A Parallel DBMS Approach to IR in TREC-3

David A. Grossman
Office of Information Technology
3E09 Plaza B
Washington, DC 20505
dgrossm1@mason1.gmu.edu

David O. Holmes
AT&T Global Information Solutions
2 Choke Cherry Road
Rockville, MD 20850

Ophir Frieder*
Department of Computer Science
George Mason University
Fairfax, VA
ophir@cs.gmu.edu

Abstract

In this our first year of TREC participation, we implemented an IR system using an AT&T DBC-1012 Model 4 parallel relational database machine. We started with the premise that a relational system could be used to implement an IR system. After implementing a prototype to verify that premise, we then began to investigate the performance of a parallel relational database system for this application. We only used the category B data, but our initial results are encouraging as processing load was balanced across the processors for a variety of different queries. We also tested the effect of query reduction on accuracy and found that queries can be reduced prior to their implementation without incurring a significant loss in precision/recall. This reduction also serves to improve run-time performance.

Finally, in a separate set of work, we implemented Damashek's n-gram algorithm for n=3 and were able to show similar results as found when n=5.

1 Introduction

For TREC-3, we implemented relevance ranking queries using unchanged SQL on an AT&T DBC-1012 (formerly Teradata) parallel database machine [4]. The purpose of this implementation was to test the following hypotheses:

- A relational system that implements standard SQL, may be used as the search engine for an information retrieval application.
- A parallel relational database machine will use an optimizer to balance the workload across multiple processors. The result will be a scalable system such that required levels of performance may be achieved with the purchase of additional hardware.
- Query reduction based on term frequency counts will dramatically improve performance without a significant degradation in accuracy.

*This work supported in part by the National Science Foundation under contract number IRI-9357785.

We have found that each of these hypothesis are true as our relational implementation provides scalable performance. Additionally, query reduction improved run time performance without a significant degradation in accuracy.

Section 2 describes related prior work that serves as the foundation behind our hypothesis. The use of the relational DBMS to model an inverted index is described in Section 3. Our means of computing a measure of relevance is described in in Section 3.3. Section 4 and Section 5 describe our runtime and accuracy results. Conclusions and suggestions for future work are given in Section 6.

2 Prior Work

We briefly describe the prior work that provides the motivation behind these hypotheses. The use of a relational system that would serve as a search engine for an IR application was first proposed by Blair [5]. In this work, it was mentioned that SEQUEL (a precursor to SQL) could be used to perform boolean keyword retrievals such as "Find all documents that contain the word 'terrorist'. " Later, Macleod presented several SEQUEL queries that performed keyword searches using unchanged SEQUEL. Additional operators were then described that could achieve relevance ranking of a set of documents to a query. Research continued in the use of the relational model as a means of providing a more robust form of information retrieval.

Most research in the area stopped when user-defined operators were defined to address certain "application specific" operations [3]. Any function required by an application that does not exist in the database system may be incorporated via a user-defined operator. Stonebraker, et al, examined the usefulness of user-defined operators in assisting a text editing application [10]. A more recent thesis was devoted to the use of user-defined operators to provide typical information retrieval functionality such as keyword searches and proximity searches [8]. Additionally, enhancements to the database optimizer were analyzed that would allow for query optimization of these user-defined operators.

We have developed algorithms using unchanged SQL that implement vector space relevance ranking, proximity searches, and Boolean retrieval. Additionally, we computed worst case disk I/O estimates for a relational implementation and a traditional IR implementation [7]. It was this work that led us to the realization that query reduction based on term selectivity would dramatically affect I/O. For TREC-3, we were able to test different levels of query reduction based on term frequency and verified the impact on disk I/O and accuracy.

3 Implementation Details

We now discuss the approach used to migrate the category B portion of the TIPSTER collection to a set of relations. The relations effectively model an inverted index which may be used to efficiently query the database. The steps used to move text to the relational model are preprocess, load, and index.

3.1 Preprocess

We have developed a text preprocessor that accepts SGML marked text as input and produces three files as output. The preprocessor applies text formatting rules for special characters as described in [1, 2]. Subsequently, for each document, the document frequency for each distinct term is computed and written to a flat file. After all of the documents have been processed, a list of each distinct

term is identified and the *inverse document frequency* for each term is computed. Implementation details of the preprocessor are found in [9].

