Transforming Very Large Models in the Cloud: a Research Roadmap

Cauê Clasen, Marcos Didonet del Fabro, Massimo Tisi

Transforming Very Large Models in the Cloud: a Research Roadmap

Cauê Clasen[1], Marcos Didonet Del Fabro[2], and Massimo Tisi[1]

[1] AtlanMod team, INRIA - École des Mines de Nantes - LINA, Nantes, France
{caue.avila_clasen, massimo.tisi}@inria.fr
[2] C3SL labs, Universidade Federal do Paraná, Curitiba, PR, Brazil
marcos.ddf@inf.ufpr.br

Abstract. Model transformations are widely used by Model-Driven Engineering (MDE) platforms to apply different kinds of operations over models, such as model translation, evolution or composition. However, existing solutions are not designed to handle very large models (VLMs), thus facing scalability issues. Coupling MDE with cloud-based platforms may help solving these issues. Since cloud-based platforms are relatively new, researchers still need to investigate if/how/when MDE solutions can benefit from them. In this paper, we investigate the problem of transforming VLMs in the Cloud by addressing the two phases of 1) model storage and 2) model transformation execution in the Cloud. For both aspects we identify a set of research questions, possible solutions and probable challenges researchers may face.

1 Introduction

Model transformation is a term widely used in Model-Driven Engineering (MDE) platforms to denote different kinds of operations over models. Model transformation solutions are implemented in general purpose programming languages or transformation-specific (often rule-based) languages such as ATL [8], Epsilon [10], or QVT [11]. These solutions access and manipulate models using existing model management APIs, such as the Eclipse Modeling Framework API, EMF [2]. Current model transformation solutions are not designed to support very large models (VLMs), i.e., their performances in time and memory quickly degrade with the growth of model size, as already identified in previous works[9].

Moving model transformation tools to a cloud may bring benefits to MDE platforms. In a cloud, a large amount of resources is shared between users (e.g., memory, CPUs, storage), providing a scalable and often fault-tolerant environment. Distribution issues are transparent to final users, which see cloud-based applications as services. Some initiatives for improving the performance of the EMF have been conducted. For instance, the Morsa [7] framework enables loading larger models, by using a storage framework based on documents (MongoDb).

While MDE techniques have been used to improve cloud-based solutions [13,3], not much work has been done the other way around. Cloud-based platforms are relatively new and researchers still need to investigate if/how/when

MDE solutions can really benefit from a cloud. This area has been called Modeling *in* the Cloud, or Modeling As A Service [1].

In this article, we present a set of research questions, possible solutions and probable challenges we may face when coupling MDE and Cloud Computing. Specifically, we concentrate on two main tasks to ultimately accomplish the execution of model transformations on the Cloud:

1. **Model storage in the Cloud:** a cloud-based and distributed storage mechanism to enable the efficient loading of VLMs to the Cloud, for subsequent querying and processing.
2. **Model transformation execution in the Cloud:** intended to take advantage of the abundance of resources by distributing the computation of the transformations to different processing units.

We will present a set of questions, benefits, and challenges that have risen in both these aspects, and possible solutions that need to be further investigated.

As future work we plan to implement a proposed solution to the problems described in this paper, in the form of a model transformation tool based on EMF and ATL. For this reason we base the examples in this paper on this technological framework.

This article is organized as follows. Section 2 presents the problem of storing/accessing models in the Cloud. Section 3 focuses on distributed model transformations in the Cloud. Section 4 concludes the paper.

2 Storage of Models in the Cloud

One of the core principles of cloud computing is to distribute data storage and processing into servers located in the cloud. In MDE, a cloud-based framework for model storage and/or transformation could bring several benefits, e.g.:

Support for VLMs. Models that would be otherwise too large to fit in the memory of a single machine could be split into several different nodes located in the cloud, for storage, processing, or both.

Scalability. The execution time of costly operations on models (e.g., complex queries or model transformations on VLMs) can be improved by the data distribution and parallel processing inherent capabilities of the Cloud.

Collaboration. A cloud-based model storage can simplify the creation of a collaborative modeling environment where development teams on different locations could specify and share the same models in real-time.

Other topics, such as transparent tool interoperability, model evolution and fault-tolerance could also benefit from the cloud computing principles and have yet to be further investigated [1].

In the next subsections we identify and discuss two main research tasks that have to be addressed to obtain an efficient mechanism for VLMs in the cloud:

1. how to access models in remote locations in a transparent way, so that existing MDE tools can make direct use of them;
2. how to distribute the storage of a VLM on a set of servers, to make use of the resources offered by the cloud.

