

**MaRVIN:**  
**a distributed platform for massive RDF inference**  
**(“a brain the size of a planet”)**

George Anadiotis, Spyros Kotoulas, Eyal Oren, Ronny Siebes, Frank van Harmelen,  
Niels Drost, Roelof Kemp, Jason Maassen, Frank J. Seinstra, and Henri E. Bal  
corresponding author: [eyal@cs.vu.nl](mailto:eyal@cs.vu.nl)

Department of Computer Science, Vrije Universiteit Amsterdam, the Netherlands

**Abstract** We have built MARVIN (**M**assive **R**DF **V**ersatile **I**nference **N**etwork), a parallel and distributed platform for performing RDF(S) inference on a network of loosely coupled machines using a peer-to-peer model. MARVIN<sup>1</sup> can be scaled to arbitrary size by adding computing nodes to the network, is robust against failure of individual components, and displays anytime behaviour (producing results incrementally over time). MARVIN is not built as a single reasoner, but rather as a platform for experimenting with different reasoning strategies. MARVIN contains instrumentation to log and visualise its run-time behaviour, and its modular design allows to vary different aspects of the reasoning strategy.

## 1 Problem Description

The volume of available Semantic Web data is now outgrowing the capacity of current reasoning engines. Through the “linking open data” initiative, and through infrastructure such as Swoogle, Watson, and Sindice, datasets in the order of  $10^6$ – $10^9$  triples are now readily available. Such volumes are indeed required for realistic tasks: the RDF encoding of the UniProt database of protein sequences contains 1.5 billion unique RDF triples<sup>2</sup>.

The recent “Berlin benchmark”<sup>3</sup>, shows that state of the art RDF inference systems can barely deal with such volumes. Just *loading* a billion triples would take many hours or even days. Computing the deductive closure of such triple sets is of course much more costly than just loading them.

**The need for inference at arbitrary scale.** To deal with such huge volumes of Semantic Web data, we should instead aim at building RDF engines that are *arbitrarily scalable*. Such arbitrary scalability can be achieved in two major ways, both of which we have exploited in MARVIN:

---

<sup>1</sup> Named after Marvin, the paranoid android from the Hitchhiker’s Guide. Marvin has “a brain the size of a planet”, which he can seldomly use: the true horror of Marvin’s existence is that no task would occupy even the tiniest fraction of his vast intellect.

<sup>2</sup> <http://seaborne.blogspot.com/2008/06/tdb-loading-uniprot.html>

<sup>3</sup> <http://www4.wiwiw.fu-berlin.de/bizer/BerlinSPARQLBenchmark/>

- using *parallel hardware* which runs *distributed algorithms* that exploit such hardware regardless of the scale, varying from tens of processors to many hundreds (as in our experiments) or even many thousands.
- designing *anytime algorithms* that produce *sound but incomplete results*, with the degree of completeness increasing over time. Such algorithms can trace the speed with which the inference process converges to completeness against the size of the dataset, while still guaranteeing eventual completeness.

Both of these roads to scalability are indeed practical: with the cost of commodity hardware in the order of some hundreds of euro’s per unit, a network of hundreds of processors is achievable for many institutions and companies; and for many realistic Semantic Web applications, full completeness of the inference process is not required. After all, on the Web, we have learned to live with incompleteness of data, so why would we insist on completeness of an inference process whose initial axioms are incomplete to start with?

MARVIN’s peers compute the materialised closure *in advance* (data-driven). RDF stores such as OWLIM and Sesame have shown that such materialisation is feasible for realistic RDF datasets without causing a worst-case exponential blow-up. But different from these other RDF stores, MARVIN computes the closure *gradually*, enabling users to query this closure at any time without having to wait for the full closure to be computed. We are not aware of any existing work that exploits large-scale hardware as we do (see section 4) to manage billions of RDF triples.

**The need for an experimentation platform.** Very little is known about the optimal choices for many of the parameters involved in the design decisions for such an anytime distributed reasoning engine: the number of processors, the initial distribution of data over the nodes, the collaboration strategy of the nodes during the computation, etc. Studying these requires a *configurable* platform which allows experiments with many aspects of its operation. The modular design of MARVIN allows varying different aspects of the reasoning strategy. Furthermore, MARVIN is equipped with tools for logging key performance indicators during runtime and with software to visualise these indicators, either live (during inferencing) or as a replay, after inferencing has been completed.

To facilitate experimenters and deployment, such a platform should abstract from different processor architectures, operating systems, or network protocols on the computing nodes; we achieve this by building MARVIN on top of Ibis, a high-performance Java-based grid programming environment.

