Towards Distributed Model Analytics with Apache Spark

Önder Babur[1], Loek Cleophas[1,2] and Mark van den Brand[1]
[1]*Eindhoven University of Technology, NL-5612 AZ Eindhoven, The Netherlands*
[2]*Stellenbosch University, ZA-7602 Matieland, South Africa*

Keywords: Model-Driven Engineering, Model Analytics, Scalability, Distributed Computing, Apache Spark, Big Data.

Abstract: The growing number of models and other related artefacts in model-driven engineering has recently led to the emergence of approaches and tools for analyzing and managing them on a large scale. The framework SAMOS applies techniques inspired by information retrieval and data mining to analyze large sets of models. As the data size and analysis complexity goes up, however, further scalability is needed. In this paper we extend SAMOS to operate on Apache Spark, a popular engine for distributed Big Data processing, by partitioning the data and parallelizing the comparison and analysis phase. We present preliminary studies using a cluster infrastructure and report the results for two datasets: one with 250 Ecore metamodels where we detail the performance gain with various settings, and a larger one of 7.3k metamodels with nearly one million model elements for further demonstrating scalability.

1 INTRODUCTION

The use of models as a basis for software engineering—whether UML models used as a basis for design and implementation, or meta-/models in model-driven engineering (MDE)—has been exponentially growing in recent years. This is witnessed by e.g. the dramatic growth of models and related artifacts present both in open source such as the GitHub repository (Kolovos et al., 2015; Hebig et al., 2016) and in industrial MDE ecosystems (Babur et al., 2017). Analogously to earlier developments in source code analytics and text mining where very high volumes of data have long emerged as a reality and a challenge, this development necessitates similar approaches for analyzing *models* at larger scales. At the same time, models inherently display more complex structure, in general being graph-structured instead of the trees with (limited) cross-links typically encountered as representations of source code and natural language. Models in turn demand efficient and scalable, even if approximate, techniques for the analysis—versus comparing them one-to-one using exact but expensive techniques (graph edit distance, similarity flooding (Melnik et al., 2002), etc.). As a result, analytics becomes more complex for the setting of models, both in terms of techniques needed, and of computation effort required. Note that the requirements and potential added value for (big) data analytics in the general sense (i.e. not for models) has for long been widely recognized by the community (LaValle et al., 2011; Zikopoulos et al., 2011), and is not further elaborated in this paper.

One way to improve the performance of model analytics (for complex analyses of large datasets) is to look at distributed settings, versus running in a sequential setting on a single machine. There recently have been a few efforts of exploiting distributed computing in the MDE community, though in a different context for model transformations (Benelallam et al., 2015; Burgueño et al., 2016). In this paper, we sketch how the existing model analytics framework SAMOS can be lifted from the latter setting to a distributed setting using the Apache Spark framework for distributed computation. In Section 2 we sketch SAMOS, while Section 3 does the same for the Apache Spark framework. Section 4 considers how SAMOS can be modified and extended to operate in the distributed setting for potentially higher performance, while Section 5 discusses initial results for two sets of Ecore metamodels from a proof of concept, i.e. a version of SAMOS lifted to the Apache Spark framework, without any specific optimizations: one dataset with 250 Ecore metamodels reporting the performance gain with various settings, and a larger one of 7.3k metamodels with nearly one million model elements for further demonstrating our scalability. Section 6 concludes the paper with several indications for future work.

Babur, Ö., Cleophas, L. and Brand, M.
Towards Distributed Model Analytics with Apache Spark.
DOI: 10.5220/0006735407670772
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 767-772
ISBN: 978-989-758-283-7

