Distributed Prolog Reasoning in the Cloud for Machine-2-Machine Interaction Inference

Radovan Zvoncek[1,2]

[1] Instituto Superior Tecnico, Lisbon, Portugal
[2] Royal Institute of Technology, Stockholm, Sweden

Abstract. Application layer interconnection of vast amounts of devices is not a trivial task due to numerous protocols and semantics devices use. Designing translating protocols and proxies mitigated the problem but did not provide a complete solution. Ontology-based middlewares have the potential to complement the gap. In this paper we propose an implementation of an ontology-based middleware aiming for massive-scale deployment. Our design extends a conventional Prolog engine by sharding its database using a DHT and consequently by migrating the goal evaluation context among different engine instances. Experiments performed with our implementation show that overhead introduced by managing a distributed system is compensated once the system load is sufficiently high.

1 Introduction

The vision of enabling Internet connectivity to every device that can potentially benefit from being connected has been almost fulfilled. The number of devices connected to the Internet has reached 10 billion and is expected to grow [1]. However, providing connectivity of devices without applications that can utilise it is not sufficient to unlock the full potential of connected infrastructure.

While many of the obstacles in connecting the devices have been successfully solved, facilitating the actual communication keeps facing several obstacles. Devices are manufactured by different vendors and therefore use vendor-specific, non-standardized protocols and semantics. Consequently, devices using the same protocol families become integrated in vertical domains, thus effectively disabling any generic interconnection initiative. As a result, the devices literally do not understand each other.

Although there has been significant work done regarding designing new protocols and implementing translating proxies aiming to bridge the gaps between different vertical domains, there is at least one more complementary approach. Giving the applications an ability to reason, learn and interpret observed behaviours in their environment based on contextual knowledge would facilitate horizontal communication without the need of detailed manual integration of disjoint vertical domains.

Previous work done by Ericsson Research resulted in creating the concept of a platform attempting to meet the aforementioned objective. Figure 1 illustrates the envisioned 3-layer architecture. In its core, the platform contains the

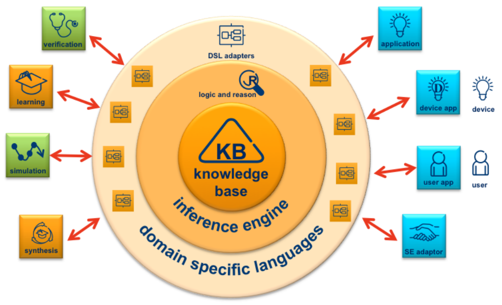


Fig. 1. Illustration of a platform offering reasoning functionality.

knowledge base gathering all data available in the system (e.g. sensor readings), as well as additional information about how to utilise the data (e.g. rules and theories). The second layer is the Prolog core facilitating the actual reasoning about the available knowledge. The outer layer consists of adapters to domain specific languages for other components present in the system (e.g. applications, machine learning).

This work presents an implementation of the described platform. We introduce a distributed system providing fault-tolerant persistent storage of the knowledge base expressed as Prolog terms while maintaining full Prolog functionality, i.e. term manipulation and goal solving. The system was designed to meet the requirements of having the ability to store very large knowledge bases and to handle high number of incoming requests.

The remainder of this paper is organised as follows. Section 2 contains brief overview of the related work. Sections 3 and 4 describe the design and implementation of the presented system. Section 5 provides evaluation of the implemented system and section 6 summarises this work and briefly introduces future work.

2 Related Work

Our work can be placed in the category of semantic and ontology-based middlewares. Examples of semantic middlewares can be found in [2] and [3]. Ontology-based middlewares [4] extend the functionality of semantic middlewares by introducing the ability to reason about the semantics of the available data. In [5], the authors use Web Ontology Language (OWL) to represent context of the environment devices operate in and reason about its changes so that the devices can adapt accordingly. In [6], the authors propose similar system, but built on combination of OWL and first order probabilistic logic. Both of the investigated systems are reported to be facing scalability issues.

The system proposed in this paper is designed using principles similar to [7]. However, we do not aim for multiple ontologies and RDF model, but consider monolithic ontology and Prolog computational model.

The novel contribution of this work lies in designing a system that can function as a middleware for massive amounts of clients while maintaining the capacity to handle large volumes of data. In addition, we support the proposed design with sound and thorough evaluation of the implementation of the proposed system.

3 System Architecture

In this section we explain the proposed architecture. We will first provide general overview of the system and then follow steps and decisions made while designing the system.

The core design idea lies in selecting an attribute of any Prolog term that can be used as an input for the distributed hash table (DHT) hashing function in order to achieve uniform and therefore efficient distribution of terms across the nodes in the system and constant algorithmic complexity of term lookup. The motivation for this decision is to limit the data movement which can potentially produce significant overhead. Instead, comparably smaller context capturing the state of Prolog goal solving will be migrating among the nodes, thus effectively bringing computation to the data.

