Distributed and Scalable OWL EL Reasoning

[1]Raghava Mutharaju [1]Pascal Hitzler [1]Prabhaker Mateti
[2]Freddy Lécué

[1]Wright State University, OH, USA.

[2]Smarter Cities Technology Centre, IBM Research, Dublin, Ireland.

12th Extended Semantic Web Conference (ESWC 2015),
Portoroz, Slovenia

Overview

Overview

Introduction

- Existing reasoners run on a single machine.
- They are constrained by the resources available to a single machine.
- Automatic generation of axioms result in very large ontologies.
  - streaming data (sensors, tweets)
  - text
- Additional axioms are generated during the reasoning process.
- Existing reasoners are overwhelmed by these large ontologies.

Overview

$\mathcal{EL}^{++}$ profile

- Most of description logic $\mathcal{EL}^{++}$ is supported, which underlies OWL 2 EL profile.
- Axioms can have one of the following forms
  - $C \sqsubseteq D$, where C and D are defined by the grammar

$$C ::= A \mid \top \mid \bot \mid C \sqcap C \mid \exists r.C \mid \{a\}$$

$$D ::= A \mid \top \mid \bot \mid D \sqcap D \mid \exists r.D \mid \exists r.\{a\}$$

  - $r_1 \circ \cdots \circ r_n \sqsubseteq r$

  where A is a concept, r, r_i are roles, and a an individual.
- Axioms of the form $C \sqsubseteq \{a\}$ are **not** supported.

Normalization

All concept inclusions have one of the forms

- $A_1 \sqsubseteq B$
- $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$
- $A_1 \sqsubseteq \exists r.A_2$
- $\exists r.A_1 \sqsubseteq B$

All role inclusions have one of the forms

- $r \sqsubseteq s$
- $r_1 \circ r_2 \sqsubseteq r_3$

Classification

- We focus on the reasoning task called *classification*.
- It is the computation of the complete subsumption hierarchy, i.e. all logical consequences of the form $A \sqsubseteq B$ involving all concept names and nominals A and B.
- We use a set of rules to classify a given $\mathcal{EL}^{++}$ ontology.

Overview

Rules

Rn	Input	Action
R1	$A \sqsubseteq B$	$U[B] \cup= U[A]$
R2	$A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$	$U[B] \cup= U[A_1] \cap \cdots \cap U[A_n]$
R3	$A \sqsubseteq \exists r.B$	$R[r] \cup= \{(X, B) \mid X \in U[A]\}$
R4	$\exists r.A \sqsubseteq B$	$Q[r] \cup= \{(Y, B) \mid Y \in U[A]\}$
R5	$R[r], Q[r]$	$U[B] \cup= \{X \mid (X, Y) \in R[r]$ and $(Y, B) \in Q[r]\}$
R6	$R[r]$	$U[\bot] \cup= \{X \mid (X, Y) \in R[r]$ and $B \in U[\bot]\}$
R7	$r \sqsubseteq s$	$R[s] \cup= R[r]$
R8	$r \circ s \sqsubseteq t$	$R[t] \cup= \{(X, Z) \mid (X, Y) \in R[r]$ and $(Y, Z) \in R[s]\}$

Table: Completion Rules

Rule Processes

- Input ontology $\mathcal{O}$ can be partitioned into eight mutually disjoint ontologies, $\mathcal{O} = \mathcal{O}_1 \cup \cdots \cup \mathcal{O}_8$
- Ontology $\mathcal{O}_i$ is assigned to a subcluster (subset of machines in the cluster) SC_i
- Rule Ri must be applied only on $\mathcal{O}_i$
- From the available machines, eight subclusters are created, one for each rule.
- $\mathcal{O}_i$ is further divided up among the machines in the subcluster (not duplicated).

Rule Processes

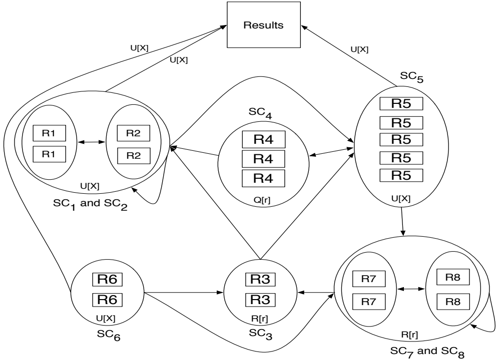

Figure: Node assignment to rules and dependency among the completion rules. For simplicity, only one node is shown to hold results.

