Cross-Lingual Information Access from Marathi to English

CLIA IIT Bombay Group
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai, India

Abstract:- ***Cross lingual Information Access is the area which combines various NLP and IR techniques to retrieve documents from a language other than the query language. In a country like India where more than 22 languages are spoken in its different parts, the language barrier problem becomes a crucial problem to be solved. This Project is a mission mode project executed by a consortium of academic and research institutions and industry partners. The system developed is called' Sandhan'. Sandhan supports six Indian languages. Currently it is supporting only tourism domain search. Given a query in Marathi (or other languages that are supported) system gives relevant documents in Marathi, Hindi and English. Along with the URLs for these documents system also provides snippet and summary of these documents in Marathi, Hindi and English.***

I. Introduction

Though initially the World Wide Web was dominated by English, now less than half of existing web pages are in English. This makes it possible to directly access previously unimagined sources of information. However in conventional information retrieval systems the user must enter a search query in the language of the documents in order to retrieve it. For using this, the user should be able to formulate queries in all possible languages and can understand documents returned by the retrieval process. This restricts the user from accessing the information available.

Cross-language information retrieval enables users to enter queries in language they are comfortable in, and uses language translation methods to retrieve documents originally written in other languages. Users who are unfamiliar with the language of documents retrieved are often unable to understand information from these documents. Additional post retrieval processing is done in order to enable user to make sense of these retrieved documents. This additional processing may take the form of machine translation of snippets, summarization of documents and subsequent translation of summaries and/or entire document. We have developed a cross lingual information retrieval system called ***Sandhan.*** The system supports Marathi, Bengali, Hindi, Tamil, Telugu, Punjabi and English.

II. Motivation

There is lot of information available on internet in different languages mainly in English. Due to language barrier most of the information is not accessible to the common people of India. They are unable to form a query in English or in language other than their native language. The main motivation of developing this system is to allow people to access the information available in English and Hindi by forming a query in Marathi or any other Indian language they are comfortable in.

III. Architecture

Sandhan enables a user to enter a query in a language in which he/she is fluent, and access documents from the web, that are written in same or other language. The Sandhan process is

mainly divided into two sub-processes: offline process and online process. The block diagram of Sandhan architecture is as shown in Figure 1

CLIA system is developed on top of Nutch Architecture. So we will first look at the framework of Lucene and Nutch.

A. Nutch and Lucene Framework

Nutch [5] is an open source apache project for searching the web. The search engine enables crawling the web to get the collection of the web pages that acts as a database of documents for searching. Nutch is designed to crawl and search the documents in web. Nutch is written in Java. The code is written as a plug-in framework so it is easy to add and test a module. There are different plug-in to modularize the functionalities of Nutch and also helps the developers in enhancing the features. The usage of Nutch brings more flexibility to the developers since complete code is available free. Nutch uses Lucene architecture to do field based searching and scoring.

Lucene [4] is an open source apache project specifically for searching a set of documents based on the fields and scoring them and presenting it to the user with high degree of relevance. Further Nutch has the flexibility to run in a single system or cluster of systems. Hadoop system is used to achieve distributed nature of crawling. Hadoop is also an open source apache project to implement Map-Reduce architecture.

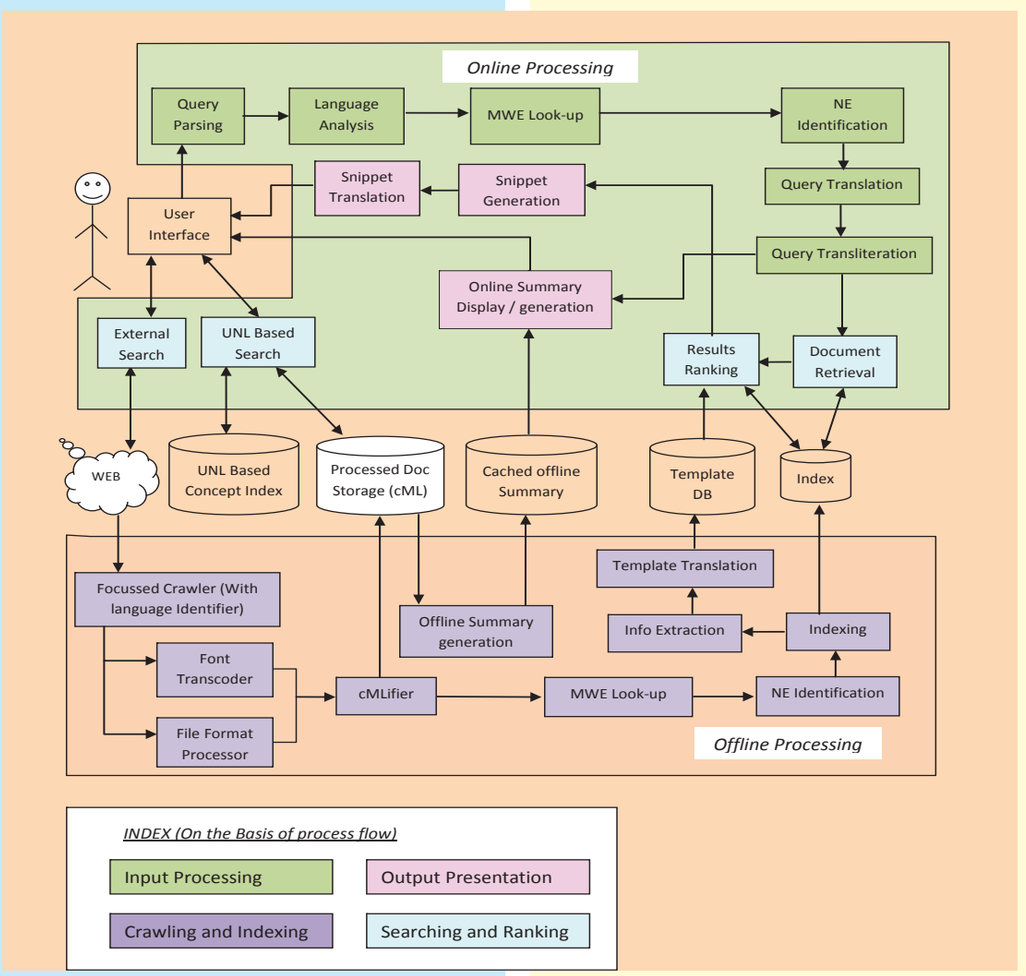

Figure 1 : Architechture of Sandhan

1) ***Nutch Architecture*** [5]

Overall, the web is considered as a directed graph with nodes being the documents and hyper-links being the edges. This helps in the representation of the collection of documents as link graph and for computational feasibility link graph is stored as adjacency lists. For every hyper-link from document A to document B indicates a link from node A to node B in the link graph. These links are used for crawling. To start with, some URLs are given as seed URLs. As shown in the Figure 2, the webDB indicates the collection of web pages being crawled in iteration. Initially webDB will be empty and Fetch Lists contains the seed URLs. Each of the iteration is a process of fetching the content for all the URLs listed in the Fetch List. These fetched documents are analyzed and processed to find the updates that need to be stored in webDB. Due to these new updates the Fetch List gets populated with more URLs and the process continues. The fetched content is used to index the terms present in the documents. These indices are used further by the searcher to map the query terms with the index terms. The searcher module can be called through any of the Web Server engines. This gives the overview of the Nutch architecture and also shows the co-ordination between crawling and searching modules.