2.1 Transparent remote model storage

The use of models in the Cloud should not hamper their compatibility with existing modeling environments. All complexity deriving from the framework implementation, such as element/node location, network communication and balance should be hidden to end users and applications. This transparency towards the MDE clients can be obtained by implementing the network communication mechanism behind the model management API.

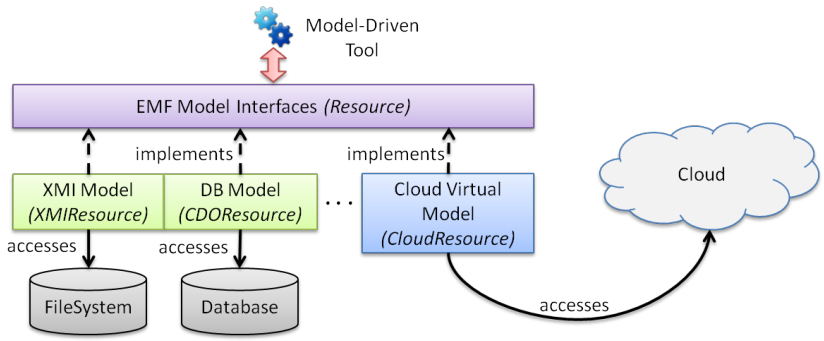

Fig. 1. Extending EMF with support of cloud storage for models.

The idea of providing alternative backends to model management APIs has already been used for local storage. For instance EMF allows applications to load and manipulate models stored as XMI files on disk when using its XMI backend, or the CDO[3] backend for models stored in databases. Clasen et al. [5] generalize this idea by introducing the concept of *virtual models* (with a direct reference to virtual databases) as a re-implementation of the EMF API to represent non-materialized models whose elements are calculated on demand and retrieved from other models regardless of their storage mechanism.

The same principle can be extended to support a cloud-based storage. The model management API can be extended/re-implemented to allow the access to a cloud-based persistence layer. Requests and updates of elements on this non-materialized model would be translated into calls to the web-services exposed by the cloud infrastructure. In our research agenda we plan to provide such a mechanism as a Cloud Virtual Model, illustrated in Figure 1.

[3] `http://www.eclipse.org/cdo/`

2.2 Distributed model storage

Data manipulated by a given cloud can come from a single data source (e.g., the client) or a distributed storage mechanism *in* the cloud. The second solution is especially useful to handle VLMs. The idea behind distributed model storage is to decompose a full model and to store subsets of its elements in different servers or physical locations (see Fig. 2). The sets of elements located in each node can be regarded as *partial models*, and from their composition the global model is constituted. The distribution strategy can be made invisible to the client application by using a virtualization layer as explained above. This way the persisted model is perceived as one single logical model.

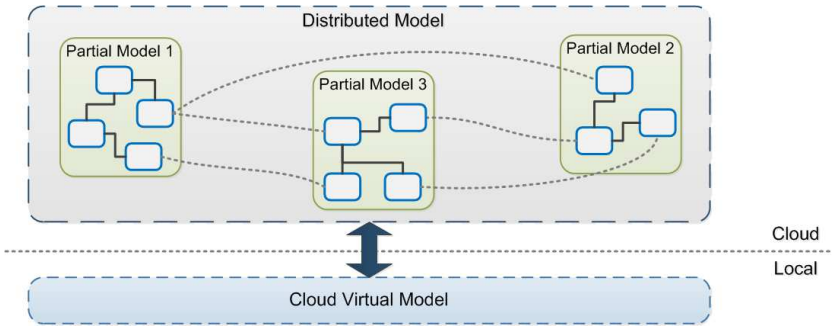

Fig. 2. A cloud virtual model that abstracts the composition of several distributed partial models. Dashed lines represent elements associations between cloud nodes.

Distributing model elements. The criteria used to define which elements are stored in each cloud node vary according to the context of use of the distributed model. For instance, when considering collaboration aspects, the model can be distributed to reduce the network costs, assigning to a given node the elements more likely to be accessed by the team located the closest to that node. As another example, in cases of parallel processing some knowledge about the computation algorithm may be used to assign model elements to nodes, to optimize parallelization. There are already approaches that study how to create partitions from graphs in a cloud for processing purposes (see the solutions from [12]). A study on the nature of MDE applications to identify correspondences with these existing approaches, in order to adapt them or to create novel solutions, has yet to be done.