Termination

repeat

  K_i := apply Ri on $\mathcal{O}_i$ once;
  broadcast(K_i);
  nUpdates := barrier-sum-of K_i;

until *nUpdates* $= 0$;

Algorithm 1: Wrapper for Ri

- K_i is associated with each Rule Ri.
- K_i is the number of updates made to result/intermediate sets.
- Barrier synchronization is used in waiting for K_i from all Ri
- If no rule process made an update, they quit; otherwise, they continue with another iteration.

Overview

Optimizations

- **Dynamic Load Balancing:** Idle nodes take (steal) work from busy nodes.
- **Rule Dependencies:** Rule Ri need not be triggered again if the output from rules it is depending on does not change.
- **Data Partitioning Strategy:** Most of the data required for rule application is available locally on each node.

Overview

Results

- Implemented in Java and is called DistEL
- Redis is the key-value store that was used
- Available from `http://github.com/raghavam/DistEL`
- Amazon's EC2 m3.xlarge instances are used (4 cores, 15GB RAM). 5GB given to JVM.
- Machine configuration meant to reflect commodity hardware.

Results

Ontology	Before	After
GO	87,137	868,996
SNOMED	1,038,481	14,796,555
SNOMEDx2	2,076,962	29,593,106
SNOMEDx3	3,115,443	44,389,657
SNOMEDx5	5,192,405	73,982,759
Traffic	7,151,328	21,840,440

Table: Number of axioms, before and after classification, in ontologies.

Results

Ontology	ELK	jCEL	Snorocket	Pellet	HermiT	FaCT++
GO	23.5	57.4	40.3	231.4	91.7	367.89
SNOMED	31.8	126.6	52.34	620.46	1273.7	1350.5
SNOMEDx2	77.3	OOM[a]	OOM[a]	OOM[a]	OOM[a]	OOM[a]
SNOMEDx3	OOM[a]	OOM[a]	OOM[a]	OOM[a]	OOM[a]	OOM[a]
SNOMEDx5	OOM[a]	OOM[a]	OOM[a]	OOM[a]	OOM[a]	OOM[a]
Traffic	OOM[b]	OOM[c]	OOM[c]	OOM[b]	OOM[b]	OOM[c]

Table: Classification times in seconds. OOM[a]: reasoner runs out of memory. OOM[b]: reasoner runs out of memory during incremental classification. OOM[c]: ontology too big for OWL API to load in memory.

Results

Ontology	8 nodes	16 nodes	24 nodes	32 nodes	64 nodes
GO	134.49	114.66	109.46	156.04	137.31
SNOMED	544.38	435.79	407.38	386.00	444.19
SNOMEDx2	954.17	750.81	717.41	673.08	799.07
SNOMEDx3	1362.88	1007.16	960.46	928.41	1051.80
SNOMEDx5	2182.16	1537.63	1489.34	1445.30	1799.13
Traffic	60004.54	41729.54	39719.84	38696.48	34200.17

Table: Classification time (in seconds) of DistEL

Results

Node	MB
R1	186.72
R2	0.81
R3	257.47
R4	0.79
R5	1970
R6	380.61
R7	0.79
R8	1470.00
Result	654.53
Total	4921.72

Table: Memory taken by Redis on each node for traffic data

Results

Nodes	Runtime	Speedup
8	544.38	1.00
16	435.79	1.24
24	407.38	1.33
32	386.00	1.41
64	444.19	1.22

Table: Speedup achieved by DistEL on SNOMED CT

Overview

Future Work

- This is a work-in-progress and more work needs to be done w.r.t performance improvements.
- Explore other ontology partitioning strategies as well as rule sets (ELK etc.).
- Fine grained analysis on larger datasets with higher number of nodes in the cluster.
- Use multi-threading
- Alternatives to Redis including custom data structures.

Overview

Conclusion

- Existing reasoners were not able to classify traffic data and other large ontologies.
- DistEL, a distributed reasoner is able to classify the large ontologies.
- It shows good speedup with increase in the number of machines in the cluster.

Thank You

Thank you