FPGA-ACCELERATED INFORMATION RETRIEVAL: HIGH-EFFICIENCY DOCUMENT FILTERING

W. Vanderbauwhede, L. Azzopardi , M. Moadeli[*]

Department of Computing Science
University of Glasgow
G12 8QQ Glasgow, UK
email: [wim,leif,mahmoudm]@dcs.gla.ac.uk

ABSTRACT

Power consumption in data centres is a growing issue as the cost of the power for computation and cooling has become dominant. An emerging challenge is the development of "environmentally friendly" systems. In this paper we present a novel application of FPGAs for the acceleration of Information Retrieval algorithms, specifically, filtering streams/collections of documents against topic profiles. Our results show that FPGA acceleration can result in speed-ups of up to a factor 20 for large profiles.

1. INTRODUCTION

Servicing millions of user search requests and processing very large volumes of information requires massive amounts of computational resources that consumes huge amounts of energy. Energy costs for computing and cooling have become the dominant cost in the operation of data centers [1]. This motivates the development of energy efficient solutions for processing large amounts of data and information. Reducing the amount of energy consumed provides a win-win situation: service providers can significantly reduce their costs by consuming less energy, and the impact upon the environment is greatly reduced.

This work describes our initial steps towards evaluating the potential of FPGAs for developing environmentally friendly Information Retrieval systems. We focus on the task of information filtering, where given a set of information needs (profiles), incoming documents are matched against these profiles [2]. This task can be performed for a number of reasons in a variety of situations, for example detecting spam in incoming emails, comparing patent applications against existing patents, monitoring communications for terrorist activity, news story topic detection and tracking, etc.

When faced with large volumes of incoming documents, processing needs to be performed in real time, and therefore time based efficiency is paramount. Therefore, our aim is to filter documents efficiently in terms of both time and energy, by implementing the most computationally intensive part of the filtering application on FPGAs.

2. INFORMATION FILTERING

According to [3], "Information retrieval (IR) deals with the representation, storage, organization of and access to information items.". The main tasks undertaken to facilitate access to information are ad hoc retrieval, classification, clustering, and filtering. Fundamentally, each task requires the matching between between information items and information needs. For instance, in ad hoc retrieval a query representing the users information need is matched against documents. In Information Filtering the task is as follows: given a collection or stream of documents, matching is performed against a topic profile that represents the user's information need: this profile is characterized by a weighted feature vector, where features usually are terms. In this work, we employ the model proposed by Lavrenko and Croft [4] for filtering. Applied to the task of information filtering, the idea is that the odds of an incoming document being relevant to the topic profile is determined using a generative probabilistic language model. If a document scores above a user defined threshold then it is considered relevant to the topic profile.

The algorithm implemented on the FPGA can be expressed as follows: a document is modelled as a "bag of words", i.e. a set D of pairs (t, f) where $f \triangleq n(t, d)$ is the number of occurrences of the term t in the document d; $t \in \mathbb{N}$ is the term identifier . The profile M is a set of pairs $p = (t, w)$ where the weight $w \triangleq \log\left(\frac{(1-\lambda)P(t|M)}{P(t)} + \lambda\right)$ (see Appendix for details). Let T be the set of terms occurring both in D and M. The score of a given document

[*]W. Vanderbauwhede acknowledges support from the EPSRC. The authors thank F. Larsson of Mitrionics for technical support with the Mitrion SDK, and the IR Facility (www.ir-facility.org) and Matrixware GmbH for supporting this project, .

against a given profile is then given by:

$$score(D, M) = \sum_{\forall k \in T} f_k w_k \quad (1)$$

This function is representative of the kernel of most filtering algorithms. The main difference being the weighting of terms in profiles.

3. SYSTEM ARCHITECTURE

3.1. Development Platform

The research was carried out using an SGI Altix 4700 machine which hosts two RC100 blades. Each blade contains two Xilinx Virtex-4 LX200 FPGAs running at 100MHz; each FPGA is connected to the host platform via the SGI NUMAlink high-speed I/O interface and has access to a local 64 MB SRAM bank over a 128-bit data bus at a maximum speed of 16GB/s. The host system is an 80-core 64-bit NUMA machine, running 64-bit Linux (OpenSuSE). The processors are dual-core Itanium-2 running at 1.6GHz; each processor has direct access to 4GB of memory, but can access the complete 320-GB memory space over the NUMAlink. It is notable that the Itanium processor consumes approximately 130 watts of power [5] whereas each Virtex-4 FPGA consumes approximately 1.25 watts of power [6].

