Scalability via Parallelization of OWL Reasoning

Thorsten Liebig, Andreas Steigmiller, and Olaf Noppens

Institute for Artificial Intelligence, Ulm University
89069 Ulm, Germany
firstname.lastname@uni-ulm.de

Abstract. Practical scalability of reasoning is an important premise for the adoption of semantic technologies in a real-world setting. Many highly effective optimizations for reasoning with expressive OWL ontologies have been invented and implemented over the last decades. This paper describes our approach for concurrent computation of the nondeterministic choices inherent to the OWL tableau reasoning procedure for $\mathcal{SHIQ}$. We present the architecture of our parallel reasoner and briefly discuss our prototypical implementation as well as future work.

Key words: OWL, Reasoning, Parallelization

1 Motivation

Tableaux-based algorithms have shown to be an adequate method in order to implement OWL reasoning services for many practical use-cases of moderate size. However, scalability of OWL reasoning is still an actual challenge of DL research. Recent optimizations have shown significant increase in speed for answering queries with respect to large volumes of individual data under specific conditions. For instance, the KAON2 [1] system achieves excellent performance for reasoning with large volumes of individuals by a clever transformation of OWL into disjunctive Datalog unless there are cardinality restrictions. A recently proposed variant of the tableau algorithm [2] has shown some speed-ups for at least certain kind of ontologies within the HermiT reasoning system. SHEER, a different, sound but incomplete approach [3] tries to reason with a condensed version of the data but is not applicable in the presence of nominals. The CB system implements a consequence driven approach [4] which is theoretically optimal at least for the Horn-fragment of $\mathcal{SHIQ}$. The bottom line is that almost all optimizations typically do come with some restriction in expressivity. This, however, adds another critical dimension to developers of semantic applications in that they need to hit the right language fragment which hopefully suits their needs but also comes with a powerful reasoning engine.

On the other hand, modern CPU's typically pool more than one processing unit on a single chip. Recent consumer desktops even come with two quad-core processors. However, research into reasoning engines which distribute their work load in such a setting just has started ([5, 6]). Clearly, parallel computation can only reduce processing time by a factor which is determined by the available

processing units but has the potential of being applicable without any restriction especially to the most "costly" cases.

On a high level there are at least two different approaches for parallelizing reasoning:

Reasoner level. Refers to the approach where the system runs independent instances of the reasoner procedure to solve some service task. For instance, computation of the class hierarchy can be delegated to a set of reasoner instances each of which check consecutively for subsumption of two classes.

Proof level. Aims at parallelizing the reasoning procedure itself by concurrent computation of inherently independent proof steps.

Our approach follows the proof level strategy because of its sophisticated tuning and optimization options. The reasoner level approach, by contrast, is a naive kind of parallelization whose synchronization interval is more or less unpredictable and therefore far from optimal. For instance, the efficient computation of the concept hierarchy should exploit previous subsumption results. In the worst case the reasoner level approach has to compute some tests multiple times due to inherent poor synchronization possibilities.

This paper describes how to parallelize the well-known tableau algorithm used within reasoning systems such as RacerPro, FaCT++, or Pellet. Parallelizing the nondeterministic choices within the standard DL tableau procedure has several advantages. First of all, nondeterminism is inherent to the tableau algorithm due to logical operators such as disjunction, at-most, or qualified cardinality restrictions. The generated alternatives from these expressions are completely independent of each other and can be computed concurrently. In case of a positive result the other sibling threads can be aborted. The parallel computation of nondeterministic alternatives also makes the algorithm less dependent on heuristics which typically choose the next alternative to process. A bad guess within a sequential algorithm inevitably will lead to a performance penalty. A parallel approach has the advantage of having better odds with respect to at least one good guess.

In the following we present our framework for distributed tableaux proofs, describe the status of our implementation and comment on first result as well as discuss future work.

