A Multi-layer Collaborative Cache for Question Answering *

David Dominguez-Sal[1], Josep Lluis Larriba-Pey[1], and Mihai Surdeanu[2]

[1] DAMA-UPC, Universitat Politècnica de Catalunya,
Jordi Girona 1,3 08034 Barcelona, Spain
{ddomings,larri}@ac.upc.edu
[2] TALP, Universitat Politècnica de Catalunya,
Jordi Girona 1,3 08034 Barcelona, Spain
surdeanu@lsi.upc.edu

Abstract. This paper is the first analysis of caching architectures for Question Answering (QA). We introduce the novel concept of multi-layer collaborative caches, where: (a) each resource intensive QA component is allocated a distinct segment of the cache, and (b) the overall cache is transparently spread across all nodes of the distributed system. We empirically analyze the proposed architecture using a real-world QA system installed on a cluster of 16 nodes. Our analysis indicates that multi-layer collaborative caches induce an almost two fold reduction in QA execution time compared to a QA system with local cache.

1 Introduction

Question Answering (QA) systems are tools that extract short answers from a textual collection in response to natural language questions. For example, the answer to the question "How many countries are part of the European Union?" is "25". This functionality is radically different from that of typical information retrieval systems, e.g. Google or Yahoo, which return a set of URLs pointing to documents that the user has to browse to locate the answer.

A Question Answering service, like some other of today's applications, demands huge amounts of CPU and I/O resources that a single computer can not satisfy. The usual approach to implement systems with such requirements consists of networks of computers that work in parallel. On the other hand, the use of caching techniques reduces significantly the computing requirements thanks to the reuse of previously accessed data. Our target is to add the infrastructure to get the benefits of parallelism and caching to a QA system.

* David Dominguez is supported by a grant from the *Generalitat de Catalunya* (2005FI00437). Mihai Surdeanu is a research fellow within the Ramón y Cajal program of the Ministerio de Educación y Ciencia. The authors want to thank *Generalitat de Catalunya* for its support through grant number GRE-00352 and Ministerio de Educación y Ciencia of Spain for its support through grant TIN2006-15536-C02-02, and the European Union for its support through the Semedia project (FP6-045032).

In this paper we propose a collaborative caching architecture for distributed QA. We propose a new multi-layer cache where we store the data according to its semantics. We identify two types of data used by the two resource-intensive processes of a QA application, and according to it, we implement a two layer cache in our system. We show that these data sets can be cached cooperatively, with an increase in the application's performance over the caching of one or the other data sets alone. Furthermore, we do not limit the architecture to local data accesses, we provide and evaluate techniques that make it possible to find and retrieve data from remote computers in order to save disk accesses and computing cycles.

We tested all our results in a real Question Answering system. We show that the use of cooperative multi-layer distributed caches improves over the use of local multi-layer caches alone. Furthermore, the results obtained in a distributed environment reach from almost two, up to five fold improvements of the hit rates; the execution time is reduced by 50% to 60% on a 16 processor paralel computer, without parallelizing the QA task.

This paper is organized as follows. In Section 2 we describe our QA system. Section 3 explains the cache architecture for the QA system. Finally, Section 4 contains the empirical analysis of the proposed cache architecture. We end with a review of related work and conclusions.

2 Question Answering

The QA system implemented uses a traditional architecture, consisting of three components linked sequentially: *Question Processing* (QP), *Passage Retrieval* (PR), and *Answer Extraction* (AE). We describe these components next.

Question Processing: the QP component detects the type of the expected answer for each input question, i.e. a numeric answer for the question above. We model this problem as a supervised Machine Learning (ML) classification problem: (a) from each question we extract a rich set of features that include lexical, syntactic, and semantic information, and (b) using a corpus of over 5,000 questions annotated with their expected answer type, we train a multi-class Maximum Entropy classifier to map each question into a known answer type taxonomy [1].