3.2. Application Architecture

The document filtering application (Fig. 1) has a client-server architecture consisting of a GUI-based client connected over TCP/IP to a Communication Server. The Communication Server acts as a proxy between the various back-end servers and the client. A typical use case starts with the user issuing a query to the Query Server, a conventional search system that returns a sorted list of hits. The user then creates a profile by selecting relevant documents from the list of hits. The complete text of all the combined documents is then used by the Profile Server to construct the profile (i.e. the list of terms and weights). The Profile Server matches the profile against the complete document collection and returns a stream of scores to the client. The modular client-server architecture facilitates benchmarking of the system as it is very easy to add a C++ reference implementation of the Profile Server running on the host CPU. As shown in Fig. 1, the FPGA-accelerated part of the application is limited to the most computationally intensive task, the matching of the documents against the profile. All other tasks are handled by the host system (Fig. 2).

The Profile server (Fig. 1) filters a stream of documents against a profile received from a client and returns a stream of scores. To evaluate the performance, both a C++ reference implementation and an FPGA-accelerated implementation were created. Both implementations have the same basic functionality: they receive the list of documents constituting the profile over a TCP/IP interface, construct the profile using a relevance model and score a memory-buffered document stream against this profile, returning a stream of document scores to the client over TCP/IP. The document stream is buffered in memory as otherwise the slow disk access would limit the performance of the application.

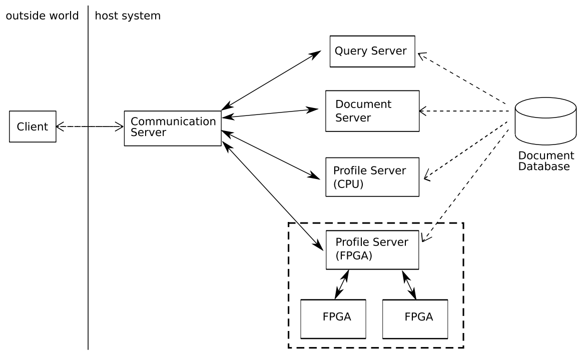

Fig. 1. System Architecture

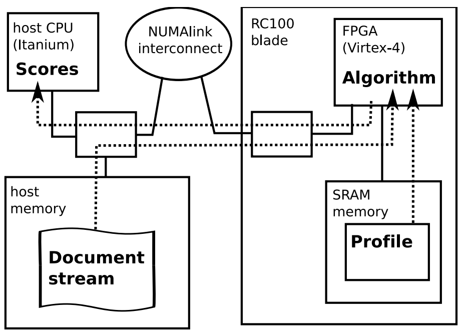

Fig. 2. FPGA SubSystem Architecture

4. APPLICATION IMPLEMENTATION

The application is implemented in C++ using the Lemur IR framework [1] and the SGI RASC libraries for interacting with the FPGA. The Lemur Toolkit is a open-source toolkit for research in IR with support for indexing and various relevancy and retrieval models. The SGI RASC (Reconfigurable Application Specific Computing) library[2] is the proprietary solution used by SGI to integrate FPGAs with host systems over the high-performance NUMAlink interconnect fabric. It defines a hardware abstaction API that provides control over each hardware element in the system.

To avoid the complexities involved in low level hardware implementation, the FPGA algorithm is implemented in Mitrion-C [7]. The Mitrion SDK[3] is used to convert Mitrion-

[1]www.lemurproject.org
[2]www.sgi.com
[3]www.mitrionics.com

C into VHDL. The produced VHDL code at this stage can be simply targeted to an FPGA device architecture. The Xilinx ISE toolchain with the XST synthesis tool are employed to create the bistream for the Virtex-4. The rest of this section provides a description of the tools, algorithms and platform adopted to implement the FPGA-based implementation of the filtering task.

4.1. Mitrion-C

The Mitrion-C language is a high-level language specifically intended for developing applications on FPGAs. It is a single-assignment dataflow language with native support for deep (vector) and wide (pipeline) parallelism. Mitrion-C provides a *stream* datatype that results in pipelined operation when used in conjunction with the *foreach* looping construct, a *vector* datatype for data-parallel operation and a *list* datatype for sequential lists. In particular, the output of a foreach loop over a stream can be filtered to produce a smaller stream (see example below). Furthermore, powerful datatypes can be constructed using the tuple construct. A final feature worth noting is the support for variable-width integer and floating-point numbers.

