

# A Distributed Retrieval System for NTCIR-5 WEB Task

Hiroki Tanioka Kenichi Yamamoto Takashi Nakagawa  
Justsystem Corporation  
Brains Park Tokushima-shi, Tokushima 771-0189, Japan  
{hiroki\_tanioka, kenichi\_yamamoto, takashi\_nakagawa}@justsystem.co.jp

## Abstract

*We developed a distributed search system with the corresponding very large scale corpora from NTCIR-5 WEB Task. And we arranged the scoring method which is based on link-structure of the Web documents to calculate lower cost. Our search system, which consists of 6 PCs could make indices for full texts size of about 1 TB. Additionally, we confirmed that our arranged scoring method made an improvement of mean average precision.*

*Also we performed experiments with the pseudo-document vectors at every pseudo-relevance feedback. Meanwhile we made a pseudo-document vector at every relevance feedback. Therefore the results had slightly better precision than raw queries even though it had not been tuned yet.*

**Keywords:** *distributed information retrieval, link-structure analysis, vector space model, inverted file, relevance feedback, pseudo-relevance feedback*

## 1 Introduction

Our purposes to participate in NTCIR-5 WEB Task is as follows.

- Research and development of search systems which are corresponding a very large scale corpora.
- Research and development of scoring methods which are based on non-text information for the Web documents.

The background of first purpose is that digital documents are increasing in recent years, while we need search systems to access these documents effectively. But traditional search systems cannot index the full text of these documents. Therefore we propose a distributed search system which is built on a distributed framework.

The background of second purpose is that we confirm the effectiveness of scoring methods which are based on link-structure of the Web documents (e.g.

PageRank[3]). We arrange the scoring method to calculate at lower cost.

The rest of this paper is divided into three sections. Section 2, we describe an architecture of our distributed processing framework and search system. Section 3, we describe results of formal runs. Section 4, we discuss about results and future works.

## 2 System Description

In this section we describe the architecture of our distributed search system and information retrieval models including some scoring methods.

### 2.1 Navigational Retrieval Subtask 2

First, we explain the system architecture and models for Navigational Retrieval Subtask 2.

#### 2.1.1 Overview

We explain a distributed search system which is based on Vector Space Model using term partitioning with an inverted file-based system, while a single inverted file is created for the document collection and the inverted lists are spread across the processors.

During query evaluation, the query is decomposed into indexing items and each indexing item is sent to the processor that holds the corresponding inverted list[2].

#### 2.1.2 Distributed Processing Framework

Cocktail Framework<sup>1</sup> is used to make the distributed search system based on Vector Space Model. The framework provides a service of agents between client and server, as broker between query and each indexing item.

Figure 1 shows an overview of this framework. To process a job<sup>2</sup> in our system, a client machine receive

<sup>1</sup>Cocktail Framework is developed for distributed processing framework by Justsystem Corporation.

<sup>2</sup>Job is described as a pair of command and argument which are processed in our system.

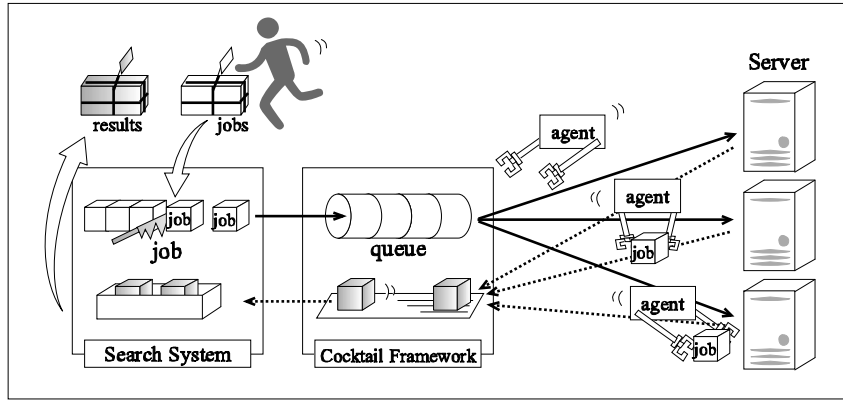


Figure 1. System Description

a job, and keep in a FIFO queue. And then, to send the job to a server machine, unconfined agents pull the job from the FIFO queue. Last, the server machine performs the job and send a result back to the client machine via same agent.

### 2.1.3 Indexing Algorithm

This system has indexing structure using an inverted file. And this inverted file-based system is based on term partitioning for whole distributed inverted files on some server machines as a single inverted file.