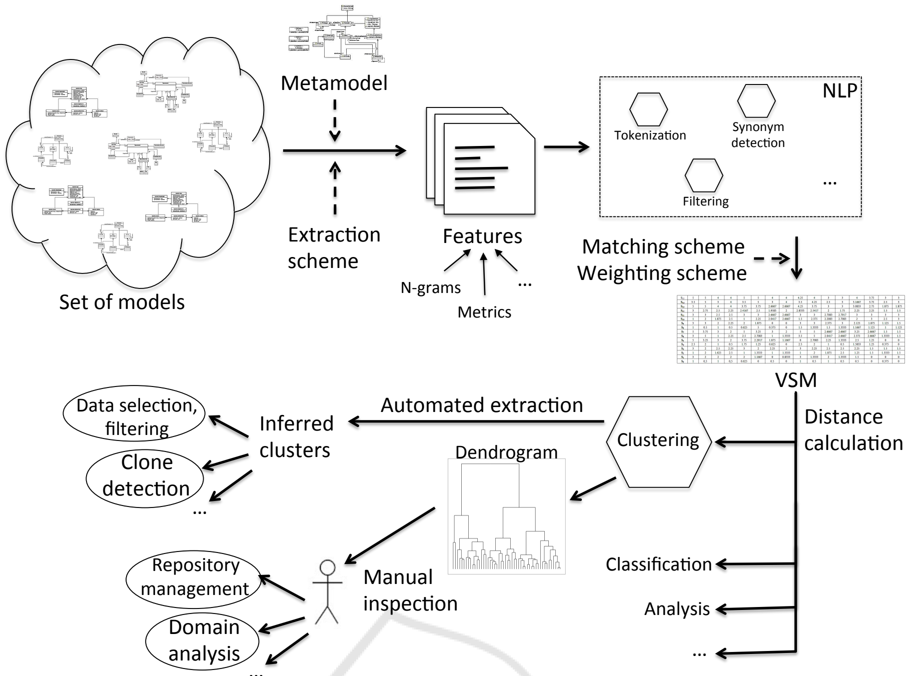

Figure 1: Overview of SAMOS workflow.

2 BACKGROUND: MODEL ANALYTICS FRAMEWORK

We outline here the underlying concepts of SAMOS (Babur, 2016; Babur et al., 2016), a framework for large-scale model analytics, inspired by the information retrieval (IR) and machine learning (ML) domains. IR deals with effectively indexing, analyzing and searching various forms of content including natural language text documents (Manning et al., 2008). As a first step for document retrieval in general, documents are collected and indexed via some unit of representation. Index construction can be implemented using a vector space model (VSM) with the following major components: (1) a vector representation of occurrence of the vocabulary in a document, named *term frequency*, (2) *zones* (e.g. 'author' or 'title'), (3) weighting schemes—such as inverse document frequency (idf)—and zone weights, (4) Natural Language Processing (NLP) techniques for handling compound terms and for detecting synonyms and semantically related words. The VSM allows transforming each document into an n-dimensional vector, thus resulting in an $m \times n$ matrix for m documents. Over the VSM, document similarity can be defined as the distance (e.g. euclidean or cosine) between vectors. These distances can be used for identifying similar groups of documents in the vector space via an unsupervised ML technique called clustering (Manning et al., 2008).

SAMOS applies this workflow to models, starting with a metamodel-driven extraction of features. Features can be, for instance, singleton names of model elements (very similar to the vocabulary of documents) or n-gram fragments (Manning and Schütze, 1999) of the underlying graph structure. N-grams originate from computational linguistics and represent linear encoding of (text) structure. In our context, an example n-gram for a UML class diagram for $n = 2$ would be a Class containing a Property (Babur and Cleophas, 2017). SAMOS computes a VSM via comparison schemes (e.g. whether to check types), weighting schemes (e.g. Class weight higher than Property) and NLP (stemming/lemmatization, typo and synonym checking, etc.). Applying various distance measures suitable to the problem at hand, it then applies different clustering algorithms (via the R statistical software) and can output automatically derived cluster labels or diagrams for visualization and manual inspection, thereby enabling exploration of large model sets. Figure 1 illustrates the workflow, showing key workflow steps of the workflow as well as application areas such as domain analysis and clone detection.

A sample output dendrogram is given in Figure 2 from (Babur et al., 2016). Given a domain analysis scenario, SAMOS is used to hierarchically group metamodels in the ATL Ecore Zoo[1]. The interpretation of this visualization is as follows: the numbers on the dendrogram correspond to the metamodels in the repository, which are gathered on (sub-)trees. The height of the tree joints represents the distance of the underlying nodes or sub-trees. Items in a subtree have similar domains (e.g. conference management metamodels), while the hierarchical structure of the dendrogram further reflects the intra-cluster similarity (e.g. word and excel subtrees under the large *office* subtree.

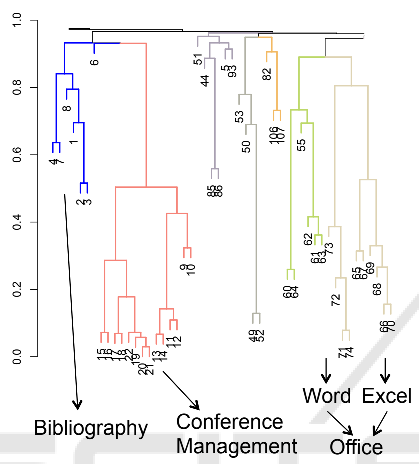

Figure 2: Excerpt of the dendrogram for domain clustering the ATL Zoo (Babur et al., 2016).