Algorithm 1 Example Mitrion-C code

```
type Word = typedef bits:128;
type Stream = typedef Word(..);
type DocId = typedef uint:64;
type Score = typedef int:48;
type DocScore = typedef tuple {DocId,Score};
type ScoreStream = typedef DocScore(..);

ScoreStream score_stream =
foreach ( Word w in Stream word_stream ) {
  score = ...  ; // calculate score
  keep = score > limit;
} keep ?  score
```

4.2. FPGA Implementation of Filtering Algorithm

To implement the operation described in Equation 1 efficiently on the FPGA, the key issues to be addressed are efficient lookup of the profile and efficient streaming I/O of the document stream.

4.2.1. Profile Lookup Table Implementation

For every term in the document, the application needs to look up the corresponding profile term to obtain the term weight. As most of the lookups will fail (i.e. most terms in most documents will not occur in the profile), it is important to discard the negatives first. For that reason we implemented a Bloom filter [8] in the FPGA block RAM. The higher internal bandwith of the BRAMs leads to very fast rejection of negatives. Because of the need for lookup, the profile must be implemented as some type of hash function. However, as the size of the profile is not known in advance, it is impossible to construct a perfect hash; imperfect hashes suffer from collisions which deteriorate the performance. Our solution is to use "trivial" hashing combined with binning. We partition the external SRAM in bins, every bin can contain a fixed number of profile terms b. The bin size determines the number of collisions that can be handled. To assign a profile term to a bin we simply take the lower part of the term id as the memory address, i.e. there is no actual hashing.

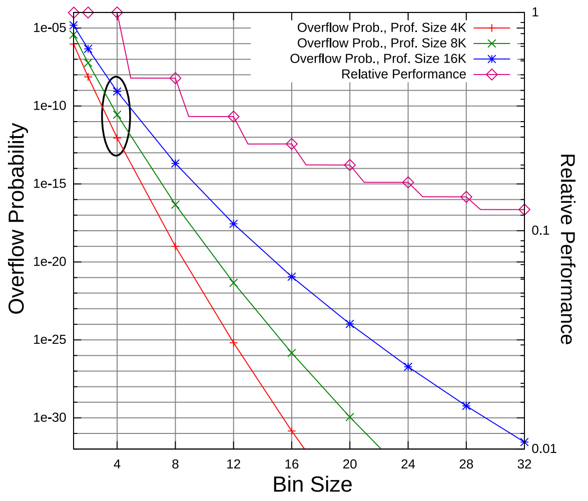

Fig. 3. Bin overflow probability for varying profile size and relative performance versus bin size

Let the SRAM memory capacity be N_M profile terms. The term id is an unsigned integer with a range depending on the vocabulary size which in our case is about 4 million terms, requiring 24 bits. The term weight is represented as an 8.32 fixed-point number, so the profile term takes 64 bits. The SRAM on the RC100 consists of 4 banks of 16MB, hence $N_M = 2^{23}$. The number of bins $n_b = \frac{N_M}{b}$ and the bin address is computed from the term id t as $(t \,\&\, (n_b - 1)).b$.

The probability x of occupancy for a bin can be expressed as follows with n_p the number of terms in the profile (combinations with replacement):

$$p(x\,|n_b, n_p) = \frac{\binom{n_b - 1 + n_p - x - 1}{n_p - x}}{\binom{n_b + n_p - 1}{n_p}} \qquad (2)$$

