TopX 2.0 at the INEX 2009 Ad-Hoc and Efficiency Tracks

–

Distributed Indexing for Top-k-Style Content-And-Structure Retrieval

Martin Theobald[1], Ablimit Aji[2], and Ralf Schenkel[3]

[1] Max Planck Institute for Informatics, Saarbrücken, Germany
[2] Emory University, Atlanta, USA
[3] Saarland University, Saarbrücken, Germany

Abstract. This paper presents the results of our INEX 2009 Ad-hoc and Efficiency track experiments. While our scoring model remained almost unchanged in comparison to previous years, we focused on a complete redesign of our XML indexing component with respect to the increased need for scalability that came with the new 2009 INEX Wikipedia collection, which is about 10 times larger than the previous INEX collection. TopX now supports a CAS-specific distributed index structure, with a completely *parallel* execution of all indexing steps, including parsing, sampling of term statistics for our element-specific BM25 ranking model, as well as sorting and compressing the index lists for our final inverted block-index. Overall, TopX ranked among the top 3 systems in both the Ad-hoc and Efficiency tracks, with a maximum value of 0.61 for iP[0.01] and 0.29 for MAiP in focused retrieval mode at the Ad-hoc track. Our fastest runs achieved an average runtime of 72 ms per CO query, and 235 ms per CAS query at the Efficiency track, respectively.

1 Introduction

Indexing large XML collections for Content-And-Structure (CAS) retrieval consumes a significant amount of time. In particular inverting (i.e., sorting) index lists produced by the XML parser constitutes a major bottleneck in managing very large XML collections such as the 2009 INEX Wikipedia collection, with 55 GB of XML sources and more than 1 billion XML elements. Thus, for our 2009 INEX participation, we focused on a complete redesign of our XML indexing component with respect to the increased need for scalability that came with the new collection. Through distributing and further splitting the index files into multiple smaller files for sorting, we managed to break our overall indexing time down to less than 20 hours on a single-node system and less than 4 hours on a cluster with 16 nodes for the complete CAS index.

As TopX originally aims at CAS queries, our basic index units are inverted lists for combined tag-term pairs, where the occurrence of each term in an XML element is propagated "upwards" the XML tree structure and the term is bound to the tag name of each element that contains it (see [8]). Content-Only (CO) queries are treated as CAS queries with a virtual $\star$ tag. Term frequencies (TF) and element frequencies (EF) are

computed for each tag-term pair in the collection individually. In summary, we used the same XML-specific extension to BM25 (generally known as EBM25) as in last years also for the 2009 Ad-hoc track and Efficiency tracks. For the EF component, we precompute an individual element frequency for each distinct tag-term pair, capturing the amount of tags under which the term appears in the entire collection. Because of the large size of the new Wikipedia collection, we approximate these collection-wide statistics by sampling over only a subset of the collection before computing the actual scores. New for 2009 was also the introduction of a static decay factor for the TF component to make the scoring function favor smaller elements rather than entire articles (i.e., the root of the documents), in order to obtain more diverse results in focused element retrieval mode (used in our two best Ad-hoc runs `MPII-COFoBM` and `MPII-COBIBM`).

2 Scoring Model

Our XML-specific extension to the popular Okapi BM25 [5] scoring model, as we first introduced it for XML ranking in 2005 [9], remained largely unchanged also in our 2009 setup. It is very similar to later Okapi extensions in [4, 6]. Notice that regular text retrieval with entire documents as retrieval units is just a special case of the below ranking function, which in principle computes a separate Okapi model for each element type individually.

Thus, for content scores, we make use of collection-wide element statistics that consider the *full-content* of each XML element (i.e., the recursive concatenation of all its descendants' text nodes its XML subtree) as a bag of words:

1) the *full-content term frequency*, $ftf(t,n)$, of term t in an element node n, which is the number of occurrences of t in the full-content of n;
2) the *tag frequency*, N_A, of tag A, which is the number of element nodes with tag A in the entire corpus;
3) the *element frequency*, $ef_A(t)$, of term t with regard to tag A, which is the number of element nodes with tag A that contain t in their full-contents in the entire corpus.