**Outline of the paper.** In this paper, we first present the requirements we followed when designing MARVIN. We then describe MARVIN’s architecture, describing the internals of each computing node and the collaboration policy between the nodes. We report on the current status of the implementation and some experiments, and finish with open challenges in the design of a distributed inferencing platform like MARVIN.

## 2 Requirements

Here we outline the requirements for a system to tackle the challenges outlined in the previous section

**Stability** With stability we mean the sensitivity of the system to weakly performing elements of it. The peer-to-peer paradigm makes weak assumptions about the performance characteristics of the individual nodes: as long as most peers perform the task well, the overall performance of the system as a whole is guaranteed. This leads to a system design where components are as autonomous as possible.

**Diversity** Given the high diversity in Semantic Web data (varying from simple RDF to expressive OWL or even richer languages like SWRL), the system should be able to use different reasoners, potentially even side by side, each optimised for a different inferential power. Cheap inferences should be drawn quickly, without being delayed by the longer runtimes of more expensive reasoners. Again the peer-to-peer paradigm seems very suitable due to the autonomous character of the individual nodes as long they fulfill the basic communication requirements.

**Scalability** As already argued in the previous section, the growth in the amount of RDF data makes single machine computations increasingly harder, or even impossible. Distributing computation and storage over several machines is the logical step.

**Modularity** Besides developing a system that is capable of dealing with vast numbers of RDF triples, another perhaps even more ambitious goal is to build it in the form of a re-usable scientific platform that can be used for experimentation with different configurations and strategies. Therefore, the system should consist of a set of loosely coupled modules that can be changed independently of each other, using a set of well-defined interfaces.

**Usability** Besides performance, usability is important in determining the success of a system. The system should be relatively easy to install, should run on many platforms and on hardware with different characteristics (e.g. CPU, bandwidth, memory and storage), and it should be easy to monitor and evaluate. The use of standard protocols and a widely deployable programming language is essential, together with support for monitoring and analysis through standard Web browsers.

## 3 Architecture

The main goal of MARVIN is to provide a platform for modular, massively scalable, RDF reasoning with support for logging and analysing the system's behaviour. In this section, we describe the architecture of our system. We start by describing the main computational loop of MARVIN, discuss each of MARVIN's modules, and we present the Ibis infrastructure which facilitates distributed computing by abstracting from physical location and communication protocols.

### 3.1 Main Loop

MARVIN operates using the following “main loop” (steps 3–5 are repeated infinitely, and continuously report operational statistics):

1. The input data is divided into smaller chunks, which are stored on a shared location.
2. A large number of reasoners is bootstrapped on the nodes of the grid<sup>4</sup>.
3. Each reasoner reads some input chunks and computes the deductive closure of this input data at its own speed.
4. On completion, each node selects some parts of the computed closure, and sends it to some other node(s) for further deduction. Asynchronous queues are used to avoid blocking communication.
5. Each node also selects some (other) parts of the computed closure, and sends it to a central storage bin, where the data can be queried by end-users. This storage bin is external to the compute cluster.

Steps 3-5 are repeated ad infinitum. Meanwhile, MARVIN logs performance statistics which can be visualised either in real-time or post mortem.

### 3.2 Modules

The above loop is implemented through a number of modules with well-defined interfaces, allowing one to exchange one implementation of a module for another. Figure 1 gives a high-level overview of MARVIN’s architecture.

Some modules are local to each node, and run on that node:

**Input Pool** This module takes care of buffering RDF data that the node receives from other nodes and ‘feeds’ them to the Reasoner. Given that the Reasoner ‘pulls’ the RDF data from this Module and the network ‘pushes’ RDF triples to it, it could be that the amount of buffered data gets too much to store in memory. Therefore the module also monitors available memory and runs a separate Java Thread to write triples to disk and fetches them when there is enough space again, similar to paging in operating systems.

**Reasoning** This module performs inferencing over the triples; as input the reasoner takes triples from the InputPool, as output it generates additional triples which are sent to the Routing Module. For all chunks of triples, inferred or not, some *score* is calculated; this score is higher if a chunk lead to more derived triples, meaning that these triples had much implicit information to be made explicit. This score is important for the decision process of the Routing Module as we will discuss next.

**Routing** This module decides what to do with the chunks of triples derived by the Reasoning Module. There are three choices, informed by the *score* of the chunk: send to OutputPool, to be processed further by other nodes, send to

---

<sup>4</sup> in our case, these are OWLIM reasoners on the DAS-3 multi-cluster, see section 4.