Using this equation we can compute the probability of

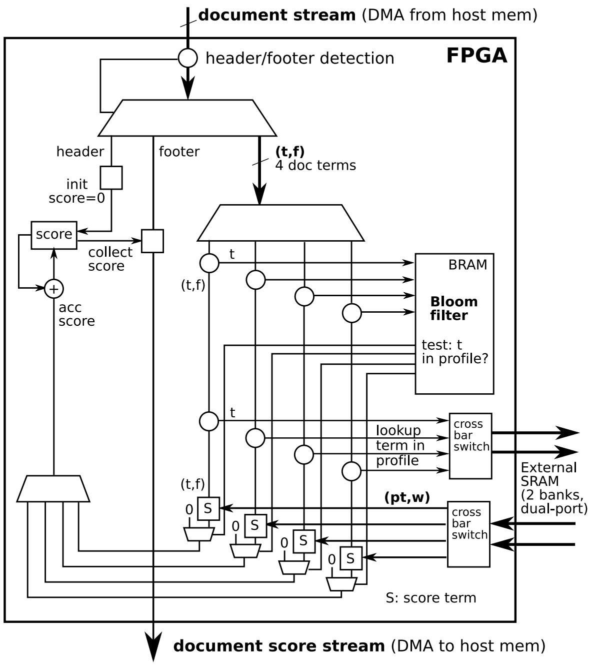

Fig. 4. Diagram of FPGA implementation of filtering application

bin overflow as a function of the bin size (and hence the number of bins, as $N_M = b.n_b$). The results are shown in Fig. 3, together with the relative performance of the lookup table. The larger the bin size, the slower the lookup; however, as the SRAM bank consists of 4 independent 64-bit addressable dual-port SRAMs, we can actually look up 4 profile terms in parallel. Consequently, the relative performance decreases as $1/\lceil b/4 \rceil$. From the figure it can be seen that even for the largest profiles (16K; in our study the largest profile was 12K, in general profiles are much smaller), the bin overflow probability for $b = 4$ (best performance) is 10^{-9}. In other words, the chance that a profile term will be discarded is less than one in a billion. It should be noted that this estimate is pessimistic because Eq. 2 assumes vocabulary size is infinite.

4.2.2. Document Stream Format

Using a bag-of-words representation (see Section 2) for the document, the document stream is a list of (document id, document term pair set) pairs. Physically, the FPGA accepts a stream 128-bit words from the NUMAlink at 1.6 GB/s. Consequently, the document stream must be encoded onto this word stream. The document term pair $d_i = (t_i, f_i)$ can be encoded in 32 bits: 24 bits for the term id (supporting a vocabulary of 16 million terms) and 8 bits for the term frequency. Thus we can combine four pairs into a 128-bit word. To mark the start and end of a document we insert header and footer words which contain the document id (64

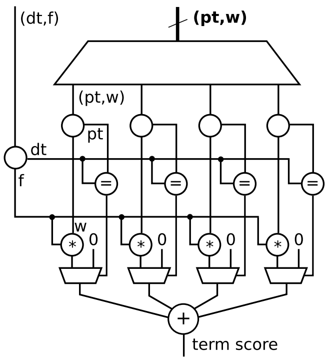

Fig. 5. Document term scoring

bits) and a marker (64 bits).

4.2.3. Lookup and Scoring

Using the lookup table architecture and document stream format as described above, the actual lookup and scoring system (Fig. 4) is quite straightforward: the input stream is scanned for header and footer words. The header word action is to set the document score to 0; the footer word action is to collect and output the document score. For every 4 terms in the document, first the Bloom filter is used to discard negatives and then 4 profile terms are read from the SRAM. The score is computed for each of these terms in parallel and added (Fig. 5). In practice, three out of four profile term ids will not match the document term id; only for the fourth the actual product is computed. The score is accumulated for all terms in the document and finally the score stream is filtered against a limit before being output to the host memory.

4.3. Host Interface

The host-FPGA interface transfers the document stream from the memory buffer to the FPGA and returns a score stream to the client. On receipt of a list of profile document ids from the client, the parent process forks off a child process that builds the actual profile, loads in onto the SRAM and runs the algorithm on the FPGA. Each child process spawns a separate output thread which buffers the scores received from the FGPA and transmits them to the client over TCP/IP, thus using the network for multiplexing the score streams. Without this thread, fluctuations in the network throughput could degrade the system performance. The main advantage of the host interface architecture is that it can easily scale to large numbers of FPGAs.

5. EXPERIMENTS

To evaluate the performance of the FPGA-accelerated filtering application, a series of experiments were performed comparing the FPGA-accelerated implementation against an optimized reference implementation written in C++ and run on the Altix. For the comparison, we used three IR test collections. Table 1 shows the collections used: TREC Aquaint, and two collection of patents from the US Patent Office (USPTO) and the European Patent Office (EPO), respectively. These collections were chosen to assess the impact of different document lengths and sizes of documents on filtering time.

