Towards Scalability Guidelines for Semantic Data Container Management

Gunnar Brataas
SINTEF Digital
Trondheim, Norway
gunnar.brataas@sintef.no

Bernd Neumayr
Johannes Kepler University Linz
Linz, Austria
neumayr@dke.uni-linz.ac.at

Christoph G. Schuetz
Johannes Kepler University Linz
Linz, Austria
schuetz@dke.uni-linz.ac.at

Audun Vennesland
SINTEF Digital
Trondheim, Norway
audun.vennesland@sintef.no

ABSTRACT

Semantic container management is a promising approach to organize data. However, the scalability of this approach is challenging. By scalability in this paper, we mean the expressivity and size of the semantic data containers we can handle, given a suitable quality threshold. In this paper, we derive scalability characteristics of the semantic container approach in a structured way. We also describe actual experiments where we vary the number of available CPU cores and quality thresholds. We conclude this work-in-progress paper by describing how more measurements could be performed so that the missing guidelines could be provided.

ACM Reference Format:
Gunnar Brataas, Bernd Neumayr, Christoph G. Schuetz, and Audun Vennesland. 2018. Towards Scalability Guidelines for Semantic Data Container Management. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion , April 9–13, 2018, Berlin, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3185768.3186302

1 INTRODUCTION

Semantic technologies help to create and manage conceptual models – also referred to as ontologies – and to apply conceptual models in large-scale and decentralized information systems to foster a common understanding of data and metadata. Semantic data container management is an ontology-based approach to organize data sets and to automate the discovery of data sets that fulfill a particular information need [8]. The semantic data container approach is currently being developed in the course of a collaborative research project, termed BEST (http://project-best.eu), in the area of air traffic management (ATM). A semantic container consists of data items and metadata that provide a high-level description of the membership condition to be fulfilled by a data item in order to be part of the semantic container. The membership condition typically describes geospatial and temporal scope of the data items, as well as other semantics, relevant for the ATM domain. For example, a semantic container's membership condition may state that the container comprises all Notices to Airmen for the route from Vienna to Frankfurt on 13 January 2018 that are relevant for heavy wake aircraft. The membership condition is expressed in terms of ontologies. A semantic reasoner then automatically organizes membership conditions in a hierarchy, which serves as an index. For example, the semantic container with Notices to Airmen for the route from Vienna to Frankfurt is subsumed (generalized) by the semantic container with Notices to Airmen for the entire European airspace. The reasoner also matches information needs expressed as membership conditions to membership conditions of available semantic containers. The computational complexity of semantic reasoning may lead to poor scalability of the semantic container approach.

We refer to scalability as a system's ability to increase the capacity by consuming more hardware and software resources [2]. Scalability analysis then investigates the scalability implications of higher load (more traffic), more work (computationally harder operations and/or more data), and stricter quality thresholds (e.g. shorter response times). In this context, we refer to scalability implications as the amount of additional hardware (CPUs, memory, network capacity) and software resources (software licences) that are required to handle increasing amounts of load or work. For example, if doubling the work requires a tenfold of underlying resources to keep within service level agreements, the system does not scale. Scalability problems are even worse if a system is not able to handle an increase in load regardless the amount of additional hardware or software resources. Such scalability problems should be identified early in the development process in order to discover the sources of the problems as well as possible solutions.

Before depending on a system architecture, it is therefore important to know its scalability implications. The contribution of this paper is an instantiation of a scalability framework [2] with respect to semantic data container management. In particular, the work dimension is elaborated in terms of ontology expressivity and various parameters for ontology size. Based on this instantiation, we provide initial measurements for producing scalability guidelines for semantic container management.

The remainder of this paper is organized as follows. Section 2 outlines the state of the art. Section 3 characterizes scalability for the semantic container approach. Section 4 describes experiments.