3.1 Sharding Prolog Database

We propose to use the concatenation of name and arity of any Prolog term as an input for the hashing function. For example, for the given Prolog term *termA(atom2,atom3) :- termA(atom1,atom2).* the input for the hashing function would be *"termA/2"*.

This would effectively lead to grouping terms of same name and arity at the same node, what is highly desirable feature due to the Prolog computational model always accessing all terms of given name and arity at the same time. Moreover, maintaining the grouping of terms facilitates manipulation with larger chunks of terms. Another expected benefit of building the storage solution atop a DHT is the seamless horizontal scalability this choice would allow because the system would not feature any centralised components that could possibly induce performance bottlenecks.

3.2 Migrating the Computation

Second fundamental design idea is the notion of migrating computation. Since the expected amount of data present in the system is envisioned to be very large compared to small amount of data associated with the context of Prolog computation, moving the computation can potentially be more efficient.

Therefore we introduced a concept of migrating computation. Figure 2 illustrates the process of solving a Prolog goal. The process goes as follows:

1. A device issues Prolog query asking if goal $termB(b, a)$ is true.

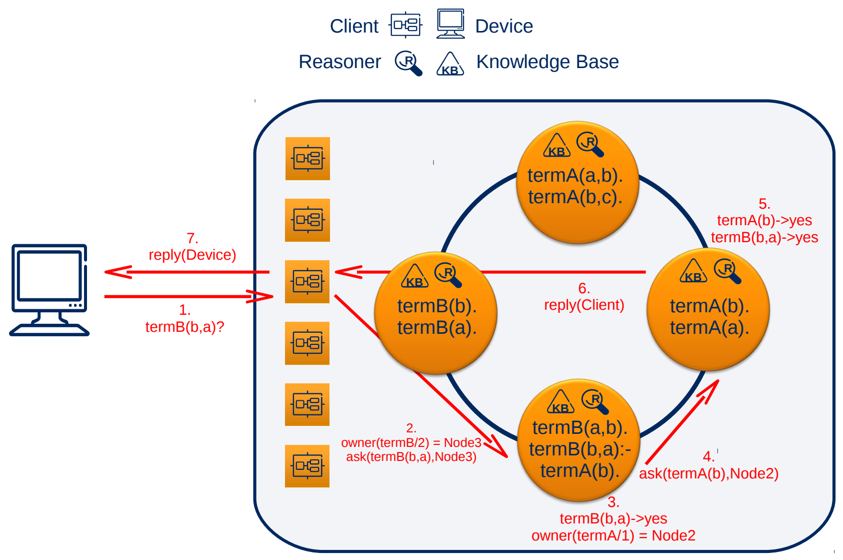


Fig. 2. Migrating computation of solution for a given Prolog goal.

2. A client receives the requests and examines the query. It reads the name and arity of the term being queried ($termB/2$) and determines a back-end node responsible for the term (*Node3*). The client then forwards the query to the corresponding node.
3. The *reasoner* at *Node3* receives the query and instantiates a new Prolog engine to handle the query. If the library containing terms $termB/2$ is not loaded, the reasoner at *Node3* loads the library with the assistance of the storage daemon and then initiates the goal solving and finds out $termB(b, a)$ is present in the knowledge base. Then it continues with the next goal, $termA(b)$. However, *Node3* is not the owner of terms $termA/1$.
4. The computation has to be sent to *Node2* which is responsible for terms $termA/1$.
5. *Node2* receives the computation, instantiates a new Prolog engine and loads it with the context related to previous computation. It successfully solves goal $termA(b)$. and discovers this concludes the whole solution process based on the information present in the computation context received from *Node3*.
6. *Node2* can therefore provide a final answer to the client.
7. Finally, the client can forward the received answer to the device.

The concept of migrating computation also facilitates effective workload sharing. Once a node does its part of computation and sends the computation further, it does not wait for any further replies. Therefore, the free resources can be used to serve other ongoing computations passing by.

3.3 Facing the CAP Theorem

The concept of migrating computation introduced a potential issue into the system. The terms involved in a solution of a goal can be modified before the solution is finalised. Such situation is a manifestation of the CAP [8] theorem as achieving consistent results of goal solving is not possible while maintaining the whole system available and partition-tolerant. Therefore we assessed the priorities of each of the attributes and obtained following outcomes.

- Partition tolerance has the highest priority because knowledge base must be monolithic.
- Consistency has the second highest priority because the system must guarantee the answer is correct.
- Availability has the lowest priority because clients can wait for the answer for a reasonable amount of time.

