Efficient Management of Very Large Ontologies*

Kilian Stoffel, Merwyn Taylor and **Jim Hendler**
University of Maryland
Computer Science Department
College Park, MD, 20742
{stoffel, mtaylor, hendler}@cs.umd.edu

Abstract

This paper describes an environment for supporting very large ontologies. The system can be used on single PCs, workstations, a cluster of workstations, and high-end parallel supercomputers. The architecture of the system uses the secondary storage of a relational data base system, efficient memory management, and (optionally) parallelism. This allows us to answer complex queries in very large ontologies in a few seconds on a *single* processor machine and in fractions of a second on *parallel* super computers. The main contribution of our approach is the open architecture of the system on both the hardware and the software levels allowing us easily to translate existing ontologies for our system's use, and to port the system to a wide range of platforms.

Introduction

Ontologies have been a part of research in AI for a long time, for example, ontology-based thesauri have been an important part of research in Natural Language Processing. In the last few years, however, ontologies have become increasingly important throughout AI. Systems such as CYC (Lenat & Guha 1991) and ISI's Sensus Project (Knight & Luk 1994) have shown the need for larger ontologies, as have newer research directions such as Knowledge Discovery in Databases (KDD) and intelligent internet-search engines, to name but two. With this growing interest, new efforts to make ontologies accessible to a larger user community, and to scale the size of these ontologies, have been undertaken. In addition, projects such as the the Knowledge Sharing Effort, which facilitate the combination of different small ontologies into larger and more complex ones, increase the need for scalable ontology support tools. Currently, most ontology-management systems cannot support the extremely large ontologies needed for such projects.

In this paper, we focus on supporting the querying and management of significantly larger ontologies. The system described here was created to manage ontologies of essentially unlimited size. In the next section, we describe different approaches to ontology management. The section after that provides some example ontologies used in practical applications. We then describe the implementation of our system and provide a series of performance results on *single* and *multi* processor machines showing that we can answer complex queries in very large ontologies in a few seconds on a *single* processor machine and in fractions of a second on *parallel* supercomputers.

Related Work: Large Ontologies

There is some dispute in the KR community as to what exactly an "ontology" is. In particular, there is a question as to whether "exemplars," the individual items filling an ontological definition count as part of the ontology. Thus, for example, does a knowledge base containing information about thousands of cities and the countries they are found in contain one assertion, that countries contain cities, or does it contain thousands of assertions when one includes all of the individual city to country mappings. While our system can support both kinds quite well, it is sometimes important to differentiate between the two. We will use the term *traditional ontology* for the former, that is those ontologies consisting of only the definitions. We use the term *hybrid ontologies* for the latter, those combining both ontological relations and the instances defined thereon. This second group may consist of a relatively small ontological part and a much larger example base, or as a dense combination of relations and instances(Stoffel *et al.* 1997).

Currently, there is a significant amount of research being done in the area of ontology development and management. Most of this work can be classified in three, often overlapping, categories: efforts to create large ontologies, to define expressive languages for representing ontological knowledge, and to implement systems which support ontology-based applications.

Given page limitations, it is not feasible for us to give an exhaustive list of all projects currently dealing with the creation of ontologies. Some significant examples include: efforts to create large ontology-based thesauri and dictionaries such as the *Sensus* project at ISI (Knight & Luk 1994), or the *WordNet* project at Princeton (Miller 1996), efforts to develop domain specific ontologies such as those used in medicine (e.g. UMLS (UMLS 1994), SnoMed (Code 1993)), and efforts to populate large common-sense ontologies such as the US *CYC* project (Lenat & Guha 1991) and the Japanese Knowledge Archive project (Inc. 1992). [1]

As well as these efforts to create specific ontologies, an important concern in ontology research is to define expressive languages which can be used to define the ontologies. This will be very important if ontologies are to become easily accessible, reusable, and combinable. Examples of current efforts include work in *Ontolingua*(Gruber 1992), *Kif*(Gruber 1990), and *Conceptual Graphs*(Sowa 1984).