2 An Approach for Parallelizing DL Tableaux Proofs

Our approach aims at parallelizing the sequential algorithm proposed in [7] for $\mathcal{ALCNH}_{R^+}$ ABoxes with GCIs, enhanced with inverse roles and qualified cardinality restrictions – referred to as $\mathcal{SHIQ}$ – and extends our previous work dealing with a parallel $\mathcal{SHN}$ reasoning system [8]

Every standard reasoning task can be reduced to a corresponding ABox unsatisfiability problem. A tableau prover will then try to create a model for this ABox. This is done by building up a tree (the tableau) of generic individuals

a_i (the nodes of the tableau) by applying tableaux expansion rules. Tableaux expansion rules either decompose concept expressions, add new individuals or merge existing individuals.

2.1 Non-Determinism within $\mathcal{SHIQ}$

The most obvious starting point for parallel proof processing are the nondeterministic tableaux rules. Nondeterministic branching yields to multiple alternatives, which can be seen as different possible ABoxes to continue reasoning with. Within $\mathcal{SHIQ}$ there are three inherent nondeterministic rules:

The disjunction rule. If for an individual a the assertion $a : C \sqcup D$ is in the ABox, then there are two possible ABoxes to continue with: either with C or with D.

The number restriction merge rule. The at-most restriction results in nondeterminism, if there are m r-successors in an ABox as well as an at-most restriction $(\leq n\ r)$ and it holds that $m > n$. In such a situation the m existing successors need to be merged to at most n r-successors. In the worst case this requires to check for all possible m on n partitions.

The choose rule. For an ABox with the qualified number restriction $(\leq n\ r.C)$ the algorithm has to add either C or $\neg C$ to any r-successor.

As there are no dependencies between the alternatives generated by the rules above. Consequently they can be evaluated within parallel threads independently.

2.2 Work Pool Architecture

In order to enable parallelism without recursively creating an overwhelming number of threads, we decided to adopt a *work pool* design as shown in Figure 1. A *work unit* with a fixed number of *work executors* is generated at the start of the tableau proof. This parametrizable number typically will be equal to or less than the number of available processing units. These executors have synchronized read access to a common queue of jobs (i.e. the ABoxes to evaluate). On start, the tableaux root node (the original query ABox) is send from a superordinate *work distributor* to a work unit (cf. step A of Fig. 1).

The unit's *work controller* will add this work package to the units work queue (step 1). The controller then starts it's executors and one of them will fetch the initial job (step 2). During processing the executors have concurrent read as well as synchronized write access to two global caches (step 3). In case of a nondeterministic rule application an executor will generate the necessary alternative ABoxes by extending the preceding ABox (step 4). This work package is prioritized and submitted to the corresponding work queue. The next available executor will fetch the most prioritized work package from the pool. The executors also report any proof relevant information such as satisfiability results to the managing work controller, which then will control other executors when appropriate (step 4).

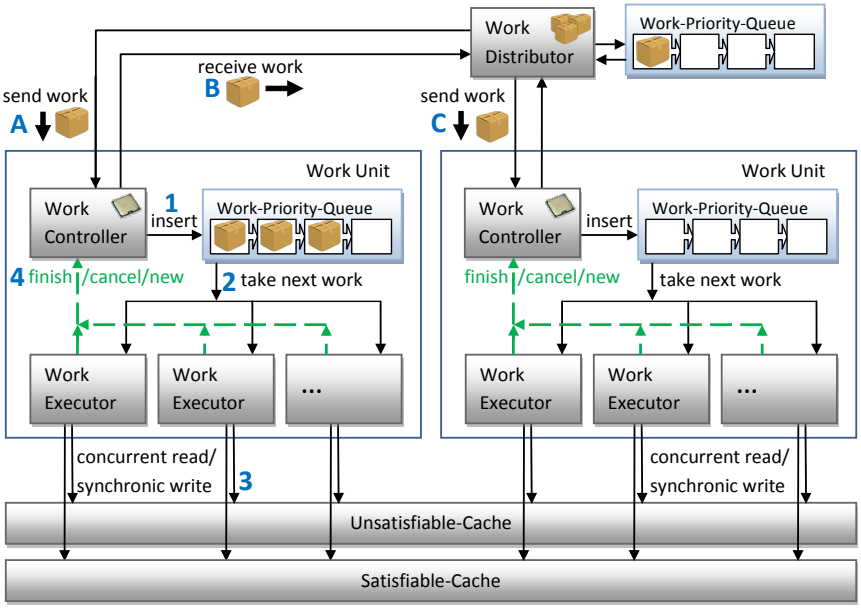

Fig. 1. Component interaction within work pool design of parallel reasoner.