3.2 Create

Relations are created on the DBC-1012. A clustered primary key ensures that the data are stored in a fashion such that all tuples for a distinct term are on contiguous data pages. Some of our experiments suggest that this approach seemed to best minimize I/O. Other research continues to investigate the best placement strategy for these data [11].

Additionally, FALLBACK mode is used for each of the relations. This results in a full replica of all the data so that the system has resilience to a disk failure.

3.2.1 Load

The DBC-1012 FASTLOAD utility is used to move the data from the flat files (residing on an Intel 80486 based machine) to the DBC-1012. The utility makes use of all of the processors and ensures that data are distributed to each processor in the fashion prescribed by the clustered index. We typically found that the FASTLOAD was able to load our largest relation at a speed of 857 rows per second. By comparison, work we have done on an Intel Pentium based processor running Microsoft SQL Server on Windows NT typically loads data (using the Bulk Copy facility) at a rate of 381 rows per second.

3.2.2 Query processing

For TREC-3, we only addressed the ad-hoc queries (Topics 151-200). The queries were preprocessed and loaded into corresponding relations using a similar method as done for the data.

3.2.3 Example

The following example illustrates our process of starting with an input document in a TIPSTER file and completing with the properly loaded relations.

Consider the following document from the TIPSTER data (a final sentence has been added to assist our presentation):

<DOC>
<DOCNO> WSJ870323-0180 <DOCNO>
<HL> Italy's Commercial Vehicle Sales <HL>
<DD> 03/23/87 <DD>
<DATELINE> TURIN, Italy <DATELINE>
<TEXT>
Commercial-vehicle sales in Italy rose 11.4% in February from a year earlier, to 8,848 units, according to provisional figures from the Italian Association of Auto Makers.
Sales for the Association are expected to rise an additional 2% in July. <TEXT>
<DOC>

The following relations will be built to model this document:

DOC:

doc_id	*doc_name*	*date*	*dateline*
1	WSJ870323-0180	3/23/87	Turin, Italy

DOC_TERM:

doc_id	*term*	itterm_freq
1	commercial	1
1	vehicle	1
1	sales	2
1	italy	1
1	rose	1
1	11.4%	1
1	february	1
1	year	1
1	earlier	1
1	8,848	1
1	according	1
1	provisional	1
1	figures	1
1	italian	1
1	association	2
1	auto	1
1	makers	1
1	expected	1
1	additional	1
1	2%	1
1	July	1

IDF

term	idf
11.4%	2.9595
2%	1.5911
8848	4.3936
according	0.7782
additional	1.0792
association	1.0792
auto	1.2788
commercial	1.0000
earlier	0.6021
expected	0.6990
february	1.3222
figures	1.2553
italian	1.8451
italy	1.6721
July	1.0414
makers	1.3010
provisional	2.5172
rose	0.7782
sales	0.6990
vehicle	1.7709
year	0.0000

The first relation, *doc* contains a tuple for each document that occurs in the text. The internal document identifier and official TIPSTER document name are stored in this tuple along with any other structured data that has a 1-1 relationship with the document. For TREC-3, we only used the *date* and *dateline* SGML marked fields, but other fields such as *source* could easily be placed into this relation.

The *doc_term* relation models a typical inverted index. An attribute *doc_freq* is used to store the number of occurrences for the term in the document. A compressed internal document identifier is used here, while the official TREC-3 name may be obtained from the *doc* relation.

Finally, the *idf* relation stores the inverse document frequency for each distinct term in the entire document collection. The *idf* is computed as:

$$idf(term) = \log_{10} df$$

where *df* is the number of distinct documents in which the term appears.

3.2.4 Overhead

Typically, storage overhead has been a justification used against relational IR implementations. Given that the document identifier and the term must be replicated numerous times in the *doc_term*

relation, it would appear that storage requirements would be substantial. The following table indicates the storage requirements for each of the three relations. Since the DBC-1012 uses hash-based indices, there is no extra storage required for each index; rather, a fixed 13 byte overhead is assigned for each table to maintain an internal hash identifier.

Storage Overhead

Name	Tuples	Megabytes
Doc	173,252	29.6
Idf	389,797	32.7
Doc_Term	33,497,912	2,037.5

For the 534 megabytes found in Category B data the relational structures required to implement it required 2.1 Gigabytes of storage. However, the DBC-1012 replicates all data to provide real time protection for a disk failure. Without this replication, only 1.05 Gigabytes would be required. The overhead ratio is 1.97:1.00.

