Web Semantics: Science, Services and Agents on the World Wide Web 39 (2016) 25–46

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

# A multigranularity locking model for RDF

Mark Douglas Jacyntho [*,1], Daniel Schwabe

*Departamento de Informática, PUC-Rio, Av. Marquês de São Vicente, 225, Gávea, Rio de Janeiro - RJ, Brazil*

ARTICLE INFO

*Article history:*
Received 20 September 2015
Received in revised form
29 March 2016
Accepted 30 May 2016
Available online 11 June 2016

*Keywords:*
Semantic web
RDF
Transaction
Transaction isolation
Concurrency control
Lock

ABSTRACT

The advent of Linked Data is spurring the deployment of applications that use the RDF data model at the information tier. In addition to querying RDF data, there is also the requirement for online updates with suitable concurrency control. Client sessions in Web applications are organized as transactions involving requests that read and write shared data. Executing concurrently, these sessions may invalidate each other's data. This paper presents a locking model, which is a variant of multigranularity locking protocol (MGL), to address isolation between transactions that manipulate RDF data. Four "hierarchically" related granules are defined, as well as new read/write operations and their corresponding lock modes, specifically created for the RDF data model. These new operations allow greater concurrency than the classical read/write operations in relational databases. We assessed the performance of the proposed locking model through model simulation.



## 1. Introduction

In the last few years, there has been an increasing interest in semantic Web applications using the RDF data model as a persistent domain layer. Moreover, there are cross-references between widely spread applications, spurred in good part by the Linking Open Data (LOD) initiative [1].

The availability of large RDF stores opens up a big opportunity for exploration on how to use this new data model in large scale, specially, how to deal with concurrent read/write accesses.

### 1.1. Motivation

In essence, a data model is just a way to view the data. The established relational model views the data through relations and tuples. The RDF graph model, based on triples, is a natural representation for various types of applications (e.g., Facebook, Twitter, recommender systems, etc.), where entities are strongly connected with each other. In contrast with legacy RDBMS, these applications consider multi-valued properties to be so desirable in modeling real-life data that they support multi-valued properties by default. Querying for multi-valued and single-valued properties is done in exactly the same way, without concerns about the need to join with a third table to model an n-to-n relationship. Furthermore, the RDF model is more convenient if the application has high heterogeneity in its schema or frequent need for schema adaptation. RDF stores simplify the development of linked data applications, and also align very well with numerous algorithms and statistical techniques developed for graphs.

This raises the issue of the *writability* of RDF and Linked Data, as corroborated by the creation of SPARQL 1.1 update standard. In [2], Tim Berners-Lee pointed out:

"*When the Web was created, the idea was for a read–write Web.[...] If the Web is genuinely to be a read–write Web, and if we are to pursue a Web of Data, then it is essential that the Data Web should not be read-only*".

The problem to be tackled is the lack of a concurrency control mechanism that properly isolates transactions that read and write shared triples. The new questions addressed here are, can the traditional locking model used in relational databases [3], be reused? Are new locking models necessary? Is the optimistic locking approach the only choice? Is there a feasible pessimistic locking approach for RDF? Is there an opportunity to improve the classical locking models?

There are several studies that address the performance issue of complex queries over large amount of triples, proposing improvements in indexing and query optimization [4–8]. The vast majority of initiatives only address querying; only a small

* Corresponding author.
*E-mail addresses:* markjacyntho@gmail.com, mjacyntho@inf.puc-rio.br (M.D. Jacyntho), dschwabe@inf.puc-rio.br (D. Schwabe).

[1] Present addresses: Universidade Candido Mendes, Núcleo de Pesquisa e Desenvolvimento, Rua Anita Peçanha, 100, Parque São Caetano, Campos dos Goytacazes - RJ, Brazil; Instituto Federal de Educação, Ciência e Tecnologia Fluminense Campus Campos-Centro, Rua Dr. Siqueira, 237, Parque Dom Bosco, Campos dos Goytacazes - RJ, Brazil.

http://dx.doi.org/10.1016/j.websem.2016.05.002
1570-8268/

number supports online updates; and are all based on versioning (optimistic protocol or snapshot isolation). When using snapshot isolation (SI) [9] the transaction sees a committed consistent state of the data as it existed at the start of the transaction, and does not see any concurrent updates. Since SI was formalized in [9] it is known that it allows non-serializable executions that could destroy the database's consistency, in particular through the *write skew anomaly*. In addition, SI uses an optimistic approach too, and to prevent *lost updates,* a write–write conflict will be raised causing the latter transaction to abort at the end.