A third set of projects are those focusing on implementations of ontology management systems. One group of such systems are knowledge representation systems which also provide ontological support, such as *Loom*(MacGregor 1994),*Classic*(Borgida & Patel-Schneider 1994), *CYC*(Lenat & Guha 1991), *Sneps*(Shapiro & Rapaport 1992), and *Kris*(Baader *et al.* 1994) (among many others). While all of these systems support their own languages, and all are very expressive, they are currently not well suited to host very large ontologies because they lack secondary memory support, database integration and other such techniques critical for scaling KR systems to extremely large ontologies (and especially to the often even larger hybrid ontologies necessary for many current applications).

Alternatively, there are some projects designed to directly examine issues in scaling KR systems such as FramerD(Haase 1996), Lee and Geller's (Lee & Geller 1996) work, GKB(Karp & Paley 1995), SHRUTI(Shastri & Ajjanagadde 1993) and PARKA(Evett, Hendler, & Spector 1994). All of these systems are scalable to a certain extent but most of them are still quite limited– FramerD is much closer to an Object Oriented Data Base System than to a KR system, GKB relies on Loom knowledge bases and is thus limited by the scalability of Loom, Lee and Geller's system is limited to a restricted set of queries on specific parallel computers (the CM2 and CM5), and SHRUTI is very limited in the number of conjunctions it can handle in a single query.

The system we will describe here, is motivated by the PARKA research done at UMD. We are extending these algorithms into a system that does not *require* supercomputers. It supports a wide variety of computer systems from PCs to Workstations although it still does scale up to high end parallel computer systems. In our work, a well-defined low-level input language enables one to write simple translators to reuse KBs/Ontologies defined for other systems. The use of secondary storage, realized by using a relational data base and efficient memory management allows us to host the largest existing ontologies. In addition, we show that the algorithms can be parallelized to provide even further scaling. (These parallel algorithms are defined using generic message passing primitives and scheduling mechanisms to provide independence from specific machine architectures, but we will not discuss this in detail in this paper.(Stoffel, Hendler, & Saltz 1995))

[1]The CYC ontology is part of a large system which includes inference algorithms and language definitions, thus spanning all three of the above categories.

Motivating Examples

In this section, we describe three ontologies we are using for various applications and for the testing of our system. We choose these three as all are publicly available to the research community.[2] We will briefly describe their contents, structure, and how they are used. Based on the definitions in the previous section, two of these ontologies, UMLS and WordNet, are "Traditional Ontologies." The third, Caper, is a hybrid ontology.

UMLS (Unified Medical Language System)

UMLS is a large medical ontology created by the National Library of Medicine. NLM developed this system to support a wide range of medical applications including descriptions of biomedical literature, categorization of clinical records, development of patient databanks, and domain definitions for knowledge-based systems. The main thesaurus of UMLS consists of 178,904 frames when represented as a semantic net. There are a total of 7 relations (link types) defined over these providing a total of 1,729,817 assertions in all. We use this ontology in a project at the Johns Hopkins Hospital to create a high level interface for browsing several lab databases as well as to do some data mining tasks on these databases(Stoffel *et al.* 1997).

Word Net

"WordNet is an on-line lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory" ((Miller 1996),page 1). Although the system was primarily developed with natural language processing researchers in mind, the ontology it defines has been proposed as a standard one for comparing knowledge-based systems. Like UMLS, the ontology is comprised of concepts in a class-subclass hierarchy and a small number of attributes relating these. These attributes and their occurrences are: Sense (167,716), Antonym (7,189), SimilarTo (20,050), ParticipleOfVerb (89), Attribute (636), AlsoSee (3,526), Pertainym (3,539), DerivedFromAdj (2,894), partHolonym (5,694), substanceHolonym (366), partMeronym (5,694), memberMeronym (11,471), memberHolonym (11,472), attribute (636), substanceMeronym 366, Entailment (435), and Cause (204). The total number of frames is 217,623 and the ontology contains 385,771 assertions.

Caper

Caper is a large hybrid ontology used by a case-based AI planning system(Kettler 1995). While it is about the same size as UMLS, it has a very different structure. In particular, Caper has significantly more attributes (112) and a shallower ontology. In all, these attributes relate 116,297 frames, and 1,768,832 total links. Three large Caper knowledge bases are available on the World Wide Web at (http://www.cs.umd.edu/projects/parka).

