Published in final edited form as:
Proc Symp Appl Comput. 2009 ; : 1257–1261. doi:10.1145/1529282.1529564.

Parallel Materialization of Large ABoxes

Sivaramakrishnan Narayanan,
Dept. of Biomedical Informatics, The Ohio State University, Columbus, OH, USA

Umit Catalyurek,
Dept. of Biomedical Informatics, The Ohio State University, Columbus, OH, USA

Tahsin Kurc, and
Dept. of Biomedical Engineering, Emory University, Atlanta, GA, 30322

Joel Saltz
Center for Comprehensive Informatics, Emory University, Atlanta, GA, 30322

Sivaramakrishnan Narayanan: krishnan@bmi.osu.edu; Umit Catalyurek: umit@bmi.osu.edu; Tahsin Kurc: tkurc@emory.edu; Joel Saltz: jhsaltz@emory.edu

Abstract

This paper is concerned with the efficient computation of materialization in a knowledge base with a large ABox. We present a framework for performing this task on a shared-nothing parallel machine. The framework partitions TBox and ABox axioms using a min-min strategy. It utilizes an existing system, like SwiftOWLIM, to perform local inference computations and coordinates exchange of relevant information between processors. Our approach is able to exploit parallelism in the axioms of the TBox to achieve speedup in a cluster. However, this approach is limited by the complexity of the TBox. We present an experimental evaluation of the framework using datasets from the Lehigh University Benchmark (LUBM).

Keywords

Parallel; Materialization; Ontology; ABox

1. INTRODUCTION

It is often desirable to represent and annotate features and data elements in a dataset using concepts and relationships from domain-specific ontologies. A knowledge-base (KB) in our context consists of two main components: a TBox and an ABox. A TBox captures higher-level knowledge about a domain using descriptions about concepts and roles in the domain. An ABox captures an instantiation of a domain and may refer to specific *individuals*. An ABox uses terms from the ontology to *assert* facts about these *individuals*. The KB *implies* additional facts from the explicitly stated ones. *Materialization* is the process of computing all implicit assertions in the KB and is frequently employed by semantic query and reasoning engines [13,4,9] to improve query performance.

With advances in tools for managing ontologies and automatic extraction of semantic information from data sources, KBs consisting of large ABoxes will become increasingly common. With increasing ABox sizes, efficiency of the materialization stage becomes important. In this paper, we investigate the use of distributed-memory parallel machines to support efficient materialization. We target ontologies expressed using a fragment of OWL-Lite [10]. Our framework implements techniques for workload partitioning and data exchange, and exploits existing semantic stores to perform local materialization. Our approach exploits inter-axiom parallelism to achieve speedup in a parallel environment. We evaluate the performance of the proposed approach using SwiftOWLIM [9] semantic store for local inferencing computations and datasets generated by the Lehigh University Benchmark (LUBM) [6].

2. APPROACH

A TBox $\mathcal{T}$ consists of a set of axioms and refers to a set of concepts $\mathcal{C}$ and a set of roles $\mathcal{R}$. In this work, we support the forms of axioms as shown in Table 1. This corresponds to a subset of OWL-Lite and a superset of RDFS [1]. For the purpose of materialization, we translate TBox axioms to rules as shown in the table. Generation of implicit assertions (materialization) happens by the repeated application of these rules to known facts as follows: (1) From existing assertions, find a rule whose LHS is satisfied by a certain binding of individuals. This is called *firing* the rule; (2) Add assertion corresponding to the RHS of the rule; (3) Repeat until no new assertions can be added. During the process of materialization, it may be possible to fire many rules simultaneously.

Our parallelization approach relies on partitioning axioms (rules) and ABox assertions. This partitioning will allow processors to fire different axioms (rules) in parallel so as to improve the materialization time. In our approach, we introduce *Home*, *Helper*, and *Generated* concepts and assertions for an axiom. For the sake of brevity, we will describe these for concepts - similar description holds for roles.

Table 1 lists the home, helper, and generated types of each axiom form supported in our implementation. Each axiom has precisely one home concept/role, one helper concept/role and one or more generated concept/roles. Note that an assertion may be a concept assertion or a role assertion. An assertion associated with the home type of an axiom is a home assertion with respect to that axiom. Similarly, an assertion may be a helper or a generated assertion with respect to an axiom. Since there are multiple axioms in a TBox, a concept may be the home concept for one axiom and a helper concept in another. Axioms are grouped based on their home concepts. This is the first stage of partitioning axioms.