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5629-9/18/04.
https://doi.org/10.1145/3185768.3186302

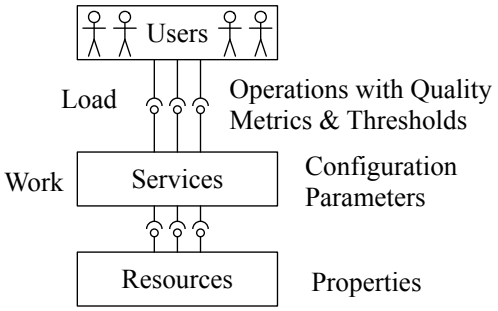

Figure 1: Scalability concepts [1]

Section 5 concludes the paper with a summary and an outlook on future work required to derive the required scalability guidelines.

2 STATE OF THE ART

This section gives a short introduction to the state of the art in scalability of semantic technologies. The Web Ontology Language (OWL 2) is a standard ontology language for the (semantic) web, and OWL 2 DL the most expressive decidable subset of the full language [5]. Further considerations on reasoning performance of OWL 2 ontologies led to the definition of a set of sublanguages – the OWL 2 profiles [6] – with reduced expressivity but also lower time complexity of common reasoning tasks when compared to OWL 2 DL. For example, the OWL 2 EL profile [6], which is sufficient in many practical situations, where the ELK reasoner [7] serves as an efficient implementation. Reasoning performance in practice also depends on the characteristics of the ontology.

Ensuring the scalability of reasoning tasks has been identified as a major challenge for implementing real-world applications using semantic technologies due to an "inherent trade-off between the expressivity of a logical representation language and scalability of reasoning" [4, p. 524]. Parallelization of reasoning tasks is a common strategy to ensure scalability. In this paper, we investigate reasoning capacity for semantic container management with the ELK off-the-shelf reasoner depending on the number of available processors on a single machine.

3 SCALABILITY REQUIREMENTS

In this section, we describe the main scalability concepts and how they are specified for semantic containers. We build on the scalability framework in [2] illustrated in Fig. 1. Ideally, all these concepts should have been explicitly present in Fig. 1, but in some cases, this is only implicitly the case, e.g. for system, described in the next sub section, which refers to services and resources in the figure.

3.1 System

When analyzing scalability, we must define which services are inside and outside of our system boundary. These boundaries also define how response times are measured. In this paper, we focus on reasoning about metadata which serves for adding/updating and querying containers. We do not investigate the population of the containers with actual data since the population of semantic containers highly depends on the specific application scenario. For example, a semantic container for Notices to Airmen may be filled with actual data using prioritization and filtering rules [3] specific to a particular airline which might be much more complex than the rules employed by another airline or the rules for another type of data such as weather forecasts. Other kinds of data, e.g., meteorological data or flight plans, require entirely different rules to be used for populating semantic containers. Scalability analysis of populating semantic containers has to be refined for each individual application scenario. Reasoning about the metadata of containers, on the other hand, is less dependent on the specific application scenario, which is the focus of this paper.

3.2 Critical Operations

An operation defines a unique and relatively similar way of interacting with a service; an operation corresponds to a request class in the context of queuing networks. Common synonyms for operations are transactions, functions and queries.

Of the metadata operations obeying the system boundaries described in Section 3.1, the following three operations are essential, i.e., must be present in order to productively employ a semantic container management system: Make a subsumption (generalization) hierarchy, extend the subsumption hierarchy, and find individual data containers, which we elaborate in the following.

3.2.1 Initialize: Make Subsumption Hierarchy. The subsumption (generalization) hierarchy of semantic containers serves as an index for the retrieval of semantic containers. A more general semantic container subsumes a more specific semantic container. For example, a semantic container with Notices to Airmen (NOTAMs) for the European airspace subsumes a semantic container with only the NOTAMs for the route from Vienna to Frankfurt.

The subsumption hierarchy derives from faceted membership conditions; a membership condition refers to one value for each facet such as geographic area and temporal scope. For example, a semantic container with NOTAMs relevant for heavy-wake aircraft on the route from Vienna to Frankfurt on the 13 January 2018 has three facets: geography, temporal, and aircraft. For each of these facets, the semantic container refers to one concept from the corresponding ontology: a concept *RouteVIE-FRA* from a geography ontology, a concept *13-01-2018* from a temporal ontology, and a concept *HeavyWakeAircraft* from an aircraft ontology.