In general, there are two methods of how to distribute inverted files on server machines. First method is that an inverted file is divided based on terms. Second method is that an inverted file is divided based on documents. And, the second method needs to search the inverted file on all server machines for the documents, but it is hard to calculate the correct *IDF* scores for each terms in a query. Thus we decide on to apply the first method.

The problem is that it is too costly to make the inverted file. It needs a large amount of memory if we execute on memory, and it needs a long time if we execute on hard disk. Therefore we take an approach using a kind of merging the partial indices increasingly. In concrete, we show the conceptual figure as figure 2. We make the inverted file on memory until reach a limit. When the size of inverted file is over the limit of memory, we store the inverted file on memory to hard disk. If there are already a previous inverted file on hard disk, the two files on hard disk and on memory are gradually merged.

The total time to generate the partial indices is  $O(n)$ , where  $n$  is the number of characters,  $m$  is the number of merging times. And thus the cost of this algorithm is as follows, where  $n'$  is the average number of characters each partial indices.

$$O(n' \cdot m(m+1)/2) \simeq O(n' \cdot m^2) \quad (1)$$

In addition, a text data extracted by MeCab[1] is included by corpora of NTCIR-5 WEB Task. Therefore, the feature of Vector Space Model contains terms of noun, verb and unknown word as part of speech from the text data.

### 2.1.4 Retrieval Model

Our retrieval model is based on Vector Space Model. And calculating formula of search score is based on simple calculation of  $TF \cdot IDF$  as follows.

$$S_Q = \frac{p_1}{T_q} \quad (2)$$

$$S_T = \log(TF_{d,t}) \cdot \log\left(\frac{N}{DF_t}\right) + S_Q \quad (3)$$

Where  $S_Q$  is the score dependent on the number of terms  $T_q$  in given query,  $S_T$  is the score of term  $t$ ,  $TF_{d,t}$  is term frequency of term  $t$  in document  $d$ ,  $N$  is the number of documents, and  $DF_t$  is document frequency of term  $t$ . To put a cap on  $\log(TF_{d,t})$  to appropriate measures against spam terms. If  $\log(TF_{d,t})$  is over  $p_3$ , we cut the numeric value to  $p_2$ .

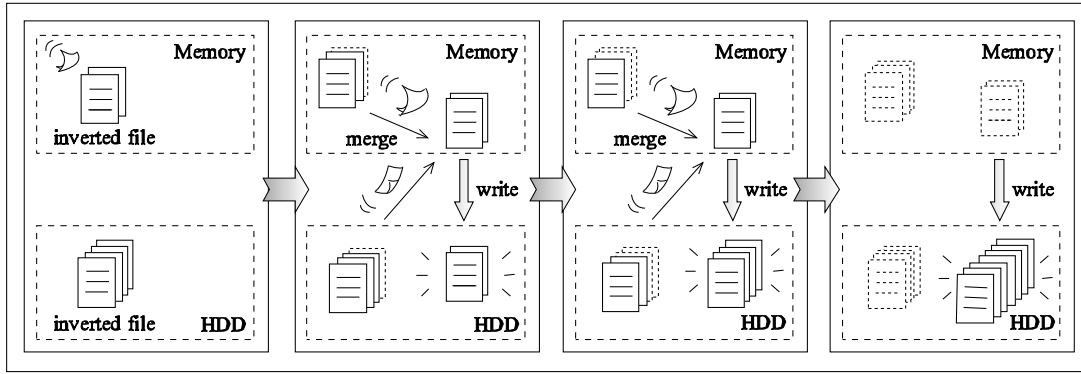
Here in the experimentations, the values of parameters are set without making an adjustment at all. Where each constant numbers are declared as follows.

$$p_1 = 100, \quad p_2 = 1, \quad p_3 = 5 \quad (4)$$

There are three differences from original  $TF \cdot IDF$  calculating formula.

- Using logarithm for term frequency :  $TF$
- Addition of score :  $S_Q$
- Limitation of logarithmic  $TF$  :  $\log(TF)$

The first difference is based on our pilot study, which shows an adverse effect a greater values of  $TF$


**Figure 2. Indexing Algorithm**

on original  $TF \cdot IDF$  calculation. The second difference is that operability method controls the score based on the number of terms in a query. And a purpose of the third difference is a solution of detecting spam terms.