The score of a tag-term pair of an element node n with tag name A with respect to a content condition of the form `//A[about(., t)]` (in NEXI [10] syntax), where `A` either matches the tag name A or is the tag wildcard $\star$, is then computed by the following BM25-based formula:

$$score(n, \texttt{//T[about(., t)]}) = \frac{(k_1+1)\ ftf(t,n)}{K + ftf(t,n)} \cdot \log\left(\frac{N_A - ef_A(t) + 0.5}{ef_A(t) + 0.5}\right)$$

$$\text{with} \quad K = k_1 \left((1-b) + b \frac{\sum_{t'} ftf(t',n)}{avg\{\sum_{t'} ftf(t',n') \mid n' \text{ with tag } A\}}\right)$$

For 2009, we used values of $k_1 = 2.0$ and $b = 0.75$ as Okapi-specific tuning parameters, thus changing k_1 from the default value of 1.25 (as often used in text retrieval) to 2.0 in comparison to 2008 (see also [1] for tuning BM25 on INEX data). Consequently,

for an `about` operator with multiple terms, the score of an element satisfying this tag constraint is computed as the sum over all the element's content scores, i.e.:

$$score(n, \texttt{//T[about(., } t_1 \ldots t_m \texttt{)]}) = \sum_{i=1}^{m} score(n, \texttt{//T[about(., } t_i \texttt{)]})$$

Moreover, for queries with multiple support elements (like for example in the query `//A//B[about(., t)]`), we assign a small and constant score mass c for each supporting tag condition that is matched (like for the tag `A` in this example). This structural score mass is then aggregated with the content scores, again using summation. In our INEX 2009 setup (just like in 2008), we have set $c = 0.01$. Note that our notion of tag-term pairs enforces a strict matching of a query's target element, while content conditions and support elements can be relaxed (i.e., be skipped on-the-fly) by the non-conjunctive query processor [8]. Also, content scores are normalized to $[0, 1]$.

2.1 2009 Extensions

Decay Factor for Term Frequencies. New for 2009 was the introduction of a static decay factor for the ftf component to make the scoring function favor smaller elements rather than entire articles (i.e., the root of the documents), in order to obtain more diverse results in focused element retrieval mode. With respect to the fairly deep structure of the new 2009 collection, we chose a relatively high decay factor of 0.925. That is, in addition to summing up the ftf values of each tag-term pair among the children of an XML element (recursively upwards to the root), we also multiply the ftf value from each of the child nodes by 0.925 before propagating these values upwards.

Sampling for Element Frequencies. With very large, diverse XML collections and very many distinct (but mostly infrequent) tag-term pairs, exact element frequencies as needed for the ef component cannot easily be kept in memory anymore. An additional difficulty in a distributed indexing setting is that these statistics need to be shared among peers. Therefore, we introduced a sampling phase for these combined tag-term frequencies, which however generates approximate statistics only. During the sampling phase, we scan (i.e., keep) only a fixed amount of tag-term statistics in memory from the XML parser output at each of the distributed nodes in the network individually. Tag-term pairs for which no statistics are kept in memory after this sampling phase are smoothed by an ef value of 1 when materializing the above scoring function.

3 Distributed Indexing

Indexing an XML collection with TopX consists of a 3-pass process: 1) parsing the XML documents (using a standard SAX parser) and hashing individual tag-term pairs and navigational tags into a distributed file storage; 2) sampling these files for the BM25-specific ef statistics for all tag-term pairs and materializing the BM25 model; and 3) sorting these BM25-scored files to obtain their final inverted block structure and compressing the blocks into a more compact binary format. While we are keeping all

intermediate index files of steps 1 and 2 in a simple (GZip'ed) ASCII format, our final block-index structure created in step 3 is stored in a customized (compressed) binary format as described in [7], which can be decompressed much faster than a GZip format.