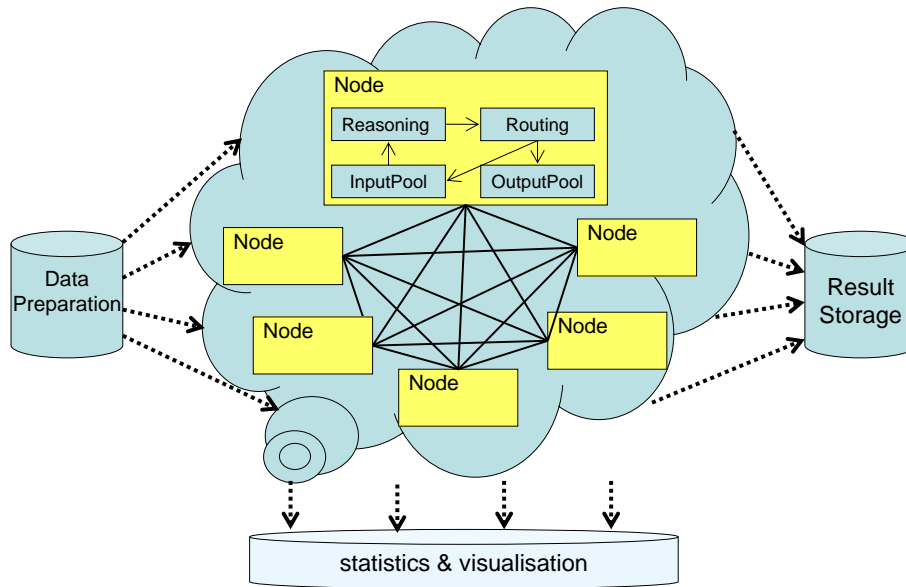


Figure 1. MARVIN's architecture. Not all modules are shown.

InputPool, to be re-processed by the same node and send to ResultPool, to be removed from the process and stored to the final output of the system.

We prevent duplicate triples being sent to the Result Storage Module, using a “one-door exit policy”: each triple is only allowed to leave the network of nodes through the single node that is “responsible” for it (based on a hash of the triple and the rank of the node). This approach spreads the load of duplicate detection among all the nodes in the network, instead of performing an expensive centralised duplicate detection on the Result Storage.

**Output Pool** This module buffers the triples that the reasoner has deduced and for which the Routing Module has decided that they are intended to serve as input to other nodes. Similar to the InputPool Module, it monitors available memory and pages triples on disk when necessary.

Other modules are system-wide, and are shared across all nodes:

**Data Preparation** This module splits the initial RDF dataset into *chunks*. These chunks are stored on a file server (currently on a single machine) and will be loaded by the nodes in the network when needed. This module can be executed independently from the rest of the modules, if necessary “off-line”, well before the main computational loop is started.

**Communication** This module arranges the communication between the nodes. This module is an abstraction over the Ibis platform, which is used for all communication between the nodes.

**Result Storage** This module stores all triples that the individual nodes derived via their Reasoner, as submitted by their Routing Modules. Storing the output, and allowing applications or users to query over this data, is not trivial at the volumes that are required.

**Statistics** All modules periodically send relevant information (such as: processor and memory load of each node, the number of triples in the various buffers, the number of received and inferred triples) to the statistics module. These statistics allow us to monitor, analyse, and understand the distributed reasoning process.

**Visualisation** The purpose of this module is to visualise the statistics via a simple web browser, which allows us to have a multi-user interface anywhere in the world, independent of the physical location of the compute cluster(s). This information is used for three different tasks: debugging the modules, performance analysis (real-time and after execution), and for demonstration.

### 3.3 The Ibis Infrastructure for Networked Computation

MARVIN is deployed on a grid in order to use as many computational resources as possible. Grid programming and deployment is difficult, as grids differ substantially from more conventional computer systems. In general, grid programmers have to use complex programming interfaces that change frequently. Also, they have to deal with hardware heterogeneity, connectivity problems (e.g. due to firewalls), and software and hardware failures. Furthermore, managing a running application is complicated, because the execution environment can change dynamically, as grid resources come and go.

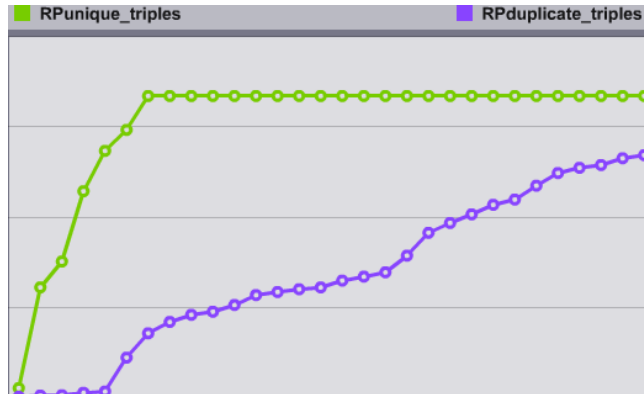
The Ibis grid software<sup>5</sup> deals with all of these problems in a fully transparent manner. It is an open-source software platform that drastically simplifies the programming and deployment process of high-performance and distributed grid applications. Ibis is unique in that it offers an integrated solution that *transparently* deals with most of the inherent grid complexities. In MARVIN all network management and node communication is based on Ibis.