### 2.1.5 Scoring Web Documents by Link Structure

In general, it is said that the scoring method based on link-structure of Web documents is useful for Web search system. Therefore we also use the score using link-structure of Web documents, as follows.

$$S_L = \frac{2}{\frac{1}{LC} + \frac{1}{DC}} \quad (5)$$

Where  $LC$  is a link count as the total number of inlinks and  $DC$  is a domain count as the number of inlinks' domain. This calculating formula is based on an assumption that the important document has more multidomain inlinks.

Furthermore if  $S_L$  grater than  $S_Q$  then we use  $S_Q$  instead of  $S_L$ . Because of total mixed ration between  $S_L$  and  $S_T$  as the main reason for this experiment. To detect spam links, we make a limitation of link count.

Finally, the score  $S_{d,q}$  of document  $d$  for query  $q$  is given by

$$S_{d,q} = \sum_{t \in q} S_t + S_L \quad (6)$$

where  $S_t$  is  $S_T$  of each term  $t$  in query  $q$ .

## 2.2 Query Term Expansion Subtask

Query expansion is often effective when queries are not so appropriate for retrieving many relevant documents. Query expansion usually needs some data; some relevant documents, a thesaurus, query logs, and so on. If manual operations are required to make these data, they should be minimized or naturally integrated in normal search operations. Thus we tried

to mix good points of both relevance feedback[2] and pseudo-relevance feedback[4, 6]. On one hand, relevance feedback has a good point to match users' needs and a bad point to need user's manual selection of some relevant documents every time. On the other hand, pseudo-relevance feedback has a good point to need no manual operation and a bad point to ignore users' preferences.

In order to mix both good points, we made a pseudo-document vector every relevance feedback. In this scheme, manual operations to select relevant documents are optional. When we select relevant documents, relevance feedback is performed and a vector for the latter pseudo-relevance feedbacks are made. In the case of the latter search with no user-selected relevant documents, pseudo-relevance feedback using these existing vectors is performed. This scheme can reflect users' preferences even in the case of pseudo-relevance feedbacks.

We made pseudo-document vectors as follows.

1. Convert relevant documents and queries to binary vectors (1 if exist, 0 if not exist).
2. Normalize these vectors as their norms equal 1.
3. Make every pseudo-document vector  $pdv$  by appending vectors of its query  $q$  and relevant documents  $r_i$  as follows.

$$pdv = q + 2 \sum_{i=1}^n \frac{r_i}{n} \quad (7)$$

We used pseudo-document vectors as follows.

1. Convert a query to a binary vector whose norm equals 1 like making pseudo-document vectors.
2. Select pseudo-document vectors with positive values of cosine between the query vector and pseudo-document vectors.

3. Make an expanded query vector  $eqv$  with the query vector  $q$  and searched pseudo-document vectors  $pdv_i$  as follows.

$$eqv = q + \sum_{i=1}^n pdv_i \cdot \cos(q, pdv_i) \quad (8)$$

4. Select 10 terms with highest scores from the expanded query vector and use them as expanded query terms.

### 3 Results

In this section we show the results of NTCIR-5 WEB Task.

#### 3.1 Navigational Retrieval Subtask 2

We show the result of Navigational Retrieval Subtask 2. Table 1 shows the difference of 4 runs we submitted.

**Table 1. Difference of each runs**

	<i>TF</i> limited	link-structure
JSWEB1	No	No
JSWEB2	No	Yes
JSWEB3	Yes	No
JSWEB4	Yes	Yes

Table 2 shows average precisions from results of each runs. But we experimented under the condition of lacking several documents. Where others is average of the other participants' average precisions. According to these results, the result of JSWEB2 was better than JSWEB1, and JSWEB4 was better than JSWEB3 throughout the results. Thus it can be said that the scoring method based on link-structure of Web documents works for this subtask, though this scoring method is simple harmonic average.

Furthermore, the result of JSWEB3 was better than JSWEB1, and JSWEB4 was better than JSWEB2 throughout the results. Thus it can be said that the limited of *TF* works for this subtask. But the results were worse than other teams' result on average. This problem requires our further works, and we will look back in Section 4.

Then we show the performance of our search system. We used 6 PCs for this subtask. Table 3 shows that specification of all PCs. And expanded information is that each network-linked PCs are connected on gigabit Ethernet.