Among the different model distribution policies two corner cases can be identified, analogous to the homonymous techniques in database systems:

– *Vertical Partitioning.* [14] Each partial model holds only elements conforming to certain types, i.e., each node has the responsibility to store only a certain

subset of concepts of the global model. For instance, a first partial model may contain only structural aspects of a UML model whereas a second may correspond to dynamic aspects.
- *Horizontal Partitioning.* [4] Each partial model holds elements of any type, and the separation conforms to a property-based selection criteria. For instance, elements representing French customers may be allocated to one node, whereas elements representing Brazilian customers may be allocated to another.

The choice of the distribution policy is not limited to partitions of the original set of model elements. Element replication could be desired to optimize the balance of network vs. memory usage [17].

Distributing associations. In most cases it is not possible to determine a partitioning in partial models that can be processed by completely independent nodes. Model elements can have different types of relationships between them (e.g., single and multi-valued references, containment references) and the computation on one node could at one point need to access an associated element contained in another one.

We first need a mechanism to store information about associations between elements located in different partial models. This information has to contain 1) pointers to locate incoming and outgoing associated element/s, and 2) the relationship type, so the distribution infrastructure can correctly interpret it. The pointers must necessarily contain both identifiers to the referenced elements within a partial model, and the location of the node that holds this partial model.

Cross-node associations can be distributed in several ways in the cloud nodes. Two main topologies are used by distributed databases and filesystems [17]:

1. The relationship metadata is centralized in a single node. All partial nodes must ask this central node for the location of the partial model containing the referenced element.
2. The relationship metadata is known by all nodes. Partial models then can directly request the referenced element to the correct node when necessary.

Both topologies have their pros and cons. Sharing the metadata in all nodes implies in extra memory usage, whereas a centralized metadata node requires extra inter-node communications. The choice of the best solution depends on several factors, as for instance node location, network bandwidth, and the quantity of cross-node associations.

Wherever the cross-node associations are stored, this information has to be correctly *interpreted* to enable navigation of the distributed model across nodes. When each node has a transparent abstraction of the full model, this navigation has to happen seamlessly. A navigation call to the virtual model of the node would have to start a resolution algorithm to: 1) retrieve the information about the cross-node association, 2) locate the requested elements from another node, and 3) return it to the MDE tool mimicking a call to a local model. This way,

when a node wants to access external model elements, it becomes itself a client of the cloud that contains it.

3 Model transformations in the Cloud

Model transformations are central operations in a MDE development process. A model transformation can be seen as a function that receives as input a set of source models and generates as output a set of target models. A transformation execution record is commonly represented in MDE as a set of *trace links*, each one connecting: 1) a set of source elements, 2) a set of corresponding target elements and 3) the section of code (e.g., rule, method) responsible for the translation.

Transformations can consume a lot of resources, in terms of memory occupation and computation time. Operations like traversing the full model or executing recursive model queries can be very expensive. When a centralized solution cannot handle the processing efficiently, one solution is to parallelize the execution of the transformations, for instance, within a cloud. The computation tasks have to be distributed on several nodes, each one in charge of generating partial outputs (i.e., models) that are later merged to obtain the full result. The expected result of a parallel computation must be the same result of its correspondent sequential transformation.

We sketch below a subdivision of the parallel transformation process in the three following steps, resulting on an overall conceptual view depicted in Fig. 3.

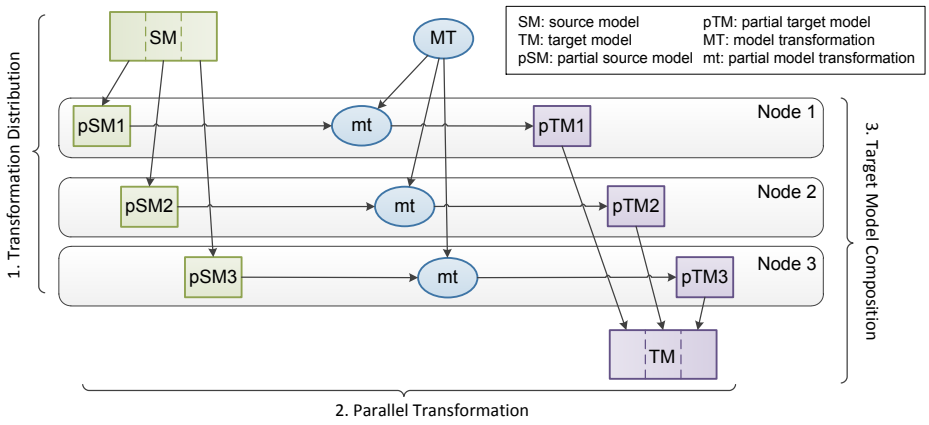

Fig. 3. Parallel Transformation Overview.

