Efficient RDF Stream Reasoning with Graphics Processing Units (GPUs)

Chang Liu
University of Maryland
USA
liuchang@cs.umd.edu

Jacopo Urbani
Vrije Universiteit Amsterdam
The Netherlands
jacopo@cs.vu.nl

Guilin Qi
Southeast University
China
gqi@seu.edu.cn

ABSTRACT

In this paper, we study the problem of stream reasoning and propose a reasoning approach over large amounts of RDF data using graphics processing units (GPU) to improve the performance. First, we show how the problem of stream reasoning can be reduced to a temporal reasoning problem. Then, we describe a number of algorithms to perform stream reasoning with GPUs.

1. INTRODUCTION

Recently, a new research area, called *stream reasoning*, has emerged to address the problem of performing reasoning for very dynamic inputs. Currently, stream reasoning is visioned as a promising research area with many potential applications such as monitoring and traffic patterns detection, financial transaction audits, wind power plant monitoring, or situation-aware mobile services [5].

So far, several approaches have been proposed to tackle this problem [1]. However, they mostly propose serial algorithms, and thus leave unsolved the problem whether it is possible to leverage modern parallel computing architectures to achieve shorter responding time and better scalability. The work presented in [3] is the first to describe a RDFS reasoning engine using a GPU architecture. However, it does not target stream reasoning and optimize the computation to perform large batch computations.

In this work, we propose a system to perform stream reasoning on RDF data using the GPU computing architecture. In this context, there is both a theoretical and a practical challenge. The first is grounded on a lack of a formalization of the problem, and we address it by showing how stream reasoning can be formalized into a temporal reasoning problem, and consequently proposing a compact representation of the data under such formalization. The second is more technical since it requires to design methods to perform reasoning very quickly before new data becomes available. Since the efficiency of a GPU hinges on the data alignment of the input, we first propose a hash-based GPU encoding algorithm that can efficiently translate the string terms into integer IDs of a fixed-length. Then, we present a number of novel parallel algorithms using the CUDA[1] programming language to perform reasoning over dynamic data. Our preliminary evaluation shows that our approach can compute the closure for an input of hundred thousand LUBM triples in a few milliseconds, making possible to reason over large streams of structured data.

2. STREAM REASONING AS A TEMPORAL REASONING PROBLEM

First, we define a RDF stream as a pair $s = \langle \mathcal{KB}, \mathcal{S} \rangle$ where $\mathcal{KB}$ is the *background knowledge*, which is an RDF graph, while $\mathcal{S}$ is the *stream*, which is defined as a sequence of (τ_i, t_i), where τ_i is a triple, and $t_i \in \mathcal{N}$ is a timestamp. We say τ_i is *observed* or *arrives* at time t_i. We assume $\forall i.t_i \leq t_{i+1}$.

In this work we consider a minimal fragment of RDFS, called ρdf [4] to perform the inference. ρdf has a deductive system $\vdash$ which can be used to compute all triples that can be entailed from a given graph in polynomial time. The deductive system contains several rules such as the following one:

$$(X, \mathtt{sp}, Y) \quad (Y, \mathtt{sp}, Z) \Rightarrow (X, \mathtt{sp}, Z)$$

Here `sp` is in the ρdf vocabulary, meaning a relationship of *subproperty* between two properties. The deductive system naturally defines a *proof* of H from G, denoted as $G \vdash H$, i.e. each triple in H is either in G, or can be derived from G using ρdf rules.

In a typical stream reasoning scenario, we are required to perform inference on a RDF graph composed by the background knowledge base and all RDF triples in a *window* of an RDF stream over time. A *window* is a time interval $[t_1, t_2]$, where $t_1 < t_2$, $t_1, t_2 \in N$. A window $[t_1, t_2]$ has size $t_2 - t_1$. The snapshot of the stream $\mathcal{S}$ in window $[t_1, t_2]$, denoted as $\mathcal{S}[t_1, t_2]$ is the set $\{\tau : \exists t.t_1 \leq t \leq t_2 \wedge (\tau, t) \in \mathcal{S}\}$.

More formally, given an RDF stream $\langle \mathcal{KB}, \mathcal{S} \rangle$, and a fixed window size w, a stream reasoning program is required to decide RDF graph G_t, such that $\mathcal{KB} \cup \mathcal{S}[t - w, t] \vdash G_t$ for each moment $t = 0, 1, ...$ [2]

2.1 From RDF Stream Reasoning to Temporal RDF Reasoning

[1] `http://www.nvidia.com/object/cuda_home_new.html`
[2] Here, for $t < w$, we treat $t - w$ as equivalent to 0

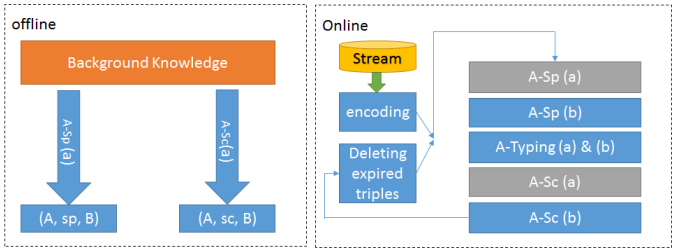

Figure 1: The general workflow of the stream reasoning engine.