We define *helper* and *generated* concepts on concept C as the union of helper and generated concepts from all axioms where C is the home concept. As a result, concepts C_1 and C_2 in a TBox may be related in the following ways: (1) Concept C_2 is a helper concept for concept C_1 if there exists an axiom where C_1 is the home concept and C_2 is the helper concept. (2) Concept C_2 is a generated concept for C_1 if there exists an axiom where C_1 is the home concept and C_2 is the generated concept.

Consider the example TBox in Table 2(a). The TBox consists of four axioms. Using Table 1, we can identify the home, helper and generated concepts/roles (types) for each axiom. Axioms 2 and 4 have the same home type (R_2). In axiom 2, R_1 is the generated type and in axiom 4, C_3 is the generated type. We generate a TBox table which summarizes this information for all concepts and roles. Table 2(b) shows the TBox table for the example TBox. The TBox table describes potential interactions between assertions in a succinct manner. The third line in Table 2(b), for instance, indicates that assertions of types C_3 and C_4 in the context of some axiom (Axiom 3) may cause an inferred assertion (of type C_2).

Our approach partitions the TBox table into n parts (here, n is the number of computation nodes) using a heuristic strategy and assigns each partition to a node in the system – each node is assigned a set of home concepts/roles. A compute node is responsible for assimilating all assertions corresponding to these types. Table 2(c) shows one possible partition of home concepts and roles. If two concepts occur in different partitions and one is a helper type for the other in some row in the TBox table, inter-processor communication may be necessary between the partitions. In the example, partition 2 (P_2) may communicate instances of type C_4 to partition 1 (P_1).

3. PARALLEL FRAMEWORK

The parallel ABox reasoning framework (shown in Figure 1) assumes a cluster system consisting of a set of storage nodes and a set of computation nodes – a node may be both a computation node and a storage node. It is assumed that the ABox containing explicit assertions is initially spread across the set of storage nodes. Reader processes are instantiated on these nodes which read the ABox files (in RDF format) and convert them to in-memory data structures. The ontology partitioner component is responsible for creating and partitioning the TBox table. Summary ABox information may be computed and utilized by the ontology partitioner. We will present the partitioning strategy employed in this work in Section 3.2. Once the partitions are computed, they are assigned to compute nodes. We will refer to compute nodes and partitions interchangeably. Reader processes read the subsets of the ABox and send them to relevant partitions. A partition inference task is executed on each computation node. Partition inference tasks perform local materialization using part of the TBox and ABox. Communication links are set up between partition inference tasks to allow transfer of ABox assertions between tasks running on different compute nodes. An end of computation signal denotes the full computation of the implicit assertions from the explicit ABox, i.e., the computation of the *materialization*.

3.1 Partition Inference Task

The partition inference task is the core of our framework and is executed on each node. The ABox assertions are partitioned on home types, and each partition is sent to the appropriate node. An assertion of the form $C_1(i_1)$ is sent to the node containing the TBox graph partition with concept C_1. Each node executes the partition inference task algorithm using the part of the TBox and ABox assigned to it. The algorithm proceeds in phases. At the end of each phase, new assertions are generated by each partition.

The algorithm keeps track of new home and helper assertions generated in the current phase and old home and helper assertions from earlier phases. When a new assertion of type C_1 is generated, it is sent to the partition containing C_1 during the exchange step. Each node is aware of the other nodes' requirement for helper assertions. If a partition has C_1 as a helper type, then the partition containing C_1 is responsible for forwarding any new instances of C_1 it sees in the previous phase.

During each phase, the old home and helper assertions and new home and helper assertions are combined and local materialization is performed using the portion of the TBox assigned to the node. To improve performance, the algorithm combines old helper assertions with new home assertions, old home assertions with new helper assertions, and new home assertions with new helper assertions. Local materialization may result in generated assertions on each node. These generated assertions may be home assertions for other partitions and must be shipped the corresponding nodes. Not all arriving assertions may be new assertions. Thus, each partition performs a difference operation to eliminate assertions that it has already seen in earlier phases.