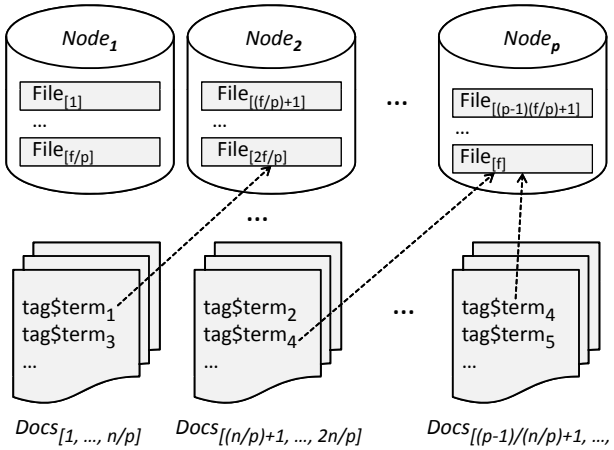

Fig. 1. Two-level hashing of tag-term pairs onto network nodes and index files.

The basic hashing phase is illustrated in Figure 1. We are given a collection of n XML documents yielding m distinct tag-term pairs, f files to hold our inverted index, and p distributed nodes (e.g., a compute cluster, or peers in a network). Before we start the actual indexing phase, the document collection is partitioned into n/p equally sized chunks, and the chunks are distributed over the p nodes. During indexing, every compute node is used for parsing and storing the index files at the same time, i.e., every node has write access to every other node in the network. Let $hash(t_i)$ be the hash code of tag-term pair t_i for all $i = 1, \ldots m$, then $(hash(t_i) \mod f)$ denotes the file identifier where the inverted list for t_i is stored. Moreover, $(hash(t_i) \mod f \mod p)$ then denotes the node identifier at which the file containing t_i is located. This ensures that all tag-term pairs from the entire collection that share the same hash code are stored on the same node and in the same index file. The reason to further partition the index into $f \geq p$ files is that, on each compute node, we can sort multiple such files concurrently in a multi-threaded fashion. Multiple smaller files can of course be sorted more efficiently than a single large file. Thus, every index file contains at least one but possibly more inverted lists. For our INEX 2009 indexing experiments, we used $f = 256$ index files which were distributed over $p = 16$ nodes.

This simple two-level hashing allows for a MapReduce-like [2] but highly specialized form of distributed indexing. After the initial parsing and hashing phase (corresponding to the Map phase in MapReduce), all files needed for materializing the above scoring function are readily available per node, and thus the sampling and scoring can be kept perfectly parallel (corresponding to the Reduce phase in MapReduce). Since all nodes can operate independently in the second phase, this approach allows for a substantially more lightweight Reduce phase than in a classical MapReduce setting. The only data structure that is finally shared across all nodes is a dictionary (see [7]) that

maps a tag-term key from a query back to a file (including the byte-offset within that file) as entry point to the respective inverted list in the distributed storage. Our dictionary maps a 64-bit hash key computed from the combined tag-term string of t_i onto a 64-bit value whose upper 8 bits encode the node id, whose middle 12 bits encode the file id that contains the corresponding inverted list, and whose lower 44 bits encode the byte offset within that file in order to mark the beginning of the inverted list. In this 64-bit setting, we can address up to $2^8 = 256$ nodes with up to $2^{12} = 4,096$ files, each file with a maximum size of $2^{44} = 16$ Terabytes. Of course, the same hash function needs to be used for both indexing and query processing.

This mapping step works particularly well for a CAS-based index, as it is using tag-term pairs as keys to access the inverted lists. In particular, the resulting index files are more uniform in length as compared to a plain (i.e., CO-style) text index because of the much larger amount of distinct tag-term keys in the CAS case. As such, it can scale to almost arbitrary amounts of index files and nodes in a network. Also, a similar hashing step is performed for the inverted tag lists, which results in a distinct set of inverted files for the navigational tag conditions as needed by TopX (see [7]). These files are less uniform in size (due to the much lower amount of distinct tag keys in the collection), but they are anyway much more compact and can benefit from a higher cache locality.

4 Query Processing

TopX is a top-k engine for XML with non-conjunctive XPath evaluations. It supports dynamic top-k-style index pruning for both CO and CAS queries. In dynamic pruning, the traversal of index list scans at query processing time can be pruned early, i.e., when no more candidate elements can make it into the top-k list anymore. Also, since TopX was designed as a native XML engine based on element retrieval, relevant passages are identified based on the XML elements that embrace them.