To implement the explained priority of attributes, we introduced phase-based operation into the system. All nodes in the system will synchronously alternate between read and write phases and therefore ensuring consistent results of the goal solving.

4 Implementation

In this section we describe how we implemented some of the described design solutions.

We have chosen Cassandra [9] as an implementation of the underlying DHT due to out-of-the-shelf persistency, fault-tolerance and uniform load distribution (provided we used Cassandras *RandomPartitioner*). In addition, open-source distribution of Cassandra gave us easy access to its hashing function from the used Prolog engine.

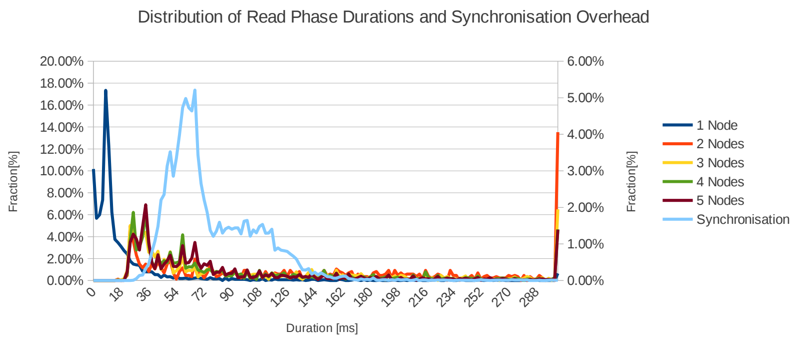
We used the tuProlog [10] as the basis for our reasoning engine. We have extended tuProlog with the functionality of checking the solved terms being locally resolvable and conditionally migrating the solution context to the node responsible for the next goal.

The introduction of phase-based operation required node synchronisation. For this purpose we implemented a leader election algorithm based on the Paxos algorithm [11] using the Zookeeper [12] coordination service.

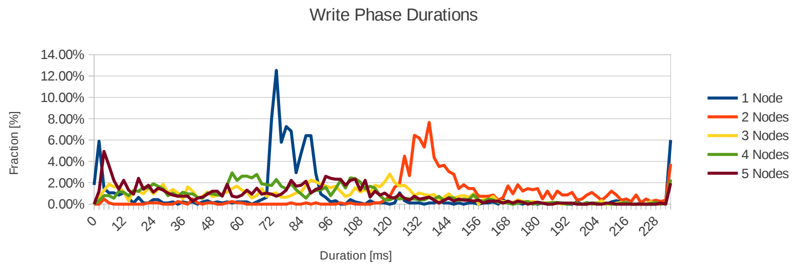
The reasoners themselves are implemented as multi-threaded processes with separate threads handling each connection, thus allowing re-using one established connections to transmit multiple messages.

5 Evaluation

The evaluation was conducted on a private cluster of 5 machines, each ofering 4-core Intel Core2 Q9400 CPU operating at 2.66 GHz and 8GB of RAM. We used one of these nodes for the one-node instalation, as well as 2 to 5 machines for



(a) Distribution of read phase durations.



(b) Distribution of write phase durations.

Fig. 3. Comparison of centralised and distributed instalations.

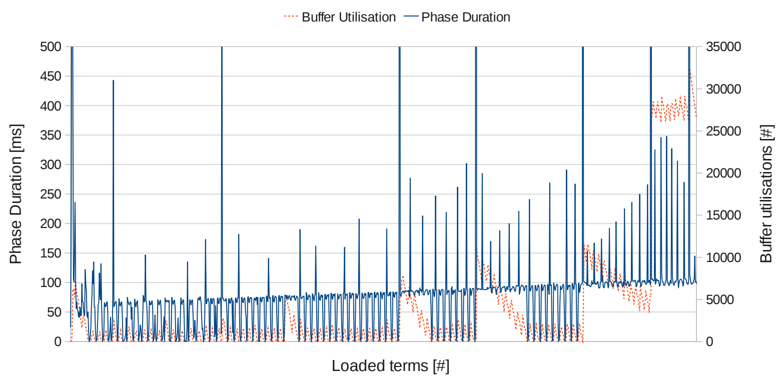
multi-node instalations. We generated two datasets containing 400k and 1,5M Prolog terms respectively and then observed durations of read and write phases manipulating with the terms at different intensity for different number of nodes participating in the system.

We have compared the multi-node installations of our system with a one-node installation obtained by taking the distributed system and striping it from all the overhead originating form the distributed mode of operation.