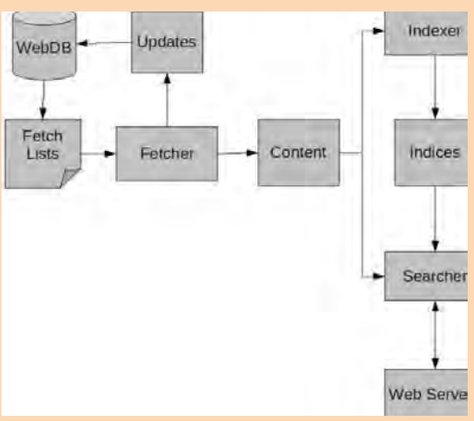

Figure 2 Architecture of the Nutch system

Now we will see in detail crawling and searching process in Nutch

Crawling

Figure 3 represents the crawling process used in Nutch. There are several components/modules involved in crawling process. Nutch provides the platform for crawling with URL filters, parsing filters, normalizers etc. as shown in the figure two different databases are used by Nutch to store the crawled data. CrawlDB contains the information about all the pages accessed by Nutch whether it is fetched or not, and if so when during crawling and linkDB contains the information about the link graph which includes the URL as well as anchor text information. The workings of different modules in the crawling process are described below:

Injector: This module is used to put the initial list of URLs in the crawlDB. There are two ways of doing this. First, Injector can directly add the list of URLs from Dmoz database. Dmoz is an open source database maintained by Mozilla for multilingual content. Second, a seed list containing the list of URLs can be used.

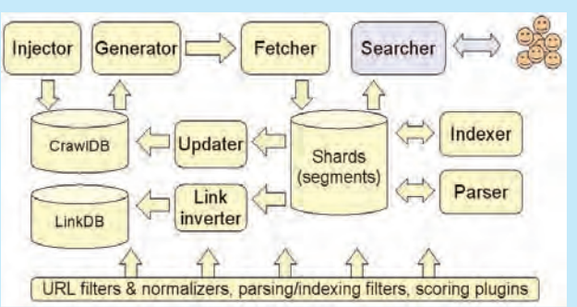

Figure 3 Crawling process in Nutch

Generator: Generates the fetch list containing the information about the pages that need to be fetched.

Fetcher: Fetches the pages for the URLs indicated in the fetch list.

Updater: Information about the newly fetched pages is updated to crawlDB.

Shards/Segments: There are set of segments during crawling. Each segment is the collection of URLs grouped together as a unit. Segments are organized as a directory structure with a segment being the directory and information about the URLs in that segment as sub-directories. These sub-directories include contents, parsed information, index etc.

Parser: Different parsers for different file formats are used to parse the content and stored in segments.

Link Inverter: All the links in the link graph is inverted so as to index the incoming anchor text information.

Indexer: Every segment is indexed with Lucene by marking the list of documents with position of occurrence for every term. The step by step process of crawling is described below:

Step 1 Injector injects the list of seed URLs in to the crawlDB.

Step 2 Generator takes the list of seed URLs from crawlDB and forms fetch list and also adds crawl_generate folder into the segments.

Step 3 These fetch lists are used by fetchers to fetch the raw content of the documents and stores in segments.

Step 4 Depending on the format of the document, corresponding parser is called to parse the contents of the document and stores the parsed content in segments.

Step 5 The links are inverted in the link graph and stored in LinkDB.

Step 6 Indexing the terms present in segments is done and indices are updated in the segments.

Step 7 Information on the newly fetched documents is updated in the crawlDB.

Searching

The process of searching involves query and also process of finding the document relevance to the query. Any web server engine can be used to give a query for the search. The query is then processed before searching. Query analyzers are used to process the query and then given to Lucene engine for searching. Lucene in combination with Nutch uses a ranking algorithm as in equation below to find the document relevance for the query [4].

$$score = queryNorm(d) * coord(q,d) \sum_{i \text{ in } q} tf(t \text{ in } d) * idf(t) * t.boost(t.field \text{ in } d) * Norm(t.field \text{ in } d)$$

Where,

tf(t in d) correlates to the term's frequency, defined as the number of times term *t* appears in the currently scored document d. Documents that have more occurrences of a given term receive a higher score.

idf(t) stands for Inverse Document Frequency. This value correlates to the inverse of *docFreq* (the number of documents in which the term *t* appears). This means rarer terms give higher contribution to the total score.

boost (*t*.field in *d*) is a search time boost of term *t* in the query *q* as specified in the query text, or as set by application calls to *setBoost()*

norm(t,d) encapsulates a few (indexing time) boost and length factors:

1. Document boost - set by calling *doc. setBoost()* before adding the document to the index. It sets document boost equal to OPIC score of that document.

2. Field boost - set by calling *field.setBoost()* before adding the field to a document. While query formulation, we set this boost values to match our ranking needs.

lengthNorm(field) - computed when the document is added to the index in accordance with the number of tokens of this field in the document.

2) Lucene Architecture [4]

The searching module in the Nutch architecture requires processing of the query given by the user and also giving weights to the documents based on the relevance. Lucene is framework for ranking the set of documents based on the query and displaying them accordingly. Lucene uses OPIC (Online Page Importance Calculation) score given during crawling to rank the documents.

This OPIC score acts as a boost factor which is a weight indicating the importance of the term in the given document during ranking. There are four modules that Lucene controls: Indexing, Searching, Scoring and Analysis.

Indexing

Inverted indexing is used to represent the set of terms from the collection of documents. Lucene sees the index to be the sequence of documents called directory, a document as sequence of fields, a field as sequence of terms and a term as text string. Lucene also stores the term frequencies and position of its occurrence in the documents for scoring purposes.

Searching

For searching purpose, Lucene provides different query types for different kinds of searching. TermQuery helps in querying with a key value, RangeQuery for querying through text with numeric range of values; BooleanQuery is used for combining multiple queries as expressions. Apart from these basic query types Lucene support several other query types such as wildcard queries and fuzzy queries.