Collection	# Docs	Avg. Doc. Len.	Avg. Uniq. Terms
Aquaint	1,033,461	437	169
USPTO	1,406,200	1718	353
EPO	989,507	3863	705

Table 1. Collection Statistics

To simulate a number of different filters, for each collection, profiles were constructed by selecting a random document, using the title as the query, then selecting up to n top documents returned by the query server as pseudo-relevant documents. The returned documents, were then used to construct a relevance model. The relevance model defined the profiles which each document in the collection was matched against (as if it were being streamed from the network). The number of documents in the profile n was varied from 1 to 50, to determine the impact on performance as the size of the profiles increased (both number of terms, and number of documents). This process was repeated 30 times and the average processing times were computed.

6. RESULTS AND DISCUSSION

The results are summarized in Table 2 and Figure 6. From the table, it is clear that the FPGA implementation is typically an order of magnitude faster than the standard implementation. From the figure, it can be seen that as the profile size increases (i.e. the number of terms that require matching increases) the standard implementation becomes slower

	Profile		**Processor**		
Collection	**# Docs**	**# Uniq. Terms**	**CPU** (secs)	**FPGA** (secs)	**Gain**
Aquaint	1	254	21.3	2.6	8.3x
	10	1,444	27.4	2.6	10.5x
	50	4,713	34.5	2.6	13.2x
USPTO	1	28	64.0	7.2	8.9x
	10	148	68.3	7.1	9.6x
	50	615	76.9	7.5	10.3x
EPO	1	1,327	107.3	8.4	12.7x
	10	4935	153.3	8.1	19.0x
	50	12,314	177.1	8.5	20.8x

Table 2. Performance Statistics

and slower, while the FPGA implementation remains relatively constant. This is because the FPGA implementation pipelines the profile scoring, resulting (in first order) in a constant latency independent of the profile size.

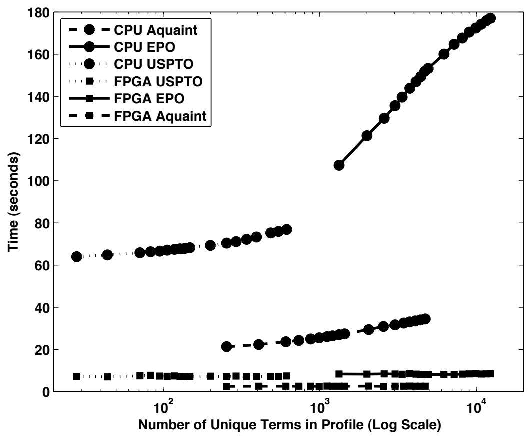

Fig. 6. Time in Secs vs. No. of Docs in Profile.

The results clearly demonstrate the potential of FPGAs for accelerating IR tasks. The speed-ups are already quite dramatic, especially for large profiles; nevertheless, there is still room for improvement. Using simulations we have verified that the FPGA algorithm scores 1 document term per 2 clock cycles. The limiting factor is the SRAM access speed which is 128 bit/cycle, requiring 2 cycles to read 4 profile terms. At a clock speed of 100MHz this means that the FPGA is capable of scoring the EPO collection in 15 seconds. The current application takes about 8.5 seconds on four FPGAs, so in principle it should be possible to increase the performance with a factor of two at least. The reason for the discrepancy lies in the streaming I/O: the document stream is transfered from user memory space to the NUMAlink via a host operating system device driver. The driver transfers a buffered block of the stream. Currently, this transfer has not been implemented optimally in terms of the transfered block size, leading to sub-optimal throughput; furthermore, using a separate thread for queuing the transfers would eliminate the transfer time from the latency.

7. CONCLUSION

In this paper, we have presented a novel application of FPGA hardware for accelerating one type of Information Retrieval task. By implementing a document filtering application using FPGAs we demonstrated reductions by up to a factor of 20 in the time taken to filter a stream of incoming documents. Furthermore, the FPGA solution delivers this speed-up at a fraction of the power of a CPU-only solution. These results demonstrate that the usage of FPGAs as "greener hardware" can deliver tremendous benefits by reducing the power consumed, and also increasing the speed of execution. Future work will be directed towards: (i) improving the performance of the current prototype, (ii) scaling the prototype up to data centre scale, and (iii) implementing more sophisticated filtering algorithms, along with other IR tasks such as ad-hoc retrieval and classification/clustering where efficiency is also paramount.

