TANKER: Distributed Architecture for Named Entity Recognition and Disambiguation

Sandro A. Coelho sandro.coelho@ice.ufjf.br
InfAI, University of Leipzig, Leipzig, Germany

Diego Moussalem moussalem@informatik.uni-leipzig.de
AKSW, University of Leipzig, Leipzig, Germany

Gustavo C. Publio publio@informatik.uni-leipzig.de
AKSW, University of Leipzig, Leipzig, Germany

Diego Esteves esteves@cs.uni-bonn.de
SDA Research Group, University of Bonn, Bonn, Germany

Keywords

Named Entity Recognition | Entity Linking | NER Architec-tures | Knowledge based systems | Joining processes

ABSTRACT

Named Entity Recognition and Disambiguation (NERD) systems have recently been widely researched to deal with the significant growth of the Web. NERD systems are crucial for several Natural Language Processing (NLP) tasks such as summarization, understanding, and machine translation. However, there is no standard interface specification, i.e. these systems may vary significantly either for exporting their outputs or for processing the inputs. Thus, when a given company desires to implement more than one NERD system, the process is quite exhaustive and prone to failure. In addition, industrial solutions demand critical requirements, e.g., large-scale processing, completeness, versatility, and licenses. Commonly, these requirements impose a limitation, making good NERD models to be ignored by companies. This paper presents TANKER, a distributed architecture which aims to overcome scalability, reliability and failure tolerance limitations related to industrial needs by combining NERD systems. To this end, TANKER relies on a micro-services oriented architecture, which enables agile development and delivery of complex enterprise applications. In addition, TANKER provides a standardized API which makes possible to combine several NERD systems at once.

1. INTRODUCTION

The Internet has been growing at an explosive rate for several years, which make harder to handle the big amount of information diffused in diverse formats, such as text, audio and, video. Currently, the Web produces more than 2.5 exabytes of data per day [1] . Therefore, the challenge of indexing, formatting, and making the information available to the users has arisen every day, which makes a critical scenario. In order to deal with such variety of contents, refined NLP techniques are required.

One of the most important NLP techniques is Named Entity Recognition and Disambiguation (NERD). The task aims at recognizing the entities and their types in raw texts and linking them to distinct Knowledge Base (KB)s [8] . In addition, NERD systems enable the processing of unstructured texts to provide useful data for information retrieval, information extraction, machine translation, question answering systems and automatic summarization tools.

Although NERD approaches have been widely researched nowadays and shown good precision, they still present inefficient time-performance algorithms and poor versatility. In an industrial environment, a given

newspaper company which desires to annotate their news can deal with different subjects in the same document. It makes the use of distinctive NERD systems and KBs harder. On the other hand, financial companies handle very large documents and the current NERD tools do not deliver the response in a reasonable time (e.g., they may take almost half day to deliver the results from a big data set [2]). For instance, DBpedia Spotlight [6] is able to process only 120 queries per minute [3] and also retrieves information only from DBpedia [5] , which is not enough for enterprise companies who deal with large documents and different subjects. Therefore, the main lack is scalability which becomes a key factor for academic solutions to be adopted by the industrial environment.

Early efforts have focused on algorithms and evaluations, resulting in stand-alone applications that aimed to solve problems in specific domains. For example, Rizzo et al. [10] and Borodino et al. [2] proposed the combination of NERD systems for industrial solutions, but they did not focus on scalability. Thus, it becomes difficult to use them in real cases. Moreover, the systems must be checked whether there is any limitation for using in distributed environments regarding their licenses.

To this end, we present TANKER, an approach to address the aforementioned gaps by combining NERD solutions through a micro-services architecture. TANKER is a REST based service that decreases drawbacks especially with regards to integration, licensing, outdated technologies and availability/scalability. The main contributions intended with our work are:

- *Scalability* : TANKER allows to start new service instances by language/domain in response to rising demand with a round-robin distribution strategy.
- *Fault tolerance* : using a client-side IPC library, a request can be handled for *N* configured available servers
- *Completeness* : under the same request, TANKER can query one-to-many services to provide better results.

The paper is structured as follows: In the next section, we present the related works. Section Section 3 presents the TANKER architecture in details and explains how TANKER address the gaps pertaining to industrial solutions. In Section Section 4 we present the primary implementation of TANKER. Finally, we give an outlook on further directions and possibilities for TANKER in Section Section " SUMMARY " .