Scoring

This is the process of measuring the document's relevance to the query. Several factors are used to score in Lucene. Term Frequency (*tf*) – Number of times the term occurs in the document. Inverse Document Frequency (*idf*) – Number of documents having the term in index. Boost factor (*boost*) – Weight resembling the importance given to the particular field. *coord* – Score factor based on number of query terms in the document. *queryNorm* – Normalization value for the query weights.

Analysis

Lucene has many in-built analyzers that are used during query processing and indexing. Analyzer is used to tokenize the terms in the query or document. *WhitespaceAnalyzer* - Tokenizes based on the whitespace. *SimpleAnalyzer* - Converts to lowercase and splits at non-letter boundaries. *Stop-Analyzer* - Converts to lowercase and removes stop words. *SnowballAnalyzer* - Converts to lowercase and applies stemming.

B. Architecture of Sandhan

As mentioned earlier the architecture of Sandhan is divided into modules and sub-modules closely organized to serve the query requests. The system has two major modules: offline processing and online processing.

1) Offline Processing

The purpose of offline processing is to crawl the web, and gather content in English and the six Indian Languages. The content is downloaded and stored as multi-lingual corpora and index table. The offline module is divided into following sub-modules:

- Crawler
  - Generator
  - Injector
  - Fetcher
  - Parser
  - Link Inverter
  - Domain identifier

- Document Processor
  - Document converter
  - IE Extraction Engine
  - CMLifier
- Indexer
  - Segment Reader
  - Language specific Analyzer
  - Index Writer
  - Index merger
  - Remove duplicates (from the index tables)

Crawler

The CLIA crawler tool is based on the Nutch crawl tool, and a family of related tools to build and maintain several types of *data structures*. The data structures built are *crawlDB, segments* and *linkDB*. The crawl database, or *crawlDB*, is a data structure for representing the structure and properties of the web graph being crawled. A *segment* is a collection of pages fetched and indexed by the crawler in a single run. *LinkDB* is a collection of inverted links from each page. The function of the Crawler is to

- Inject few seed URLs into its database.
- Generate number of fetch lists and put each of them in a new directory called segment.
- Fetch the pages corresponding to URLs in the fetch lists and put them in segments.
- Parse the fetched content in the segments.
- Update crawlDB with web pages, and new URLs and meta-data from parsed segments.
- Create a linkDB from parsed segments, associating incoming anchor text with corresponding URLs (Link inverter).

The generate-fetch-update cycle is designed to repeat until the required depth is reached, maintaining an up-to-date image of the Web graph.

Domain Identifier:

In Sandhan system a domain identifier is developed, that looks for pages that belong to a domain. Only the pages belonging to a certain domain are indexed.

Document Processor

This module consists of many language processing tools which are used in processing the documents. The tools are described briefly below.

Document converter: This converts the documents from different formats into Text format.

IE Extraction Engine: Information Extraction is a process, which takes unseen texts as input and produces fixed-format, unambiguous data as output. It runs as a background process that extracts the pertinent information from text into predefined templates. The templates are domain specific.

Translation Engine: This engine gives the target language equivalence to IE template and transforms it into a grammatical sentence in the target language. The target languages are Bengali, Marathi, Punjabi, Tamil and Telugu.

CML-ification: CLIA Markup Language (CML) is developed to store and transport document across the modules. CML is similar to Extensible Markup Language (XML). Once the crawling phase has been completed, and all the necessary information from the web pages has been stored, the CMLifier module looks up the segments and the linkDB to generate CML format from all the documents. This process is invoked one time after crawling and is an offline process.

Indexer

The *index* is the inverted index of all of the pages the system has retrieved, and is created by merging all of the individual segment indexes. The ordinary index would contain for each document, the index terms within it. But the inverted index stores for each term the list of documents where they appear. The benefit of using an inverted index comes from the fact that in IR we are interested in finding the documents that contain the index terms in the query. So, if we have an inverted index, we do not have to scan through all the documents in collection in search of the term. Often a hash-table is associated with the inverted index so that searching happens in O(1) time. Inverted index may contain additional information like how many times the term appears in the document, the offset of the term within the document. Along with this information, field information corresponding to each term is stored. That is, to which field the term belongs to and accordingly some weightage can be given to a term while ranking. The fields are as follows.

1. MWE
2. anchor
3. boost
4. content
5. digest
6. host
7. language
8. segment
9. site
10. title
11. url

The function of the Indexer is to

- Index each parsed segment and keep it in index segment.
- Remove duplicates (pages at different URLs with the same content) from the set of index segments.
- Merge all index segments into a single segment index for searching.

Segment Reader: Segment reader reads the segment created during indexing.

Language Specific Analyzer: Before indexing we need to we need to analyze the document for tokenization, stop word removal, stemming, POS tagging, etc. End results of this analysis are term to be indexed. We will discuss each of these processes in some detail for Marathi.

1. Tokenization
2. Morphological analyzer
3. POS tagging and Chunking
4. Stop word removal
5. Stemming
6. Name entity recognition
7. Multi-word expression recognition

1. Tokenization: The main function of this sub module is to identify and separate the tokens present in the query, in such a way that every individual word as well as every punctuation mark will be a different token. The sub module considers abbreviations, acronyms, numbers with decimals, or dates in numerical format, in order not to separate the dot, the comma or the slash (respectively) from the preceding and/or following elements.

2. Morphological analyzer: The morphological analyzer identifies the root and the grammatical features of a given word. The grammatical feature includes category, gender, number, person, case, etc. The morphological analyzer is done in two steps first is segmentation and second is parsing. The paradigms and lexicon are used for segmentation purpose. The Paradigm-based approaches for Morphological Analyzers have good success rates for Indian languages [2].

Example:

मुलगा (Boy)	Root – मुलगा Category – Noun Gender – Male Number - Singular Person – 3rd , 1st Case – Direct

3. POS Tagger and Chunker: Part of Speech (POS) tagging is the process of assigning a part of speech tag to words in the sentence. Identification of the parts of speech such as nouns, verbs, adjectives, adverbs for each word of the sentence helps in analyzing the role of each word in a sentence. The POS tagger has been developed by CRF-based approach. [2]

Example:

मी	Pronoun
मुलगा	Noun
आहे	Verb

Chunking is the task of identifying and segmenting the text into syntactically correlated word groups. Chunking identifies simple noun phrases, verb groups, adjectival phrases, and adverbial phrases in a sentence. This involves identifying the boundary of chunks and the label.

Example:

जेवत आहे {jevat aahe} {eating}	Verb Chunk (VGF)
औद्योगिक प्रशिक्षण संस्था {aaudyogic prashikshan sanstha}{institute of technology}	Noun chunk (NNP)

It is natural to view each of these sub-tasks as a sequence labeling task of assigning POS/ Chunk labels to a given word sequence. Indian languages do not enjoy the resource abundance of English; their linguistic richness can be used to offset this resource deficit. Specifically, in such languages, the suffixes carry a lot of information about the category of a word which can be harnessed for shallow parsing [6].