3.3 Query Processing

Having constructed the relations, we were able to implement a variety of experiments to learn more about performance of the DBC-1012 for this application. To learn more about accuracy, we reduced the size of TIPSTER queries based on the precomputed *idf* values. The premise was that a very frequently occurring term increases the I/O but decreases the accuracy of the query. Hence, this form of query reduction is based on the same premise as a stop word list and can be viewed as a tailored stop word list.

3.4 Building the Query Threshold Relation

A new query relation is generated called *query_threshold*. The original *query* relation is joined with the *idf* relation such that the *idf* is obtained for each term in the query. The terms in the query are then sorted in decreasing order of their *idf*.

The relation is then exported to a flat file and a simple utility is invoked to determine the number of terms in the query and develop new queries based on a given threshold. The threshold indicates the percentage of terms that are found in the new query. A threshold of ten indicates that the new query contains ten percent of the terms in an unmodified query. For a one hundred term query, a threshold of ten would result in a query composed of those terms that are ranked (by *idf*) one through ten. We developed a *query_threshold* relation that contains tuples that represent queries for thresholds of .1, .2, .25, .3, .33, .5, and 1.0.

3.5 Implement the Query with Unchanged SQL

We have previously identified queries that use standard SQL to implement both the inner product and cosine measures of relevance [7]. These queries used a *query* relation as described in Section 3. A slight modification is required to implement different query thresholds. Additionally, we found performance improved when we denormalized the *query_threshold* table to contain the *idf* as well. The *doc* table is used to obtain the official doc name after the internal document identifier is matched. The query used to compute a vector inner product between a query and all document vectors for the TREC-3 data is:

```
SELECT c.doc_name, sum(q.cntidf*b.term_freq)
   FROM QUERY_THRESHOLD a, DOC_TERM b, DOC c
 WHERE a.term = b.term AND
 b.docid = c.docid AND
          THRESHOLD = ? AND
 QUERY_NUM = ?
 GROUP BY b.doc_id
 ORDER BY 2 DESC
```

4 Run Time Performance

We developed macros to execute queries 151-200 for threshold of .10, .20, .25, .30, .33, .50, and 1.0. Each query was run with one and three concurrent sessions. The reason for this is that the processors were only being used at a capacity of between twenty and thirty percent with a single session. Increasing the number of concurrent sessions gives the machine more work to do and results in increased throughput.

Figure 1 gives the average response time for all fifty queries for each of the query thresholds. Separate lines depict results for one and three sessions. Figures 2 and 3 provide CPU and disk I/O results. It can be seen that for thresholds less than .5, workload increases linearly, and over .5, we experience an exponential change. This is consistent with Zipf's law [12].

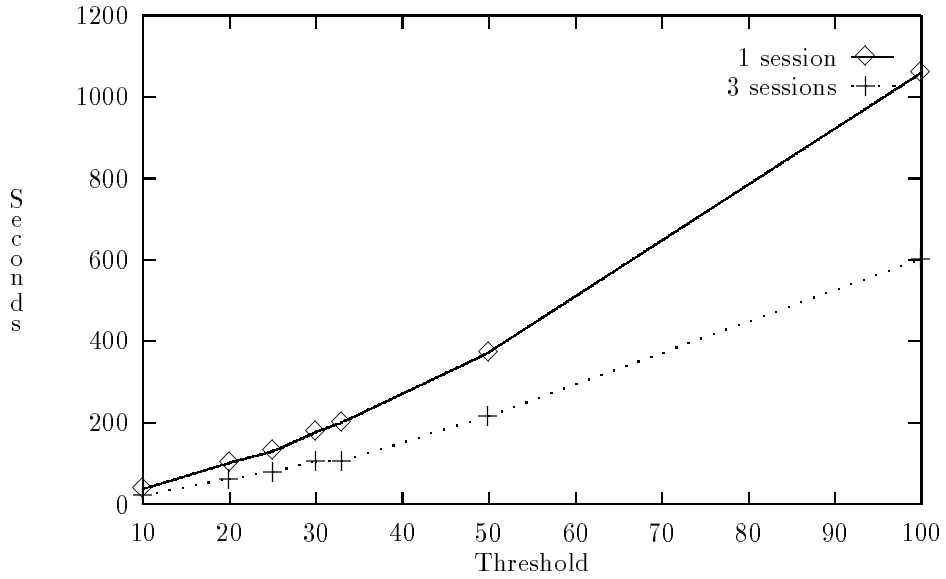

Figure 1: Avg. Response Time for Varying Query Thresholds

To further illustrate the cause of the exponential increase in run time performance, Figure 4 provides the number of tuples found in the *doc_term* relation that match a term in the query. Again, the behavior is the same, and we can see that thresholds of below .5 runs substantially faster than over .5.