2. RELATED WORK

To the best of our knowledge, there is no work proposing a distributed architecture based on micro-services, especially focusing on its infrastructure and scalability. However, there are two works which had proposed distinct architectures for combining NERD systems without relying on machine learning (ML) algorithms. They are as follows:

- *NERD framework* : In 2012, proposed a generic framework which groups either commercial and research approaches among several entity recognition tools. The Named Entity Recognition (NER) tools are made available via Web APIs and they use a hybrid approach for presenting the different outputs of each NER tool via a unique response. Thus, providing users the opportunity to easily query each of these services through the same setup and compare their outputs. In addition, NERD is tailored for entity recognition of Twitter streams and has a public web API [4] .
- *HERMES framework* : The authors proposed a novel NLP framework dubbed HERMES which focuses on addressing the performance at the infrastructure layer. HERMES provides an Entity Recognition and Disambiguation service, enhanced with three features (topic extraction, topic labeling and topic explanation). It uses Apache Kafka [5] to deal with message queues. However, their implementation comprises of asynchronous modules which does not consider the response time. This implies several challenges for developers, as it is hard to deal with different kinds of failure in asynchronous scenarios. Although Hermes architecture is based on modules, there is no web service or API provided by this solution.

In terms of components, i.e. NER, Entity Linking (EL) and NERD models, there are a plenty of available tools that could be integrated in our architecture. However, for the sake of space, we only introduce the approaches

which are included in TANKER.

- *NER - Stanford NER* [6] [4] is a Java implementation of a NER system. It labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names. It comes with well-engineered feature extractors for Named Entity Recognition, and many options for defining feature extractors. Stanford NER implements Conditional Random Fields (CRF) sequence models to perform NER tasks in pre-existing training sets, and one can also train a new model.
- *NED – AGDISTIS* [7] [11] is an open source named entity disambiguation framework. Its early version can link entities by combining the HITS algorithm with label expansion strategies and string similarity measures. The newer version of it includes a new algorithm called MAG [7] . MAG is a multilingual and deterministic algorithm which disambiguates entities from a given knowledge base by using HITS and PageRank along with an in-depth context search based on TF-IDF statistics. Based on this combination, it can efficiently detect the correct URIs for a given set of named entities within an input text. Furthermore, AGDISTIS is agnostic of the underlying knowledge base.
- *NERD* - One of the first semantic approaches published in 2011, *DBpedia Spotlight* [8] [6] is a tool which combines NER and NED approaches for automatically annotating mentions of DBpedia resources [3] in texts. In addition, Spotlight contains programmatic interfaces based on a vector-space representation of entities and cosine similarity for phrase spotting, i.e., recognition of phrases to be annotated. Moreover, it can export the results in various output formats such as XML, JSON/JSON-LD, RDF, NIF, and N3.

3. ARCHITECTURE

TANKER was designed with portability and efficiency in mind. It was also designed to be customizable and extensible w.r.t. its user interface, functionality and the integration of underlying components. The overall architecture of TANKER is shown in TANKER is built using a microservice architecture [9] . Microservices have been getting a lot of attention and popularity in the recent years [9] because of its significant benefits, especially w.r.t enabling the agile development, improving scalability, reliability and failure tolerance.

TANKER does not depend on any specific NERD service and is generic enough to be connected to any replacement microservice that abides by given service specification. This allows for simple configuration as well as adds a way to extend the functionality of the system - by adding or removing the microservices we can easily tailor the final user experience.

Additionally, microservices enforces a modularity level which is much faster to develop, and easier to understand and maintain. This architecture enables the development of each server independently by letting the architects free to choose appropriate technologies for different kind of problems. Thus, being possible to combine different programming languages in one single solution. The other components of TANKER are described in the sequence.

