A 3D Distributed Index(*)

Gabriel H. Tolosa[1,2]

[1]Departamento de Computacion, FCEyN, Universidad de Buenos Aires
(estudiante del Doctorado en Cs. de la Computación, director: Dr. Esteban Feuerstein)
[2]Departamento de Cs Básicas, Universidad Nacional de Lujan
(*) En colaboración con Yahoo! Research Santiago de Chile

The size of available text collections grows at very high rates. At the same time, there is a fast increment in the number of users who use information retrieval systems to find documents in that collections. Furthermore, it becomes even important to provide fast response to the users. To handle huge data volumes and high query throughput rates (for example, in a Web Search Engine), it is necesary to use parallel systems in which the data is split across a set of processors.

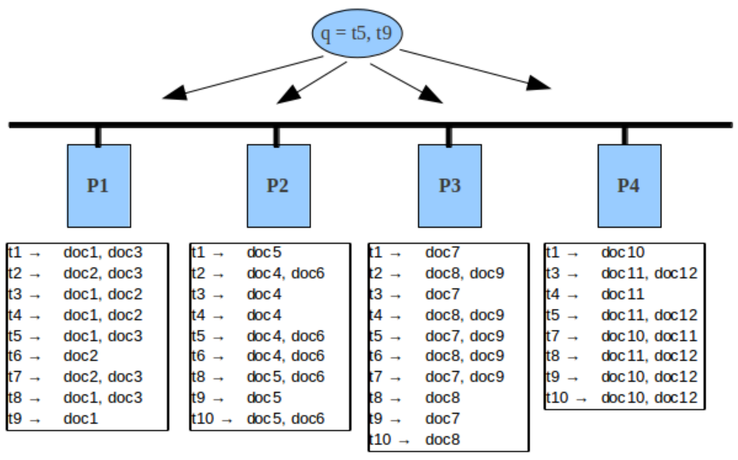
Inverted files are the most widely used data structure to implement efficient search systems that can deal with a high traffic of queries upon web-scale text collections. In its simplest form, an inverted index consists of a vocabulary table which contains the set of relevant terms found in the text collection, and a set of posting lists that contain the document identifiers. One interesting issue is the distribution of the whole inverted file onto P processors. There exist two basic approaches: document partition (local index) and term partition (global index) but some mixed approaches have been developed for special cases. This work focuses in this research problem and we propose a novel distributed index architecture.

Example: Assume a collection with twelve documents and a parallel system with four processors.

doc1 (t1, t3, t4, t5, t8, t9)
doc2 (t2, t3, t4, t6, t7)
doc3 (t1, t2, t5, t7, t8)
doc4 (t2, t3, t4, t5, t6)
doc5 (t1, t8, t9, t10)
doc6 (t2, t5, t6, t8, t10)
doc7 (t1, t3, t5, t7, t9)
doc8 (t2, t4, t6, t8, t10)
doc9 (t2, t4, t5, t6, t7)
doc10(t1, t7, t9, t10)
doc11(t3, t4, t5, t7, t8)
doc12(t3, t5, t8, t9, t10)

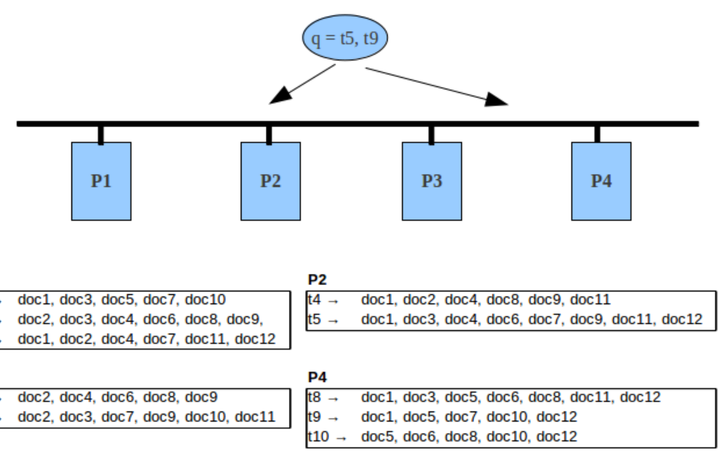
Document-partitioned index

Each processing node stores the index corresponding to a subset of the documents and every node participates in the resolution of a query.



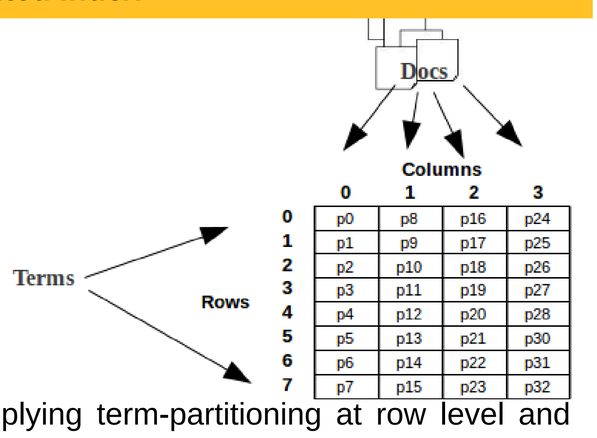
Term-partitioned index

Each processing node maintains complete index information for a subset of the terms in the collection, and each query is referred to the subset of the nodes that hold relevant information.