[2]Other very large examples have been used in applications of our system, but they contain private and/or licensed material.

System Structure

We are developing a system which supports the storage, loading, querying and updating of very large ontologies such as those above. Although this paper concentrates mainly on the querying, as this is crucial in enabling these other capabilities, in this section we describe the overall structure of our approach.

The system consists basically of three layers. The lowest level of the system is based on a relational data base management system (RDBMS). This layer manages all I/O operations, as well as some simple relational operations such as selects, joins and projections used in processing the ontology queries. This layer is also used to maintain (insert, delete, update) the relationships in the ontology and, in the case of hybrid ontologies, the lower-level instance data.

The second level consists of a set of efficient inferencing algorithms. As well as simple inheritance, these algorithms also support inferential distance ordering inheritance (Horty, Thomason, & Touretzky 1987), transitive closures on arbitrary predicates, and transfer-through inferences(Lenat & Guha 1991). This level sends basic requests (I/O and relational operations) to the database level. The relational table returned is used by the inferencing modules and the result produced is is sent back to the first level for storage in the RDBMS.

The third level is a general-purpose user interface. It allows the user to insert, delete and update information in the ontology and also to pose queries. These operations can be done through a graphical user interface. However, to allow large ontologies to be embedded in other computer systems (such as our datamining program (Stoffel *et al.* 1997)), the system also can be invoked via an API which allows front-end processes to access the system. The API offers a superset of the functionality of the graphical interface in the form of Remote Procedure Calls (RPCs).

Internal Data Structures

All information contained in the ontology is stored in the relational database system. In this section we describe the two important specialized datastructures used to maintain this information.[3]

For efficiency, we separate out those links used in the often-called inheritance algorithms, and load these directly into memory as soon as an ontology is loaded into the system. These *structural links*, which encode the class/subclass hierarchy, are converted from a relational representation (table) into the first datastructure, a DAG (directed acyclic graph) which is kept in memory because it is repeatedly accessed during the inferencing, and it would be too inefficient to load it from disk repeatedly. The DAG is encoded as an array of integers which provides for extremely efficient access. This is an example of how our approach optimizes performance for ontological, as opposed to database, use.

The second important datastructure maintains all the other, non-structural, attributes. These are stored in binary relational tables in the database. For each attribute, there exists a table with two columns with the first column containing an integer "ID" corresponding to a specific frame, and the second contains the ID for the value for this attribute. The advantage of keeping all attributes in separate binary tables is that only the minimal amount of data has to be loaded when information for a concept is requested. If multiple attributes were stored in a single relation, the whole table would have to be loaded even if only one attribute was of interest. Again, this is an optimization designed to support large ontologies.

Serial Algorithm

The serial algorithm to process ontological queries consists of two main parts. First, a preprocessing step divides a complex query into its underlying constituents and generates a join tree specifying the order in which to execute and combine the subqueries that is expected to best optimize the performance. The preprocessor also schedules substring matches for queries in which variables specifying a partial string are used (an example of this is query UMLS2, defined below).

Second, a loop is used to fetch the data needed for each join and to perform the requisite inferencing. This second part consists of three main steps. First, the relational tables needed for a particular subquery's inferencing are loaded from the RDBMS. These tables can be the relations specifying a particular attribute or may be intermediate results produced by a previous iteration. Following this, the inference algorithms (inheritance, closure, etc) are applied in core using the data loaded in the previous step. Finally, the resulting data is joined with previous intermediate results and stored back into the database. This algorithm repeats until all of the subqueries in the join tree are completed.

A key feature of this algorithm, keeping it efficient for very large ontologies, is this use of the relational database to store the ontology. Since these tables, which can be very large, are not kept in primary memory, the inferencing algorithms can be made very efficient. In particular, for very large ontologies we can exploit space/time tradeoffs by using a maximal portion of the online memory to process inferencing instead of using it to store the ontology and/or the large intermediate datastructures which may be generated in the loop described above.[4]

Example Queries and Results

To test our system, we have run it on a wide range of queries over the large ontologies described above and a number of others. In this paper, we present a small sample of these that appear to be typical of our performance, but are easily described. These queries are expressed with respect to the terms and attributes described previously. Each query is a conjunctive query that consists of variable and constants

[3] In the actual implementation there are also a number of other datastructures used to maintain indices, translation tables, and other such low-level data, but space precludes the discussion of these details.