The work controller itself gives notice to the work distributor in case

i) an ABox that represents a complete tableau is found, or
ii) no satisfiable alternative was found and there are no alternatives left to process.

The work distributor of our system architecture is designed to be able to coordinates more than one work unit. For instance, in case of an empty work queue of one unit the distributor will level the queues of its units such that there is no idle unit as long as there are work packages to process.

So far the design is tailored to a SMP (symmetric multi processor) architecture, where all processing cores have access to one main memory. However, our approach also allows for distribution over many computing systems. In such a setting multiple work distributors can coordinate their work between each other.

An important decision in this design is the choice of the units work pool organization. The commonly used queue is unsuitable in this setting as it promotes a breadth-first style evaluation order. Thus, ABoxes which were created earlier (generated by fewer applications of nondeterministic rules) are preferred, and the discovery of complete ABoxes is delayed. The usage of a stack would not reliably lead to a depth-first oriented processing order either, because several executor share one pool and push new work package when the occur.

We therefore have chosen to use a priority queue in order to be able to explicitly influence the processing order. We use a simple heuristic to control the processing order:

- The priority of the original ABox is set to 0.
- ABoxes generated from an ABox with priority n are given the priority $n+1$.

This allows for a controlled depth-first oriented processing order. More sophisticated heuristics or even some sort of A*-algorithm would also be possible. For example, FaCT++ also utilizes a priority queue for its ToDo list [9], weighting tableaux rules with different priorities.

2.3 Implementation Status and Future Work

The architecture as described above has been implemented in C++ utilizing the Qt libraries[1]. Qt allows for platform independent development and supports parallel processing via dedicated thread libraries. So far we only have a reasoner core in the sense that there is a $\mathcal{SHIQ}$ proof procedure only. However, we plan to build a complete reasoning system by adding the well-known pre-processing mechanisms (such as GCI absorption) and inference services (such as taxonomy computation, basic asks, etc.). Initial benchmarks also have revealed encouraging results which are planned to be published very soon.

References

1. Hustadt, U., Motik, B., Sattler, U.: Reasoning in Description Logics by a Reduction to Disjunctive Datalog. Journal of Automated Reasoning (2007) To appear.
2. Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research **36** (2009) 165–228
3. Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., Srinivas, K.: The Summary Abox: Cutting Ontologies Down to Size. In: Proceedings of the International Semantic Web Conference (ISWC 2006), Athens, GA, USA (2006) 343–356
4. Kazakov, Y.: Consequence-Driven Reasoning for Horn $\mathcal{SHIQ}$ Ontologies. In: Proceedings of the 21st International Conference on Artificial Intelligence (IJCAI 2009). (July 11-17 2009) 2040–2045
5. Schlicht, A., Stuckenschmidt, H.: Peer-to-peer Reasoning for Interlinked Ontologies. In: Proc. of Int. Conference on Web Reasoning and Rule Systems. (2009)
6. Aslani, M., Haarslev, V.: Towards parallel classifcation of tboxes. In: Proc. of the 2010 International Workshop on Description Logics (DL-2010). (2010)
7. Haarslev, V., Möller, R.: Expressive ABox Reasoning with Number Restrictions, Role Hierarchies, and Transitively Closed Roles. In: Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2000). (2000) 273–284
8. Liebig, T., Müller, F.: Parallelizing Tableaux-Based Description Logic Reasoning. In: Proc. of the Int. Workshop on Scalable Semantic Web Systems (SSWS). Volume 4806 of LNCS., Springer (2007) 1135–1144
9. Tsarkov, D., Horrocks, I., Patel-Schneider, P.F.: Optimising Terminological Reasoning for Expressive Description Logics. Journal of Automated Reasoning **39**(3) (2007) 277–316

[1] `http://qt.nokia.com/`