Passage Retrieval: our PR algorithm uses a two-step query relaxation approach: (a) in the first step all non-stop question words are sorted in descending order of their priority using only syntactic information, e.g. a noun is more important than a verb, and (b) in the second step, the set of keywords used for retrieval and their proximity is dynamically adjusted until the number of retrieved documents is sufficient, e.g. if the number of documents is zero or too small we increase the allowable keyword distance or drop the least-significant keyword from the query. The actual information retrieval (IR) step of the algorithm is implemented using a Boolean IR system[3] that fetches only documents

[3] http://lucene.apache.org

that contain *all* keywords in the current query. PR concludes with the elimination of the documents whose relevance is below a certain threshold. Thus the system avoids applying the next CPU-intensive module for documents that are unlikely to contain a good answer.

Answer Extraction: the AE component extracts candidate answers using an in-house *Named Entity Recognizer* [2], and ranks them based on the properties of the context where they appear. We consider as candidate answers all entities of the same type as the answer type detected by the QP component. Candidate answers are ranked using a formula inspired from [3].

This QA system obtained state-of-the-art performance in one recent international evaluation. More details on the system architecture and the evaluation framework are available in [4].

QA systems are excellent solutions for very specific information needs, but the additional functionality provided, i.e. query relaxation and answer extraction, does not come cheap: using the set of questions from previous QA evaluations[4] we measured an average response time of 75 seconds/question. The execution time is distributed as follows: 1.2% QP, 28.8% PR, and 70.0% AE. This brief analysis indicates that QA has two components with high resource requirements: PR, which needs significant I/O to extract documents from the underlying document collection, and AE, which is CPU-intensive in the extraction and ranking of candidate answers.

3 A Multi-layer Distributed Cache for Question Answering

As previously mentioned, QA has two resource intensive components that can benefit from caching: PR and AE which are I/O and CPU intensive respectively. This observation motivates the architecture proposed in this paper: we introduce a two-layer distributed caching architecture for QA, with one layer dedicated to each resource-intensive component.

Figure 1 depicts the architecture of the distributed QA system extended with multi-layer caching on each node. We consider that each node has a finite amount of memory for the cache. Figure 1 shows that, in each node, the cache is divided into two layers: the first dedicated to caching PR results and the second used to cache AE information. The PR layer caches only the raw text documents retrieved from disk. The AE layer stores passages, which are the retrieved documents as well as their syntactic and semantic analysis. An entry is stored either in the PR layer or in the AE layer because the cache architecture is inclusive: a document cached in the PR layer will hit in the PR phase but miss in the AE phase, while a passage cached in the AE layer will hit in both.

Note that: (a) due to the filtering at the end of PR, only a subset of the documents accessed in PR reach AE and (b) an AE cache unit is significantly

[4] http://trec.nist.gov/data/qa.html

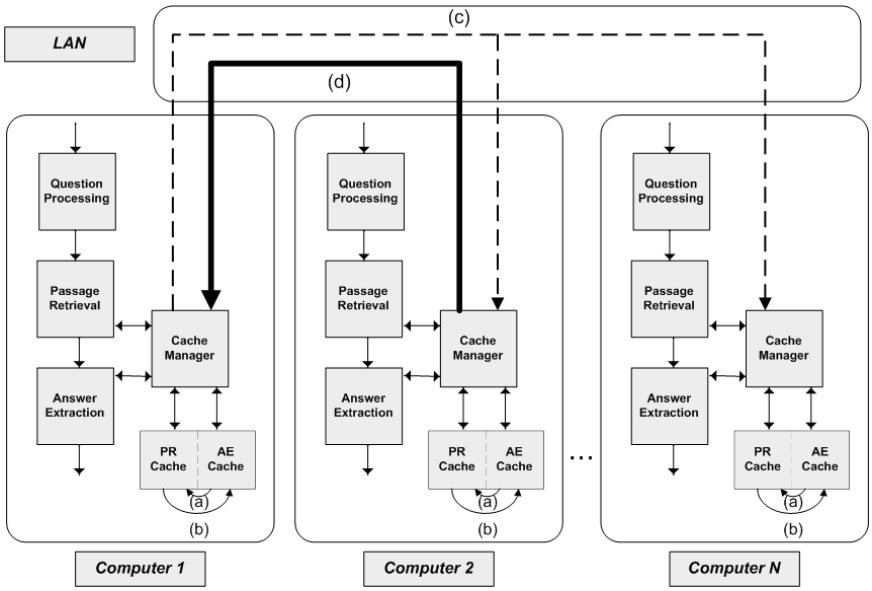

Fig. 1. *Diagram of the QA system. QA is composed of three sequential blocks QP, PR and AE. PR and AE can request and put information into the cache using the cache manager. The cache manager implements several operations: (a) Demote: An entry from the AE cache is moved to the PR cache and the natural language analysis is reset; (b) Promote: An entry from the PR cache is moved to the AE cache; (c) Petition: broadcast request message to other caches; (d) Response: the node that can resolve the petition replies with the requested information.*

larger than the corresponding PR cache unit (approximately five times larger in our setup). These observations justify the need for the multi-layer cache.