Deriving the subsumption hierarchy of semantic containers is a two-step process. First, the reasoner derives concept hierarchies for the ontologies used for the facets of the semantic container description. For example, the reasoner determines that *SuperHeavyWakeAircraft* is more specific than *HeavyWakeAircraft* and that *A380* is more specific than *SuperHeavyWakeAircraft*. For each facet ontology, the concept hierarchy may come from a separate reasoner, or be asserted. By "asserted" we mean predefined for that ontology rather than inferred through logical properties. Then, having the subsumption hierarchy for each facet ontology, the reasoner determines the subsumption hierarchy of semantic containers, which have a reference to one concept for each facet. For example, a semantic container has a value *HeavyWakeAircraft* for the aircraft facet and *January_2018* for the date facet. Another semantic container

has facet values *Aircraft* and *2018*. The latter container subsumes the former.

3.2.2 Add Semantic Container: Extend Subsumption Hierarchy. The subsumption hierarchy of semantic containers needs to be updated to accommodate new semantic containers. The employed reasoning algorithm works only for a certain complexity of the semantic container descriptions: The more expressive the ontology, the more complex the expressed semantic container descriptions, the more complex the reasoning algorithm. For simple ontologies, we can do incremental reasoning, whereas for more complex ontologies we may have to make the subsumption hierarchy from scratch. In this paper, we employ the ELK reasoner to study scalability of the semantic container approach. The ELK reasoner is capable of incremental reasoning. Adding or removing an axiom does not necessitate a full recalculation of the subsumption hierarchy.

3.2.3 Find Semantic Containers. Different semantic containers fulfill different information needs. Technically, an information need is represented by a membership condition in the same ontology language as the subsumption hierarchy of semantic containers – in our case an OWL EL class expression. The task of finding semantic containers that satisfy a given information need is referred to as semantic container discovery.

3.3 Work

Work characterizes the amount of data to be processed, stored or communicated when invoking one operation. Ultimately, work characterizes the amount of hardware resources consumed when invoking one operation. The set of operations is of course an important part of the characterization of work. In addition, when considering scalability, we are also interested in how the work for one operation varies. This variation is connected to sizes of relevant objects, e.g., the number of documents and their average size; such parameters are referred to as work parameters. For scalability, the highest values of the work parameters are most relevant. Whereas load typically goes up and down during the day, week and month in complex patterns, work parameters are simpler as they typically only increase in value. For operations which encompass ontological reasoning, the most relevant work parameters are ontology expressivity and ontology size.

3.3.1 Ontology Expressivity. Ontology expressivity concerns the complexity of the axioms in the ontology and thus the complexity of automatic reasoning. A more expressive ontology language allows to describe more precisely the contents of a semantic container as well as information needs. This often comes with high computational costs. Ontology language profiles restrict the expressivity of ontologies in order to allow for more efficient reasoning. The OWL EL ontology language profile disallows, for example, the use of the OR operator in class expressions. In this paper, we consider the use of OWL EL. In our experiments, we further restrict the expressivity to class hierarchies (subclassOf axioms) and to class definitions with intersectionOf class expressions.

3.3.2 Ontology Size. In the context of semantic container management, the work parameter of ontology size can be further broken down into the following work parameters:

- Number of containers. The actual size of the container is not relevant, since we are working with meta-data.
- Number of facets: A facet is a property of all the data items in the container, for example location and time. The complexity of the container hierarchy is determined by the complexity of the facet hierarchy.
- Number of classes per facet: For a spatial facet, the number of locations (represented by bounding boxes) will determine the number of classes.
- Depth and complexity of facet hierarchy: The classes of each facet will form a hierarchy – a tree or a (semi-) lattice – possibly at different levels, where the number of levels is the depth of the hierarchy.

