Semantics for Big Data
AAAI Technical Report FS-13-04

Three Laws Learned from Web-scale Reasoning

Jacopo Urbani
Department of Computer Science
Vrije Universiteit Amsterdam, The Netherlands
jacopo@cs.vu.nl

Abstract

Reasoning on a web-scale is one of the most important problems of the emerging semantic web. In this short paper, we look into our previous experience in this field to identify whether there are important lessons that we have learned so far. From this process, we extracted three important principles that we identified as three "laws" that arguably hold on the current Semantic Web.

We believe that these "laws" can be useful to solve the remaining issues that still prevent the applicability of reasoning on a web-scale. To this end, we briefly analyze two of such issues, performance prediction and reasoning approximation, and explain how these three laws can drive the research towards a truly reason(able) Web.

Introduction

The problem of web-scale reasoning is a crucial obstacle in transforming the original vision of a "Semantic Web" into reality. The current amount and the exponential growth of semantic web data call for high-performance reasoning methods that can be applied over very large inputs such as the entire Web.

This problem can be tackled at different levels, which range from a theoretical analysis to the physical implementation. At each of these levels, web-scale reasoning poses several research questions of a fundamental importance. For example, from a theoretical perspective it is important to define algorithms with a computational complexity that remains afforded on a large scale. On a lower and more technical level, engineering-related research questions that are concerned with, for example, efficient storage and retrieval of the data, become more relevant and have a high-priority.

Because of multi-level nature of this problem, extensive research must be conducted at every level to provide a general and elegant solution to each of its issues. In this context, we argue that research should not only be conducted "intra-level", but also "inter-level" because findings discovered at one particular level might be useful also at the other ones. For example, high-level research (e.g. an analysis or a standardization process) might benefit from being aware of what works and what does not when reasoning is implemented on a large scale.

To facilitate an "inter-level" research, in this paper we take a distance from our previous technical contribution, and analyze it from a higher perspective, with the intention of abstracting it into some high-level considerations that might serve to this purpose. To this end, it is important to choose the right level of abstraction: in fact, if we abstract too much then our conclusions might become either trivial or too vague. In contrast, if we do not abstract enough then such conclusions might loose their potential of being useful in a different context.

Given such premises, we decided to reformulate a number of practical considerations that arose from our experience as three *laws* that we believe hold in our context and perhaps even in a more general setting. We title these laws as follows:

- **1st Law:** Treat schema triples differently;
- **2nd Law:** Data skew dominates the data distribution;
- **3rd Law:** Certain problems only appear at a very large scale.

We must warn the reader that there is a degree of speculation in our claims since such laws are only based on initial empirical findings. Therefore, they should be seen more as a number of lessons learned from experience instead of absolute laws that hold in all their generality.

At this point, the reader might find the word "law" misused in such context, and wonder why we do not simply refer to them as "lessons". Such concern might be partly fueled by the fact that in a scientific world we mainly deal with natural laws, which require much more than initial empirical evidence to be accepted as true.

In our case, the reason why we call them *laws* rather than *lessons* is two-fold: Firstly, we believe that the importance that these "lessons" have in the *current* research landscape is such that their application is mandatory, and by using the word *law* we intend to remark the unavoidable constraint that they pose.

Secondly, we believe that such laws are (partly) due to precise choices of the scientific community of representing the knowledge with the current semantic technologies instead than with others. Therefore, we appeal to the juridical meaning of "law", and propose them as implicit laws that follows the more explicit laws such as, for example, the ones defined in standardized RDF and OWL languages.

In the following sections, we discuss each of these laws in more detail. After this, we briefly introduce two important problems of web-scale reasoning and discuss how such laws can help us in finding a solution for them.

1st Law: Treat schema triples differently

In our previous works (Urbani et al. 2012a; 2011; 2012c), and in (Weaver and Hendler 2009), the reasoning process makes a fundamental difference in handling the schema triples from the other ones. With schema triples we generically refer to those triples that define the domain of the knowledge. This distinction is very similar (if not identical) to the T-Box and A-Box distinction in description logic. For example, the triple `(:Student :subclassOf :Person)` is a schema triple because it simply states that all students are also persons.

In both works, the schema triples are always replicated on each node and loaded in main memory to minimize the data transfer during the computation. In contrast, generic triples are not being replicated but rather partitioned according to some criteria. This division is beneficial for the computation: it reduces the data transfer and allows the implementation of clever techniques to speed up the execution. In doing so, we heavily rely on the assumption that the generic triples significantly outnumber the schema triples on the Web. However, although the number of these triples is small, they are fundamentally important because they are frequently used in inference rules as they appear in almost all of the rules antecedents.