The nodes communicate the number of assertions generated. If any partition has received new home or helper assertions, a new phase of execution begins and the above steps are repeated. Eventually, at the end of a phase, no node will be able to generate new assertions, and the algorithm will terminate.

The algorithm is guaranteed to terminate. In each phase, axioms are activated only if at least one new home or helper assertion is detected. Thus, at the end of every non-final phase, the total number of assertions increases monotonically. The total number of materialized assertions is bounded by the number of concepts, roles and individuals and, therefore, is finite. This ensures that the parallel materialization eventually terminates.

Different partitioning of the TBox table may produce assertions in different orders. However, the algorithm will generate correct results for any partitioning scheme. The performance of this partitioned approach depends on the quality of the partitions produced. In each phase, different partitions may have different amount of work to do. Ideally, we would like to achieve load balance in every phase while minimizing communication between partitions. We discuss a partitioning strategy we employed in this work that keeps these goals in mind.

3.2 TBox Partition Strategy

We employ a min-min approach to partition the TBox table. Algorithm 1 describes the partitioning process. To aid in this process, summary information of the explicit ABox is utilized. Specifically, the number of explicit instances of each type is counted and is represented by the function *Count*.

The algorithm iterates over all combinations of types (concepts or roles) and partitions to determine the best concept/role, partition – (c, p) – pair. To determine the best pair, the cost of adding a concept to a partition is computed. If a type is assigned to a partition, the home assertions of that type are stored in that partition. Helper assertions may have to be shipped to this partition for the purpose of materialization. The algorithm chooses the pair (c, p) such that after this assignment, W_p has the lowest weight. This approach has been taken to ensure that not all types are assigned to a single node and load balance is achieved. The cost computation depends on whether helper types are local or remote. This aims to co-locate helpers with home types when possible.

The proposed min-min approach has its limitations. The explicit counts of types are employed in cost computation. It is possible that a type A may have few explicit instances, but numerous implicit instances. The number of implicit instances are not known a-priori. In some applications, it may be possible to approximate the final counts of types in the TBox. In such a scenario, it will be possible to employ the final counts to achieve better load balance. The above approach also has a complexity of the order of $O(|S|^2.N)$ which can be a problem when there are very high number of concepts and roles. In our experimental scenarios, however, the cost of computing partitions was negligible.

3.3 Implementation

An important step in our parallelization is performing local inference in a phase involving a fragment of the ABox and TBox. One of our design and implementation decisions was to use existing single node semantic stores like SwiftOWLIM [9]. We employ these systems as black-box reasoners and interact with them using adapters. There are advantages and disadvantages to this approach. The key advantage is that we can reuse the optimizations that these systems provide. The main disadvantage is that we do not access the internal data structures of these systems directly and, therefore, need to manage the ABox outside of these reasoners. This causes an overhead of translating ABox fragments to and from the reasoner's internal representation. It also increases our memory footprint. We plan to work on these shortcomings in our future work.

4. EXPERIMENTAL RESULTS

We performed an experimental evaluation of the proposed approach using a distributed memory parallel machine which consists of a set of storage nodes and a set of compute nodes. The two types of nodes are connected through an Infiniband Switch. Each compute node has an dual AMD 250 Opteron processor running at 2.8GHz with 8 GB of main memory. The storage nodes have the same CPU and memory configuration, but each storage node is connected to a 3.5 TB SATA disk array in RAID 5 mode. Each storage node has a sequential read bandwidth of about 295 MB/s. We implemented the framework using Data Cutter [3], a component-based middleware framework that enables execution in a distributed environment. We employed the Lehigh University Benchmark (LUBM) to generate datasets for performance evaluation. The LUBM was developed to facilitate evaluation of semantic queries and inference computations against large datasets [6]. It uses a *University* ontology. Examples of concepts in this ontology are *Publication*, *Person* and *Grad Student*. Examples of roles are *publication Author* and *Advisor*. This ontology employs anonymous concepts and we rewrote some axioms and assign names to these types. This does not affect the correctness of the results produced but makes the axioms amenable to our partitioning techniques. There were 83 concepts and roles in the final TBox. To generate an ABox using the dataset generation tool of the LUBM, the user specifies the number of universities. The size of the ABox (i.e., the number of explicit assertions) grows linearly with the number of universities. The 10-university dataset has about 1.3 million explicit assertions and the 450-university dataset has approximately 58.5 million explicit assertions. In our experiments, we used SwiftOWLIM v2.8.4 as the reasoning engine with options owl-horst, partialRDFS, noPersist with the index size parameter set to 4 million for improved performance. The cost of computing partitions using the min-min approach was not significant due to the small size of the TBox and we do not include it in the results.