8. REFERENCES

[1] C. L. Belady, "In the data center, power and cooling costs more than the IT equipment it supports," *Electronics Cooling*, vol. 13, no. 1, 2007.

[2] N. J. Belkin and W. B. Croft, "Information filtering and information retrieval: two sides of the same coin?" *Com. ACM*, vol. 35, no. 12, pp. 29–38, 1992.

[3] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval.* Edinburgh Gate: Pearson Education Limited., 1999.

[4] V. Lavrenko and W. B. Croft, "Relevance based language models," in *Proc. of the 24th ACM SIGIR Conference*, 2001, pp. 120–127.

[5] C. McNairy and R. Bhatia, "Montecito: A dual-core, dual-thread itanium processor," *IEEE MICRO*, vol. 25, no. 2, pp. 10–20, 2005.

[6] S. Sharp. (2005, dec) Virtex-4 Dynamic Power Comparison - A Case Study. [Online]. Available: www.xilinx.com

[7] V. Kindratenko, R. Brunner, and A. Myers, "Mitrion-C application development on SGI Altix 350/RC100," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, 2007, pp. 239–250.

[8] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Com. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

Appendix: Relevance Based Language Model

While there been have been numerous kernel functions proposed for matching, one algorithm developed which provides state of the art performance is the so called "Relevance Based Language Model" proposed by Lavrenko and Croft [4]. Formally, this model can be expressed as:

$$\frac{P(D|R)}{P(D|N)} \approx \prod_{t \in D} \frac{P(t|R)}{P(t|N)}^{n(t,d)} \tag{3}$$

where D is the document, and R and N denote relevance and non-relevance to the topic profile, respectively, and the $n(t,d)$ is the number of times the term occurs in the document. The odds is computed by treating the document as set of terms which are drawn independently and identically from the relevance model $P(t|R)$ and non-relevance model $p(t|N)$. i.e. these distributions represent the probability of seeing term t given the relevance and non-relevance to the topic. If the probability of a term is high in the relevance model and low in the non relevance model, a document that contains the term will attract a higher score, and vice versa. If the odds of a document is greater than one, then a document is more likely to be relevant to the topic/profile. By setting a threshold, documents can be filtered or classified accordingly, as for example, spam, or not spam. As multiplication is a costly operation and the probability values can be very small, it is standard practice to transform the computation into a summation, by taking the log of both sides, so that the log odds ratio calculated, instead. i.e.

$$\begin{aligned} \log \frac{P(D|R)}{P(D|N)} &\approx \log \prod_{t \in D} \frac{P(t|R)}{P(t|N)}^{n(t,d)} \\ &= \sum_{t \in D} \log n(t,d) \frac{P(t|R)}{P(t|N)} \end{aligned} \tag{4}$$

The $P(t|N)$ is assumed to be probability of the term occurring in the background collection $P(t)$, which is estimated using the maximum likelihood estimator, i.e. the number of times the term occurs in the collection divided by the total number of term occurrences in the entire collection. The relevance model is estimated by :

$$P(t|R) = (1 - \lambda)P(t|M) + \lambda P(t) \tag{5}$$

where, $P(t|M)$ is the probability of the term occurring in relevant documents (see [4] for full estimation details).

Smoothing of $P(t|M)$ with $P(t)$ is required because there maybe some values of $P(t|M)$ that are zero. This would then result in the zero probability problem, where a document contain such a term would be assigned a zero probability, regardless of how many matching terms it actually contained. The impact of this smoothing on the scoring of a document, means that the odds can be split into two parts: the score of a document where the document terms are non-zero in $P(t|M)$, and the score of a document where the document terms are zero in $P(t|M)$:

$$\begin{aligned} \log \frac{P(D|R)}{P(D|N)} &\approx \sum_{t \in D \wedge M} (t,d) \log \frac{(1-\lambda)P(t|M) + \lambda P(t)}{P(t)} \\ &+ \sum_{t \in D \wedge \neg M} n(t,d) \log \frac{\lambda P(t)}{P(t)} \\ &\approx \sum_{t \in D \wedge M} n(t,d) \log \left(\frac{(1-\lambda)P(t|M)}{P(t)} + \lambda \right) \end{aligned} \tag{6}$$

To further reduce the complexity of the scoring algorithm, matching is reduced to only scoring the terms that occur in the document and in the model $p(t|M)$. Therefore we drop the second addend and the final score is an approximated by Equation 6.