4. Stop word removal: Stop word lists for Marathi are collected using corpus. This list was initially obtained by selecting terms with high document frequency as potential stop words. This list was then manually cleaned to obtain the final list of stop words, which is used to remove stop words from the document and query.

5. Stemming: Stemming is used to reduce variant word forms to common roots and thereby improve the ability of the system to match query and document vocabulary. The variety in word forms comes from both inflectional and derivational morphology.

Inflectional morphology: Inflection is the combination of a word stem with a grammatical morpheme, usually resulting in a word of the same class as the original stem, and usually filling some syntactic function like agreement [3]. This results into change of features like gender, number, person, case, tense, mood, aspect, *etc.* In example: घर (Home) – घराना (to the homes) (Inflection to घर as घरां + suffix ना) and in दार (door) - दारे (the doors) (inflection to the word दार gives plural form).

Derivational morphology: Derivation is the process of combination of a stem with a derivational morpheme to form a new word generally belonging to a different grammatical category. The derived words may have a category different from the stem using which they are derived. In case of घरासमोरच्याने {gharaasamorchyaane} {one who in front of the home}, घर{ghar} {a home} is a noun, which when affixed with the postposition 'समोर' becomes घरासमोर {gharaasamor}{in front of home} is an adverb of place, which is then affixed with 'चा' becomes घरासमोरचा {gharaasamorchaa}{one

who lives in front of the home}, an adjective. This adjective can also be used as a noun and as a noun it has taken the case marker 'ne' and becomes घरासमोरच्याने {gharaasamorchyaane} {one who lives in front of the home}.

6. Name Entity Recognition (NER): Entities which refer to particular objects such as places, persons, organizations, time etc. These entities should explicitly be the name of the referring object. These entities are given some tag which describes the set of classes to which named entities are associated. For example, Person, organization, place, quantity, dates, etc. Conditional Random Fields (CRFs), a machine learning technique an undirected graphical model where the conditional probabilities of the output are maximized for a given input sequence is used to identify name entities.

Example: औद्योगिक प्रशिक्षण संस्था is a Name Entity (Organization)

7. Multi-word expression (MWE) recognition: Generally queries need to be understood as multi word expressions rather than token of words. A multiword expression (MWE) is a lexeme made up of a sequence of two or more lexemes that has properties that are not predictable from the properties of the individual lexemes or their normal mode of combination. A multiword expression can also be a fragment of a sentence, or a complete sentence. The group of lexemes make up a MWE can be continuous or discontinuous.

Fetch, parse, invert links, index and deleting duplicates may be distributed across multiple machines and run in parallel. Inject, generate and updated require single process exclusive access to the crawlDB. Merge requires that a single process have exclusive access to all segments data.

2) Online Processing

The purpose of online processing is to process the query that is in one of the six Indian languages supported by Sandhan. The query would be processed and a relevant documents list would be generated to present it to the user. For each of the document retrieved, snippet generation and translation is done. Index table generated in offline processing module is used during searching. Online Processing consists of the following modules:

1. Input Processing Subsystem
2. Ranking
3. Output Processing

1. Input Processing Subsystem

The Sandhan Input Processing subsystem consists of following modules and sub-modules.

- Language Analyzer
  - Tokenization
  - NER
  - MWE
  - Stop Word Identification
  - Stemmer
- Query Translation And Query Transliteration
  - IL-IL Dictionaries
  - IL-English Dictionaries
  - CRF++

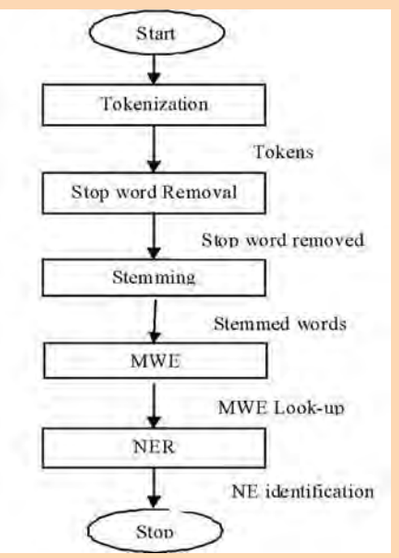

Figure 4: Input Processing

Input processing takes in a query given by a user through the user interface and analyses using language processing tools, expands the query to add more relevant terms and based on its analysis either translates or transliterates all the query terms to target languages and then provides this as input to the search modules. It consists of many modules which are used in retrieving the documents. The modules used are described briefly below.

Language Analyzer

This performs all source language analysis tasks on the given query input. Typical language analysis tasks for an IR engine involves tokenization, stemming, stop word elimination and in some cases some shallow processing for proper name identification. All these processes are same as that done during offline processing. The only difference is that during online processing we do list lookup to identify multiword expressions as we want quick response. For this we maintain a list of multiword identified during the offline process.

Query Translation & Query Transliteration

Query Translation

The pre-processed query has to be translated or transliterated for cross language information access. Query is processed in two different ways. If the query terms can be translated then it is passed to translation module otherwise it is transliterated. We will see the methods adopted for each of the process.

Query Translation approach for CLIR is among the most widely used approach. Accuracy wise, Document translation approach performs better than the query translation approach, but because of practical limitation it is not being used [1].

In query translation, the source language query is being translated in the target language. For the task resources like bi-lingual dictionary, parallel corpora are used. Some of the benefits are as follow:

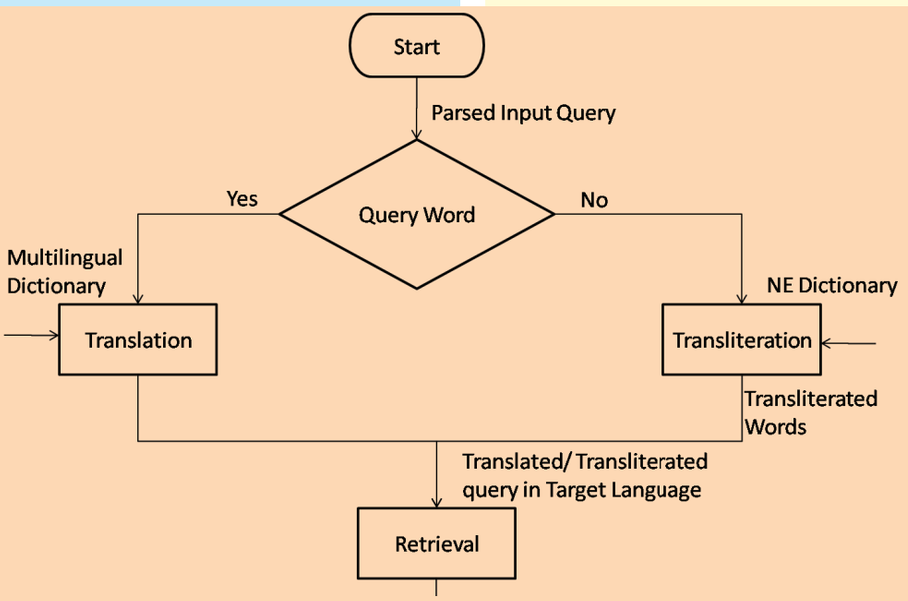

Figure 5 : Flow for Query Translation & Transliteration

1. In general, queries are shorter than document, so number of terms need to translate the query is smaller, which is practically possible at the run time.