We observe that, in general, the approach for concurrency control used in Web applications is optimistic locking, with verification of conflicts at the end of the user interaction. Indeed, as presented in detail in Section 8, the current state of the art for locking in RDF stores are the optimistic and snapshot mechanisms. No doubt these are simpler approaches, suitable for the stateless behavior of the Web. Certainly in cases where short write transactions conflict minimally and long-running transactions are likely to be read-only, the optimistic approach should present better results. But it probably is not good for long-running write transactions competing with high-contention short transactions, since the long-running transactions are unlikely to be the first writer of everything they write, and so will probably be aborted. So, there are also many circumstances where the use of optimistic locking is improper, for example: when there are the so-called *hotspots*—a small subset of the data which is updated very frequently; in reservation systems with limited inventory items (car reservations, tickets to events, plane seats, etc.); or for complex data entry forms that consume a long time to be filled in.

From the end user's perspective, it is unacceptable to make him/her spend significant effort entering complex or extensive information, only to discover at the end of the submission process that the desired item is no longer available or that the underlying database was significantly changed by a concurrent session, while editing was being carried out (the so-called "*user thinking time*"). This occurs because another faster competing user gets ahold of the referenced item (loading, editing and submitting it), while the user in question is still inputting his/her information. Moreover, there are critical use cases when the user needs to work with the last committed state of the database, thus making the use of snapshots impracticable.

We highlight three scenarios that require a pessimistic locking approach for RDF, inspired by the realistic examples that motivate the read–write Linked Data Platform architecture [10]:

1. Healthcare
   "For physicians to analyze, diagnose, and propose treatment for patients requires a vast amount of complex, changing and growing knowledge. This knowledge needs to come from a number of sources, including physicians' own subject knowledge, consultation with their network of other healthcare professionals, public health sources, food and drug regulators, and other repositories of medical research and recommendations. To diagnose a patient's condition requires *current data* on the patient's medications and medical history" [10]. Current data means up-to-date committed data.
2. Collaborative Model Driven Engineering (MDE)
   In MDE, models, in their vast majority, are essentially graphs of properties and values. The version control approaches do not work well with models under concurrent access (collaborative development). The version control systems are geared to traditional textual artifacts (e.g. source code), managing them line by line. But for models, line by line management is not appropriate, but structural management, properties and relationships. In general, creating/modifying a model (e.g. business process workflow) demands considerable effort and complex data entry. It would be unacceptable to use the optimistic approach and detect conflicts at end and have to redo the edition.
3. Knowledge in emergency management systems
   In [11], an architecture is described that uses Linked Open Data (LOD) in the design of an Emergency Management System, where different types and sources of information are combined to be used by the command team, designated to manage an emergency, for decision making and for directing field team operations. Three types of knowledge are combined: the previous personal knowledge (team's experience), formal knowledge (general information originated from government agencies) and the *current* contextual knowledge. *Current* contextual knowledge is information generated during the emergency evolution process. Situation assessment done by field agents, including information about victims and damages, orders issued by the command and their effects are examples that help the command decision-making. This knowledge is very dynamic and must to be up-to-date. It changes all the time and thus it has to be constantly updated. In this case, few data items (*hotspots*) are frequently updated and conflicts are so likely that optimistic concurrency control wastes effort in rolling back conflicting transactions.

From this discussion, it is clear that there is a need for pessimistic locking model (where the user acquires the exclusive rights to access a resource before changing it) for RDF. Today's native RDF stores do not support a fully pessimistic protocol. This pessimistic model should explore the RDF data model to improve concurrency, when compared to the traditional read and write locking model. Improvement here means to find, for RDF, the adequate definition of: lockable granules, possible operations (lock modes) on those granules and, finally, the protocol for transactions to acquire and release locks.

Selecting the right granule to lock requires a non-trivial balance between locking overhead and the level of concurrency allowed. Using coarse (i.e., large) granules incurs in low lock management overhead, but may decrease concurrency, since larger portions of the data may be locked. Conversely, finer (i.e., smaller) granules improve concurrency, but require higher overhead for lock management, since more locks are typically requested. It is possible to benefit from both sides of this trade-off by means of the multigranularity locking protocol (MGL), introduced by Gray et al. [12]. MGL allows each transaction to use the granule size most appropriate for its access profile. For example, if a transaction accesses many records of a file, it simply locks the whole file rather than locking only required records. Long transactions can lock coarse granules. Short transactions can lock fine granules. In this fashion, long transactions do not waste time setting up too many locks, and short transactions do not artificially interfere with others by locking portions of the dataset that they do not access.

### 1.2. Contribution

In this paper, we propose a novel pessimistic concurrency control model for RDF, defined by:

- four "hierarchically" interrelated granules;
- six new lock modes inspired by the basic principle of removal and insertion of RDF triples, and
- a protocol to set and release locks for transactions that is an adaptation of the MGL protocol.

It is important to clarify that the goal is to isolate updates in a centralized dataset, under a single control. At this stage, we are not addressing updates on multiples geographically distributed, independent datasets, such as in the *Web of Linked Data* [1]. Due to its open nature, RDF data is often treated as incomplete, following the Open World Assumption (OWA). On the other hand, the SPARQL language interprets RDF data under Closed-World Assumption (CWA). This semantic distance opens up an avenue for