

## СОЗДАНИЕ СПЕЦИАЛИЗИРОВАННОЙ ПОИСКОВОЙ СИСТЕМЫ НА БАЗЕ ОБЛАЧНЫХ ТЕХНОЛОГИЙ

Глибовец Андрей, Ситмамбетов Назим

**Абстракт:** В статье сделан обзор основных технологий, которые могут быть использованы при построении поисковой системы с использованием облачных технологий. Исходя из декомпозиции задачи, для каждой подзадачи предоставляется аргументированное обоснование выбора средств и платформы для её решения. Для построения системы выбраны технологии: J2EE, PDFBox, Lucene и платформа облачных вычислений Google App Engine. Базируясь на выбранных технологиях, создан и описан прототип ядра специализированной поисковой системы и развёрнут на базе Google App Engine облака.

**Keywords:** поисковая система, облачные технологии, Google App Engine, Lucene, JSF, прототип поисковой системы.

**ACM Classification Keywords:** D.3.2, D.2.2 Design – search engine

---

### Введение

Системы поиска информации являются неотъемлемой частью современного мира информационных технологий, ведь количество информации растёт непрерывно, а время для нахождения необходимой информации должно оставаться неизменным. Универсальные поисковые системы на данный момент уже не в состоянии удовлетворить потребности всех пользователей. Скорость, удобство и главное релевантность, - те характеристики, которым должна соответствовать специализированная поисковая система. Целью этой работы, является выбор средств и построение ядра специализированной поисковой системы на базе облачных технологий. Наша поисковая система должна предоставлять сервис поиска в коллекции научных публикаций, опубликованных на русском или украинском языке.

Построение поисковой системы можно логически разбить на следующие подзадачи:

- построение crawler-a – программы, которая собирает объекты поиска;
- подготовка объектов;
- индексация объектов;
- реализация поисковых алгоритмов;
- создание пользовательского интерфейса.

---

### Выбор технологий

Для решения поставленной задачи основным языком разработки был выбран язык программирования Java, потому что:

- существует достаточно большое количество библиотек, находящихся в свободном использовании и предназначенных для индексации, поиска, преобразования документов, что позволяет использовать концепцию ООП, использовать библиотеки и расширять их для решения поставленной задачи;

- возникает возможность использовать Java 2 Enterprise Edition (J2EE) для решения задач по созданию серверной платформы поисковой системы, а также разработки пользовательского веб-интерфейса.

Построение так называемого crawler-а было рассмотрено в предыдущей работе [Глибовець, 2010] и поэтому мы не будем дополнительно останавливаться на этой проблеме.

На этапе подготовки объектов все документы должны быть приведены к единому формату. На начальном этапе мы ограничились работой только с форматом pdf, ведь большинство научных публикаций являются документами именно этого типа. Но понятно, что позже система должна быть расширена дополнительными функциями для обработки документов и в других форматах (Word, Xml, HTML, RDF).

Результатом индексации документов системы является реверсивный индекс. Процесс индексации стандартизирован и не зависит от типа и структуры документов, но несмотря на это, в процессе индексации индексатор должен выявлять специализированные метаданные документа: автор, название, список использованной литературы и т.д..

Для подготовки объектов, собранных crawler-ом, было принято решение использовать java библиотеку PDFBox. PDFBox - это open source средство для работы с PDF документами, что дает возможность для создания, чтения и модификации существующих PDF документов.

Проанализировав существующие библиотеки, которые предоставляют функционал индексации и поиска документов (Compass, Oxyus, BDDBot, Egothor, Nutch, Lucene, Zilverline), была выбрана библиотека Lucene [Hatcher, 2004]. Lucene - open source библиотека для высокоскоростного полнотекстового поиска, написанная на Java и поддерживаемая Apache Foundation. Данная библиотека может быть использована для поиска в Интернет.