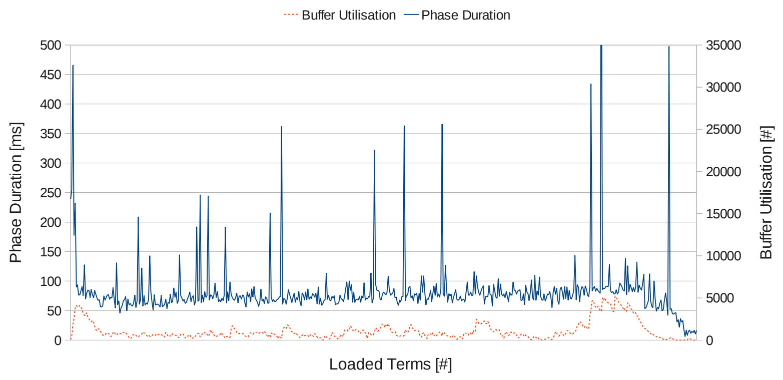
Figure 3(a) shows the distribution of read phase durations for different node setups together with the observer synchronisation overhead. It can be seen that the distributed installations are approximately 20 to 80 ms slower than the 1-node installation. However, the slowdown is almost exclusively caused by the synchronization overhead. The remaining slowdown is caused by the computation migration. It is possible to conclude the slowdown is present but tolerable.

Figure 3(b) shows the distribution of write phase durations. It can be seen that from 3-node onwards the phase durations are comparable with 1-node installation, but more widely distributed in the interval of 0 to 120 ms. This can be explained by multi-node setups splitting the work needed to assert incoming clauses.

Figure 4(a) shows how 1-node system behaves when facing increased workload. It is visible how the system begins to experience longer phases as it approaches the limit of 400k loaded terms and eventually crashes, even after lowering the number of loaded terms thanks to unloading least recently used libraries. In contrast, Figure 4(b) shows how 5-node system manages to handle the same amount of terms, but with the increased inbound rate of 9k operations per second.



(a) 1-node system under heavy load.



(b) 5-node system under heavy load.

Fig. 4. Experiments with hight work load.

6 Conclusion and Future Work

This paper presented a design and implementation of an ontology-based middleware aiming to support application-level interoperability of large number of heterogeneous devices. We have described the design process and motivation behind every design decision. The evaluation has shown that even though the proposed system is burdened by the synchronisation overhead, this overhead is compensated once the workload becomes sufficiently high.

The possible directions to continue development of this system can focus on mitigating the synchronisation overhead, or implementing additional features such as fault-tolerant query execution, query result caching or automated scalability.

References

1. Ericsson: More than 50 billion connected devices. Online (February 2011) Whitepaper.
2. Song, Z., Cárdenas, A.A., Masuoka, R.: Semantic middleware for the internet of things. In: Internet of Things (IOT), 2010, IEEE (2010) 1–8
3. Gómez-Goiri, A., López-De-Ipiña, D.: A triple space-based semantic distributed middleware for internet of things. In: Proceedings of the 10th international conference on Current trends in web engineering. ICWE'10, Berlin, Heidelberg, Springer-Verlag (2010) 447–458
4. Kiryakov, A., Simov, K.I., Ognyanov, D.: Ontology middleware: Analysis and design. On-To-Knowledge (IST-1999-10132) Deliverable **38** (2002)
5. Gu, T., Wang, X.H., Pung, H.K., Zhang, D.Q.: An ontology-based context model in intelligent environments. In: Proceedings of communication networks and distributed systems modeling and simulation conference. Volume 2004. (2004) 270–275
6. Qin, W., Shi, Y., Suo, Y.: Ontology-based context-aware middleware for smart spaces. Tsinghua Science & Technology **12**(6) (2007) 707–713
7. Anadiotis, G., Kotoulas, S., Siebes, R.: An architecture for peer-to-peer reasoning. In: Proc. of the First Int. Workshop" New forms of reasoning for the Semantic Web: scalable, tolerant and dynamic", co-located with ISWC 2007 and ASWC 2007. Volume 291. (2007)
8. Gilbert, S., Lynch, N.: Perspectives on the cap theorem. Computer **45**(2) (2012) 30–36
9. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review **44**(2) (2010) 35–40
10. Piancastelli, G., Benini, A., Omicini, A., Ricci, A.: The architecture and design of a malleable object-oriented Prolog engine. In Wainwright, R.L., Haddad, H.M., Menezes, R., Viroli, M., eds.: 23rd ACM Symposium on Applied Computing (SAC 2008). Volume 1., Fortaleza, Ceará, Brazil, ACM (16–20 March 2008) 191–197 Special Track on Programming Languages.
11. Lamport, L.: Paxos made simple. ACM SIGACT News **32**(4) (2001) 18–25
12. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: Zookeeper: wait-free coordination for internet-scale systems. In: Proceedings of the 2010 USENIX conference on USENIX annual technical conference. Volume 8. (2010) 11–11