Figure 2 shows the end to end materialization time using a standalone SwiftOWLIM and the parallel approach using 2,4 and 8 nodes. In the standalone scenario, we used SwiftOWLIM as a library and created an inferencing repository and read the datasets from a local disk. In the parallel scenario, we distributed the datasets over 5 storage nodes and employed compute nodes to execute partition inference tasks.

The first thing to note is that the serial materialization time appears to grow quadratically. The reason behind this is that many axioms require a join of instances from two types (see Table 1). When the input sizes are doubled, the join operation increases by a factor of 4. The ontology in question contains 8 axioms that involve a join operation. This cost of these joins dominate for higher ABox sizes and this is the reason for a quadratic increase in the serial materialization time. The partitioning strategy places these axioms in different partitions and this reduces execution times linearly up to 8 nodes.

Figure 3 shows the speedup of materialization on up to 16 nodes. Performance of this approach worsens if more than 8 nodes are employed. There are two reasons for this observation. The first is that the performance of the parallel approach depends on the number of independent, expensive axioms in the TBox. In this dataset, there are 8 such axioms and therefore we see a benefit up to 8 nodes. The reason for worsening of performance is that our partitioning strategy attempts to balance the load. In doing so, it sacrifices co-location of home and helper types. This results in home and helper types of expensive axioms to be assigned to different nodes and a lot of communication across nodes. Furthermore, due to data dependencies, may nodes are idle in many phases of execution. This is the cause of the drop in performance when more nodes are added.

An important cost involved with using a shared-nothing cluster is the cost of communication. Communication costs involve latency and bandwidth. The order of data exchanged in these experiments is of the order of hundreds of megabytes. In our cluster configuration, the high-bandwidth interconnect allows transferring these data sizes in the matter of a few seconds. The majority of the cost of communication comes from latency, i.e. the cost of serializing and deserializing ABoxes. This cost is seen in the *Exchange Home Assertions* and *Exchange Helper Assertions* calls in the partition inference task. To guage the effect of latency in our setting, we compute the latency cost as the sum of maximum serialization/deserialization time in any partition over all phases of execution.

$$LatencyCost = \sum max(serialization + deserialization)$$

As expected, latency increases linearly with dataset sizes (see Figure 4). Furthermore, latency is lower when more nodes are employed. A shared-nothing architecture necessitates serialization and deserialization of ABox structures and a latency cost is unavoidable. The impact of latency on overall materialization time is lesser in larger datasets since computation of materialization grows quadratically and dominates overall cost. Therefore, the shared-nothing approach shows better speedup for large ABoxes.

5. RELATED WORK

While the well-defined semantics of OWL makes it machine-processable, the reasoning process is a computationally expensive process; OWL-Lite has exponential complexity, for example. Systems such as FaCT [8] and Racer [7] implement in-memory tableaux algorithms that work well for realistic TBoxes and small ABoxes. In many applications, the explicit ABox is much larger than the TBox in question. Tableaux systems tend to perform poorly in these situations. Forward-chaining engines like SwiftOWLIM [9] and Jena [13] are being increasingly used for ABox reasoning. These reasoners are sound and efficient (compared to tableaux systems), but incomplete. Their reasoning capabilities are, however, are sufficient for a variety of applications. Our approach employs such engines to perform local materialization.