Finally, we measured the amount of load balancing that takes place in using the four pro-

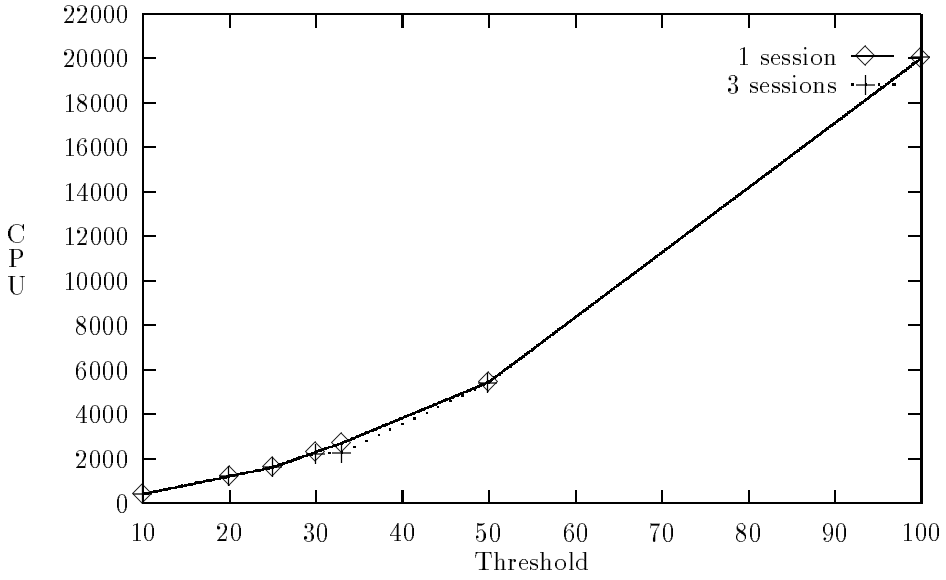

Figure 2: Avg. CPU Time for Varying Query Thresholds

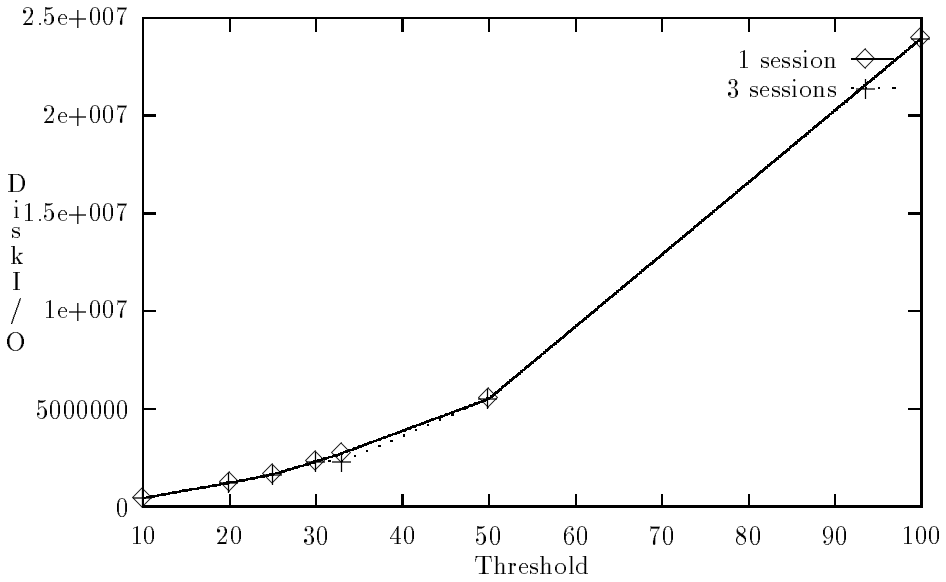

Figure 3: Total Disk I/O for Varying Query Thresholds

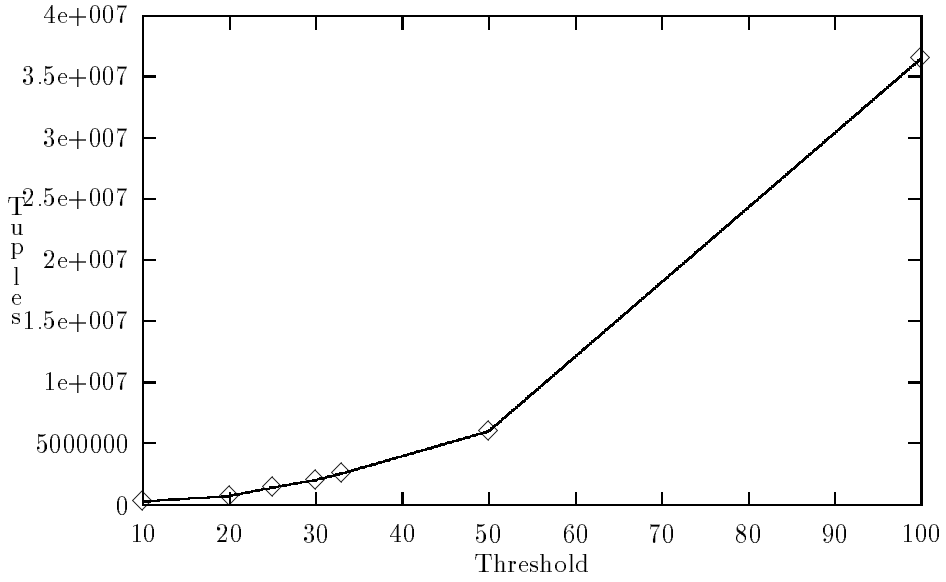

Figure 4: Total tuples in DOC_TERM accessed for Varying Query Thresholds

cessors on the DBC-1012. The table below indicates the amount of processor imbalance for CPU time: $\frac{maxcpu-mincpu}{avgcpu}$ measured at each query threshold. It can be seen that for all workloads, the processors are less then ten percent out of balance. Given this high degree of load balancing, it is reasonable to suspect that additional processors may be added to achieve required response time.

Percent of Processor Imbalance :

threshold	*CPU Time (1 session)*	*CPU Time (3 sessions)*
10	7.36	7.88
20	8.39	7.29
25	9.38	4.81
30	4.79	2.02
33	5.02	3.02
50	8.19	2.38
100	5.13	4.51

5 Accuracy

We measured precision/recall for each of the queries 151-200 for thresholds of .10, .25, .33, .50, and 1.0. Our hypothesis was that higher thresholds would result in reduced precision/recall as the query would be searching for terms that were very common across the corpus and would do little for differentiating a relevant document from an irrelevant document.

Figure 5 illustrates the number of relevant documents that were retrieved for all fifty queries using varying thresholds. Separate lines for result sets of size 100 and 200 are presented. It can be seen that as the threshold increases from ten to twenty-five, more relevant documents are found. This is reasonable to expect as a threshold of ten or twenty may omit many query terms that assist in identifying relevant documents. As the threshold increases beyond twenty-five the number of