[4] In practice, the efficiency gained by the algorithms more than outweighs the overhead of moving data to and from the disk.

that must unify against the ontology. For each query we present the number of variables in the query and the number of matches found in the ontology.

UMLS queries are used in medical systems as described above. We choose two queries that are typical of the keyword search and ontological inferencing used in medical informatic systems.

UMLS1: Find everything that *isA* streptococcaceae and a *sibling* of lactococcu and *is qualified by* chemistry. (1 variable, 4 results)
UMLS2: Find everything that *isA* organism and that contains the string *virus*. (1 variable, 346 results)

The WordNet system comes with a query engine which can be used for finding synonyms, hierarchical relations, etc. This query engine is significantly stressed by queries which must explore large subontologies. We demonstrate our system on two of these, particularly finding all animals and mammals respectively in WordNet. The third query demonstrates the efficiency of our algorithm on a complex query combining two attributes (hyponyms and member meronyms). Such queries are difficult to express to the WordNet query engine and extremely inefficient.

WordNet1: Find all senses of *animal* and all of their *hyponyms*. (1 variable, 7691 results)
WordNet2: Find all senses of *mammal* and all of their *hyponyms* (1 variable, 2231 results)
WordNet3: find all senses of *tree* and all of their *hyponyms* which are *member meronyms* of all senses of *genus citrus*. (1 variable, 62 results).

The queries used in Caper were generated by a case-based planning system as part of its problem solving as reported in (Kettler 1995). These three queries are more complex than the previous in that they contain 5-10 attributes relating up to 10 variables, which we believe is typical of useful queries in these sorts of hybrid ontologies. (For these queries we use a paraphrase rather than the form above, so we also specify the number of different attributes they include.)

Caper1: Find all plans in which a train delivered a package to a particular location. (5 variables, 6 attributes, 101 results).
Caper2: Find all top-level plans using a Regular truck (6 variables, 7 attributes, 269 results).
Caper3: Show me the cities in which a tanker has been used for delivering a liquid cargo. (7 variables, 8 attributes, 23 results).

Results Using our system, the queries above were run on an IBM RS6000 workstation. We report the timings in Table 1. We were able to issue queries WordNet1 and WordNet2 to the query engine supplied with the WordNet ontology. The WordNet engine evaluated WordNet2 in 17 seconds, as compared to our 3.2 seconds. For WordNet2, our system again takes about 3 seconds, while the WordNet query engine took 4 minutes, followed by a request to restrict the query. Further, in our system the timings for these two queries are very similar differing by only .2 seconds despite the large difference in number of results returned, while WordNet shows a great difference particularly when the subontology necessary to a query is also large.

Query	Time	Query	Time
WordNet 1	3315	Caper 1	713
WordNet 2	3123	Caper 2	1051
WordNet 3	4440	Caper 3	1776
UMLS 1	1845	UMLS 2	23

Table 1: Sequential results (times in ms.)

As can be seen, these single processor results are quite good, and they are significantly better than those reported in the literature for the systems discussed in the related work section. Unfortunately, direct comparisons are impossible because the cited papers do not report exactly what queries were used with WordNet, UMLS, or other publicly available ontologies.

Parallel Algorithms

The most obvious way to parallelize the basic algorithm described above is a "task parallelization" approach, evaluating each subquery in parallel and combining all results as they become available (c.f. (Andersen *et al.* 1994)). However, this parallelization tends to utilize only a small number of processors and can leave other processors idle especially when queries have fewer subqueries than the available number of processors. (For example WordNet1 and 2 would only use one of *n* available processors). It is our experience, however, that in handling large ontologies, these sorts of queries are quite prevalent. For example, all of the queries reported above would allow processors to go idle even on a 16-node machine.