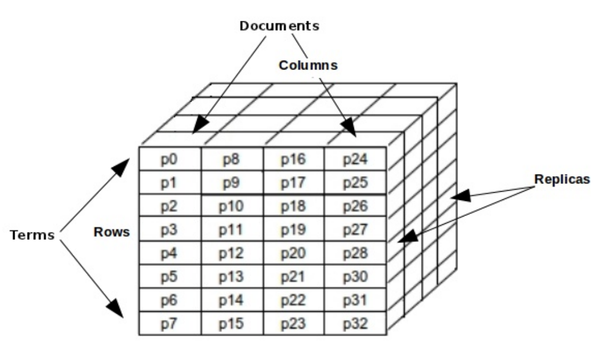
2D Distributed Index

Many different methods for distributing the inverted file onto P processors or computers and their respective query processing strategies have been proposed in the literature, focusing on optimizing for particular situations. In previous work, Feuerstein et al. introduced a novel distributed architecture for indexing, named the 2D index, that consists in arranging a set of processors in a two-dimensional array, applying term-partitioning at row level and document-partitioning at column level.



3D Distributed Index

The work presented here is an evolution of the 2D index in which we include also replication which is a common strategy to achieve fault tolerance and increase query throughput. We extend the RxC model adding D processors which act as replicas, building a 3D index. In this architecture, every processor is responsible of a set of queries but the index partitioning is replicated D times.



Results of "2D" strategy showed that it is possible to obtain significant improvements in the performance with the same number of processors but choosing an adequate number of rows (R) and columns (C). The best configuration achieves an optimal trade-off between communication cost and computation overhead, minimizing the total processing cost for a set of queries. We assume a similar behaviour in the 3D index but in this new case the optimal configuration relies on three parameters (R, C and D).

Evaluation and Preliminar Results

3D Index Performace

We tested several configurations of Rows, Columns and Replicas (RxCxD) with P=[32..2048] and found a trade-off value for each case.

Figure 1 shows the configurations with the best performance for each value of P as a function of the number of columns (x-axis).

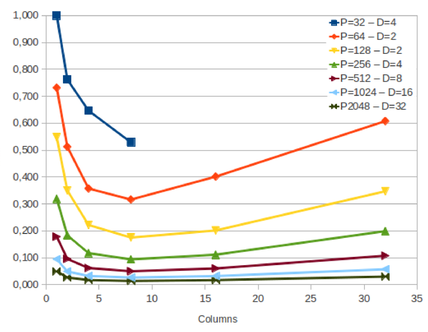


Figure 1

Besides, we found that cost reduction when P increases can be approximated with $f(x) = 23\ P^{-0.84}$ (Figure 2)

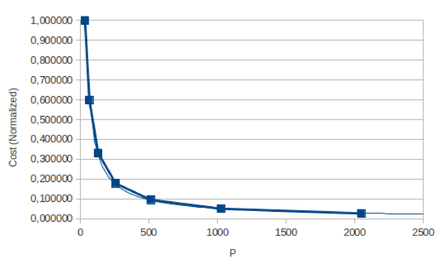


Figure 2

Escalability

To study the behaviour of the 3D Index as the collection size varies we started with the original data (base collection) and scaled it up/down. To perform this modification, we re-compute the sizes of the postings lists and made necessary adjustments to obtain the sizes of intersections for 50%, 200% y 1000% of the base collection (N=0.5, 2 and 10 respectivelly).

We run simulations for P = 512, 1024 and 2048. For each value of P we found a logaritmic curve (with coefficients of 0.29, 0.15 and 0.06, respectively) that models the increment of the total cost while N grows (Figure 3).

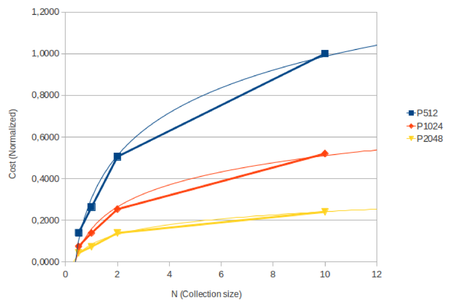


Figure 3

Pipelined Caching

We propose a novel idea for managing an "intersection cache" (where each one keeps frequently needed intersections of posting lists). This schema is applied to the set of processors that form a "column" slice of our 3D matrix (Figure 4). All processors in the slice will share their intersection cache.

For example, for a four-term query t1∩t2∩t3∩t4 the column-broker in each column looks for the intersection t1∩t2∩t3∩t4 in a particular processor of the column. If that fails, in a second step, it looks for t1∩t2∩t3 on the corresponding processor and so on. When a hit is found the partial result of the intersection is retrieved, and a schedule is prepared to complete the missing intersections.

Figure 4

Though we are conducting exhaustive tests, preliminar results showed important improvements in the performance, exchanging some extra communication cost for much less computation and disk accesses.

References

[1] E. Feuerstein, M. Marín, M. Mizrahi, V. G. Costa, and R. A. Baeza-Yates. Two-dimensional distributed inverted files. SPIRE 2009, vol 5721, Lecture Notes in CS, pp 206-213. Springer, 2009.
[2] Q. Gan and T. Suel. Improved techniques for result caching in web search engines. In Proc.of the 18th international conference on World wide web, WWW '09, pages 431-440, 2009. ACM.
[3] X. Long and T. Suel. Three-level caching for efficient query processing in large web search engines. In Proceedings of the 14th international conference on World Wide Web, WWW '05, 2005. ACM.
[4] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. Information Retrieval, 10:205-231, 2007.