2. Once we have a Translated query, any monolingual search engine can be used for retrieving documents.

For query translation a bilingual dictionary is used. Bilingual dictionary lists the meanings of words of one language in another. For example a Marathi to English bilingual dictionary will contain a list of Marathi words and their corresponding English translations. A query is tokenized and each term is considered for translation. If the term is present in the bilingual dictionary then corresponding English translations (in case of Marathi to English translation) are taken. Now the question is, if there are more than one translation available then which one should be considered. For this we take each of these translated terms and search for the relevant documents. The term for which we get maximum relevant documents is considered as most appropriate translation for the Marathi term. The terms which are not present in the bilingual dictionary we transliterate them assuming that dictionary contains all the Marathi words and the words which are not present may be name entities or English or Hindi words for which there is no corresponding Marathi word.

Query Transliteration

Transliteration task is considered to be non-trivial as its output decides the quality of the results retrieved. The terms which cannot be translated are transliterated. We have maintained a list of Marathi name entities and their corresponding English and Hindi transliteration. But the list is not exhaustive so we need to transliterate the terms which are not in this list. The flow is explained in the Figure 5.

Transliteration module is built on a grapheme based model in which transliteration equivalents were identified by mapping the source language names to their equivalents in a target language index, instead of generating them. CRF++ was used for training and testing by generating the transliterations models from the aligned and mapped characters from parallel corpus of Marathi data using GIZA++. The GIZA++ model is then used to get the transliteration of the query term.

2. Ranking

Sandhan ranking is basically Field based Nutch ranking with some modification to the field weights. The modification includes taking care of the query formulation and reducing the adverse effect of OPIC score. Query formulation is very important to get required precision and recall. For web search engines usually precision is more important than recall as the total number of relevant documents is too huge to be considered by the user. While designing different strategies for query formulation, we keep this in mind. We formulate the query such that while ranking it uses a fall back strategy to handle precision and recall tradeoff.

While translating the query, the pruning of translation candidates is important to increase precision@k. We devised a strategy for the pruning. We also designed a weighing scheme for the phrasal queries that boosts the documents that contain all the query terms.

We noticed that Nutch by default gives too much importance to the OPIC score while ranking. This was affecting the retrieval performance adversely. Hence we put a cap on the OPIC score while indexing to reduce the adverse effect.

We will describe the Nutch ranking scheme. Nutch uses Lucene ranking to rank the retrieved results for a query. Lucene uses two IR model to retrieve results for a query and rank them in descending order. Given a query, Lucene first

apply Boolean model of Information retrieval on the query and initial set of results is retrieved. On this initial set, it applies Vector space model of Information retrieval to rank them against the given query.

The score of a document in a set of initial query result, "score(q,d)" is sum of the score for each term of the query ("t in q"). A term score in a document is the sum of the term run against each field that are present in the document (remember a document is a collection of different fields e.g. URL, title, content, etc.). For a field and a term score is calculated as given in below equation.

$$score = queryNorm(d) * coord(q,d) \sum_{i\ in\ q} tf(t\ in\ d) * idf(t) * t.boost(t.field\ in\ d) * Norm(t.field\ in\ d)$$

Where,

tf(t in d) correlates to the term's frequency, defined as the number of times term t appears in the currently scored document d. Documents that have more occurrences of a given term receive a higher score. The default computation for *tf (t in d)* is:

$$tf(t\ in\ d) = \sqrt[2]{frequncey}$$

idf(t) stands for Inverse Document Frequency. This value correlates to the inverse of docFreq (the number of documents in which the term t appears). This means rarer terms give higher contribution to the total score. The default computation for idf(t) is:

$$idf(t) = 1 + \log \frac{NumDocs}{docFreq + 1}$$

*boos*t (t.field in d) is a search time boost of term t in the query q as specified in the query text, or as set by application calls to setBoost().

norm(t,d) encapsulates a few (indexing time) boost and length factors:

- Document boost - set by calling doc. setBoost() before adding the document to the index. It sets document boost equal to OPIC score of that document.
- Field boost - set by calling field.setBoost() before adding the field to a document. While query formulation, we set this boost values to match our ranking needs.
- lengthNorm(field) - computed when the document is added to the index in accordance with the number of tokens of this field in the document.

When a document is added to the index, all the above factors are multiplied. If the document has multiple fields with the same name, all their boosts are multiplied together:

However the resulted norm value is encoded as a single byte before being stored. At search time, the norm byte value is read from the index directory and decoded back to a float *norm* value. This encoding/decoding, while reducing index size, comes with the price of precision loss - it is not guaranteed that decode (encode(x)) = x. For instance, decode (encode(0.89)) = 0.75.

queryNorm(q) is a normalizing factor used to make scores between queries comparable. This factor does not affect document ranking (since all ranked documents are multiplied by the same factor), but rather just attempts to make scores from different queries comparable. This is a search time factor computed by the Similarity in effect at search time.

coord(q,d) is a score factor based on how many of the query terms are found in the specified document. Typically, a document that contains more of the query's terms will receive a higher score than another document with fewer query terms. This is a search time factor computed in coord(q,d) by the Similarity in effect at search time.

3. Output Processing

Output processing consists of Snippet Generation, Summary Generation and Snippet and Summary Translation

Snippet Generation

The Snippet Generation Module generates the snippet for each retrieved document. This module receives the parse text of each retrieved document and the Query words from the search engine as input and generates the snippet of the document within a fixed maximum character length. The generated snippet is displayed on the output screen.

The Title and META Description content (if available) of the retrieved web page is extracted from the HTML source file, the query words are highlighted and then forwarded to the Snippet Translation module for generating the snippet in the query language when the document language is either in English or Hindi. If no META description content is available then the snippet will be generated using the following sequence of functions:

- The Title and the Text content of the web page are extracted from the HTML source file.
- Sentences in the text content are identified and ranked based on the number of query words, key words and domain ontology words present in the sentence. The top three sentences with the query words highlighted, are selected as the snippet. If the specified snippet length is exceeded, sentences are segmented into snippet units. These snippet units are separately ranked and selected to generate the final snippet.