3.4 Load

Load is how often an operation is invoked. The term load refers to the frequency of invocation of an operation. In this paper, we focus on work and leave the analysis of the influence on load to future work. Load is most important in the case of finding semantic containers since that operation is more frequently invoked in day-to-day work than making the subsumption hierarchy.

3.5 Quality Metrics and Thresholds

A quality metric defines how we measure a certain quality and is a key part of an SLA (Service Level Agreement). At an overall level, response times and throughput are traditional scalability quality metrics. Quality thresholds (QTs) describe the border between acceptable and non-acceptable quality for each operation.

Quality thresholds in our domain are measured in 90th percentile response times, since in this way outliers will not affect the capacity. Quality thresholds will be measured in seconds and hours. We can tolerate a longer time for making the subsumption hierarchy than extending the subsumption hierarchy. Finding data containers should be done even faster.

3.6 Resources and Capacity

We have active as well as passive resources. Active resources are hardware for processing (CPUs), storage (primary memory (RAM), flash memories, disks) and communication (network). Passive resources represent semaphores, buffers and pools, typically associated with storage. When considering scalability, passive resources are crucial, and not surprisingly, storage often represent scalability limitations. The cost of software licenses may also be important.

The highest workload fulfilling quality thresholds is the capacity of a system. In our case, we want to vary one work parameter. To get one single number of capacity, we must fix the remaining work parameters as well as quality metrics and quality thresholds. Then, the highest work parameter for the average operation which fulfills the quality thresholds, becomes the capacity.

4 EXPERIMENTS

We conducted experiments for the operation "make subsumption hierarchy". We were interested in how the capacity varied with the number of cores (varied from 1 to 16 cores) and a given quality threshold. We employed 90th percentile response time as our quality metric and varied the quality threshold between 0.5 sec, 5 sec and 20

seconds. Capacity is the number of semantic containers for which the subsumption hierarchy can be derived within the time given by the quality threshold.

4.1 Setup

As our hardware configuration, we used a Sun Fire X4600 (a machine from year 2008) with 8 CPUs (16 cores) of type AMD Dual-Core Operton 885 2.6 GHz and with 58 GB of RAM.

As our software configuration, we use Linux (CentOS Release 6.9), Java (JRE 8 Update 151), ELK reasoner (Version 0.4.3) and a custom-made Java program which is invoked with a maximum heap size of 50 GB. The Java program includes the ELK reasoner as a library.

A shell script turns on and off cores (to vary the number of cores from 1 to 16) and invokes the Java program with different quality thresholds (0.5, 5, and 20 seconds).

To find the capacity for a given number of cores and a given quality threshold, a custom-made Java program performs binary search. In the binary search, for a number of semantic containers (the tested capacity), the "make subsumption hierarchy" operation was executed up to ten times allowing one run exceeding the time limit (0.5 sec, 5 sec, or 20 sec). To save experimentation time, the binary search was stopped when an additional round of ten runs would affect the resulting capacity by less than 10 percent.

For each run, the Java program generates an OWL EL ontology according to given work parameters and performs subsumption reasoning for this ontology. In this experiment, we varied the number of semantic containers during the binary search. We fixed the other parameters as follows: 3 facets with a hierarchy depth (number of levels) of 5, and 3 children per parent; giving a tree of 364 classes per facet, i.e., $1 + 3 + 3^2 + 3^3 + 3^4 + 3^5$.

4.2 Results

Our results are shown in Fig 2. The x-axis is the number of cores while the y-axis is the highest number of containers where the quality thresholds are still obeyed. We measure this for 0.5 sec, 5 sec and 20 sec.

Based on these measurements, with a 40 times increase in the quality threshold, the capacity increases from 9000 to 50000 containers, i.e. approximately a six-time increase. This is not surprising since we get a more congested system. When it comes to the number of cores it is not easy to draw any conclusions based on these measurements, except that until approximately 7 cores the capacity increases.

5 CONCLUSIONS AND FURTHER WORK

In this paper, we have described how structured scalability analysis can be applied to semantic technologies and we have also described conducted experiments. To actually provide useful guidelines more experimental results are required.