Основные возможности Lucene:

- масштабируемая и высокоскоростная индексация
  - размер индекса может составлять 95GB
  - требует небольшого размера оперативной памяти - 1MB
  - индекс занимает примерно 20-30% исходного текста
- мощный, точный и эффективный поисковый алгоритм
  - алгоритм ранжирования базируется на векторно-пространственной модели, что позволяет возвращать достаточно четкие результаты за приемлемое время. Существует модуль, позволяющий реализовать вероятностную систему ранжирования – BM25
  - поддерживается большое количество различных типов запросов: фразовые запросы, шаблонные (wildcard) запросы, поиск с учетом интервалов
  - поддерживается зонный поиск, который очень важен для поисковой системы научных материалов (зоны: титул, название, автор, текст)
  - возможность работы с распределенными индексами с последующим объединением результатов
  - возможность одновременного поиска и обновления индекса
- является кроссплатформенным решением
  - исходный код полностью написан на Java
  - существует возможность использования других языков программирования (Python, C++, C#).

Для создания интерфейса (java web приложения) было принято решение использовать Java Server Faces (JSF [Mann, 2004]). JSF - это фреймворк для веб-приложений, написанный на Java. Он служит для того, чтобы облегчать разработку пользовательских интерфейсов для Java EE приложений. В отличие от прочих MVC фреймворков, которые управляются запросами, подход JSF основывается на использовании компонентов. Состояние компонентов пользовательского интерфейса сохраняется, когда пользователь запрашивает новую страницу и затем восстанавливается, если запрос повторяется. Для отображения данных обычно используется JSP, Facelets.

Технология JSF включает:

- Набор API для представления пользовательского интерфейса и управления состоянием, обработкой событий и валидацией данных;
- Специальную библиотеку JSF тегов для представления JSF страницы.

Для создания связей между объектами было принято решение использовать Java EE Context Dependency Injection (CDI). CDI - фреймворк Java контейнера (контейнера сервлетов EJB), который позволяет гибко и элегантно реализовать принцип программирования Inversion of Control. Inversion of control уменьшает связность между объектами. Dependency Injection - один из способов его реализации.

На май 2012 года в Интернет существует большое количество сервисов (Platform as a Service), предоставляющих услуги платформы для облачных вычислений. Наиболее известными являются:

- Amazon Web Services - PaaS & IaaS. Предоставляет сервисы для хранения данных, аренды виртуальных серверов, предоставление вычислительных мощностей
- Windows Azure - PaaS. Предоставляет такие же сервисы, как и Amazon Web Services. Исключительной особенностью является то, что оплата услуг ведется только за использованные ресурсы (данные, процессорное время, сетевые расходы и т.п.).
- Google App Engine (GAE [Davis, 2011]) - PaaS. Предоставляет контейнер сервлетов и такие сервисы как хранение данных, кэширование, поиск, конвертация форматов и т.п. Платформа предоставляет лимитированное количество бесплатных операций (до некоторого момента системой можно пользоваться совершенно бесплатно).

Для решения нашей задачи была выбрана платформа Google App Engine. Основными преимуществами данной системы является:

- возможность бесплатного тестирования решения сразу на платформе;
- поддержка языка Java;
- поддержка J2EE Servlets API.

Среди основных возможностей GAE необходимо отметить также:

- поддержка JRE6;
- поддержка JPA & JDO;
- поддержка JCache;
- поддержка java mail;
- blobstore - сервис хранения больших файлов;
- conversion - сервис работы с различными типами файлов;
- OAuth - сервис авторизации через OpenID.

## Детали реализации

Lucene предоставляет широкий набор средств для создания инвертированного индекса. Lucene является фреймворком а потому допускает расширение функционала через переопределение функциональных единиц.

Расширяя функционал Lucene мы создали свой класс ScientificWorkIndexator (Рис.1.) который отвечает за процесс индексирования.

```
package ukma.sciir.indexation.impl;
import java.io.Reader;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;
import ukma.sciir.indexation.FieldType;
import ukma.sciir.indexation.Indexator;
/* Class, which encapsulates logic of indexing text streams
 * @author nazim.sitmanbetov
 */
public class ScientificWorkIndexator implements Indexator {

    @Override
    public Document createDocument(String docName, Reader contents) {
        Document result = new Document();
        result.add(new Field(FieldType.Contents.name(), contents));
        result.add(new Field(FieldType.Name.name(), docName, Store.YES,
            Index.ANALYZED));
        result.add(new Field(FieldType.Author.name(), "Nazim Sitmanbetov",
            Store.YES, Index.ANALYZED));
        result.add(new Field(FieldType.Description.name(),
            "This is a small description of a great work", Store.YES,
            Index.ANALYZED));
        return result;
    }
}
```