A temporal RDF triple $\tau : \lambda$ is a pair of an RDF triple τ and a temporal annotation λ. A temporal annotation λ is a union of a finite number of intervals.

Temporal RDF has a deductive system which is an extension of ρdf rules. The temporal RDF deductive system also naturally defines a proof of H from G, denoted as $G \vdash_t H$.

Given an RDF stream $s = \langle \mathcal{KB}, \mathcal{S} \rangle$, the *temporal correspondence* of s at moment t denoted as $\mathcal{T}(s,t)$, is a temporal graph defined by $\{\tau : [0,+\infty] | \tau \in \mathcal{KB}\} \cup \{\tau : [t', t' + w] | (\tau, t') \in \mathcal{S} \wedge t' \leq t\}$.

We can show the following result: Given an RDF stream $s = \langle \mathcal{KB}, \mathcal{S} \rangle$, an RDF graph G_t, and a moment t, $\mathcal{KB} \cup \mathcal{S}[t-w, t] \vdash G_t$, if and only if $\forall \tau.\tau \in G_t \Leftrightarrow \exists \lambda.t \in \lambda \wedge \mathcal{T}(s,t) \vdash_t \tau : \lambda$. This result allows us to use a temporal RDF deductive system to compute the closure avoiding the expensive deletions required in the incremental process. To further speed up the inference process, we use a compressed representation of the temporal annotation: Instead of storing the temporal interval, we only store $t' + w$ for a triple τ arriving at t'.

3. STREAM REASONING USING CUDA

If we treat a RDF graph as a three-column table, then the execution of a rule translates in computing a relational join over the tables. The premises of each rule contains at most one ABox triple (i.e. the triples that do not contain `sp`, `sc`, etc.). Furthermore, the number of TBox triples (i.e. the triples that are not ABox triples).

First, our system executes the transitive closure rules over the ABox triples in the static knowledge base. Then, it executes the following steps after new triples arrive in the stream: Removing the expired triples; Encoding the new triples; Executing the rules. The system workflow is reported in Figure 1, and in the following we describe each phase in more detail.

Removing the expired triples. As discussed above, all expired triples are those $\tau : t'$ where $t' + w < t$. Therefore when a triple $\tau : t'$ is arrived, we keep track of $t' + w$ as τ's annotation. Then, we remove all triples whose annotation is less than the current time t. This operation can be computed in parallel using a parallel stream compaction algorithm like the one described in [2].

Compression of the RDF Stream. Before performing reasoning, our system encodes the textual terms of the triples into numeric IDs. We propose a two-phase hash-based algorithm to perform this operation: in the first phase, we compute the hash value of each variant-length string, while in the second phase we assign a unique ID to each hash value. The hash function is carefully chosen (e.g. SHA-1) so that the collision probability is negligible. While this still does not guarantee each string is mapped to a unique ID, we argue that it is a good compromise for efficiency.

3.1 Rules Execution on a RDF Stream

Our system applies each rule in the order illustrated in Figure 1 on both the stream and background knowledge data. This is the same execution order as the one used in [3].

However, differently from [3], our system needs to deal with the incremental part. Since TBox is usually much smaller than ABox, we cache all TBox triples in the GPU memory. We implemented two strategies in case the TBox is either static or present in the stream.

Static TBox. If the TBox is static, then we can omit all rules that deal with only TBox triples (i.e. the rules in grep box in Figure 1). All the other rules require a join between TBox and ABox. In our case, the TBox is cached in the GPU memory before the stream arrives, and this allows us to perform some preprocessing on the TBox (e.g. sorting and building hash table), so that computing the join between a dynamic ABox (only the incremental part) and the static TBox can be executed using well-known join algorithms like sort-join or hash-join. In our experiment, an hash-join has achieved the best performance.

Dynamic TBox. If the TBox is not static, then we cannot omit the TBox rules like the ones required to compute the transitive closure of `sp` and `sc`. Since computing the transitive closure requires an iterative process until fixpoint where each iteration produces redundant computation. To reduce such a redundancy, we employ incremental algorithms that computes only the join result over triples derived within two iterations. This reduces the computation and allows a substantial increase of performance.

4. REFERENCES

[1] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. Incremental Reasoning on Streams and Rich Background Knowledge. In *Proc. of ESWC '10*, pages 1–15, 2010.

[2] M. Billeter, O. Olsson, and U. Assarsson. Efficient stream compaction on wide SIMD many-core architectures. In *Proc. of HPG '09*, 2009.

[3] N. Heino and J. Z. Pan. RDFS Reasoning on Massively Parallel Hardware. In *Proceedings of ISWC*, pages 133–148, 2012.

[4] S. Muñoz, J. Pérez, and C. Gutierrez. Minimal Deductive Systems for RDF. In *Proceedings of ESWC*, pages 53–67, 2007.

[5] E. D. Valle, S. Ceri, F. van Harmelen, and D. Fensel. It's a Streaming World! Reasoning upon Rapidly Changing Information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.