This process is divided in three major steps:

1. **Transformation Distribution.** An algorithm is defined to distribute the transformation computation over the available nodes. This phase may include a physical partitioning of the source model into partial models to send

to each server. The step is optional, e.g., in the case of transforming source models that are already distributed on nodes.
2. **Parallel Transformation.** The transformation code of each node is fed with a source model and runs in parallel with the others, generating a set of partial target models. When implementing the transformation engine running on the nodes, it could be advisable to re-use its sequential implementation, and to add a communication mechanism between nodes to access unavailable information. Both aspects have been discussed in Section 2.2.
3. **Target Model Composition.** The partial target models generated by each node are composed into a full target model.

While in the following we describe the three steps as sequential, they don't have to be necessarily executed eagerly. For example, in scenarios requiring costly data processing, but where a cloud-based storage infrastructure is not available, source models can be loaded and transformed lazily (i.e., on demand). Even when the target partial model have been computed, they don't necessarily have to be returned to the client as a full model, but a virtual model can be used to lazily retrieve needed model elements only when they are requested. The subject of lazy execution has been investigated in [15], and its application to the phases of parallel transformations is another pointer for future research.

3.1 Transformation distribution

The phase of transformation distribution is responsible for the assignment of parts of the transformation computation to each node. At the end of the parallel transformation the global transformation record will be constituted by the same set of trace links of the equivalent sequential transformation, since the correspondence between source and target model elements is not changed by the parallelization. Each node is responsible for a subset of the trace links, having translated only a subset of the VLM. Thus, distributing the computation of the transformation is equivalent to partitioning the set of trace links in groups assigned to the nodes.

To implement this partitioning, a distribution algorithm has to communicate to the nodes the needed information to determine which trace links to generate. Being a trace link uniquely determined by a set of source elements and a section of transformation code, the nodes, in general, have to receive information about the model elements they are responsible for and the transformation code to apply to each of them.

This information can be determined according to several different strategies. We classify the possible strategies based on the knowledge they exploit:

- In a first class of strategies, the transformation distribution algorithm assigns computation to nodes without ever loading the model. These approaches avoid the problem of loading a VLM that could exceed the limited memory of the client. Distribution can be still performed based on:

    - A partitioning of the transformation code (e.g., each node executes a single transformation rule). All nodes have access to the full source model (or its replica), but only execute a subset of the transformation code. This approach is called *transformation slicing*.
    - A low-level parsing of the serialized model, that in some situations can be split in consistent chunks without being fully loaded in memory. The chunks are then only loaded when they arrive to their assigned nodes.
  - A second class of transformation distribution algorithms allow to load the model (and its metamodel) and to select which computation to assign based on properties of the model elements. This category is called*model slicing*. Vertical and horizontal model partition algorithms (see Section 2.2) can be used as transformation distribution approaches of this type.
  - A more sophisticated class of algorithms would be based on static analysis of the transformation code, to determine a partitioning that can optimize parameters of the parallel execution, e.g., total time, throughput, network usage. Static analysis can for instance identify dependencies between transformation rules that can be exploited to maximize the parallelization of the computation. Similar dependencies have already been computed in related work, e.g., in [16] with a focus on debugging.
  - Finally several external sources of data can be used to drive the distribution, like usage statistics on model elements, or information about the cloud topology and resources.

An analogous characterization can be done for the algorithms of target model composition. For instance, for some transformations, the nodes could provide a perfect partition of the target model, without requiring the composition algorithm to load and analyze the produced partial models. Alternatively, some processing could be required during composition, e.g., to remove redundant elements, or to bind missing references.

Optimal algorithms in these classes may be a promising research subject.

3.2 Coupling model transformations with MapReduce

A well-known large scale data processing framework that may be adapted to implement distributed (especially on-demand) model transformations on the Cloud is the MapReduce framework[6]. MapReduce has three key aspects:

Input format: is always a pair *(key, value)*. The *key* is used to distribute the data. The *value* is processed by the framework. However, it is necessary to implement import/export components to be able to inject different formats (e.g., text files, databases, streams) into the framework.

Map tasks: receives each *(key, value)* pair and processes them in a given node. The result is another *(key, value)* pair.

Reduce tasks: receives the output of the Map tasks and merges them into a combined result.

Once the *(key,value)* pairs are defined and these two tasks are implemented, the framework takes care of the complex distribution-inherent details such as load balancing, network performance and fault tolerance.