Sometimes, determining the set of all the schema triples is a difficult operation. This is not the case with WebPIE (Urbani et al. 2012a), because in there schema triples can only be derived after the execution of MapReduce jobs which store the output in files. Since the data resides on a distributed file system, it is simple to retrieve it in the following reasoning steps. However, for backward-chaining reasoning described in (Urbani et al. 2011; 2012c) the situation is not so simple, because schema triples can be inferred at a later stage. This required us to introduce a procedure at the beginning of the computation (and provide for a theoretical ground to verify its correctness) to ensure that this operation is performed without loosing derivations.

Regardless of how challenging it is to calculate the set of schema triples, in both cases empirical evaluation shows that treating schema triples differently is beneficial to increase the performance. Therefore, in our first law we state that reasoning methods should make a distinction between schema and other triples in designing the reasoning algorithms. Applying this law allows the application of several optimizations to make the computation more efficient and scalable.

2nd Law: Data skew dominates the data distribution

In the previous law we stressed the importance of the schema triples and reported on our strategy of replicating them on all the nodes, under the assumption that the number of these triples is small on realistic data.

In this law, we focus instead on the generic triples. Since a large collection of data contains many of them, a strategy based on replication and loading into main memory is not acceptable because the local space on single machines is likely not large enough. Therefore, we are obliged either to store the data on disk or to partition the input across several machines.

From a high-performance point of view, the last option is preferable since network access is often faster than disk access. In this case, the partitioning criterion has a fundamental importance in determining the overall performance of the system. Unfortunately, there is no universal partitioning scheme that fits all needs, and the chosen criterion determines how and, (even more important) where the computation occurs.

For example, in MapReduce the data is stored in files that reside on the distributed filesystem HDFS where raw data blocks are replicated on a limited number of nodes (three by default). When the Hadoop scheduler receives a request of executing a map task, it schedules it on one of the nodes that has the input locally stored. In this way, Hadoop moves the computation rather than the data and this limits the problem of large data transfer. This is an example of how the partitioning of the data influences the computation because if the data would not be equally split among the nodes, then the scheduler would not be able to provide for an effective way to parallelize this operation.

Our experiments confirms a finding described in (Kotoulas et al. 2010) that current Web data is highly skewed and this makes it more difficult to choose a partitioning criterion to impose a fair balancing of the computation between the machines. Therefore, the impact of data skewness is of great importance on the overall performance and should not be underestimated.

In WebPIE (Urbani et al. 2012a; 2012b), we tackled the problem of data skewness in several ways. First, we sample the data to determine which terms can introduce a load balancing problem. Furthermore, we reduce the effect of data skewness by partitioning the data using more than one resource, to avoid cases where the data is grouped by a single popular resource.

Furthermore, we have observed that data skewness does not always occur. From our empirical analysis, we noticed that skewness appears more frequently on the distribution of the predicates and the objects rather than on the subjects. This means that while there is not a large variance in the number of triples that share the same subject, there is a much larger difference in the number of triples that share the same predicate or object (think for example about all the `(rdf:type)` triples). Such a consideration can drive an implementation where, given a generic input query, data is accessed through the subjects rather than the predicates or objects. Also, the current structure of web data can be used to optimize other tasks than reasoning: For example, in (Neumann and Moerkotte 2011) such principle is used to build *characteristic sets*, which are sets of features that are used to describe large subgraphs of the input.

3rd Law: Certain problems only appear at a very large scale

In the context of web-scale reasoning, an empirical evaluation of the performance is crucial to assess how the methods would perform in a real scenario. In fact, even though a theoretical analysis provides for a good metric to establish the properties of a method, modern computer architectures are very complex and only an experimental evaluation can determine whether one approach indeed "works".

In this context, building a prototype that is able to deal with a massive amount of data is not an easy task. In developing the prototypes used for our experiments we had to address many technical issues and we noticed how particularly important the implementation of a specific algorithm is in order to evaluate its scalability and real performance.

From a research point of view, we are tempted to consider such issues as unimportant because they often do not introduce any theoretical research question. As a result, researchers frequently implement simple prototypes and use them to verify the quality of their contribution. We argue that on a web-scale such approach would be a dangerous mistake, because certain problems appear only at a very large scale (e.g. data skewness) and if the prototype is unable to scale to this extent, then we are unable to verify what is indeed its real performance.