Summary Generation

The Summary Generation Module generates the summary of the retrieved document based on user request. This module gets the parsed text of the retrieved document and the Query from the search engine. The summary of the document is generated with specified maximum summary length and then displayed on the output screen.

An Extraction based to summary generation has been adopted. Initially, the title and the text content of the web page are extracted from the HTML source file. Sentences in the text content are identified and ranked based on the number of query words, key words, title words and domain ontology words present in the sentence. The top ranked sentences with the query words highlighted, are selected as the summary until the specified maximum summary length is exceeded

Snippet and Summary Translation

The snippet generation and summary modules generate English / Hindi Snippet and the snippet translation module converts it into an Indian language.

C. Evaluation Subsystem

The aim of the Evaluation sub-system is to create a framework for quantitatively evaluating various components of the Sandhan system, most importantly the ranking of search results returned by the system. Such a framework is important for the following reasons. It can be used to tune various parameters of the Sandhan system in order to improve performance.

More generally, it can be used to compare different models, algorithms, and techniques in a fair and uniform way. This allows us to measure progress. The availability of large-scale, standardized datasets also provides the impetus necessary to drive new research.

We have seen the overall architecture of the Sandhan system. Following figure shows how the Sandhan system is developed and the flow of offline and online processes.

IV. Case Study

We will see the working of Sandhan system through an example. As the Sandhan system is divided into two subsystems we will divide case study into two parts, offline and online.

A. Offline Process

Input for the offline process is seed URLs. Given a URL, the web page is fetched and parsed. After that a dump is created. This dump contains information about the document as in when it was fetched, URL of the document, content type (HTML/text), raw HTML content, parsed content, metadata, out-links, etc.

This dump file is given as input to CMLifier. CMLifier do parsing for Indian languages and gives back content in Unicode format. This is required as many of the Indian language content on web are not in Unicode format which is required for further processing. Language and domain of the content is identified. CMLifier creates a parse object with fields like title, content, language, etc. Figure 6 gives the architecture of Sandhan offline system form code prospective.

A. Online Process

Input to the online process is query from the user. The query goes through following processes. The input and output of each process is also shown. We will see few example queries how they are processed and what are the documents retrieved for those queries.

Query 1: अकोल्यातील प्रेक्षणीय स्थठ्ठ (tourist places in Akola)

1. Tokenization
   Input: अकोल्यातील प्रेक्षणीय स्थठ्ठ
   Output: अकोल्यातील, प्रेक्षणीय, स्थठ्ठ
2. Stop word removal
   Input: अकोल्यातील, प्रेक्षणीय, स्थठ्ठ
   Output: अकोल्यातील, प्रेक्षणीय, स्थठ्ठ
3. Stemming
   Input: अकोल्यातील, प्रेक्षणीय, स्थठ्ठ
   Output: अकोला, प्रेक्षणीय, स्थठ्ठ
4. Name entity recognition
   Input: अकोला, प्रेक्षणीय, स्थठ्ठ
   Output: अकोला, प्रेक्षणीय, स्थठ्ठ
5. Multi-word expression recognition
   Input: अकोला, प्रेक्षणीय, स्थठ्ठ
   Output:

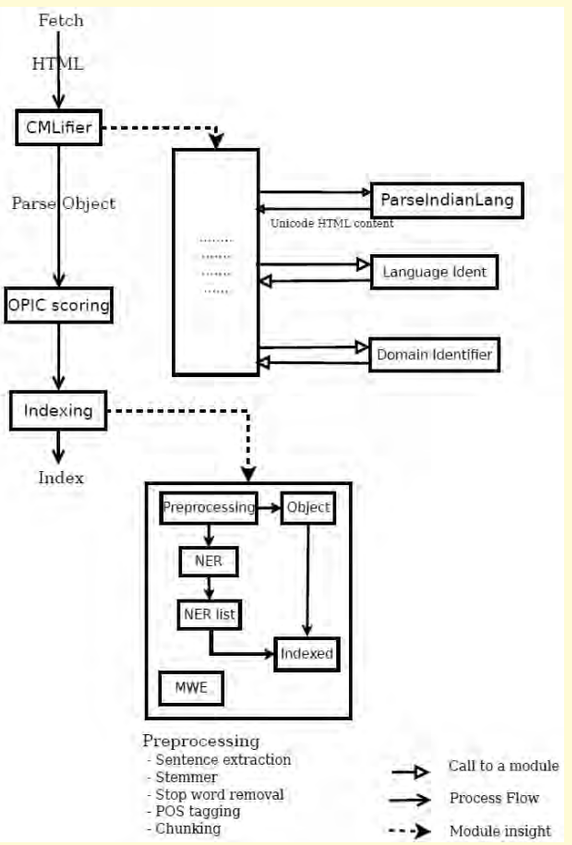

Figure 6: Flow of offline processes of Sandhan

6. Lucene query formation

   Input: अकोला, प्रेक्षणीय, स्थळ

   Output:(url:अकोला^3.0 anchor:अकोल^0.0
   content:अकोल^10.0 title:अकोल^3.0)

+(url:प्रेक्षणीय^3.0 anchor:प्रेक्षणीय^0.0 content:प्रेक्षणीय^10.0 title:प्रेक्षणीय^3.0) +(url:स्थट्ठ^3.0 anchor:स्थट्ठ^0.0 content:स्थट्ठ^10.0 title:स्थट्ठ^3.0) url:"अकोला प्रेक्षणीय स्थट्ठ"~2147483647^3.0 anchor:"अकोला प्रेक्षणीय स्थट्ठ"~4^0.0 content:"अकोला प्रेक्षणीय स्थट्ठ" ~2147483647^10.0 title: "अकोला प्रेक्षणीय स्थट्ठ"~2147483647^3.0 +lang:mr

As you can see the query is tokenized, stemmed correctly. There are no stop words in this query. Lucene query is formed giving weightage to each term and combinations of term. These weights are helpful in ranking the documents.

The retrieved documents are shown Figure 7. We can see that documents retrieved in Marathi are not relevant to the query. The documents only talk about the tourist places but no the tourist places in Akola. This might be due to the fact that Akola is not a tourist place or information regarding tourist places in Akola is not present in the crawled Marathi content. But documents in English talks about Akola and places around it. This shows that Marathi documents crawled does not cover all places but still we are able to access the information about it using Sandhan. Figure 8 and Figure 9 shows the documents retrieved in English and Hindi.