Рис.1. Класс ScientificWorkIndexator

Фамилию автора документа и название документа необходимо получить из исходного текста с помощью алгоритмов распознавания отдельных частей текста. На данный момент мы работаем над этой проблемой. Основная проблема состоит в том, что нет единого стандарта оформления документов. Специализированные зоны (название, автор, аннотация, список литературы) могут находиться в разных частях документа и быть отформатированы различными стилями. Мы работаем над алгоритмом, который базируясь на данных PDF документа (размер и тип шрифта, расположение на странице, расположение в документе) и машинном обучении позволит выделять эти зоны.

Нами была проиндексирована одна тысяча документов в формате PDF, и размер индекса составил 30% от исходного текста, что является довольно не плохим результатом.

GAE поддерживает сохранение данных в двух видах: объекты в базе данных и большими блоками двоичных данных в Blobstore. Соответственно индекс нашей системы может быть размещён только в Blobstore в связи со своим размером. В стандартном наборе классов Lucene для чтения индекса нет механизма для работы с этим специализированным хранилищем. Поэтому был создан адаптер (Рис.2.а Рис.2.б.), реализующий стандартный интерфейс Lucene - IndexReader и работающий с GAE Blobstore.

```

package org.apache.lucene.store;
/** Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.
 * The ASF licenses this file to You under the Apache License, Version 2.0 */
import java.io.IOException;
import com.google.appengine.api.blobstore.BlobInfo;
import com.google.appengine.api.blobstore.BlobInfoFactory;
import com.google.appengine.api.blobstore.BlobKey;
/** A memory-resident {@link IndexInput} implementation like the
 * {@link RAMInputStream}.
 * $Id: GAEIndexInput.java 25 2009-09-14 02:10:51Z thelper $ */
public class GAEIndexInput extends IndexInput implements Cloneable {
    public static final long BUFFER_SIZE = 1015807;
    private byte[] buffer;
    private int bufferPosition = 0;
    private int bufferIndex = 0;
    private BlobInfo blobInfo;
    private GaeCache cache;
    GAEIndexInput(GAEFile f, GaeCache cache) throws IOException {
        super(f.getName());
        this.cache = cache;
        this.blobInfo = new BlobInfoFactory().loadBlobInfo(new BlobKey(f
            .getBlobKey()));
        this.buffer = cache.get(blobInfo, bufferIndex);
    }
    private GAEIndexInput(String resourceName, byte[] buffer,
        int bufferPosition, int bufferIndex, BlobInfo blobInfo, GaeCache cache) {
        super(resourceName);
        this.buffer = buffer;
        this.bufferPosition = bufferPosition;
        this.bufferIndex = bufferIndex;
        this.blobInfo = blobInfo;
        this.cache = cache;
    }
    public void close() { }
    public byte readByte() throws IOException {
        if (bufferPosition >= buffer.length) {
            getNextChunk();
        }
        return buffer[bufferPosition++];
    }
    private void getNextChunk() {
        bufferPosition = 0;
        ++bufferIndex;
        this.buffer = cache.get(blobInfo, bufferIndex);
    }
    @Override
    public void readBytes(byte[] b, int offset, int len) throws IOException {
        while (len > 0) {
            if (bufferPosition >= buffer.length) {
                getNextChunk();
            }
            int remainInBuffer = buffer.length - bufferPosition;
            int bytesToCopy = len < remainInBuffer ? len : remainInBuffer;
            System.arraycopy(buffer, bufferPosition, b, offset, bytesToCopy);
            offset += bytesToCopy;
            len -= bytesToCopy;
            bufferPosition += bytesToCopy;
        }
    }
    public long getFilePointer() {
        return bufferPosition + bufferIndex * BUFFER_SIZE;
    }
}