3 BACKGROUND: APACHE SPARK

Apache Spark[2] is an open-source distributed data processing engine (Zaharia et al., 2016), also used for Big Data Analytics. It offers a stack of technologies for both fundamental components such as cluster management and fault-tolerant distributed data storage (in the form of Resilient Distributed Datasets — RDDs), and for advanced ones such as streaming and distributed machine learning. Spark further provides rich APIs for various programming languages including Java, Python and Scala. It improves on the previously popular MapReduce (Dean and Ghemawat, 2008) computational paradigm (e.g. on Apache Hadoop[3]) with a more efficient distributed data and memory management system, leading to a higher comparative performance and scalability (Zaharia et al., 2016).

Spark typically operates with a master *driver* node, which coordinates several worker nodes (see Figure 3). Each worker node is allocated parallel tasks to process the specific parts of the distributed data (e.g. on the distributed file system). A central cache in each worker node can further improve efficiency by for instance maintaining some data in memory for faster access in multiple cores and for repeated/iterative tasks.

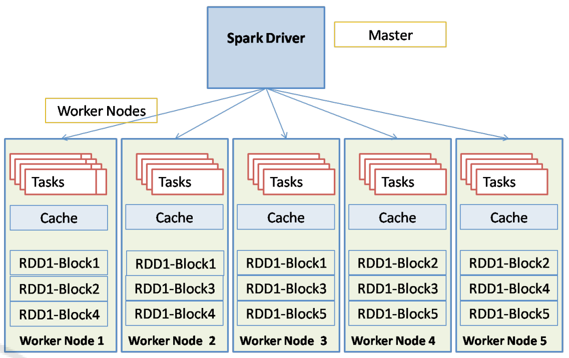

Figure 3: Overview of Spark architecture[4].

4 DISTRIBUTED VSM COMPUTATION

As outlined in Section 2, SAMOS relies on the calculation of a vector space by comparing the extracted features of each model against the set of all features (i.e. columns of the vector space). We can identify several components of this approach:

- extraction of desired features from the models, mapping each model to the feature set P_i,
- calculation of the set of all unique features (F, dimensions of the VSM),
- given F, comparing each feature in P_i against each feature in F to calculate a row or *vector* in the VSM.

The vector represents the model in the high-dimensional vector space, to be used for distance calculation and other statistical analyses. The bottleneck of this approach is the quadratic number of feature comparisons (in contrast with the previous steps which have linear complexity), which makes this the target for our parallelization effort to increase scalability. Note that the efficiency and scalability of the distance calculation for the resulting large sparse matrix is a relatively lesser problem and left as future work.

[1]http://web.emn.fr/x-info/atlanmod/index.php?title=Ecore
[2]https://spark.apache.org/
[3]http://hadoop.apache.org/

[4]http://spideropsnet.com/site1/blog/2014/12/09/igniting-the-spark/

On the other hand, thinking in the context of the Spark architecture (cf. Figure 3), we can map our approach to the distributed setting as follows:

- **data:** feature sets (P_i's) residing as distributed data (i.e. RDDs of model-feature pairs),
- **cache:** maximal feature set (F) precomputed and distributed to each worker node to be held in memory cache,
- **tasks:** feature comparison as the atomic unit of parallel execution.

While feature-to-feature comparison is the atomic unit for parallelization in this setting, for practical reasons we aim for a coarser granularity: we perform a single pass for each feature, comparing it with the maximal set. Each parallel task in return consists of (1) pair-wise comparing a feature p in P_i versus F and computing an intermediate vector, and (2) computing the final VSM vector for the corresponding model, e.g. via summing the intermediate ones (*frequency* setting of SAMOS (Babur and Cleophas, 2017)). To exemplify, a model consisting of m features is processed m-way and eventually integrated to calculate a single row of the VSM corresponding to that model.

Note that a great deal of the necessary functionality for distributed operation are provided by Spark: partitioning and distribution (shuffling) of the data, synchronisation of the tasks and the workflow, data collection and I/O, and so on. The necessary modifications for SAMOS were mostly wrapping the related building blocks (e.g. parsing and extraction, feature comparison) into parallel Spark RDD operations with minimal glue code around them.