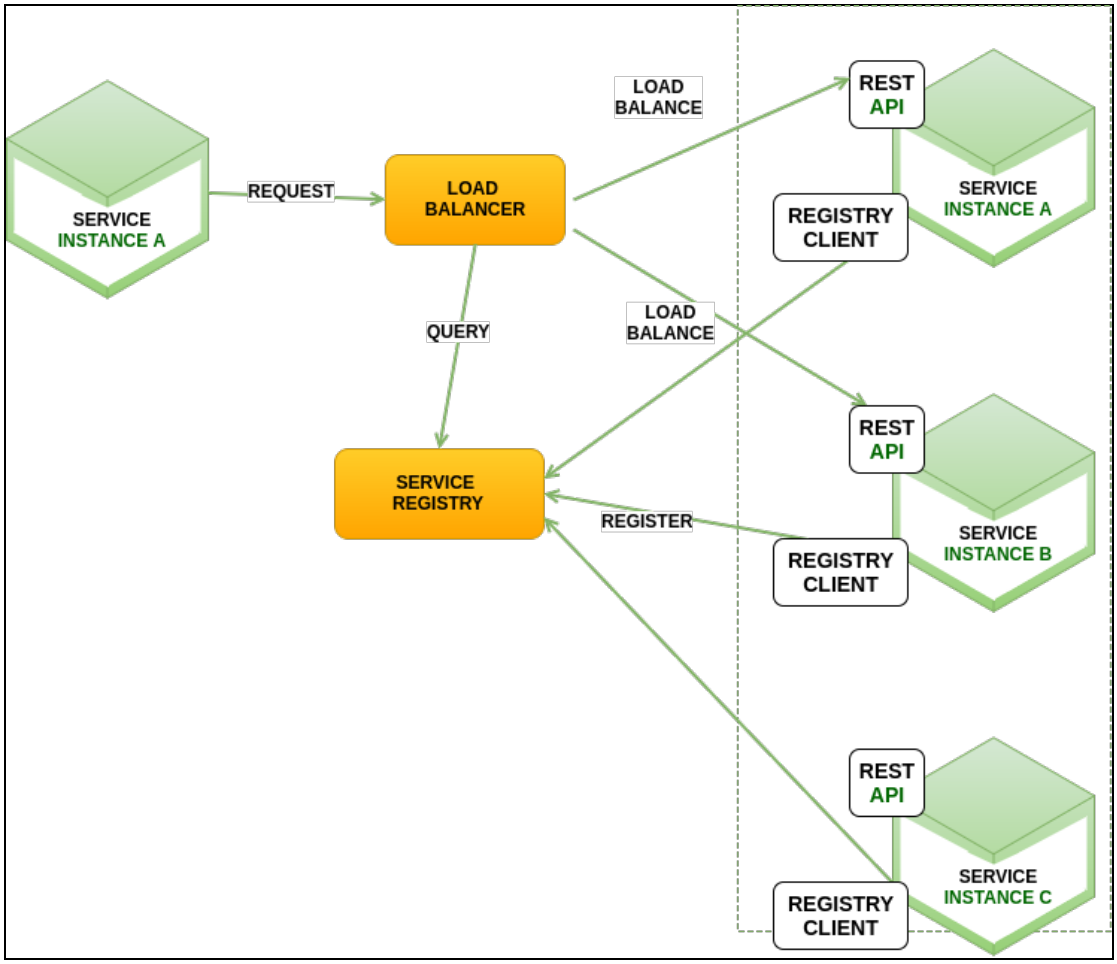

Figure 1. Overview of TANKER architecture

3.1. Scalability and Fault tolerance

Scalability is the capability to handle a growing amount of processes within a computational system in a graceful manner, providing minimal interruptions to ongoing operations. This feature is mandatory for enterprise systems in addition to fault-tolerance, scaling up mode and some others, to deliver a good user experience at a minimum cost.

Most of NERD solutions are usually developed under synchronous or asynchronous requests. Both requests wait for a response, however, the synchronous might block the user interaction while the asynchronous does not. Commonly, distributed applications consider services interactions to decide which one will be used in its ecosystem. For instance, in synchronous requests, HTTP REST or Thrift [10] are adopted, on the other hand, the Advanced Message Queuing Protocol [11] is used for asynchronous.

These request types infer on Inter-Process Communication (IPC) mechanisms. IPC software is a central piece of the architecture to ensure that microservices will scale and have fault-tolerance. This component is usually designed to be highly configurable and supports running in hybrid environments that are multi-region and multi-zone. To this end, TANKER supports synchronous interactions relying on Ribbon framework [12] as our IPC. Ribbon framework offers client-side software load balancing algorithms and a good set of configuration options such as connection timeouts and retry algorithms that fills in our requirements for NERD environments.