4.1 Retrieval Modes

Article Mode. In Article mode, all CO queries (including `//*` queries) are rewritten into `/article` CAS conditions. Thus we only return entire `article` elements as target element of the query. This mode conforms to a regular document retrieval mode with our BM25 model collapsing into a document-based scoring model (however including the per element-level decay factor of the TF component). Article mode can thus be seen as a simplest-possible form of focused element retrieval, as entire articles are always guaranteed to be overlap-free.

CO Mode. In CO mode, any target element of the collection is allowed as result. CO queries are processed by TopX equivalently to queries with `//*` as structural condition and exactly one `about` operator with no further structural constraints as filter predicate. The $*$ is treated like a virtual tag name and is fully materialized into a distinct set of corresponding $*$-term pairs directly at indexing time. That is, a $*$-term pair is always generated in addition to any other tag-term pair, with individual TF and EF statistics for these $*$ tags, which roughly doubles the index size. As a new extension in our 2009

experiments, we cut off very small elements of less than 24 terms (as a form of static index pruning) from these CO-related index lists. At query processing time, a CO query can be processed by directly accessing these precomputed (inverted) CO lists.

CAS Mode. In CAS mode, TopX allows for the formulation of arbitrary path queries in the NEXI [10] or XPath 2.0 Full-Text [10] syntax. As an optimization, leading $\star$ tags are rewritten as `article` tags, for CAS queries that were otherwise led by a `//`$\star$ step.

4.2 Optimizations

Focused, Best-In-Context, and Thorough Modes. We performed experiments with Focused, Best-In-Context and Thorough runs. Element overlap in the *Focused* and *Best-In-Context* retrieval modes is eliminated on-the-fly during query processing by comparing the pre- and post-order labels [9] of elements already contained in the top-k list with the pre- and post-order label of each element that is about to be inserted into the top-k list. During query processing, this top-k list is a constantly updated queue. Thus, if an element is a descendant of another parent element that already is in the top-k queue and the descendant has a higher score than its parent, then the parent is removed from the queue and the descendant is inserted; if otherwise an element is a parent of one or more descendants in the top-k queue, and the parent has a higher score than all the descendants, then all the descendants are removed from the top-k queue and the parent is inserted. In *Thorough* retrieval mode, this overlap check is simply omitted.

Caching. Entire index lists (or their prefixes in case of dynamic top-k pruning) can be cached by the TopX engine, and the decoded and decompressed data structures for each index list can directly be reused by the engine for subsequent queries. Thus, when running over a *hot cache*, only joins and XPath evaluations are carried out over these cached index lists at query processing time, while physical disk accesses can be almost completely avoided for query conditions that were already processed in a previous query.

5 Results

We next present the results for our Ad-hoc and Efficiency track experiments. While indexing was performed on a distributed cluster of 16 nodes, the final runs were performed on a single 3Ghz AMD Opteron Quad-Core machine with 16 GB RAM and a RAID5 storage, with all index files being copied to the local storage.

5.1 Ad-Hoc Track

Figures 2, 3 and 4 depict the iP[x] and gP[x] plots of all runs submitted to the Ad-Hoc track as functions of the recall x, for Focused, Best-In-Context and Thorough modes, respectively (see also [3] for an overview of current INEX metrics). At iP[0.01] (Focused and Thorough) and gP[0.01] (Best-In-Context), the best TopX runs rank at positions 3, 3 and 7, respectively, when grouping the runs by participant id (runs denoted

as `p10-MPII-COFoBM` and `p10-MPII-COBIBM`). For all retrieval modes, we generally observe a high early (interpolated or generalized) precision rate (i.e., for iP[0.01] and gP[0.01], each at a recall of 1%), which is an excellent behavior for a top-k engine. Both our top runs used a true focused element retrieval mode, with CO queries being rewritten into `//`$\star$ CAS queries, i.e., any result element type was allowed as result. Thus our CO runs achieved higher iP[0.01] (and MAiP) values than our Article runs, as opposed to 2009.