To get better utilization for large ontologies, we move to a "data parallel" approach. In particular, each processor only loads parts of the relational table for a given query, and subquery processing on these parts can occur in parallel. The algorithm, for n processors:

```
On one processor, preprocess the query and broadcast
this to all other nodes
    For each subquery
        On processor P, load the Pth partition of the
            necessary relational tables (where a partition
            is 1/n tuples)
        On processor P, execute the required inferences
            using data in partition P
        Processors all-to-all broadcast partial results
            for this subquery
        On processor P, for each partial result received
            join the result with previous intermediate
            result and write this into the database
            as a new intermediate result
When no more subqueries, processors scatter partial
    results and gather these on Node0.
Return the result gathered on Node0.
```

This approach has a significantly higher degree of parallelism then the former. In theory, the maximal number of processors that could be used is only limited by the size of

Query	1	2	4	8	16
WordNet1	3315	1756	981	655	497
WordNet2	3123	1552	888	532	404
WordNet3	4440	3014	1405	917	739
UMLS1	1845	800	414	223	144
Caper1	713	542	434	381	363
Caper2	1051	904	883	909	1201
Caper3	1776	1293	1036	955	1058

Table 2: Parallel Results (times in ms.)

the largest relation. In practical applications, this number is typically much higher then the number of available processors. There is, however, a trade-off between the degree of parallelism and the time for the all-to-all broadcast used to exchange partial results. The amount of data sent per processor is approximately $\frac{(n-1)*|result|)}{n}$, which is roughly the same amount of data each node will receive in total. (This is only an approximation because we have no guarantee that the data is perfectly distributed.) Thus, for a given node the amount of data sent after each subquery grows with the number of processors and for large *n* is about the same size as the size of the whole result (i.e. as n gets large, nearly n^2 data must be broadcast). On the other hand the amount of total disk I/O operations per processor is linearly reduced and these operations are often much more expensive than the broadcasts. In addition, the total inferencing part also speeds up, because the amount of data the inference algorithms are applied to is reduced by a factor of *n*.

Parallel Results

The queries presented previously were evaluated by our system in parallel using 2, 4, 8, and 16 processors on an IBM SP2 parallel computer. We report the timings in Table 2.[5] For UMLS and WordNet, the "traditional ontologies," the query evaluation times decreased as the number of available processors increased. The times for WordNet3, for example, drop from 4.4 seconds to .74 seconds on 16 processors (about 40% efficiency).

The query times for the hybrid ontology, Caper, do not exhibit the same behavior. Our system performs well on the other two primarily because of the efficiency of our inferencing algorithms. For the examples on the hybrid ontology, our system speeds up through eight nodes but degrades slightly thereafter. This is because the hybrid ontologies require significantly more complex database-like processing. Since our system is currently optimized for larger ontologies, the current database algorithms do not speed up enough to overcome the additional communication burden. This can be overcome by doing a better data distribution, and we are currently exploring this issue.

[5]UMLS2, which took 23 milliseconds on a serial machine was not used as there is no parallel advantage for such simple queries.

Conclusion and Future Work

We have presented efficient algorithms for query processing in several existent very large ontologies. We have shown how using secondary storage in the form of a relational database, using efficient memory management and using high performance computing technology enabled us to handle very large ontologies for both single and parallel processing applications. We demonstrated the efficiency of these algorithms using three very large ontologies each containing hundreds of thousands of assertions, showing how we process queries in a few seconds on a single-processor and in fractions of seconds on a multi-processor system, showing the high degree of scalability for our work.

The next steps in the development of the system will be to automate the translation of ontologies from/to some of the standard representation languages such as Ontolingua and Kif. We also intend to extend the number of KBs and sample queries (with details and results) which we have already made available on the WWW (http://www.cs.umd.edu/projects/parka). We are also continuing to push the application of this technology, especially to the use of large ontologies in KDD.

As discussed above, our system is efficient for hybrid ontologies, but may not scale as well to very large parallel supercomputers. To solve this problem, we are are exploring the integration of our system with other, more advanced, databases particularly parallel DB systems. (For example, the binary structure of the relations we use to record attributes is similar to those used by the parallel RDBMS MONET(Holsheimer, Kersten, & Siebes 1996), and thus map nicely onto that system.) Similarly, other aspects of our system resemble features of object-oriented databases and we are exploring OSQL and other such languages for integration with these.

Acknowledgments

This research was supported in part by grants from ONR (N00014-J-91-1451), ARPA (N00014-94-1090, DAST-95-C003, F30602-93-C-0039), and the ARL (DAAH049610297). Dr. Hendler is also affiliated with the UM Institute for Systems Research (NSF Grant NSF EEC 94-02384).