3.2. Configuration

The configuration of TANKER aims to reduce the complexity of management processes by using cloud services. Therefore, TANKER is based on Spring Cloud Config [13] which offers a client-side application for exposing configuration of a distributed system. It is integrated with Spring ecosystem, but it can also be used

with any application running in any programming language. This service uses the human-readable data serialization language (YAML [14]), a widely spread format to describe services parameters and exposes all of them under a REST/API (see Listing 1).

```
spring:
    profiles: eureka primary
    cloud:
        config:
            uri: http://localhost:8001
    eureka:
        instance:
            preferIpAddress: true
            enableSelfPreservation: false
        client:
            name: eureka
. . .
```

Listing 1. YAML example

Moreover, the delivered configuration through agnostic technologies will reinforce TANKER architecture pliability, by allowing approaches written in non-JVM technologies to reuse parameters and quickly integrate to the IPC.

3.3. Service Registry

Distributed systems need to localize the network address of each service. Services assign network locations dynamically. Moreover, the set of services instances also changes periodically because of auto-scaling, failures, and upgrades. This reinforces the need to have an elaborated service framework that uses a discovery pattern.

There are two service registry patterns: client-side discovery and service-side discovery. In client-side discovery pattern, clients query the service registry to select an available resource and perform a request. In the server-side discovery pattern, clients make a request via router, which queries the service registry and forward the request to an available instance [1] .

TANKER uses Eureka [15] for service discovery. Eureka is a REST server-side discovery service that locates services for load balancing and fail over of middle-tier servers.

3.4. Completeness

TANKER architecture offers a pluggable platform that allows the whole community to interconnect a set of services and approaches, making it available into a powerful standardized API. To this end, TANKER comprises of scaffoldtechnique. This technique provides a basis configuration that allows a quick development by using parameters from our setup services and by connecting to our service registry infrastructure.

4. PROTOTYPE

On the primary version, we support both recognition and linking of named entities using DBpedia as a knowledge base. We chose DBpedia as primary KB for our prototype because the most of NERD approaches can handle it. Also, we intend to evaluate TANKER in GERBIL which comprises of many datasets using DBpedia as a KB. Therefore, we include DBpedia Spotlight and Stanford NER for carrying out the recognition part and AGDISTIS framework to disambiguate the recognized entities (see Figure 2).