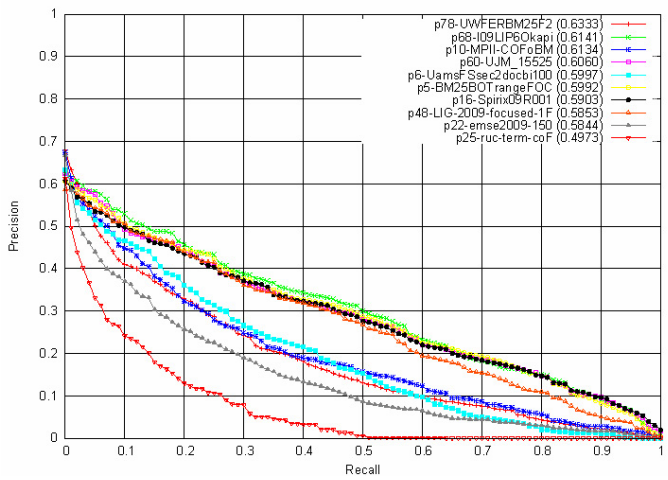

Fig. 2. iP[x] for top runs in Focused mode, Ad-hoc track.

5.2 Efficiency Track

Figure 5 depicts the iP[x] plots for all Focused type A runs submitted to the Efficiency track. The best TopX run (`TopX2-09-ArHeu-Fo-150-Hot`) is highlighted. Figures 6 and 7 depict the iP[0.01] plots of all Focused runs submitted to the Efficiency track in comparison to their runtime. Our fastest runs achieved an average runtime for type A topics of 72 ms per CO query, and 235 ms per CAS query at the Efficiency track, respectively (denoted as `TopX2-09-ArHeu-Fo-15-Hot` and `TopX2-09-CASHeu-Fo-15-Hot`). TopX ranks among the top engines, with a very good retrieval quality vs. runtime trade-off (compare also Tables 1 and 2). Our fastest runs operated over a hot cache and employed a heuristic top-k stopping condition (denoted as *Heu*, thus terminating query evaluations after the first block of elements was read and merged from each of the inverted lists that are related to the query (see [7]). Unsurprisingly, our best Article runs on type A topics (70 ms) were slightly faster than our best CO runs (72 ms), and about a factor of 3 faster than our CAS runs (235 ms), at a comparable result quality as in the Ad-hoc track. For type B topics, containing CO topics with partly more than 90 keywords, average runtimes were more than an order

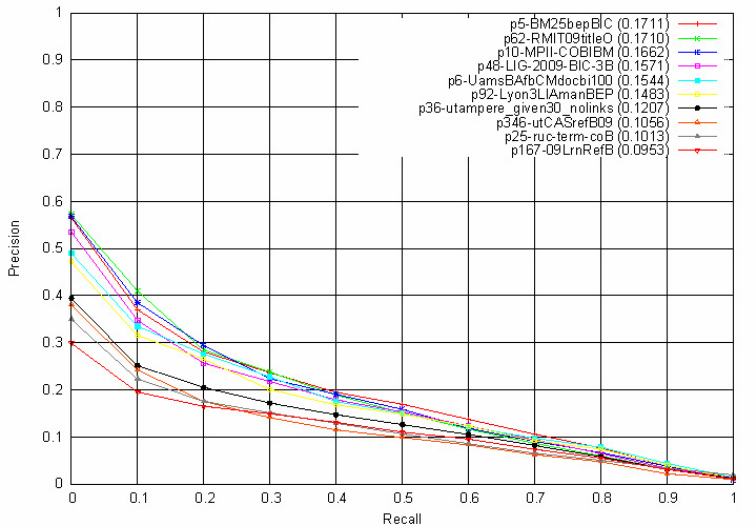

Fig. 3. gP[x] for top runs in Best-In-Context mode, Ad-hoc track.

of magnitude worse than for type A topics. Altogether, TopX was the only engine to submit other than Article runs to the Efficiency track.