The precision of the current experiments could be improved by replicating the measurements. Confidence intervals could then also be established.

In our measurements, we used quality thresholds in the order of seconds. With a higher and more realistic quality threshold, in the order of hours, the time to do these measurements would increase, but then the value of these experiments could also be higher. We also plan to measure the scalability of the other two operations "Add Semantic Container" and "Find Data Containers". Here realistic response times are in the order of second, so this should not be too time consuming. Also, the consequences of load should be studied.

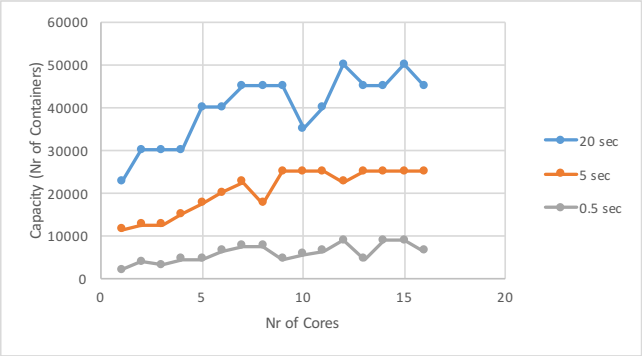

Figure 2: Capacity measured in number of containers with 0.5 sec, 5 sec and 20 sec quality thresholds

More experiments are needed in order to identify definitive guidelines for the choice of specific semantic technologies to realize the semantic container approach. How would, for example, different semantic technologies with different ontology expressivity affect the ontology sizes (described in Section 3.3.2) we are able to handle?

ACKNOWLEDGEMENT

This project has received funding from the SESAR Joint Undertaking under grant agreement No 699298 under the European Union's Horizon 2020 research and innovation program. The views expressed in this paper are those of the authors.

REFERENCES

[1] Gunnar Brataas and Tor Erlend Fægri. 2017. Agile Scalability Requirements. In *ICPE, ACM/SPEC International Conference on Performance Engineering*. ACM.

[2] Gunnar Brataas, Nikolas Herbst, Simon Ivansek, and Jure Polutnik. 2017. Scalability Analysis of Cloud Software Services. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 285–292. https://doi.org/10.1109/ICAC.2017.34

[3] Felix Burgstaller, Dieter Steiner, Bernd Neumayr, Michael Schrefl, and Eduard Gringinger. 2016. Using a Model-Driven, Knowledge-Based Approach to Cope with Complexity in Filtering of Notices to Airmen. In *Proceedings of the Australasian Computer Science Week Multiconference (ACSW 2016)*. 46:1–46:10.

[4] Dieter Fensel, Frank van Harmelen, Bo Andersson, Paul Brennan, Hamish Cunningham, Emanuele Della Valle, Florian Fischer, Zhisheng Huang, Atanas Kiryakov, Tony Kyung-il Lee, Lael Schooler, Volker Tresp, Stefan Wesner, Michael Witbrock, and Ning Zhong. 2008. Towards LarKC: A Platform for Web-Scale Reasoning. In *Proceedings of the 2nd IEEE International Conference on Semantic Computing*. 524–529.

[5] Pascal Hitzler, Peter Patel-Schneider, Sebastian Rudolph, Markus Krötzsch, and Bijan Parsia. 2012. *OWL 2 Web Ontology Language Primer (Second Edition)*. W3C Recommendation. W3C. http://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

[6] Ian Horrocks, Zhe Wu, Achille Fokoue, Boris Motik, and Bernardo Cuenca Grau. 2012. *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation. W3C. http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.

[7] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. 2014. The Incredible ELK. *J. Autom. Reason.* 53, 1 (June 2014), 1–61. https://doi.org/10.1007/s10817-013-9296-3

[8] Bernd Neumayr, Eduard Gringinger, Christoph Schuetz, Michael Schrefl, Scott Wilson, and Audun Vennesland. 2017. Semantic Data Containers for Realizing the Full Potential of System Wide Information Management. In *Proceedings of the IEEE/AIAA 36th Digital Avionics Systems Conference*.