Because of this, we argue that simple proof-of-concepts often do not implement all the necessary elements to be representative. To support our claim, we report some considerations about the development of our prototypes, WebPIE and QueryPIE, highlighting why some purely engineering issues played a fundamental role in achieving high performance. Such issues are not typical of our specific case but are commonly known in the domain of high-performance applications. Therefore, discussing them is certainly relevant and potentially useful for similar problems in our domain.

Complexity of the prototype. First of all, both WebPIE and QueryPIE are written in Java and consist of respectively about 15 and 27 thousand lines of code. The amount of lines can already give an impression of the complexity of the prototypes. We chose Java because all the supporting frameworks and libraries (namely Hadoop and Ibis) were written with this language.

Memory management. The main limitation of using Java for our purpose consisted of the automatic garbage collector, which is an excellent feature of the language but with the limitation that it cannot be controlled by the programmer. Because of this, we developed algorithms that avoided to create new objects but rather reuse existing ones to reduce the impact of the garbage collection phase. Furthermore, since web-scale reasoning is essentially a data-intensive problem, an efficient memory management strategy is almost always necessary in order to exploit all the resources of the hardware. In our case, we realized that some standard Java data structures were not appropriate since they become prohibitively expensive if the input is too large. For example, already storing a few millions of objects in the standard Java Hash-map was enough to fill several gigabytes of space in main memory. To solve this problem, we had to implement custom hash maps which were less memory expensive and rely on data marshaling on byte buffers to avoid the creation of new objects.

Tuning the parameters. Finally, tuning the configuration of the execution environment has also proven to be crucial in the evaluation. For example, the number of Hadoop mappers and reducers per node or the activation of the data compression for the intermediate results can radically change the performance of WebPIE. In QueryPIE, the size of some internal buffers turned out to be a very important parameter, because larger buffers are more difficult to allocate (and might require a call to the garbage collector), while smaller ones get quickly filled.

All these considerations are examples of the purely technical problems that we had to solve. These do not have a real scientific value since they are well-known in high-performance computing. Nevertheless, they are necessary components in the scientific process and this makes the problem of web-scale reasoning very vulnerable to software engineering issues.

We would like to point out that this "law" has substantial consequences for the evaluation. For example, it is impossible to perform a formal verification of a complex implementation, and the correctness can be measured only with an empirical analysis. Also, more complex implementations have higher chances to contain bugs (current estimates indicate that industry-delivered code contains on average 10-50 bugs per 1000 lines of code[1]).

Nevertheless, in the context of web-scale reasoning too many external factors can influence the performance, and each of them must be properly addressed. Because of this, with our third and final law we intend to redefine reasoning not only as a theoretical research question but also as an important engineering problem. In doing that, we aim to elevate the engineering efforts that are necessary to implement web-scale reasoning as an important contribution to solve this problem, since they might radically change the evaluation and therefore mislead the judgment over the quality of the proposed method.

Towards a Reasonable Web

Even though latest developments in this research area have shown that web-scale reasoning is indeed possible, there are still important open issues that prevent us from having a truly reason(able) Web.

Lack of Performance Prediction Techniques. One major problem that we currently have is that we are unable to predict the performance of reasoning beforehand. In fact, there can be cases for which reasoning might become so computationally expensive to be simply unfeasible with current technologies. This can happen for example when there are mistakes in the input or when the query is very complex.

To the best of our knowledge, there is a lack of effective methods to detect such cases, and this prevents us to design reasoning algorithms that are robust enough to properly handle such cases. To this end, we must first define a proper

[1]http://amartester.blogspot.nl/2007/04/bugs-per-lines-of-code.html

cost model for reasoning, where each operation is ranked in terms of difficulty. In this case, we can consider the first law and assume that the reasoning engine has already calculated the closure of the schema. This allows us to give a lower cost to the operation of processing such information.

In a similar way, the third law encourages us to define the cost model considering also the actual implementation of the algorithm. For example, retrieving data from a Java Hash-map has a constant cost. On the contrary, if we use a Java Tree-map the cost is $O(log_2 n)$. Such details should not be ignored, otherwise and accurate estimate is not possible.