Part.ID	Run ID	Type	iP[0.00]	iP[0.01]	iP[0.05]	iP [0.10] [ms]	MAiP [ms]	avg total [ms]	k	Mode
10	TopX2-09-Ar-Fo-15-Hot	Article	0.598	0.583	0.494	0.397	0.178	84.0	15	CO
10	TopX2-09-ArHeu-Fo-1500-Hot	Article	0.597	0.586	0.530	0.475	0.275	301.2	1500	CO
10	TopX2-09-ArHeu-Fo-150-Hot	Article	0.598	0.588	0.531	0.474	0.252	87.2	150	CO
10	TopX2-09-ArHeu-Fo-15-Hot	Article	0.589	0.577	0.482	0.398	0.175	69.8	15	CO
10	TopX2-09-CAS-Fo-15-Cold	Focused	0.546	0.480	0.423	0.355	0.138	467.7	15	CAS
10	TopX2-09-CAS-Fo-15-Hot	Focused	0.545	0.493	0.418	0.350	0.137	379.2	15	CAS
10	TopX2-09-CASHeu-Fo-15-Hot	Focused	0.525	0.468	0.358	0.304	0.124	234.9	15	CAS
10	TopX2-09-CO$-Fo-15-Hot	Focused	0.645	0.567	0.406	0.285	0.135	125.6	15	CO
10	TopX2-09-CO-Fo-15-Hot	Focused	0.641	0.564	0.405	0.291	0.130	338.2	15	CO
10	TopX2-09-COHeu-Fo-15-Hot	Focused	0.507	0.429	0.306	0.196	0.079	71.8	15	CO

Table 1. Summary of all TopX Efficiency runs, type A queries.

6 Conclusions

TopX was one of the few engines to consider CAS queries in the Ad-hoc track over the new 2009 Wikipedia collection, and even the only engine in the Efficiency track that processed CAS queries at all. In our ongoing work, we already started looking into further XPath Full-Text operations, including phrase matching and proximity-based ranking for both CO and CAS queries, as well as top-k support for more complex XQuery

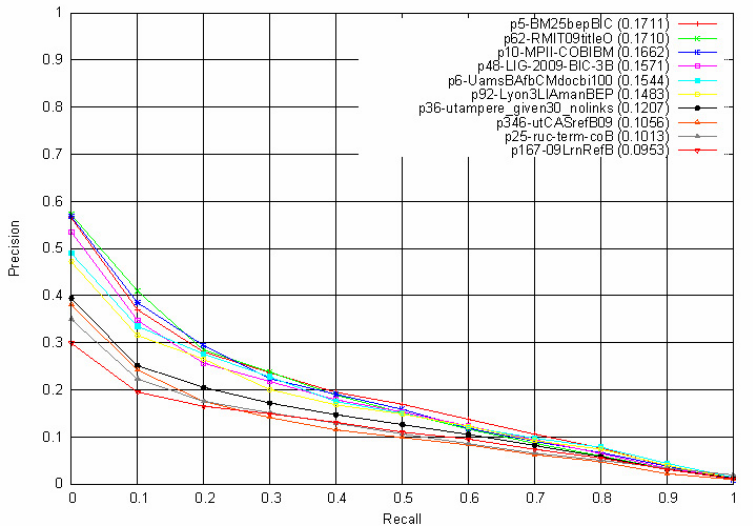

Fig. 4. iP[x] for top runs in Thorough mode, Ad-hoc track.

Part.ID	Run ID	Type	iP[0.00]	iP[0.01]	iP[0.05]	iP [0.10] [ms]	MAiP [ms]	avg total [ms]	k	Mode
10	TopX2-09-Ar-TOP15-Hot	Article	0.440	0.427	0.362	0.315	0.119	4163.2	15	CO
10	TopX2-09-ArHeu-TOP1500-Hot	Article	0.443	0.434	0.381	0.358	0.206	2412.0	1500	CO
10	TopX2-09-ArHeu-TOP150-Hot	Article	0.443	0.433	0.381	0.358	0.193	2260.2	150	CO
10	TopX2-09-ArHeu-TOP15-Hot	Focused	0.431	0.418	0.344	0.303	0.118	2205.4	15	CO
10	TopX2-09-CAS-TOP15-Cold	Focused	0.442	0.428	0.363	0.316	0.119	4352.3	15	CAS
10	TopX2-09-CAS-TOP15-Hot	Focused	0.440	0.427	0.357	0.315	0.118	4685.1	15	CAS
10	TopX2-09-CASHeu-TOP15-Hot	Focused	0.431	0.418	0.344	0.303	0.118	2293.1	15	CAS
10	TopX2-09-COHeu-TOP15-Hot	Focused	0.364	0.329	0.221	0.175	0.066	497.8	15	CO