## 4 Current status

We are currently running experiments using MARVIN to complete the full deductive closure of the Challenge dataset. We perform these experiments on the Distributed ASCI Supercomputer 3 (DAS-3), a five-cluster grid system, consisting in total of 271 machines with 791 cores at 2.4Ghz, with 4Gb of RAM per machine. The experiments were not complete at the submission deadline, but are expected to be available well before the Challenge event in Karlsruhe.

We have also run MARVIN in a simulated network of eight P2P nodes, on a subset of the Challenge dataset, using a single machine with 8 2.3GHz cores and 32Gb RAM. This has shown that MARVIN indeed provides the required functionality.

<sup>5</sup> see <http://ibis.cs.vu.nl/> and <http://doi.acm.org/10.1145/583810.583813>



**Figure 2.** anytime performance profile of MARVIN

Figure 2 is a screen-shot of our Visualisation Module, and shows a typical run of MARVIN. The  $x$ -axis shows runtime, the  $y$ -axis shows the number of unique derived triples collected in the Result Storage (green), and duplicate derived triples that have been detected by the Routing modules of the nodes (and are hence not forwarded to the Result Storage, since they are duplicates of earlier derivations, purple line). The graph shows that MARVIN behaves as expected: after a short initial period, the number of derived triples arriving at the Result Storage begins to increase, until at some point apparently the full closure has been derived, and no new triples arrive at the Result Storage (the green line stays flat). Initially, all derived triples are unique, and no duplicates are being detected (the purple line is 0). At some point during the derivation of the closure, the nodes start to generate triples that have been generated before, hence the purple line increases. If we keep MARVIN running indefinitely, the purple line will continue to rise, since ever more duplicates are being derived, detected, and stopped from going to the Result Storage.

Future experiments must show which configurations of MARVIN will lead to the best shapes of such performance profiles under different circumstances, and for datasets of different sizes and with different properties.

## 5 Summary and Challenges

We have built MARVIN, a platform for massive distributed RDF inference. MARVIN uses a peer-to-peer architecture to achieve scalability to arbitrary size, either by adding computational resources, or by trading scale for completeness. MARVIN guarantees eventual completeness of the inference process, but produces its results gradually, so that users can already query partial results during the computation (anytime behaviour). Through its modular design and its built-in instrumentation, MARVIN allows experimentation with many configurations.

MARVIN provides an exciting basis for answering many further questions concerning many aspects of distributed RDF reasoning. Interesting technical questions concern choices in the initial data-distribution across nodes (can we improve over random, e.g. by clustering triples based on name-space usage?), the utility function used to decide if triples should be routed to other nodes, the routing behaviour itself (how to decide which triples should be sent to which peers?), and in general how to optimise the anytime performance profile of the platform, generating a steep convex completeness curve over time.

Besides these technical questions, MARVIN also raises methodological questions, such as how its output quality should be measured (simple notions of recall and precision lose their meaning on datasets which are so large that even simple counting becomes expensive, and where the “correct” answer is not known), and how to improve our current instrumentation. Which performance characteristics should be logged, and what is the best way of storing and analysing them?

Finally, we aim to deploy MARVIN in a setting that we call *thinking@home*, for its resemblance with the well-known *SETI@home* project. In this setting, MARVIN would perform distributed inference using spare CPU cycles of PCs around the globe. This would change the network characteristics in a fundamental way: a low-bandwidth, high-latency, wide-area network of loosely-coupled machines with high churn-rates. Using the Ibis platform, this setup should not present great technical difficulties, but it would require further experiments to determine the optimal algorithm parameters. The *thinking@home* setup would bring the ultimate vision of a “self-computing Semantic Web” closer to reality. We consider MARVIN to be a step in that direction.

## 6 Online material and acknowledgements

This work is part of the LarKC project, funded under grant nr. FP7-215535 of the European 7th Framework Programme. MARVIN is released open source: the code, experimental results, and a replay of some experiments is available at <http://www.larkc.eu/marvin>. A big thanks is due to the IT Group at VUA, notably Henk Schut and Gert Huisman, who provided support for our work well beyond the call of duty.