The caching unit is a collection document or a passage. This approach increases the probability of a hit because: (a) even though the same natural language question can be reformulated in many distinct variants, they all require the same documents/passages, and (b) the documents/passages needed by questions that are different but on the same topic are likely to overlap significantly.

3.1 The Cache Manager

Both layers are managed by the same cache manager, which implements the local intra and inter-layer operations and the collaboration protocol between nodes. The operations of the cache manager are logically grouped in two protocols: the *local protocol*, which defines the operations within a node, and the *distributed protocol*, which defines the collaboration between nodes.

Local Operations

With respect to the cache management protocol, we employ two policies:

– One for the internal management of a layer, which defines the *intra-layer* operations. We use a least recently used (LRU) algorithm. This means that entries in a layer are ordered in decreasing order of their usage time stamps.
– A second policy, which defines the *inter-layer* operations. We use a promotion/demotion algorithm. When a document is read from disk for the first time it is promoted in the PR layer; then, the oldest document is demoted from PR (i.e. it is completely removed from the cache). When we miss a passage in the AE layer but the document exists in the PR layer, we promote the entry to the AE layer. Then, the oldest entry in the AE layer is demoted to the PR layer. When we demote a unit from the AE to the PR layer the syntactico-semantic information is deleted.

Distributed Protocol

Question Answering is an expensive task. In other words, a single computer can not process the amount of user queries that a web search engine receives. The solution, as with information retrieval engines, is distributed computing [5]. We present an extension of our cache to a collaborative cache for a cluster based architecture, where each node runs a copy of the QA system. We compare several alternatives for implementing the distributed system:

Local: A multi-layer cache is deployed locally in each cluster node. We call this implementation *Local*. There is no communication between nodes.

*Broadcast Petition (*BP*):* We build a collaborative cache using a protocol that asks for remote work units using a network broadcast (operation (c) in Figure 1). If another node has already processed that work unit, it answers with the information stored in its cache (operation (d) in Figure 1). The response includes information of subsequent layers if the remote node knows it. The QA process continues when all requests for a query have been fullfilled or the request times out, in which case the request is processed locally. Results received are locally cached using the local caching protocol.

For example, if a node asks for a document (that is missing in the local cache) and another node has the document and the passage, the remote node will answer with the information from both the AE and PR layer because this information will be probably needed later. Sending all available information once reduces the number of network connections and the response time.

*Broadcast Petition Recently (*BPR*):* This model is similar to *BP*, but it locally caches work units received from other nodes only if they were repeatedly accessed in the same node (at least twice) in the past x seconds. For example: the first time that a node retrieves remotely the document A, the node will not store it locally. The next time the node must retrieve A, if it occurs before x seconds, a duplicate copy will be cached locally because the entry is considered to be frequently accessed. Otherwise, the local cache will not store it locally. The protocol is efficiently implemented with a queue with the ids and the timestamps

of petitions. When a node receives the information remotely, it checks if it has previously requested the same data recently with a lookup to a queue of the most recent petitions.[5].

This method minimizes the chance that documents that are not frequently accessed have multiple copies in the network. This is an important feature for QA where storing a cache unit is expensive, e.g. documents may be large and the natural language information takes significant space.

Our collaborative solutions avoid failure problems because all nodes are equal: if one machine crashes, the system continues to run. Furthermore, we can increase the processing power of our QA cluster simply by warm plugging more computers. To balance the load, we dispatch questions in a round robin fashion among the nodes in the system. However, note that in a real-world implementation the dispatcher does not have to be part of the distributed QA system itself. It can be implemented within a Domain Name Service (DNS) server, which maps a unique name to all the IP addresses in the QA cluster.