In contrast to our approach, some authors have proposed partitioning the ABox to perform reasoning. Guo and Heflin [5] describe an approach to partition large OWL ABoxes with respect to a TBox so that reasoning may be performed on each partition separately and results trivially combined. They employ partitioning for a different reason. Their objective is to utilize tableuax systems like RACER on larger ABoxes [7]. They employ a fine-grained ABox partitioning strategy for this purpose. They use a serial algorithm to compute the partitions which involves building an ABox connectivity graph. There are two issues which make this approach unsuitable for our purpose. The first is that the serial algorithm is expensive (it takes 21 min to partition 10 university dataset). The second is that the partitions generated are overlapping. This leads to redundancy in the inference process. In contrast, we employ a coarse-grained approach wherein we exploit parallelism in TBox axioms. Amir and McIlraith [2] suggest a partition-based approach to logical reasoning. They target reasoning on first-order logic theories using a forward message-passing algorithm. They generate graphs capturing relationships between symbols and employ vertex min-cut to partition a theory. We take a coarse-grained approach of partitioning the TBox because of the large scale of ABoxes.

In this work, we translate axioms to rules to identify interacting types. Prolog [11] programs are comprised of definite clauses and any question in prolog is a goal. In our work, we focus on materializing instances of all types as opposed to reaching a single goal. To this end, we employ forward-chaining while prolog systems employ backward-chaining. Ueda [12] describes Guarded Horn Clauses to describe a parallel logic programming language. The goal of that work was to annotate prolog programs to achieve parallelism during backward-chaining.

6. CONCLUSIONS

This paper has presented a framework and strategy for parallel computation of inferred statements from a large ABox on a shared-nothing architecture. Our results show that the proposed approach achieves good speedup on small number of processors using SwiftOWLIM, a very fast rule-based reasoning system, for local computations. This approach attempts to exploit axiom-level parallelism in the TBox. This approach will benefit applications where the TBox contains many axioms with few dependencies. In a data-integration scenario, it is likely that the resultant TBox will contain many such axioms which may be processed using the approach in the paper. This approach to parallelization will not benefit applications which use very small TBoxes. A limitation of our current implementation is that it uses the existing reasoning engine as a black-box for local inference computations. While this design allows us to plug-in other reasoning engines, it introduces overheads. We plan to investigate mechanisms to alleviate these issues by a tighter integration of reasoning engine with parallelization strategy and by schemes that will partition work associated with a single concept/role across processors in a future work.

Acknowledgments

This research was supported in part by the National Science Foundation under Grants #CNS-0403342, #CNS-0426241, #ANI-0330612, #CCF-0342615, #CNS-0203846, #ACI-0130437, #CNS-0615155, #CNS-0406386, and Ohio Board of Regents BRTTC #BRTT02-0003 and AGMT TECH-04049 and NIH grants #R24 HL085343, #R01 LM009239, and #79077CBS10.

References

1. RDF Vocabulary Description Language 1.0: RDF Schema. 2004. http://www.w3.org/TR/rdf-schema/
2. Amir, E.; McIlraith, SA. In: Cohn, AG.; Giunchiglia, F.; Selman, B., editors. Partition-based logical reasoning; Proceedings of the Conference on Principiles of Knowledge Representation and Reasoning (KR-00); S. F. Apr. 11–15 2000; Morgan Kaufman Publishers; p. 389-400.
3. Beynon MD, Kurc T, Catalyurek U, Chang C, Sussman A, Saltz J. Distributed processing of very large datasets with DataCutter. Parallel Computing. Oct; 2001 27(11):1457–1478.
4. Broekstra, J.; Kampman, A.; van Harmelen, F. Sesame: A generic architecture for storing and querying RDF and RDF schema. International Semantic Web Conference, number 2342 in Lecture Notes in Computer Science; Springer Verlag; July. 2002 p. 54-68.
5. Guo, Y.; Heflin, J. A scalable approach for partitioning owl knowledge bases. Proceedings of the 2nd International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2006); 2006. p. 47-60.
6. Guo Y, Pan Z, Heflin J. LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics. 2005; 3(2–3):158–182.
7. Haarslev, V.; Möller, R. Racer: A core inference engine for the semantic web. 2nd International Workshop on Evaluation of Ontology-based Tools (EON 2003), volume 87 of CEUR Workshop Proceedings. CEUR-WS.org; 2003.
8. Horrocks, I. The FaCT system. Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-98), volume 1397 of LNAI; Berlin. May 5–8 1998; Springer; p. 307-312.