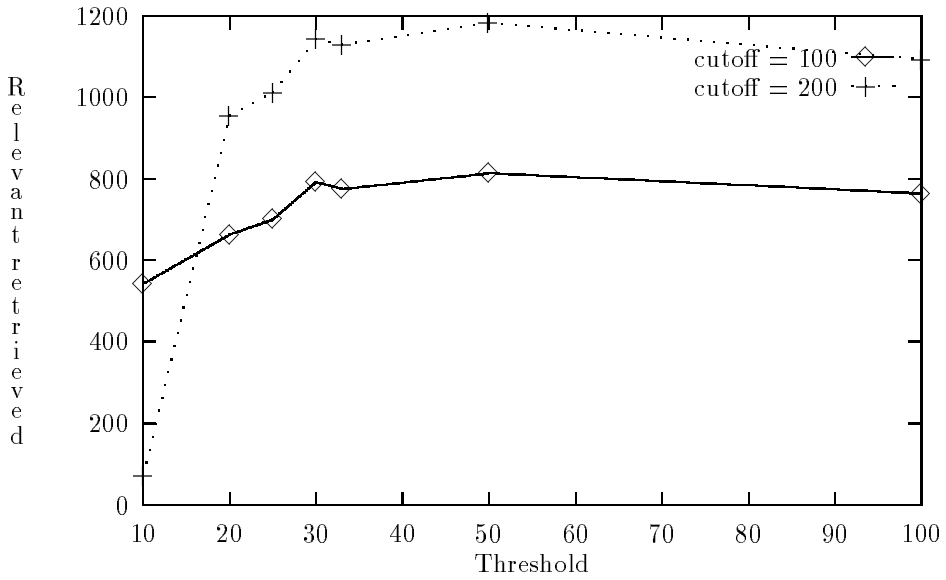

Figure 5: Effect on Query Reduction on Number Docs Retrieved that are Relevant

relevant documents retrieved drops. This is also understandable as a higher threshold will result in increased noise within the query. Hence, we have found that a query threshold of between twenty-five to thirty-three may yield good accuracy as well as dramatically improved run time performance over a query that is not filtered at all (one hundred percent threshold). These results are unofficial as they were determined after the TREC-3 submission deadline.

6 Conclusions and Future Work

Our TREC-3 effort has served as a proof-of-concept of our prior ideas that an unchanged relational DBMS may be used to serve as an engine for IR. Our experiments found that overhead was somewhat high, but tolerable for a large scale machine. An overhead of 1.97:1.00 may be reasonable for applications that have performance requirements large enough to justify a large scale parallel architecture. Given the lack of scalable parallel algorithms for parallel information retrieval, our work may be one means of easily spreading workload across large numbers of processors.

Another advantage of our approach is that structured data and text may be easily integrated. Although we only used *date* and *dateline* in our prototype, other structured fields could have easily been used. Queries that integrate both structured data and text are relatively straightforward extensions to the queries we have discussed.

We have also shown that query reduction using term frequencies may be a viable means of reducing disk I/O without significantly affecting accuracy.

Finally, it should be noted that the results given here are unofficial. Our official results came as a result of implementing Marc Damashek's approach using n-grams of size three instead of size five [6]. Interestingly, our results for the ad-hoc collection were markedly similar to those submitted by Damashek's group. We plan to incorporate this work into our relational implementation by developing algorithms that implement this work using unchanged SQL.

References

[1] E. Adams. *A Study of Trigrams and Their Feasibility as Index Terms in a Full Text Information Retrieval System.* PhD thesis, George Washington University, Department of Computer Science, 1991.

[2] E. Adams. Trigrams as index elements in full text retrieval observations and experimental results. *Proceedings of the First Annual Conference on Information and Knowledge Managementnt (CIKM '92)*, October 1992.

[3] M. Stonebraker Jeff Anton and Eric Hanson. Extending a database system with procedures. *ACM Transactions on Database Systems*, 12(3):350–376, September 1987.

[4] AT&T Global Information Systems. *Teradata DBC-1012 Concepts and Facilities*, March 1992.

[5] D. Blair. Square (specifying queries as relational expressions) as a document retrieval language). Written while working on the System R project., 1974.

[6] M. Damashek. Gauging similarity via n-grams: Text sorting, categorization, and retrieval in any language. Submitted to Science, 1994.

[7] D. Grossman. Using the relational model and part-of-speech tagging to implement text relevance. *Proceedings of the First Annual Conference on Information and Knowledge Managementnt (CIKM '92)*, October 1992.

[8] C. Lynch and M. Stonebraker. Extended user-defined indexing with application to textual databases. *Proceedings of the 14th VLDB Conference*, pages 306–317, 1988.

[9] E. Pulley. A preprocessor for integrating structured data and text. Technical Report CfIA-94-003, Center for Image Analysis, George Mason University, 1994.

[10] M. Stonebraker, H. Stettner, N. Lynn, J. Kalash, and Antonin Guttman. Document processing in a relational database system. *ACM Transactions on Office Information Systems*, 1(2):143–158, April 1983.

[11] A. Tomasic and Hector Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document information retreival systems. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information*, pages 8–17, 1993.

[12] G.K. Zipf. *Human Behaviour and the Principle of Least Effort.* Addison-Wessley, 1949.