5 PRELIMINARY RESULTS AND DISCUSSION

We performed some preliminary experiments for our technique. First of all, we used SURFSara [5], the computational infrastructure for ICT research in the Netherlands. SURFSara provides a Hadoop cluster with Spark support, which consists of 170 data/compute nodes with 1370 CPU-cores for parallel processing and a distributed file system with a capacity of 2.3 PB.

Next, as for SAMOS, we chose bigrams of attributed nodes (for the clone detection scenario (Babur, 2018)), as one of the more computationally intensive setting (e.g. compared with extracting simple word features for domain analysis (Babur et al., 2016)). As for the dataset, we mined GitHub for (1) a limited set of 250 Ecore[6] metamodels, and (2) a large set of 7312 Ecore metamodels (after exact duplicates and files smaller than 2KB removed). Table 1 shows some details on the sizes of the two datasets. A further SAMOS framework setting to mention is that we have turned off expensive NLP checks for semantic relatedness and synonymy for this preliminary experiment.

Normally, we have a simplistic *all or none* strategy for NLP-caching; for small datasets we iterate over all the model elements to compute and keep in memory the word-to-word similarity scores (i.e. full caching). For the distributed execution we have disabled this feature as we cannot fit the relevant data for very large model sets (the goal is to process tens of thousands of models) into the memory, so we completely disabled NLP-caching. As future work, we plan to investigate various more sophisticated approaches to caching to circumvent this issue.

Table 1: Description of the datasets: number of metamodels, total file size and number of model elements.

dataset	#models	file size	#model elem.
1	250	4.8MB	$\sim$50k
2	7312	133.6MB	$\sim$1 million

Performance for Dataset 1. On dataset 1, we ran the single-core local version of SAMOS, with and without NLP-caching, and the distributed version with 1, 10, 50, 100 and 250 and 500 executors without NLP-caching. Figure 4 depicts the results. For the single-core case, local execution has the best performance, especially with NLP-caching enabled. We have included the single-core distributed case to roughly assess the overhead: 17.1 hours (distributed) versus 13.8 hours (local). It is evident that as the number of executors increase, the performance increases as well, though with diminishing returns.

Performance for Dataset 2. As a bigger challenge for our approach, we made an attempt to run dataset 2 with the same (expensive) settings as above. We could argue going for more approximate, hence cheaper, settings (unigrams instead of bigrams, ignoring instead of including attributes, etc.) for such a large dataset but we performed this experiment in order to load-test and assess the limits of our technique. We successfully calculated the VSM using a total of $\sim$1500 executors (215 executors with 7 cores and 8GB memory each) processing the 5000-way partitioned data on SURFSara and obtained the resulting

[5] https://www.surf.nl/en/about-surf/subsidiaries/surfsara/

[6] https://www.eclipse.org/modeling/emf/

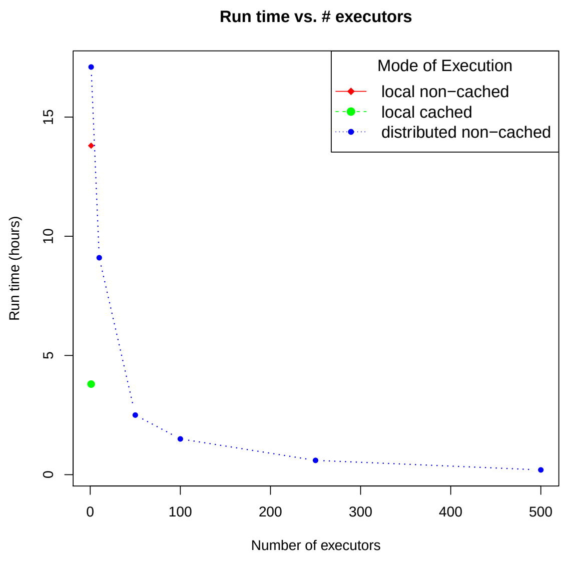

Figure 4: Performance for dataset 1.

VSM (5000-part on the distributed file system) in approximately 17.9 hours.

Discussion. The results indicate that the distributed execution mode has the potential to increase the applicability of SAMOS for large datasets. We can consider this as a first but important step towards the in-depth analysis of thousands of models for application scenarios such as repository mining (notably our ∼7k Ecore metamodel set from GitHub and the ∼93k Lindholmen UML dataset (Hebig et al., 2016)), large-scale clone detection and model evolution studies.