The performance of our system was without text analyzing. Because we used the corpus analyzed by MeCab. For this reason, the indexing time which was

**Table 2. Result of each runs**

	A	AB	A_mod	AB_mod
JSWEB1	0.0094	0.0107	0.0094	0.0107
JSWEB2	0.0182	0.0180	0.0144	0.0146
JSWEB3	0.0099	0.0111	0.0099	0.0111
JSWEB4	0.0194	0.0190	0.0157	0.0158
others	0.1117	0.1078	0.0936	0.0902

**Table 3. Specification of PCs**

	CPU[GHz]	Ram[GB]	OS
A	Celeron 2.4	1	WinXP Pro SP2
B	Celeron 2.4	1	FedoraCore3
C	Celeron 2.4	2	FedoraCore3
D	Celeron 2.2	1	FedoraCore3
E	Celeron 2.2	1	FedoraCore3
F	Celeron 1.7	1	WinXP Pro SP2

without the morphological analysis time. On that basis, the time for indexing of about 1 TB full text corpus was in 7.4 days. And the time to search was in about 2.7 second by a query. As a result, we thought that the distributed search system had a significant impact on high-speed processing for information retrieval. Moreover we can expect to speed up by adding on more PCs.

**Table 4. Performance of search system**

Indexing time	7.4 days
Search time	2.7 sec.

#### 3.2 Query Term Expansion Subtask

Table 5 shows slightly improved precisions with expanded query vectors. We used our original search system as a baseline and a processor of expanded queries. MAP shows a mean average precision.

Next, we explored factors relating to the MAP improvement. We couldn't find any relations between the MAP improvement and query terms. However, we found some relation between the MAP improvement and relevant documents. Figure 3 shows the slightly negative correlation (correlation coefficient = 0.40) between the MAP improvement and the number of relevant documents to make the pseudo-document vector for the query.

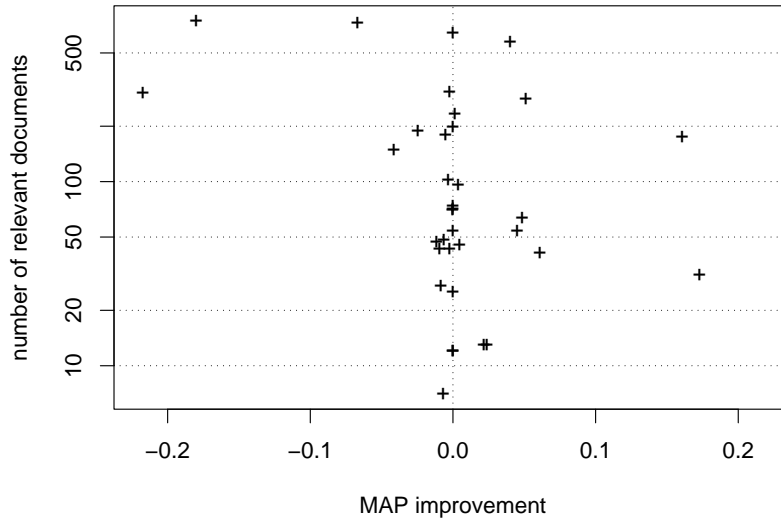


Figure 3. Relation between MAP Improvement and Number of Relevant Documents

Table 5. Result of query expansion

	Baseline	Expanded queries
MAP	0.0962	0.0976

$$S_T = \log(TF + 1) \cdot \log\left(\frac{N}{DF}\right) + S_Q \quad (10)$$

## 4 Discussions

### 4.1 Navigational Retrieval Subtask 2

We proposed the distributed search system and indexing method using merging the partial indices. Also we evaluated the distributed and indexing methods, and showed that these methods have high-speed in the subtask. Therefore we can say that restriction of corpus size is solved in theoretical.

We also proposed to use a scoring method based on link-structure of Web documents, and the limited of  $TF$ . When these scoring methods were used, the results were improved slightly from original  $TF \cdot IDF$ . Additionally the distributed processing framework and search system, which were very flexible to adjust to various experimental environments.

All our purposes are accomplished, but there are still unsatisfactory results. We show the calculating formula again.

$$S_T = \log(TF) \cdot \log\left(\frac{N}{DF}\right) + S_Q \quad (9)$$

Here in the above formula, there are problem that  $\log(TF) = 0$  when  $TF = 1$ . Thus we improved the calculating formula as follows.



Table 6 shows the results improved calculating formula. When all experimental conditions are same as original conditions, new results were better than all original results.