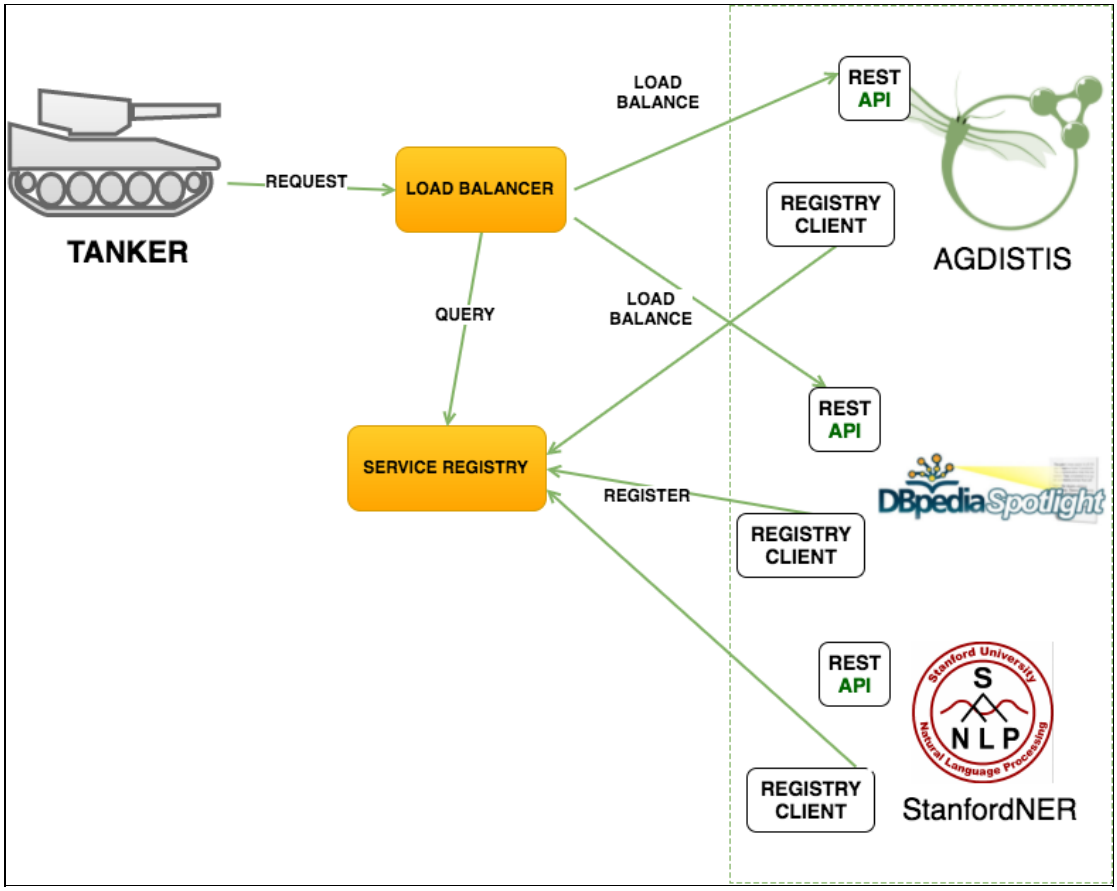

Figure 2. Prototype schema with integrated services

The described tools will be available into the TANKER infrastructure, under three endpoints: annotate, disambiguate and recognition.

- *Annotate* . This parameter performs both name entity recognition and disambiguation (for an example, see Listing 2). TANKER can deliver both the types of entities and their resources links
- *Disambiguate* . Once the entities are already recognized in texts, this parameter only disambiguates them.
- *Recognition* . This parameter is only able to recognize the types of entities.

```
{
  "text": "Angela met Obama in New York"
                  "resources": [
  {
    "surface-form": "New York",
    "offset": "20:28",
    "score": "0.96",
    "type": "dbo:Location",
    "origin-tool": "Spotlight"
  {"disambiguate": [
  {
    "uri": "dbr:New_York",
    "types": "dbo:Location",
    "surface-form": "New York",
    "offset": "20:28",
    "similarity-score": "0.86",
    "percentage-second-rank": "2",
    "origin-tool": "SpotLight"
  }
  {
    "uri": "dbr:New_York_City",
    "types": "dbo:Location",
    "surface-form": "New York",
    "offset": "20:28",
    "similarity-score": "0.92",
    "percentage-second-rank": "1",
    "origin-tool": "AGDISTIS"
  }]}
...
}
```

Listing 2. Annotating New York as entity.

When a client performs a request, all the available tools in the service discovery will be queried, and their results will be consolidated in the response. As you can see in the Listing 2, the entity New York got different resources from the disambiguation tools, but TANKER provides both and rank them according to their scores to let the user chooses. TANKER initially supports content-negotiation for JSON, JSON-LD, NIF, and N3.

4.1. Challenges

After the deployment of our first prototype we identified three challenges to be addressed. First, to rank the response when the tools diverge the results from a same entity even more when one of the tools does not provide any score. Second, to combine different KBs at once in a reasonable response time. Finally, to manage the outcomes and configurations of experiments. To bridge this gap, we plan to integrate the MEX vocabulary [14] and stored the configurations of experiments and respective outcomes in the WASOTA repository [13] .

SUMMARY

We presented TANKER, a distributed architecture for combining NERD systems. In a preliminary overview, our approach can deal with a large-scale processing and a high number of requests. In addition, TANKER responds to the queries in an appropriate response time thus addressing the aforementioned gaps. As an immediate work, we intend to integrate more NERD systems in order to improve the fault tolerance and evaluate TANKER using GERBIL [12] to see the real performance of it compared to other NERD systems. As a future work, we plan to include a KB management service for enabling TANKER to process different KBs altogether. Furthermore, in order to facilitate data management and follow best practices in terms of reproducibility of experiments, we will integrate TAKER within state of the art ML vocabularies and metadata repositories.

ACKNOWLEDGEMENTS

This paper's research activities were funded by grants from the FP7 & H2020 EU projects ALIGNED (GA-644055) and from the project Smart Data Web BMWi project (GA-01MD15010B) and CNPq foundation (scholarships 201808/2015-3 and 206971/2014-1).