Table 2. Summary of all TopX Efficiency runs, type B queries.

constructs. Our long-term goal is to make TopX a full-fledged, open-source indexing and search platform for the W3C XPath 2.0 and XQuery 1.0 Full-Text standards. While we believe that our distributed indexing strategy already scales to future XML indexing needs (with many Terabytes potential index size), making also the search process truly distributed will be a challenging topic for future work.

References

1. C. L. A. Clarke. Controlling overlap in content-oriented XML retrieval. In R. A. Baeza-Yates, N. Ziviani, G. Marchionini, A. Moffat, and J. Tait, editors, *SIGIR*, pages 314–321. ACM, 2005.
2. J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI 2004*, pages 137–150, 2004.

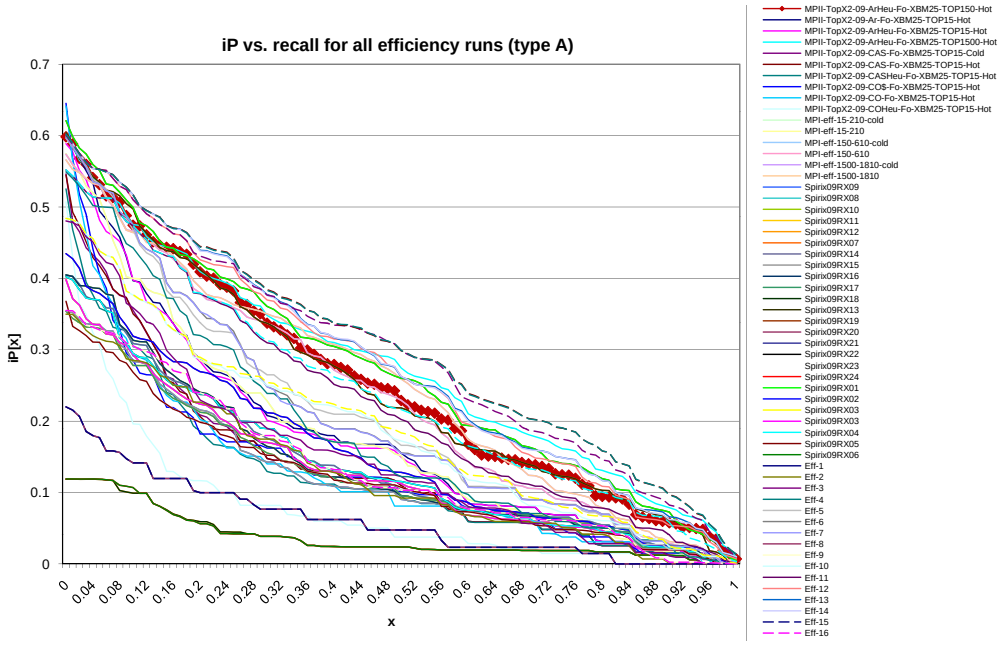

Fig. 5. iP[x] for all Efficiency runs, type A queries.