Given the preliminary nature of this work, there is certainly a lot of room for further optimization. These would involve not only optimizations with respect to the inner mechanisms of Apache Spark, but also improvements within SAMOS, which is itself a research prototype. Nevertheless, to our knowledge we do not know of another comparable model analytics approach or tool in the literature which is capable of such scalability.

Threats to Validity. In this work, we only deal with VSM calculation (which we assume to be the major bottleneck) and leave tackling the rest of the workflow—such as distance calculation and statistical analyses—as future work. We plan to proceed with parallel or scalable techniques for this as well. Another threat to validity is that our approach at this point has not been tried on larger scales, so it should be investigated how it performs with e.g. hundred thousands of models towards Big Data.

6 CONCLUSION AND FUTURE WORK

In this paper we present a novel approach for distributed model analytics. We have extended the SAMOS framework to operate on Apache Spark infrastructure and exploit its powerful distributed data storage and processing facilities. Using the SURFSara cluster, we have performed preliminary experiments using two sets of Ecore metamodels. On the smaller one, we have reported in detail the performance of the different execution modes and number of executors. For the larger one, we have tested the scalability of our approach in the case of nearly a million model elements.

There is a large volume of potential future work. The immediate next step would involve extending the SAMOS workflow with scalable, distributed techniques for distance calculation and statistical analyses. More advanced statistical analyses, including predictive and prescriptive ones, are among the notable targets for future work. Noting the benefits of NLP-caching, we also would like to investigate compromising NLP-caching strategies, applicable for large amounts of data. Moreover, we believe there is a lot of room for optimization for this approach, which can be considered in parallel to the other proposed items. A more in-depth discussion of the distributed vs. local execution in terms of performance gain, optimal number of executors, etc. would also be beneficial.

ACKNOWLEDGMENTS

This work is supported by the 4TU.NIRICT Research Community Funding on Model Management and Analytics in the Netherlands. We also would like to thank SURF, the collaborative ICT organisation for Dutch education and research, for providing us with a computational infrastructure and support.

REFERENCES

Babur, Ö. (2016). Statistical analysis of large sets of models. In *31th IEEE/ACM Int. Conf. on Automated Software Engineering*, pages 888–891.

Babur, Ö. (2018). Clone detection for ecore metamodels using n-grams. In *The 6th International Conference on Model-Driven Engineering and Software Development, to appear.*

Babur, Ö. and Cleophas, L. (2017). Using n-grams for the automated clustering of structural models. In *43rd Int. Conf. on Current Trends in Theory and Practice of Computer Science*, pages 510–524.

Babur, Ö., Cleophas, L., and van den Brand, M. (2016). Hierarchical clustering of metamodels for comparative analysis and visualization. In *Proc. of the 12th European Conf. on Modelling Foundations and Applications, 2016*, pages 3–18.

Babur, Ö., Cleophas, L., van den Brand, M., Tekinerdogan, B., and Aksit, M. (2017). Models, more models and then a lot more. In *Grand Challenges in Modeling, to appear*.

Benelallam, A., Gómez, A., Tisi, M., and Cabot, J. (2015). Distributed model-to-model transformation with ATL on MapReduce. In *Proc. of the 2015 ACM SIGPLAN Int. Conf. on Software Language Engineering*, pages 37–48. ACM.

Burgueño, L., Wimmer, M., and Vallecillo, A. (2016). Towards distributed model transformations with LinTra.

Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Hebig, R., Ho-Quang, T., Chaudron, M. R. V., Robles, G., and Fernndez, M. A. (2016). The quest for open source projects that use UML: mining GitHub. In *Proc. of MODELS 16*, pages 173–183. ACM.

Kolovos, D. S., Matragkas, N. D., Korkontzelos, I., Ananiadou, S., and Paige, R. F. (2015). Assessing the use of eclipse mde technologies in open-source software projects. In *OSS4MDE@ MoDELS*, pages 20–29.

LaValle, S., Lesser, E., Shockley, R., Hopkins, M. S., and Kruschwitz, N. (2011). Big data, analytics and the path from insights to value. *MIT sloan management review*, 52(2):21.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Melnik, S., Garcia-Molina, H., and Rahm, E. (2002). Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. of the 18th Int. Conf. on Data Engineering*, pages 117–128. IEEE.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., et al. (2016). Apache Spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

Zikopoulos, P., Eaton, C., et al. (2011). *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media.