```

Рис.2.а. Адаптер, реализующий стандартный интерфейс Lucene

```

public void seek(long pos) throws IOException {
    if (pos >= blobInfo.getSize()) {
        throw new IOException("Index '" + pos
            + "' is out of buffer length " + buffer.length);
    }
    bufferIndex = (int) (pos / BUFFER_SIZE);

    bufferPosition = (int) (pos % BUFFER_SIZE);

    this.buffer = cache.get(blobInfo, bufferIndex);
}

@Override
public Object clone() {
    return new GAEIndexInput(blobInfo.getFilename(), buffer,
        bufferPosition, bufferIndex, blobInfo, cache);
}

@Override
public long length() {
    return blobInfo.getSize();
}
}

```

Рис.2.б. Адаптер, реализующий стандартный интерфейс Lucene

Таким образом мы смогли связать алгоритмы Lucene с файловым хранилищем GAE Blobstore.

Для построения интерфейса пользователя было выбрано JSF и основным Java Bean-ом является SearchBeanImpl (Рис.3.а, Рис.3.б.).

```

Package ukma.sciir.web.admin.impl;
import java.io.IOException;
import java.io.StringReader;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.inject.Named;
import javax.jdo.PersistenceManager;
import org.apache.lucene.analysis.CachingTokenFilter;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.search.highlight.Highlighter;
import org.apache.lucene.search.highlight.InvalidTokenOffsetsException;
import org.apache.lucene.search.highlight.QueryScorer;
import org.apache.lucene.search.highlight.SimpleFragmenter;
import org.apache.lucene.search.highlight.SimpleHTMLFormatter;
import org.apache.lucene.util.Version;
import ukma.sciir.indexation.FieldType;
import ukma.sciir.web.SearchBean;
import ukma.sciir.web.SearchResult;
import ukma.sciir.web.SearchResultFactory;
import ukma.sciir.web.admin.IndexBean;
import ukma.sciir.web.admin.IndexListener;
import ukma.sciir.web.db.PMF;

```

Рис.3.а. Класс SearchBeanImpl

```

@ApplicationScoped
@Named("searchBean")
public class SearchBeanImpl implements SearchBean, IndexListener {
    private IndexSearcher indexSearcher;
    @Inject
    private IndexBean indexBean;
    @Inject
    private SearchResultFactory searchResultFactory;
    @PostConstruct
    public void init() {
        indexSearcher = indexBean.getIndexSearcher();
    }
    @Override
    public List<SearchResult> search(String query) {
        if (indexSearcher == null) { return Collections.emptyList();}
        List<SearchResult> searchResults = Collections.emptyList();
        try {
            QueryParser qp = new QueryParser(Version.LUCENE_36,
                FieldType.Contents.name(), new StandardAnalyzer(Version.LUCENE_36));
            Query q = qp.parse(query);
            TopDocs hits = indexSearcher.search(q, 10);
            if (hits.totalHits > 0) {
                searchResults = new LinkedList<SearchResult>();
                PersistenceManager pm = PMF.get().getPersistenceManager();
                try {
                    for (ScoreDoc scoreDoc : hits.scoreDocs) {
                        Document d = indexSearcher.doc(scoreDoc.doc);
                        SearchResult sr = searchResultFactory.getSearchResult(d,pm);
                        searchResults.add(sr);
                    }
                } finally {
                    pm.close();
                }
            }
        } catch (IOException e) {
        } catch (ParseException e) {
        }
        return searchResults;
    }
    boolean isIndexLoaded() { return indexSearcher != null; }
    @Override
    public void indexUploaded() { indexSearcher = indexBean.getIndexSearcher();}
    @Override
    public void indexDeleted() { indexSearcher = null; }
    private static String highlightField(Query query, String fieldName,
        String text) throws IOException {
        CachingTokenFilter tokenStream = new CachingTokenFilter(
            new StandardAnalyzer(Version.LUCENE_36).tokenStream(fieldName,
                new StringReader(text)));
        SimpleHTMLFormatter formatter = new SimpleHTMLFormatter();
        Highlighter highlighter = new Highlighter(formatter, new QueryScorer(
            query));
        highlighter.setTextFragmenter(new SimpleFragmenter(Integer.MAX_VALUE));
        tokenStream.reset();
        String rv = "";
        try {
            rv = highlighter.getBestFragments(tokenStream, text, 1, "(FIELD TEXT TRUNCATED)");
        } catch (InvalidTokenOffsetsException e) {}
        return rv.length() == 0 ? text : rv;
    }
}

```

Рис.3.6. Класс SearchBeanImpl

SearchBeanImpl является «слушателем» индекса, и после загрузки индекса изменяет свое состояние и начинает работу. Ключевым методом является List <SearchResult> search (String query), который собственно и осуществляет поиск в индексе. После выполнения, результаты отсылаются в HttpServletResponse и отображаются пользователю (рис. 4).

