングによりアクセス集中時のほうが大幅に短い．特に出発バス停検索では，大幅な減少が見られる．また，全体の処理時間も約 7 秒から約 1 秒程度まで減少する．本システムの処理速度が低い主な原因は路線検索にあると考えられる．行方向展開した路線検索を実装したが，膨大なデータから検索を行っているため，処理に時間がかかっている．また，コンポジットインデックスを用いて範囲検索を行っている出発バス停の時刻表検索も時間がかかっている．検証結果としては、路線検索と時刻表検索について改善を行うことで処理速度の向上があると思われる．

第6章　終論

本研究では，最新のバスデータを用いて，地図によるバス停，バス路線，時刻表検索システムを空間情報検索の API である JTS を用いて GAE での実装を行った．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，JTS やコンポジットインデックスを用いることで検索処理時間が大幅に短縮できた．そして新たにバス路線の近隣の沿線情報検索システムを実装し，Google+との連携を取ることでユーザーにとっても有益なシステムになった．

今後の課題としては，利便性や検索における精度，速度の向上，また，本システムを公開して多くのユーザーに評価していただき，その結果を基にシステムを改善していくことが挙げられる．近年，携帯端末の発展，普及により，その高機能な長所を生かすために，外出先でも利用できるよう携帯端末にも対応していくことも必要である．

第7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．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] https://maps.google.co.jp/

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

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

[4] 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

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

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

[7] 日経 SYSTEMS 実践セミナー DVD クラウドコンピューティング設計・開発 Google App Engine 編, 日系 BP

[8] http://www.vividsolutions.com/jts/jtshome.htm

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

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

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

[12] https://developers.google.com/maps/documentation/localsearch/?hl=ja

[13] https://plus.google.com/