Query 2: ताजमहल आगरा

1. Tokenization
   Input ताजमहल आगरा
   Output ताजमहल आगरा
2. Stop word removal
   Input: ताजमहल आगरा
   Output: ताजमहल आगरा
3. Stemming
   Input: ताजमहल आगरा
   Output: ताजमहल आगरा
4. Name entity recognition
   Input: ताजमहल आगरा
   Output: ताजमहल आगरा
5. Multi-word expression recognition
   Input: ताजमहल आगरा
   Output:
6. Lucene query formation

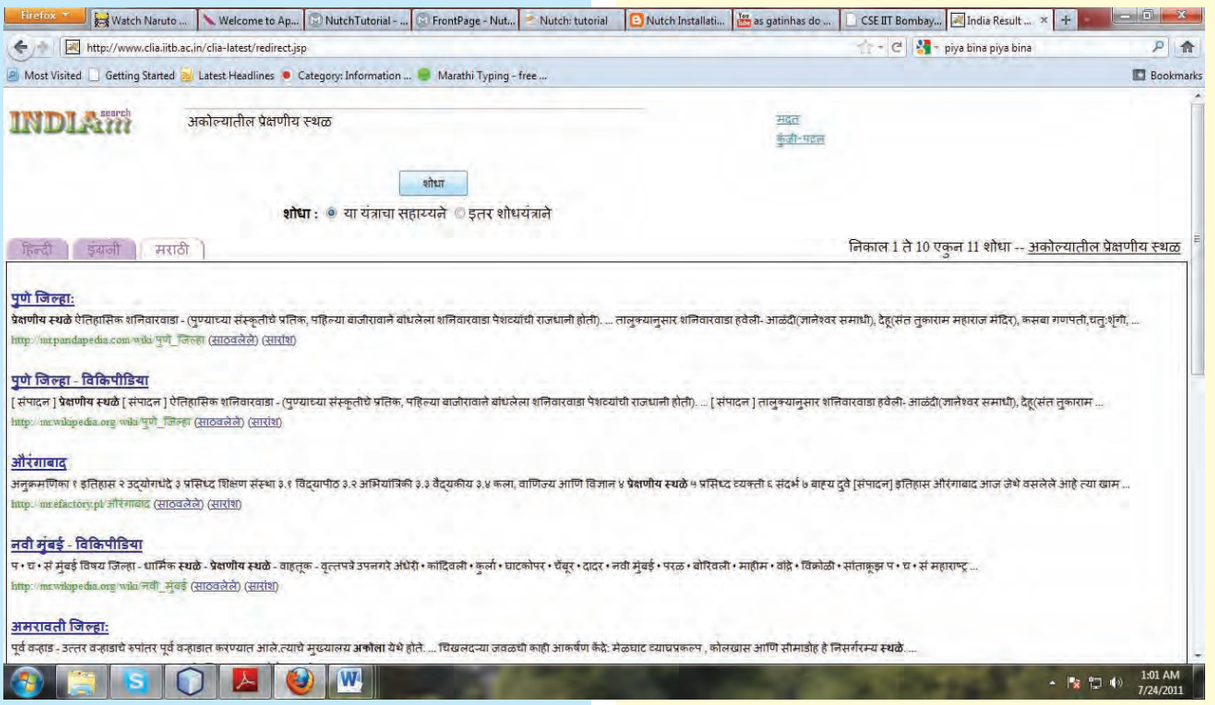

Figure 7 : Screenshot of Sandhan retrieving Marathi documents for a Marathi query

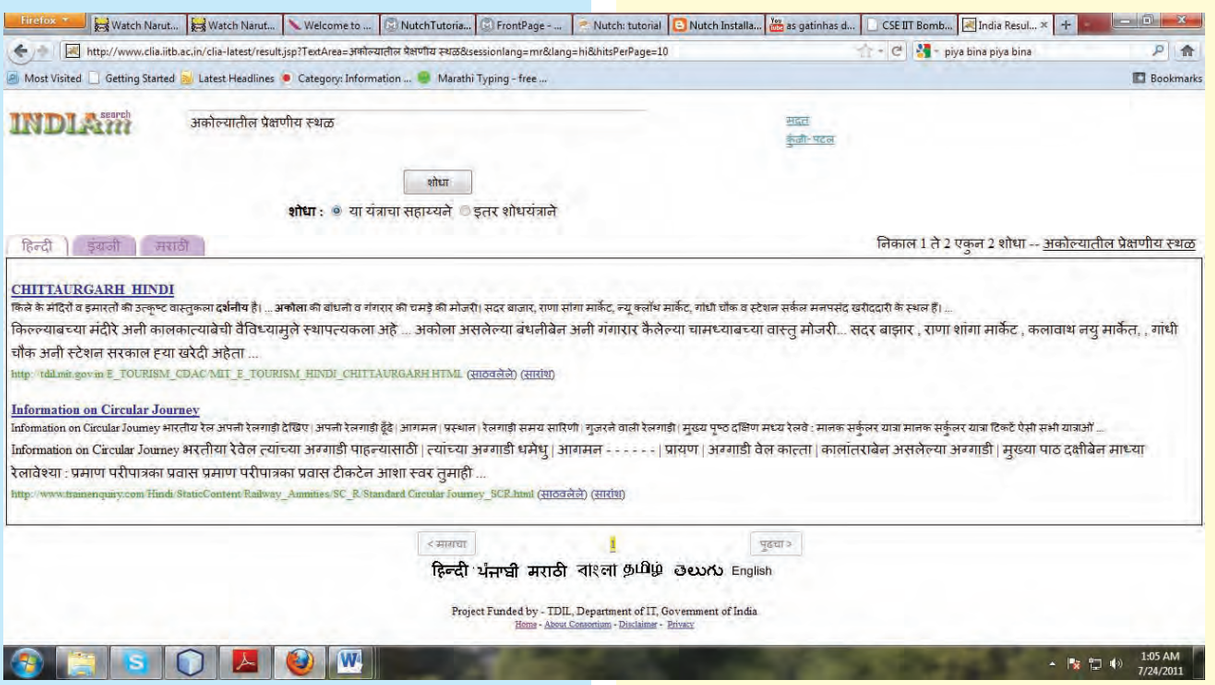

Figure 8 : Screenshot of Sandhan retrieving Hindi documents for a Marathi query

Figure 9 : Screenshot of Sandhan retrieving English documents for a Marathi query

(url:ताजमहाल^3.0 anchor:ताजमहाल^0.0 content:ताजमहाल^10.0 title:ताजमहाल^3.0) +(url:आग्रा^3.0 anchor:आग्रा^0.0 content:आग्रा^10.0 title:आग्रा^3.0) url:"ताजमहाल आग्रा"~2147483647^3.0 anchor:"ताजमहाल आग्रा"~4^0.0 content:"ताजमहाल आग्रा"~2147483647^10.0 title:"ताजमहाल आग्रा"~2147483647^3.0 +lang:mr

The documents retrieved for query2 are shown in Figure 10. We can see that the documents are relevant to the query.