9. Kiryakov, A.; Ognyanov, D.; Manov, D. OWLIM - A pragmatic semantic repository for OWL. WISE Workshops, volume 3807 of Lecture Notes in Computer Science; Springer; 2005. p. 182-192.
10. McGuinness, DL.; van Harmelen, F. OWL web ontology language overview. 2004. http://www.w3.org/TR/owl-features/
11. Sterling, L.; Shapiro, EY. The Art of Prolog - Advanced Programming Techniques. MIT Press; 1986.
12. Ueda, K. Guarded horn clauses. In: Wada, E., editor. Logic Programming, number 221 in LNCS. Springer-Verlag; 1986. p. 168-179.
13. Wilkinson, K.; Sayers, C.; Kuno, HA.; Reynolds, D. Efficient RDF storage and retrieval in Jena2; Proceedings of Very Large Data Bases (VLDB) Workshop on Semantic Web and Databases; 2003. p. 131-150.

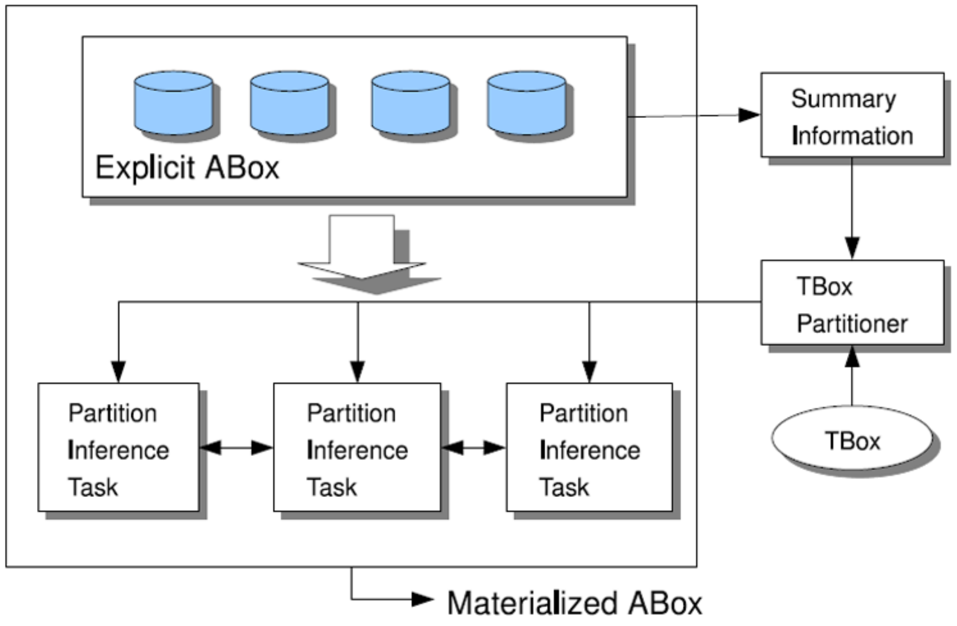

Figure 1.
Parallel ABox reasoning framework.