4 Experiments

In this section we test the performance of the multi-layer cache and the different collaborative protocols using the previously described QA system.

We used as textual repository the TREC document collection which has approximately 4GB of text in approximately 1 million documents. The questions sent to the system were selected from the questions that were part of former TREC-QA evaluations (700 different questions).

We tested the multi-layer cache in our QA system for Zipf question distributions. [6–8] show that web servers and IR systems follow zipfian distributions with estimations of parameter α between 0.60 and 1.45. Considering that QA systems are similar to IR systems from the usage point of view, it is reasonably safe to assume that question distributions for real-world QA systems will also follow a similar Zipf distribution.

4.1 Experiments With Multi-layer Cache

Setup: One single PC dual Xeon 2.4Ghz with 1.5GB of RAM. The question set contains 150 queries randomly selected following a $\text{Zipf}_{\alpha=1.0}$ distribution.

Experiments: In this experiment we show that multi-layer caches perform better than single level caches. We assigned to the QA system a fixed pool of memory for storing the PR and the AE layers. We executed several runs with different partitions of the available memory between the two layers. We set the size of an entry in the AE layer as 5 times the size of the entry in the PR layer to accommodate the additional information generated by the natural language analysis. A

[5] This can be implemented efficiently in several ways. For example, we can add a hash if the number of recent requests is known to be large.

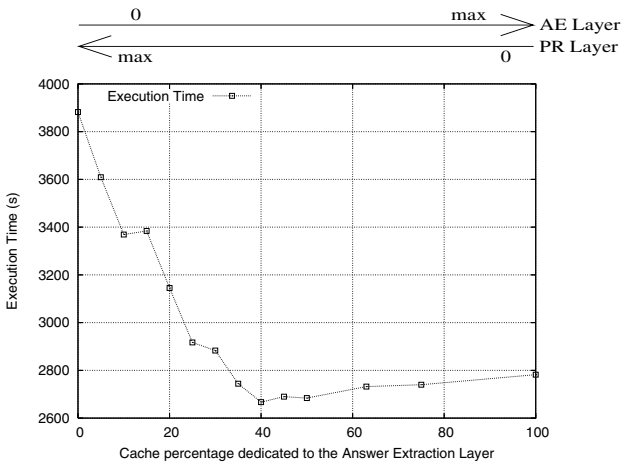

Fig. 2. *Execution time of the QA system in a single computer with different partitions.*

QA system with a single cache layer corresponds to partitioning all the available memory for either PR or AE.

In Figure 2 we plot the execution times for different partitions of the cache. The vertical axis is the total execution time. The horizontal axis represents the partitioning of the available memory between cache layers. The extremes of the graph correspond to single level caches: the left end is a PR only cache, and the right end is an AE only cache. Our experiments show that the best execution time corresponds to partitioning about 40% of the cache to the AE layer and the rest to the PR layer. All other partitions yield higher execution times. Figure 2 also proves that single layer caches, i.e. when the cache is complely dedicated to either PR or AE (left and right-most points in Figure 2), are less efficient than multi layer caches.

4.2 Distributed Protocol Experiments

Setup: A cluster with 16 nodes, each equipped with two Xeon 2-Ghz processors and 2GB RAM. All nodes were interconnected with a 1GB Ethernet. We dispatched the questions to the QA system using a separate client process that runs in the same LAN. The client process sends questions following the chosen Zipf distribution with a pause of 10 ms between questions. The question set contains 700 questions distributed in a round robin fashion among the nodes.

We performed several experiments with different cache sizes. In each node we set the maximum cache size as $\frac{1}{16}$ of the size required to cache all the work units in a single node[6]. Hence, when all memory is used, the overall cache can store all the data generated in the experiment if there were no duplicates. We

[6] This value corresponds to the right end in Figures 3 and 4

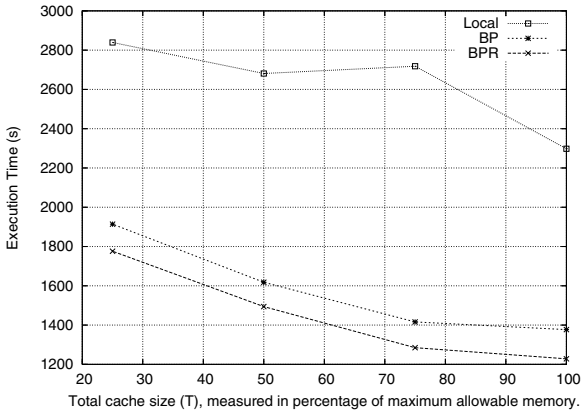

Fig. 3. *Execution time of the distributed QA system with 16 nodes. Question distribution is Zipf* $_{\alpha=1.0}$.

partitioned the cache memory between the PR and AE layers using the best configuration from the experiment in Figure 3 (60% for PR and 40% for AE).