Query 3: लोणावळ्यातील तलाव

1. Tokenization
   Input लोणावळ्यातील तलाव
   Output लोणावळ्यातील, तलाव
2. Stop word removal
   Input: लोणावळ्यातील, तलाव
   Output: लोणावळ्यातील, तलाव
3. Stemming
   Input: लोणावळ्यातील, तलाव
   Output लोणावळा, तलाव
4. Name entity recognition
   Input लोणावळा, तलाव
   Output लोणावळा, तलाव
5. Multi-word expression recognition
   Input: लोणावळा, तलाव
   Output:
6. Lucene query formation

(url:लोणावळा^3.0 anchor:लोणावळा^0.0 content:लोणावळा^10.0 title:लोणावळा^3.0) +(url:तलाव^3.0 anchor:तलाव^0.0 content:तलाव^10.0 title:तलाव^3.0) url:"लोणावळा तलाव"~2147483647^3.0 anchor:"लोणावळा तलाव"~4^0.0 content:"लोणावळा तलाव"~2147483647^10.0 title:"लोणावळा तलाव"~2147483647^3.0 +lang:mr

Retrieved documents for this query are shown in Figure 11. Some of the retrieved documents talk about lakes in Lonavala and some talk about lakes in different places. The set of documents which talk about lake and not the lakes in Lonavala are retrieved as these documents contain the word तलाव in the title field which is given more weightage than content field. We need to refine the ranking algorithm to avoid this kind of results.

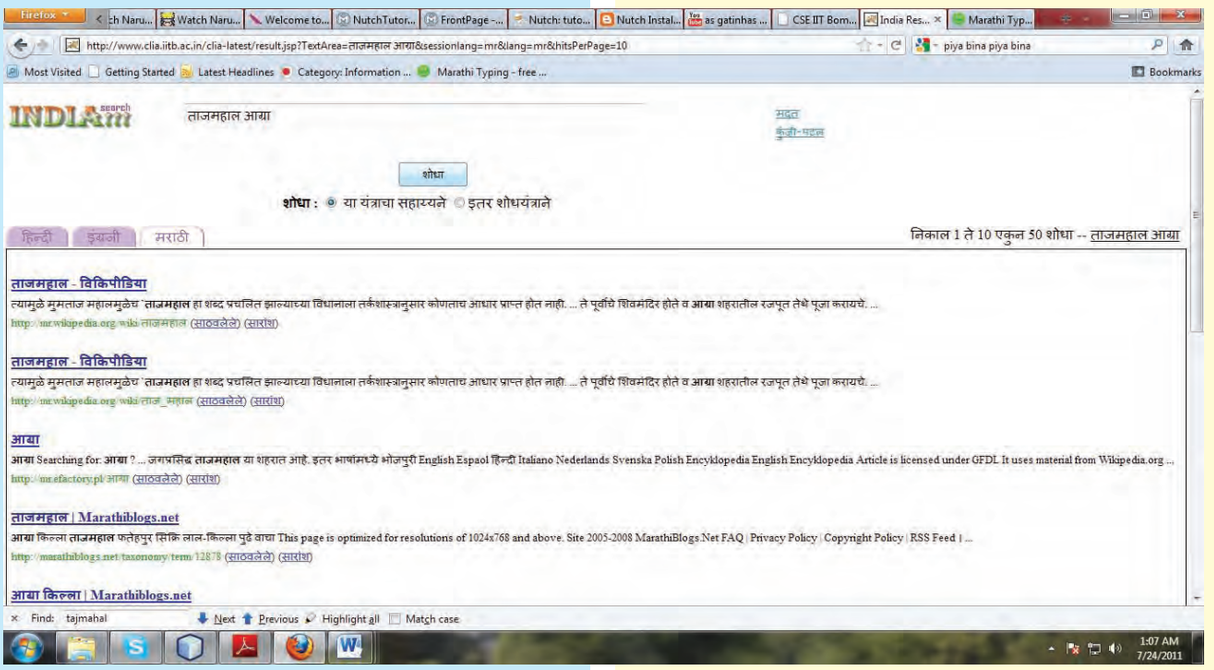

Figure 10: Screenshot of Sandhan giving good results for tourism query

Figure 11: Screenshot of Sandhan giving partially relevant documents

As you can see the current NER and MWE modules are not working well. Development of these modules is in progress.

Precession of the system is calculated for 25 Marathi queries. The precession values are shown in Table 1.

	P@5	P@10
Marathi – Marathi	0.24	0.15
Marathi-English	0.75	0.69
Marathi-Hindi	0.25	0.21

Table 1: Precession of Sandhan

V. Conclusion

We have seen how Nutch and Lucene works and what are the additional things we are doing to make it support cross lingual information access. Few of these changes are the language processing modules. Language processing modules like morphological analysis, translation and transliteration are essential but at the same time they add to the error as each of these processes are not perfect and include some error. Other important element to cross lingual information access is resources in the supported languages. For resource scarce languages like Marathi, it is very difficult to find the resources like parallel corpora for translation. We have also seen that Marathi is morphologically reach language which also makes it difficult to stem and get the root form of the words and formation of sentence during translation.

We can see from Table 1 precession values for Marathi to Marathi and Marathi to Hindi retrieval are less compared to Marathi to English retrieval. The reason for this is the lack of tourism related data in Marathi and Hindi languages.

VI. Acknowledgement

The research was supported by the Department of Information Technology, Govt. of India, through the sponsored project on Cross Lingual Information Access, code 10DIT012.

VII. References

[1] N.kando, K.kishida. "Two stage Refinement of Query Translation in a pivot language Approach to CLIR : An Experiment at CLEF 2003." *CLEF.* 2003.

[2] Ganesh Bhosale. "Marathi to Hindi Machine Translation". Masters project report, Indian Institue of Technology Bombay, Mumbai. 2011

[3] Jurafsky, D., & Martin, J. H. *Speech and Language Processing.* Prentice-Hall. 2000.

[4] Gospodnetic, Otis; Erik Hatcher. *Lucene in Action* (1st ed.). Manning Publications. pp. 456. ISBN 978-1-932394-28-3. 2004.

[5] Nutch Wiki http://wiki.apache.org/nutch/

[6] Singh, Smriti, Kuhoo Gupta, Manish Shrivastava, and Pushpak Bhattacharyya. 2006. Morphological Richness Offsets Resource Demand – Experiences in Constructing a POS Tagger for Hindi. In Proceedings of ACL-2006

[7] Vishal Vachhani. "Cross Lingual Search". Masters project report, Indian Institue of Technology Bombay, Mumbai. 2009

[8] Mugdha Bapat, Harshada Gune and Pushpak Bhattacharya. *A Paradigm-Based Finite State Morphological Analyzer for Marathi*. The 23rd International Conference on Computational Linguistics, Beijing,, China.2010.