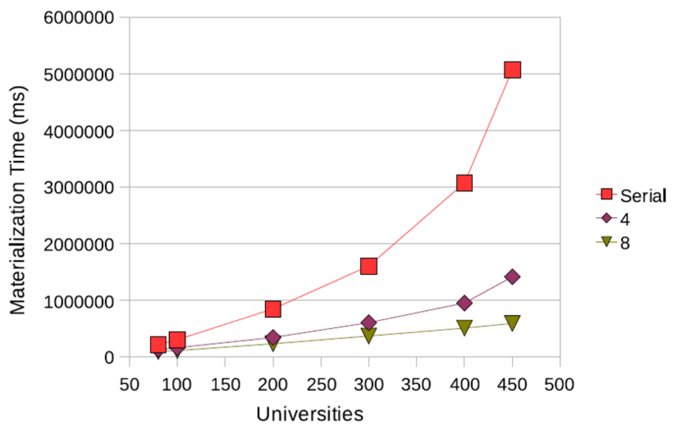

Figure 2.
Comparison of materialization times

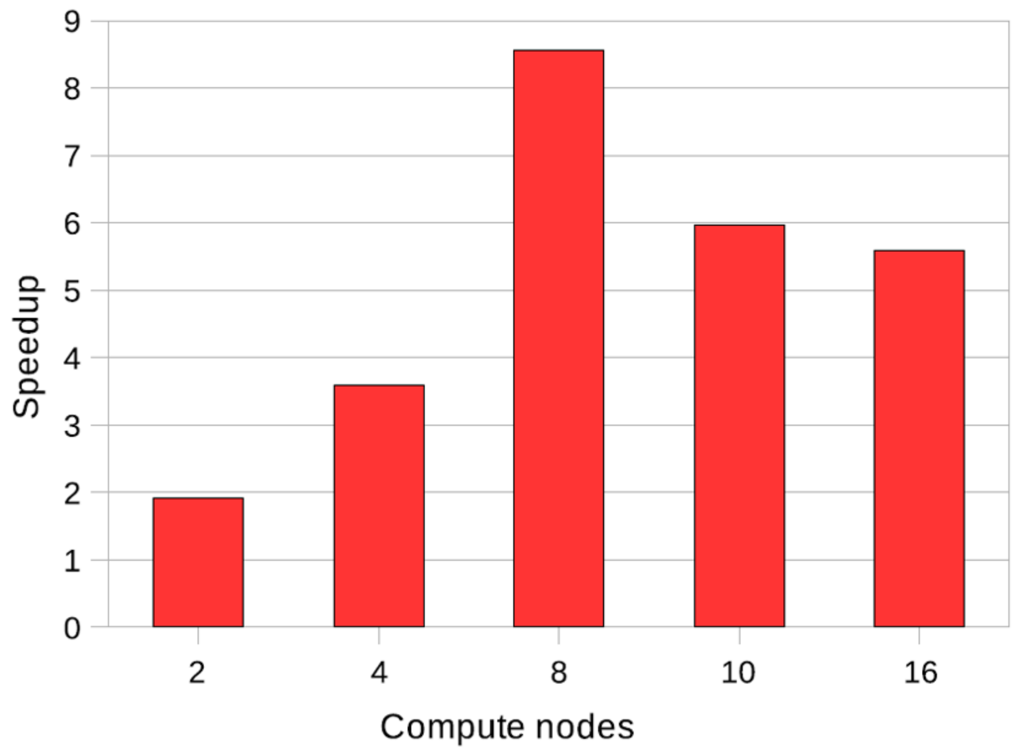

Figure 3.
Speedup on 2,4,8,10,16 nodes

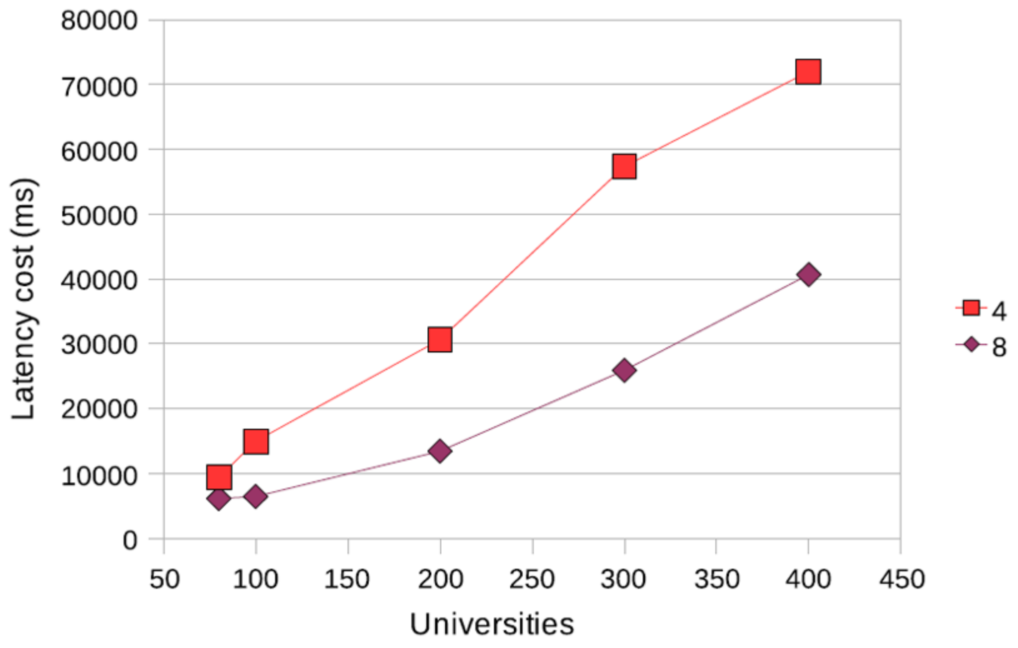

Figure 4.
Latency as a percentage of computation time

Table 1

Supported axiom forms and corresponding forward-chaining rules with home, generated and helper concepts/roles