References

1994. *Proceedings of the 12th National Conference of the American Association for Artificial Intelligence*, AAAI Press, MIT Press.

Andersen, W.; Evett, M.; Hendler, J.; and Kettler, B. 1994. *Massively Parallel Matching of Knowledge Structures*. Massively Parallel Artificial Intelligence. AAAI/MIT Press.

Baader, F.; Hollunder, B.; Nebel, B.; Profitlich, H.-J.; and Franconi, E. 1994. An empirical analysis of optimization techniques for terminological representation systems or "making KRIS get a move on". *Applied Intelligence* 4(2):109–132.

Borgida, A., and Patel-Schneider, P. 1994. A semantics and complete algorithm for subsumption in the classic description logic. *Journal of Artificial Intelligence Research* 1.

Code, R. 1993. *Systemized Nomenclature of Human and Veterinary Medicine: SNOMED*. College of American Pathologists, American Veterinary Medical Association.

Evett, M.; Hendler, J.; and Spector, L. 1994. Parallel knowledge representation on the connection machine. *International Journal of Parallel and Distributed Computing* 22(2).

Gruber, T. 1990. he development of large, shared knowledge-bases: Collaborative activities at stanford. Technical report, Knowledge Systems Laboratory.

Gruber, T. R. 1992. Ontolingua: A mechanism to support portable ontologies. Technical report, Knowledge Systems Laboratory.

Haase, K. 1996. Framerd: Repersenting knowledge in the large. *IBM Systems Journal*.

Holsheimer, M.; Kersten, M. L.; and Siebes, A. 1996. Data surveyor: Searching for nuggets in parallel. In Fayyad, U. M.; Piatetski-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds., *Advance in Knowledge Discovery and Data Mining*. AAAI/MIT Press.

Horty, J.; Thomason, R.; and Touretzky, D. 1987. A skeptical theroy of inheritance in nonmonotonic semantic networks. Technical Report CMU-CS-87-175, Carnegie Mellon, Department of Computer Science, Pittsburgh, PA, USA.

http://www.cs.umd.edu/projects/parka.

1995. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, San Francisco, CA.

Inc., E. D. R. 1992. A plan for the knowledge archives project. Technical report, Technical Report distributed by the Japan Society for the Promotion of Machine Industry, English language addition.

Karp, P. D., and Paley, S. M. 1995. Knowledge representation in the large. In IJCAI-95 (1995), 751–758.

Kettler, B. B. 1995. *Case-based Planning with High-Performance Parallel Memory*. Ph.D. Dissertation, University of Maryland, College Park.

Knight, K., and Luk, S. 1994. Building a large knowledge base for machine translation. In AAAI-94 (1994).

Lee, E., and Geller, J. 1996. Parallel transitive reasoning in mixed relational hierarchies. In Aiello, L. C.; Doyle, J.; and Shapiro, S., eds., *Principles of Knowledge Representation and Reasoning*.

Lenat, D. B., and Guha, R. 1991. *Building Large Knowledge-Based Systems*. Addison-Wesley.

MacGregor, R. M. 1994. A description classifier for the predicate calculus. In AAAI-94 (1994).

Miller, G. A. 1996. Human language technology. Technical report, Psychology Department, Green Hall, Princeton University.

Shapiro, S. C., and Rapaport, W. J. 1992. The sneps family. *Computers & Mathematics with Applications* 243–275.

Shastri, L., and Ajjanagadde, V. 1993. From simple associations to systematic reasoning. *Behavioral and Brain Sciences* 16(3):417–494.

Sowa, J. F. 1984. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley.

Stoffel, K.; Saltz, J.; Hendler, J.; Dick, J.; Merz, W.; and Miller, R. 1997. Semantic indexing for efficient grouping. In *submitted*.

Stoffel, K.; Hendler, J.; and Saltz, J. 1995. Parka on mimd-supercomputers. In *3rd International Workshop on Parallel Processing in AI*. Montral, Canada: IJCAI.

UMLS. 1994. *Unified Medical Language System*. National Library of Medicine.