## ІНТЕЛЕКТ

[136274.pdf](#)

Author: Nazim Sitmanbetov

Description: This is a small description of a great work

[143894.pdf](#)

Author: Nazim Sitmanbetov

Description: This is a small description of a great work

[143861.pdf](#)

Author: Nazim Sitmanbetov

Description: This is a small description of a great work

[144201.pdf](#)

Author: Nazim Sitmanbetov

Description: This is a small description of a great work

Рис. 4. Интерфейс пользователя

Среднее время выполнения поискового запроса составляет - 700 мс. Этот результат не является лучшим, и существуют методы, позволяющие ускорить работу сервера, но на данный момент бесплатное использование платформы GAE накладывает определенные ограничения.

Прототип системы можно найти по адресу - <http://index-4.nazloid.appspot.com/user/search.jsf>

На данный момент ведется работа над:

- распознаванием специализированных зон (название, автор, аннотация, список использованной литературы);
- улучшением алгоритма ранжирования, включением весов специализированных зон и применением веса публикации;
- улучшением интерфейса.

## Выводы

Предоставление качественного специализированного поиска - сложная архитектурная и алгоритмическая задача. Предоставление привычного интерфейса, выдача релевантных результатов, быстрые методы индексирования и анализа документов, ограниченность ресурсов – все это очень усложняет задачу построения специализированной поисковой системы. Развертывание в облаке позволяет преодолеть ограниченность ресурсов, использование Lucene предоставляет быстрые методы индексирования и поиска, JSF позволяет предоставить адекватный интерфейс. Сочетание указанных технологий упрощает разработку и позволяет программисту сосредоточиться непосредственно на бизнес логике. Созданный



прототип ядра поисковой системы позволяет относительно легко расширяться и усовершенствоваться, что и будет сделано в ближайшее время.

---

### Благодарности

---

Работа опубликована при финансовой поддержке проекта **ITHEA XXI** Института информационных теорий и приложений FOI ITHEA Болгария [www.ithea.org](http://www.ithea.org) и Ассоциации создателей и пользователей интеллектуальных систем ADUIS Украина [www.aduis.com.ua](http://www.aduis.com.ua).

---

### Библиография

---

[Глибовець, 2010] Глибовець А.М., Шабінський А.С., Ольшевський Р.Я. Побудова пошукового робота українськомовних наукових матеріалів. Наукові праці МДУ ім. Петра Могили. Комп'ютерні технології. — Випуск 130. — том 143. — 2010. — с. 81-87.

[Hatcher, 2004] E. Hatcher, O.Gospodnetic. Lucene in Action – Manning, 2004 – 456 p.

[Mann, 2004] Kito D.Mann. Java Server Faces in Action - Manning, 2004 – 744 p.

[Davis, 2011] G.Davis, N.Johnson. Google App Engine in Action - Manning, 2011 – 578 p.

---

### Информация о авторах

---



**Глибовец Андрей** – кандидат физ.-мат. наук, доцент кафедры сетевых технологий Факультет информатики Национальный университет «Киево-Могилянська Академія» 04655, Украина, Киев, ул. Сковороды; e-mail: [andriy@glybovets.com.ua](mailto:andriy@glybovets.com.ua)

Основные области научных исследований: разработка интеллектуальной поисковой системы, построение онтологий, разработка систем искусственного интеллекта



**Ситмамбетов Назим** – Java разработчик в Intro Pro, студент магистерской программы «Информационные управляющие системы» факультета информатики национального университета «Киево-Могилянська Академія»; e-mail: [nasitmanbetov@gmail.com](mailto:nasitmanbetov@gmail.com)

Основные области научных исследований: реализация сложных распределённых вычислений, облачные технологии, информационный поиск