We experimented empirically with several values of recency for the *BPR* algorithm, between 30 and 180 seconds, but we obtained similar results in this range. This is caused by the exponential nature of the Zipf distribution, which yields large gaps in access frequencies between the very frequent questions and the remaining ones. For example, the most frequent questions in our experiments repeated up to eight times per node, sometimes even consecutively. On the other hand, there is a big set of questions that will appear more than once in the question set but are rarely accessed twice by a node. Due to this large gap in access frequencies, *BPR* achieves its goal of avoiding caching the latter class of questions without fine tuning the recency parameter. In the experiments reported here the value is set to 120 seconds.

Experiments: In Figure 3 we plot the execution time for different memory size configurations and cache algorithms. The vertical axis represents execution time, and the horizontal is the size of the memory pool allocated to the system. This experiment is done using a $\text{Zipf}_{\alpha=1.0}$ distribution. Figure 3 confirms that both collaborative approaches have better performance than the *Local* method. The execution time of the *local* protocol (top line in Figure 3) is up to two times slower than the execution time of the collaborative methods.

Figure 4 explains in detail the better performance of the collaborative caching algorithms (*BP* and *BPR*). The figure shows the cache hit rates for the PR and the AE layers for the three models proposed. We show both the local cache hits (lines with clear marks) and the total hits, i.e. local + remote (lines with solid marks). At a first glance, one can see that collaborative caches have better performance, i.e. their hit ratio is approximately 25% larger than the hit ratio of local protocols. If we compare *BP* and *BPR* in the first row of Figure 4, we see

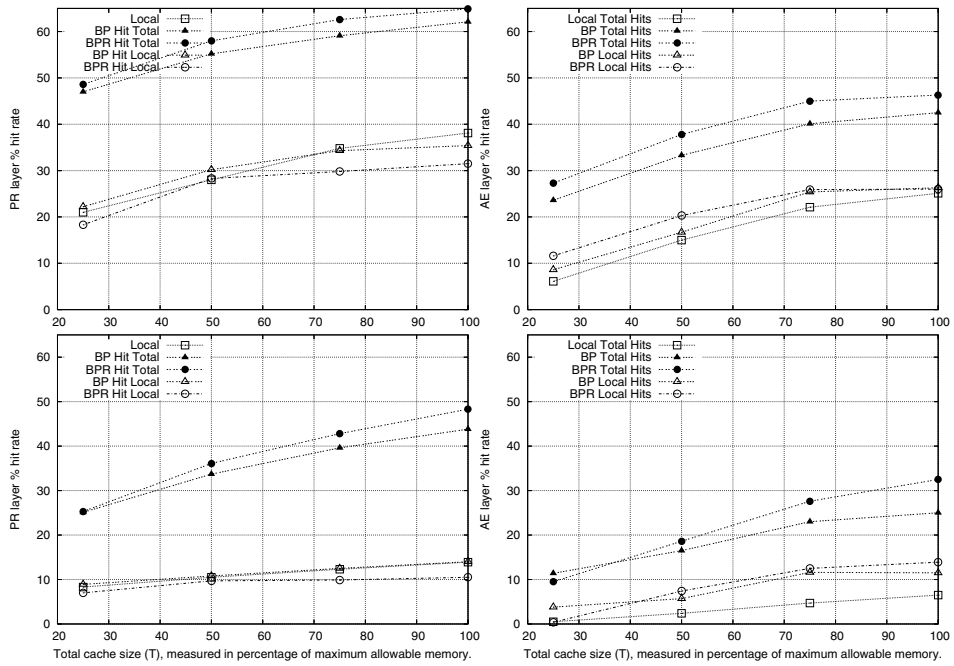

Fig. 4. *Hit rates for a distributed QA system with 16 nodes. The left column shows the PR layer and the right column is the AE layer. First row plots results for a Zipf$_{\alpha=1.0}$, second row for a Zipf$_{\alpha=0.6}$.*