REFERENCES

1. K. Bakshi. Microservices-based software architecture and approaches. In 2017 IEEE Aerospace Conference, pages 1-8, March 2017.
2. Bordino, A. Ferretti, M. Firrincieli, F. Gullo, M. Paris, S. Pascolutti, and G. Sabena. Advancing nlp via a distributed-messaging approach. In Big Data (Big Data), 2016 IEEE International Conference on, pages 1561-1568. IEEE, 2016.
3. J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes. Improving e ciency and accuracy in multilingual entity extraction. In Proceedings of the 9th International Conference on Semantic Systems, I-SEMANTICS '13, pages 121-124, New York, NY, USA, 2013. ACM.
4. J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In ACL, 2005.
5. J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. SWJ, 2014.
6. P. N. Mendes, M. Jakob, A. Garcia-Silva, and Bizer. DBpedia Spotlight: Shedding Light on the Web of Documents. In 7th International Conference on Semantic Systems (I-Semantics), 2011.
7. D. Moussallem, R. Usbeck, M. R. oder,• and A.-C. N. Ngomo. Mag: A multilingual, knowledge-based agnostic and deterministic entity linking approach, 2017.
8. D. Nadeau and S. Sekine. A survey of named entity recognition and classification. Lingvisticae Investigationes, 30:3-26, 2007.
9. S. Newman. Building microservices. " O'Reilly Media, Inc., USA, 2015.
10. G. Rizzo and R. Troncy. Nerd: a framework for unifying named entity recognition and disambiguation extraction tools. In Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics, pages 73{76. Association for Computational Linguistics, 2012.
11. R. Usbeck, A. N. Ngomo, M. Roder,• D. Gerber, S. A. Coelho, S. Auer, and A. Both. AGDISTIS - graph-based disambiguation of named entities using linked data. In P. Mika, T. Tudorache, A. Bernstein, Welty, C. A. Knoblock, D. Vrandecic, P. T. Groth, F. Noy, K. Janowicz, and C. A. Goble, editors, The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I, volume 8796 of Lecture Notes in Computer Science, pages 457-471. Springer, 2014.
12. R. Usbeck, M. Roder,• A. N. Ngomo, C. Baron, Both, M. Brummer,• D. Ceccarelli, M. Cornolti, D. Cherix, B. Eickmann, P. Ferragina, C. Lemke, A. Moro, R. Navigli, F. Piccinno, G. Rizzo, H. Sack, R. Speck, R. Troncy, J. Waitelonis, and L. Wesemann. GERBIL: general entity annotator benchmarking framework. In A. Gangemi, S. Leonardi, and A. Panconesi, editors, Proceedings of the 24 th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015, pages 1133-1143. ACM, 2015.
13. Neto, C. B., Esteves, D., Soru, T., Moussallem, D., Valdestilhas, A., & Marx, E. (2016). WASOTA: What Are the States Of The Art?. In SEMANTiCS (Posters, Demos, SuCCESS).
14. Esteves, D., Moussallem, D., Neto, C. B., Soru, T., Usbeck, R., Ackermann, M., & Lehmann, J. (2015, September). MEX vocabulary: a lightweight interchange format for machine learning experiments. In

Proceedings of the 11th International Conference on Semantic Systems (pp. 169-176). ACM.

Footnotes

1 http://www.northeastern.edu/levelblog/2016/05/13/how-much-data-produced-every-day/ [back]

2 See the average time in http://gerbil.aksw.org/gerbil/experiment?id=201701260017 [back]

3 https://github.com/dbpedia-spotlight/dbpedia-spotlight/wiki/User's-manual [back]

4 http://nerd.eurocom.fr [back]

5 http://kafka.apache.org [back]

6 https://nlp.stanford.edu/software/CRF-NER.shtml [back]

7 http://aksw.org/Projects/AGDISTIS.html [back]

8 http://www.dbpedia-spotlight.org [back]

9 https://www.oreilly.com/ideas/the-evolution-of-scalable-microservices [back]

10 https://thrift.apache.org/ [back]

11 https://www.amqp.org/ [back]

12 https://github.com/Netflix/ribbon [back]

13 http://cloud.spring.io/spring-cloud-static/spring-cloud.html [back]

14 http://yaml.org/ [back]

15 https://github.com/Netflix/eureka [back]