3. J. Kamps, J. Pehcevski, G. Kazai, M. Lalmas, and S. Robertson. INEX 2007 evaluation measures. In N. Fuhr, J. Kamps, M. Lalmas, and A. Trotman, editors, *INEX*, volume 4862 of *Lecture Notes in Computer Science*, pages 24–33. Springer, 2007.
4. W. Lu, S. E. Robertson, and A. MacFarlane. Field-weighted xml retrieval based on bm25. In N. Fuhr, M. Lalmas, S. Malik, and G. Kazai, editors, *INEX*, volume 3977 of *Lecture Notes in Computer Science*, pages 161–171. Springer, 2005.
5. S. E. Robertson, S. Walker, M. Hancock-Beaulieu, M. Gatford, and A. Payne. Okapi at TREC-4. In *TREC*, 1995.
6. S. E. Robertson, H. Zaragoza, and M. Taylor. Simple BM25 extension to multiple weighted fields. In D. Grossman, L. Gravano, C. Zhai, O. Herzog, and D. A. Evans, editors, *CIKM*, pages 42–49. ACM, 2004.
7. M. Theobald, M. AbuJarour, and R. Schenkel. TopX 2.0 at the INEX 2008 Efficiency Track. In S. Geva, J. Kamps, and A. Trotman, editors, *INEX*, volume 5631 of *Lecture Notes in Computer Science*, pages 224–236. Springer, 2008.
8. M. Theobald, H. Bast, D. Majumdar, R. Schenkel, and G. Weikum. TopX: efficient and versatile top-k query processing for semistructured data. *VLDB J.*, 17(1):81–115, 2008.
9. M. Theobald, R. Schenkel, and G. Weikum. An efficient and versatile query engine for TopX search. In K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson, and B. C. Ooi, editors, *VLDB*, pages 625–636. ACM, 2005.
10. A. Trotman and B. Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In N. Fuhr, M. Lalmas, S. Malik, and Z. Szlávik, editors, *INEX*, volume 3493 of *Lecture Notes in Computer Science*, pages 16–40. Springer, 2004.

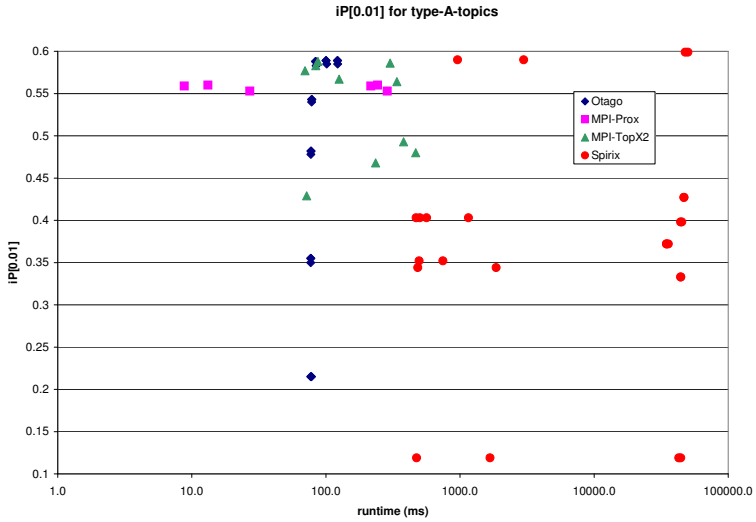

Fig. 6. Runtime vs. iP[0.01] for all Efficiency runs, type A queries.

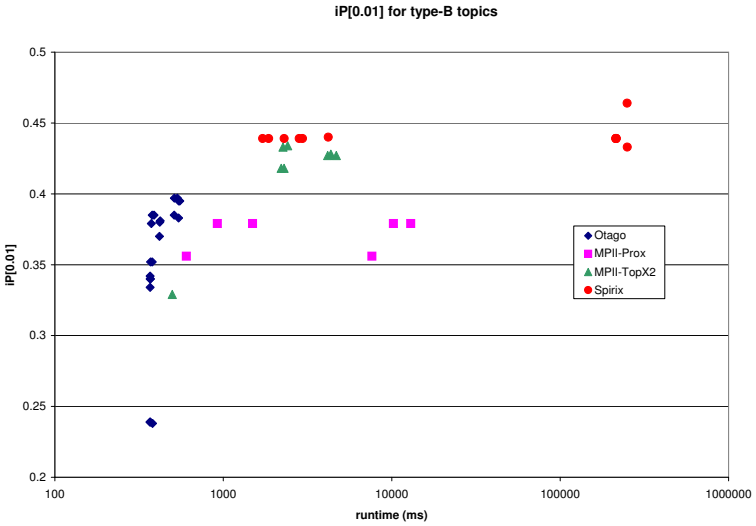

Fig. 7. Runtime vs. iP[0.01] for all Efficiency runs, type B queries.