that *BPR* has a higher ratio of total hits. In the PR layer, *BPR* has a 2-7% lower hit ratio than *BP* because when the system asks for a remote document it must be requested more than once in order to cache it locally. On the other hand, *BPR* yields a more efficient global cache by avoiding useless copies in the network, so the system has a higher number of total hits. In the AE layer, collaborative algorithms perform slightly better than the local protocol for local accesses. The explanation is the inclusive nature of the proposed caching architecture: if a node requests a document from the PR cache and a remote node node has the only corresponding passage in the AE cache, it replies with the complete passage. Hence the original node will have a remote hit in the PR phase and a local hit in AE. Figure 3 confirms that the better hit rate makes the *BPR* protocol up to 12% faster than *BP*.

As expected, when we decrease the cache size, the number of local hits becomes considerably smaller. Nevertheless, even if we can not allocate sufficient memory for caching results, the use of a collaborative cache mantains a large number of total hits in the system.

We tested the algorithms with another distribution, setting $\alpha = 0.6$, which is a lower bound for typical usages of web search engines and other IR systems. The results are shown in the last row of Figure 4. The trend is the same as for

$\alpha = 1.0$ as we saw above. The most notable difference is that the local hit rates are lower but the number of remote hits does not decrease at the same rate due to the collaboration between caches. We attribute the lower local hit rates to the length of the question set, which lets few repeated questions per node. The general trend of the double layer collaborative cache is that as distribution becomes more uniform the local hit rate keeps decreasing, but the cache hits on remote documents or passages smooth the decrease of the total hit ratio of the system.

5 Related Work

Many applications moved to distributed frameworks in order to increase their performance. Specifically, web servers with large numbers of visits distribute their contents across multiple computers [9]. Web search engines use big clusters to satisfy the user requests [5]. Even the underlying file systems that support large applications work in distributed mode [10]. From our point of view, all these systems may benefit from the proposals of this paper.

In terms of distributed data caching, many applications take advantage of managing a pool of memory that stores past data [11]. Information Retrieval systems, for instance, save in cache the indexes that point to documents stored in disk [12], and, in some environments, a second cache layer that stores the answers to frequent queries may be beneficial [13]. In our work, we generalize the use of more than one cache and propose a collaborative model for the caches that helps improve their performance. Furthermore, we show that a correct allocation of the available memory is crucial if we want to get the maximum performance of our application.

Our approach should not be confused with multi-level caches [14, 15]. A multi-level cache is oriented towards a client-server architecture where the cached data can be stored in different computers in the route between the client and the server. Our multi-layer cache proposes a management protocol for an application with several pipelined stages. Our extensions for a distributed environment also differ from multi-level because the computers do not have to be organized in a hierarchical topology, which is inherent of multi-level caches.

Question Answering has been a prolific research field mainly devoted to improving the accuracy of the results obtained [16–18]. The work on improving the quantitative performance of the QA problem is scarce. [19] proposed a distributed QA architecture but their focus is on load-balancing algorithms rather than caching strategies. Our work can be seen as a necessary complement to their load distribution techniques. Some QA systems implement caches for previous answers [20] or take advantage of the caches of IR systems to reduce the Passage Ranking execution time [21]. In this paper, we give a distributed solution for caching the different consuming phases in QA, and the granularity of our solution allows for the efficient reuse of the results from different previous questions.

6 Conclusions

To our knowledge, this paper is the first analysis of caching architectures for Question Answering. QA, usually implemented with a pipeline architecture, has resource requirements that are distinct and much larger than its historical predecessor, Information Retrieval: Passage Retrieval, which extracts the content of relevant documents, has high I/O usage, and Answer Extraction, which identifies the exact answer in the relevant documents, is CPU intensive. Driven by these observations we proposed and analyzed a caching architecture that: (a) is multi-layer, i.e. each resource intensive pipeline stage is allocated a segment of the cache, and (b) is collaborative, i.e. the overall system cache is transparently spread across all nodes of the distributed system. We proposed two different protocols for the management of the distributed cache.