In some cases, calculating the cost of reasoning might not be possible since parameters such as n might not be known. For example, suppose that one algorithm might request the storage of a subset of the input like all the "*rdf:type*" triples in a temporary Tree-map. Here, we are unable to estimate the cost unless we physically count all the "type" triples.

To solve this issue, we need to apply statistical methods to estimate n. In this case, the second law becomes very important since it warns that current data has high skewness, and therefore the statistical method should take this into account to provide for an accurate estimate.

Reasoning Approximation. Once we have estimated the computation, reasoning might still turn out to be unfeasible. This can frequently happen on the Web, since corrupted data can generate exponential number of derivations (Hogan, Harth, and Polleres 2009). In these cases, several types of approximation can be applied to reduce the computation. For example, we can avoid computation that is most likely redundant, or which will most likely fail. Furthermore, another type of approximation would calculate only "interesting" derivations and discard the "trivial" ones (notice that in this case we are required to first define what is "interesting" and what is "trivial").

Also in this case the three laws can help us in developing more effective techniques. For example, the first law can be exploited to verify whether some computation leads to some conclusion exploiting the availability of the schema. In our previous work (Urbani et al. 2011), we applied this principle to prune the reasoning during backward-chaining, but such process can be "relaxed" to consider cases where the computation will *most likely* fail.

The second law allows another type of approximation, which considers the high skewness of the data to avoid trivial derivation. In this case, we could define "trivial" as the conclusions that could be derived from a process that was applied on a large part of the input (e.g. the inference that concludes that every subject is a *rdfs:Resource*). This law facilitates the detection of the triples than can trigger such "trivial" reasoning by looking at the few frequent terms. This makes such approximation faster and more effective.

Finally, the third law can drive the approximation to consider not only the reasoning algorithm, but also its implementation. For example, suppose that reasoning is implemented in a distributed setting, where the data can be retrieved either from main memory (fast), network (slower), or disk (slowest). In this case, the approximation method could apply reasoning only on the data that is stored in main memory, and avoid more expensive operations like reading from disks or the network.

Conclusions

In this short paper we propose three laws which we learned in studying the problem of applying reasoning on a web-scale. These laws concern the difference between schema and generic information, the skewness of the data, and the importance of the actual implementation.

Even though these laws might not be universally true, we believe that they hold on the current (semantic) web, and therefore they can contribute in solving the major issues that prevent us from having a truly "reason(-able) web". To this purpose, we briefly outlined how they could have been exploited for two of such problems, which are the inability to estimate the complexity of reasoning and the consequent approximation if the computational resources are insufficient.

Extensive research is still necessary to solve the ultimate problem of web-scale reasoning, and the abstraction of technical contributions in terms of more abstract principles can be a useful and effective practice to reach this goal. Therefore, we hope that more of such laws will be discovered. It is time that the "constitution" of web-scale reasoning is finally being written.

References

Hogan, A.; Harth, A.; and Polleres, A. 2009. Scalable Authoritative OWL Reasoning for the Web. *International Journal on Semantic Web and Information Systems* 5(2).

Kotoulas, S.; Oren, E.; van Harmelen, F.; and van Harmelen, F. 2010. Mind the Data Skew: Distributed Inferencing by Speeddating in Elastic Regions. In *Proceedings of the International World-Wide Web Conference*, 531–540.

Neumann, T., and Moerkotte, G. 2011. Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins. In *Proceedings of the International Conference on Data Engineering*, 984–994.

Urbani, J.; van Harmelen, F.; Schlobach, S.; and Bal, H. E. 2011. QueryPIE: Backward Reasoning for OWL Horst over Very Large Knowledge Bases. In *Proceedings of the International Semantic Web Conference (ISWC)*, 730–745.

Urbani, J.; Kotoulas, S.; Maassen, J.; Harmelen, F. V.; and Bal, H. 2012a. WebPIE: A Web-scale Parallel Inference Engine using MapReduce. *Journal of Web Semantics* 10(0):59 – 75.

Urbani, J.; Maassen, J.; Drost, N.; Seinstra, F.; and Bal, H. 2012b. Scalable RDF data compression with MapReduce. *Concurrency and Computation: Practice and Experience*.

Urbani, J.; Piro, R.; van Harmelen, F.; and Bal, H. 2012c. Hybrid Reasoning on OWL RL. *Under submission to the Semantic Web Journal. Available online at http://bit.ly/19JxrJl*.

Weaver, J., and Hendler, J. 2009. Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples. In *Proceedings of the International Semantic Web Conference (ISWC)*.