Axiom	Forward-chaining rule(s)	Home	Generated	Helper
$C_1 \sqsubseteq C_2$	$C_1 (i_1) \rightarrow C_2 (i_1)$	C_1	C_2	-
$C_1 \sqsubseteq \forall R_1. C_2$	$C_1 (i_1) \wedge R_1 (i_1, i_2) \rightarrow C_2 (i_2)$	C_1	C_2	R_1
$\exists R_1. C_2 \sqsubseteq C_1$	$C_2 (i_2) \wedge R_1 (i_1, i_2) \rightarrow C_1 (i_1)$	C_2	C_1	R_1
$C_1 \sqsubseteq C_2 \sqcap C_3$	$C_1 (i_1) \rightarrow C_2 (i_1), C_1 (i_1) \rightarrow C_3 (i_1)$	C_1	C_2, C_3	-
$C_2 \sqcap C_3 \sqsubseteq C_1$	$C_2 (i_1) \wedge C_3 (i_1) \rightarrow C_1 (i_1)$	C_2	C_1	C_3
$\top \sqsubseteq \forall R_1. C_2$	$R_1 (i_1, i_2) \rightarrow C_2 (i_2)$	R_1	C_2	-
$\top \sqsubseteq \forall R_1^{-}.C_2$	$R_1 (i_1, i_2) \rightarrow C_2 (i_1)$	R_1	C_2	-
$R_1 \sqsubseteq R_2$	$R_1 (i_1, i_2) \rightarrow R_2 (i_1, i_2)$	R_1	R_2	-
$R_1 \sqsubseteq R_2^{-}$	$R_1 (i_1, i_2) \rightarrow R_2 (i_2, i_1)$	R_1	R_2	-
$R_1^{+} \sqsubseteq R_1$	$R_1 (i_1, i_2) \wedge R_1 (i_2, i_3) \rightarrow R_1 (i_1, i_3)$	R_1	R_1	R_1

Table 2

(a) An example TBox. (b) TBox table for axioms and roles from the example TBox. (c) An example partition of the TBox table.

	Home	Helper	Generated	
$C_1 \sqsubseteq \forall R_1. C_2$	C_1	R_1	C_2	
$R_2 \sqsubseteq R_1$	C_2	-	-	P_1: $\{C_1, C_2, C_3\}$
$C_3 \sqcap C_4 \sqsubseteq C_2$	C_3	C_4	C_2	P_2: $\{C_4, R_1, R_2\}$
	C_4	-	-	
$\top \sqsubseteq \forall R_2. C_3$	R_1	-	-	
	R_2	-	C_3, R_1	

Algorithm 1

Min-min Partition

```
Require: TBox T, Number of partitions N
Ensure: Partitions of TBox $\{P_1, P_2, \ldots, P_N\}$
  $P_i = \varnothing$ for $i \in \{1, 2, \ldots, N\}$
  $W_i = 0$ for $i \in \{1, 2, \ldots, N\}$
  $S = Concepts(T) \cup Roles(T)$
  while $S \neq \varnothing$ do
    $c = NULL$ {Chosen type}
    $p = 0$ {Chosen partition}
    $minCost = \infty$
    for $i \in S$ do
      for $j \in \{1, 2, \ldots, N\}$ do
        $Cost = W_j + Count(i)$
        for $H \in Helper(i)$ do
          if $H \notin P_j$ then
            $Cost+ = Count(H)$
        if $Cost \leq minCost$ then
          $c = i, p = j, minCost = Cost$ {Best pair so far}
    $P_p = P_p \cup \{c\}$
    $S = S - \{c\}$
    $W_p = minCost$ {Update partition weights}
Return
```