We empirically analyzed the proposed cache architecture using a real-world QA system and question distributions in both single-processor and distributed environments (16 nodes). Our first experiments show that the multi-layer approach is better than a single-layer architecture where all cache memory is allocated to either of the resource-intensive components. The QA system with the best multi-layer cache configuration has a response time up to a 30% smaller than the system with a single-layer. Secondly, we showed that, apart from allowing for warm plugs and unplugs, the distributed approach improves the hit ratio of the multi-layer collaborative caching, and also reduces the amount of communication by keeping information that is frequently reused locally. Overall, the distributed QA system enhanced with our best cache architecture has almost twofold gains in execution times from the system with local caching.

References

1. Xin, L., Roth, D.: Learning question classifiers: the role of semantic information. Natural Language Engineering **12**(03) (2005) 229–249
2. Surdeanu, M., Turmo, J., Comelles, E.: Named Entity Recognition from Spontaneous Open-Domain Speech. Proceedings of Interspeech (2005)
3. Pasca, M.: Open-Domain Question Answering from Large Text Collections. CSLI Publications Stanford, Calif (2003)
4. Surdeanu, M., Dominguez-Sal, D., , Comas, P.: Performance Analysis of a Factoid Question Answering System for Spontaneous Speech Transcriptions. Interspeech (2006)
5. Barroso, L., Dean, J., Hölzle, U.: Web search for a planet: The google cluster architecture. IEEE Micro **23**(2) (2003) 22–28
6. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web caching and zipf-like distributions: Evidence and implications. INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE (1999) 126–134
7. Markatos, E.P.: On caching search engine query results. Computer Communications **24**(2) (2001) 137–143
8. Scime, A. and Baeza-Yates, R.: Web Mining: Applications and Techniques. Idea Group (2005)

9. Cardellini, V., Casalicchio, E., Colajanni, M., Yu, P.: The state of the art in locally distributed Web-server systems. ACM Computing Surveys **34**(2) (2002) 263–311
10. Ghemawat, S., Gobioff, H., Leung, S.: The Google file system. Proceedings of the nineteenth ACM symposium on Operating systems principles (2003) 29–43
11. Raunak, M.: A survey of cooperative caching. Technical report (1999)
12. Alonso, R., Barbara, D., Garcia-Molina, H.: Data caching issues in an information retrieval system. ACM Transactions on Database Systems (TODS) **15**(3) (1990) 359–384
13. Saraiva, P., de Moura, E., Ziviani, N., Meira, W., Fonseca, R., Riberio-Neto, B.: Rank-preserving two-level caching for scalable search engines. ACM SIGIR (2001) 51–58
14. Zhou, Y., Philbin, J., Li, K.: The multi-queue replacement algorithm for second level buffer caches. In: Proceedings of the General Track: 2002 USENIX Annual Technical Conference, Berkeley, CA, USA, USENIX Association (2001) 91–104
15. Wong, T., Wilkes, J.: My cache or yours? making storage more exclusive. In: Proceedings of the General Track: 2002 USENIX Annual Technical Conference, Berkeley, CA, USA, USENIX Association (2002) 161–175
16. Chu-Carroll, J., Czuba, K., Duboue, P., Prager, J.: Ibm's piquant ii in trec2005. In: Proceedings of the Fourthteen Text REtrieval Conference (TREC). (2005)
17. Robert, E.N.: Javelin i and ii systems at trec 2005. In: Proceedings of the Fourthteen Text REtrieval Conference (TREC). (2005)
18. Moldovan, D., Pasca, M., Harabagiu, S., Surdeanu, M.: Performance issues and error analysis in an open-domain question answering system. ACM Transactions in Information Systems **21**(2) (2003) 1–22
19. Surdeanu, M., Moldovan, D., Harabagiu, S.: Performance analysis of a distributed question/answering system. Transactions on Parallel and Distributed Systems (2002)
20. Lam, S., Ozu, M.: Querying Web data-the WebQA approach. Web Information Systems Engineering. WISE 2002. (2002) 139–148
21. Zheng, Z.: AnswerBus Question Answering System. Human Language Technology Conference (HLT 2002) (2002) 24–27