Table 6. Improved result of each runs

	AB_mod
JSWEB1	0.0107
JSWEB2	0.0146
JSWEB3	0.0111
JSWEB4	0.0158
new JSWEB1	0.0112
new JSWEB2	0.0158
new JSWEB3	0.0115
new JSWEB4	0.0161
others	0.0832

However the calculating formula is written in a component of search system. Then we can these additional experiments without re-indexing the inverted file.

Next, we discuss the result for each query. Figure 4 shows example of good or bad queries. Good query shows better average precisions than averages of the other participants' average precisions. Bad query shows worse average precisions than averages of the

Type	Query ID	Terms	Diff
Good Query 	1283	<u>真鍋 かかり 日記</u>	0.57
	1104	<u>カブ-エンジョイ</u>	0.19
	1084	<u>アイフル</u>	0.12
	1140	<u>ドクター 中松</u>	0.07
	1189	<u>阿蘇山</u>	0.06
Bad Query 	1345	<u>東京 駅 銀 の 館</u>	- 0.26
	1202	<u>花やしき</u>	- 0.31
	1059	<u>non - no</u>	- 0.34
	1017	<u>IT 用語 辞典 e - words</u>	- 0.38
	1005	<u>ABC 振興 会</u>	- 0.42

The under line part is one term.

Figure 4. Example of Good / Bad Queries

other participants' average precisions. Where "diff" is the difference of between our average precisions and averages of the other participants' average precisions.

We analyze some queries of bad results and categorize these queries as follows.

- Our system can search the relevant domain page, but cannot search the top page. [ex. 1005, 1017]
- Our system cannot search the relevant page because of segmenting complex term. [ex. 1059]
- Our system cannot search the relevant page because of low term frequency. [ex. 1202, 1345]

We think that countermeasures against each type is as follows.

- We arrange the scoring method which is based on link-structure or the page weighting method for the domain top page.
- We deal properly with a complex term.
- We adopt the algorithm of normalized term frequency.

We think a reason of good query is that our search system is the full text search system. If query term appear only in body text, our system is more advantageous than the other system which indexes only anchor text or title text.

Finally our future work is the improvement of search result. We must experiment various calculating formula and parameters. Although we will optimize quickly, we can simply change some components.

## 4.2 Query Term Expansion Subtask

Our scheme to expand query vectors made slightly higher average precisions than the baseline search system. However, these differences are not so clear. Also,

figure 3 implies problems to make pseudo-document vectors with many relevant documents. We didn't use *TF* or *IDF* to make or use pseudo-document vectors. So, expanded query terms with many relevant documents are apt to contain only frequent terms with little information. We didn't use *IDF* or other corpus-dependent parameters because we want to share pseudo-document vectors among different corpora. However, this result might suggest that we should adopt a new method with *IDF*-like parameter for our corpus-independent scheme.

We had not tuned the parameters and detailed methods to expand them yet. Thus we will explore better parameters and detailed methods and test how better it is. We suppose the merits of this scheme as follows.

- Reduction of every time users' operations from relevance feedback.
- Adapting users' preferences in the case of a query without user selected relevant documents.
- Good performance and scalability to apply it to large corpus such as Web documents.

## Acknowledgement

We appreciate our colleagues Kayoko Tono and Daisuke Motohashi for their encouragement.

## References

- <http://chasen.org/taku/software/mecab/>.
- R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*, chapter 5,9. Addison-Wesley, 1999.
- S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *In Proc. of the 7th Int. WWW Conference, Brisbane, Australia*, April 1998.
- D.Evans and R.Lefferts. Design and evaluation of the clarit-trec-2 system. *In D.K.Harman editor, Proceedings of the Second Text REtrieval Conference (TREC-2). NIST Special Publication*, pages 500-548, 1994.
- M. R. Henzinger. Hyperlink analysis for the web. *IEEE INTERNET COMPUTING JANUARY FEBRUARY*, 2001.
- A. M.Mitra and C.Buckley. Improving automatic query expansion. *In SIGIR '98*, pages 206-214, 1998.
- B. C. Salton G. Term-weighting in information retrieval using the term precision model. *Journal of the Association for Computing Machinery*, 152-170(29), 1982.
- Y. C. S. Salton G., Wong A. A vector space model for automatic indexing. *Communications of the ACM*, 613-620(18), 1975.
- J. Wu, H. Tanioka, S. Wang, D. Pan, K. Yamamoto, and Z. Wang. An improved vsm based information retrieval system and fuzzy query expansion. *In FSKD (1)*, pages 537-546, 2005.