In order to use MapReduce to execute model transformations, we need to precisely define how to represent models and model transformations in terms of these components. We can identify a clear correspondence in Fig. 3 between those steps and the MapReduce mechanism:

1. The source model needs to be divided into appropriate *(key,value)* pairs. Values should be partial source models.
2. The Map functions execute the transformations rules in parallel, on each one of the partial source models, generating partial target models as output data.
3. The Reduce functions combine the result of all Maps (i.e. partial target models) into a final result (i.e. a full composed model).

The exploitation of the MapReduce framework seems a feasible starting point towards parallel model transformations by allowing the research focus to be on MDE-related issues, while the framework is in charge of handling all cloud-inherent complications.

4 Conclusion and Future Work

In this article, we have discussed a set of research questions to port modeling and transformation frameworks into a cloud-based architecture. We have described different paths that need further investigation. Based on our previous experience on the use, development and research of model transformations, we have identified key aspects and divided them in two phases. First, an efficient model storage and access mechanism on the cloud needs to be investigated. The main difficulties are related on how to efficiently distribute the model elements and the relationships between them. Second, a parallel processing mechanism by distributed model transformations has to be provided. The main difficulties are about the distribution of the transformations coupled with the models that are going to be processed and how to combine the distributed results.

In our future work we plan to propose a solution, among the illustrated alternatives, in the form of a cloud-based engine for the ATL transformation language. The main design features for this engine will be: cloud transparency, re-use of the standard ATL engine, node inter-communication, and support for pluggable distribution algorithms. We also want to study how the static analysis of ATL transformations can help in optimizing the distribution algorithm for our engine. Finally, we hope that this article will promote discussion and involve other researchers to the task of moving MDE to the Cloud.

References

1. H. Brunelière, J. Cabot, and F. Jouault. Combining Model-Driven Engineering and Cloud Computing. In *MDA4ServiceCloud'10 (ECMFA 2010 Workshops)*, Paris, France, June 2010.
2. F. Budinsky. *Eclipse modeling framework: a developer's guide.* Addison-Wesley Professional, 2004.
3. S. Ceri, P. Fraternali, and A. Bongio. "Web Modeling Language (WebML): a modeling language for designing Web sites". *Computer Networks*, 33(1–6):137 – 157, 2000.
4. S. Ceri, M. Negri, and G. Pelagatti. Horizontal data partitioning in database design. In *ACM 1982 SIGMOD International Conference*, pages 128–136, Orlando, USA, 1982. ACM.
5. C. Clasen, F. Jouault, and J. Cabot. VirtualEMF: A Model Virtualization Tool. In *Advances in Conceptual Modeling. Recent Developments and New Directions (ER 2011 Workshops)*, LNCS 6999, pages 332–335. Springer, 2011.
6. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.
7. J. Espinazo Pagán, J. Sánchez Cuadrado, and J. García Molina. Morsa: A Scalable Approach for Persisting and Accessing Large Models. In *MODELS 2011*, LNCS 6981, pages 77–92. Springer, 2011.
8. F. Jouault and I. Kurtev. Transforming Models with ATL. In *MoDELS 2005 Workshops*, LNCS 3844, pages 128–138. Springer, 2006.
9. F. Jouault and J. Sottet. An AmmA/ATL Solution for the GraBaTs 2009 Reverse Engineering Case Study. In *5th International Workshop on Graph-Based Tools, Grabats*, Zurich, Switzerland, 2009.
10. D. Kolovos, R. Paige, and F. Polack. The Epsilon Transformation Language. In *ICMT 2008*, LNCS 5063, pages 46–60. Springer, 2008.
11. I. Kurtev. State of the Art of QVT: A Model Transformation Language Standard. In *AGTIVE 2007*, LNCS 5088, pages 377–393. Springer, 2008.
12. J. Lin and C. Dyer. Data-Intensive Text Processing with MapReduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177, 2010.
13. I. Manolescu, M. Brambilla, S. Ceri, S. Comai, and P. Fraternali. Model-Driven Design and Deployment of Service-Enabled Web Applications. *ACM Transactions on Internet Technology*, 5(3):439–479, Aug. 2005.
14. S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical Partitioning Algorithms for Database Design. *ACM Transactions on Database Systems*, 9(4):680–710, 1984.
15. M. Tisi, S. Martínez, F. Jouault, and J. Cabot. Lazy Execution of Model-to-Model Transformations. In *MODELS 2011*, LNCS 6981, pages 32–46. Springer, 2011.
16. Z. Ujhelyi, A. Horvath, and D. Varro. Towards Dynamic Backward Slicing of Model Transformations. In *ASE 2011*, pages 404 –407. IEEE, nov. 2011.
17. P. Valduriez and M. Ozsu. *Principles of Distributed Database Systems.* Prentice Hall, 1999.