LEARNING LOAD BALANCING FOR SIMULATION IN HETEROGENEOUS SYSTEMS

Irina Bernst, Christof Kaufmann and Jörg Frochte
Dept. of Electrical Engineering and Computer Science
Bochum University of Applied Science
42579 Heiligenhaus, Germany

ABSTRACT

Distributed computing is an important key technology for simulation technologies like finite elements. Dedicated homogeneous parallel servers are well-scaling but very expensive solutions, while heterogeneous systems are a challenge for efficient computing. The work we present deals with the problem to provide a learning assistance system for load balancing in FEM simulations. Our method uses a two-stage architecture to minimize additional computational costs. The approach does neither require the labeled data nor a teacher for the initial setup and can improve itself unsupervised. For the FEM application case we introduce a novel feature set and perform an evaluation for several problem sets.

KEYWORDS

Machine Learning, Data Mining, Load Balancing, Assistant System, Distributed Computing, FEM

1. INTRODUCTION

Machine learning and data mining techniques have proven to be of great practical value in many application areas. However, there are fields, which have not got much attention with respect to machine learning. The field of simulation in engineering software is one of these fields, and many methods could be improved by these kinds of techniques; especially when general challenges like accuracy and stability meet requirements concerning security. The presented work connects among others to the work of Burrows et al. (2013), Bernst et al. (2014) dealing with the problem set described in Frochte et al. (2014). This means, that we investigate approaches to distribute parts of a simulation using the finite element method (FEM), see e.g. Brenner and Scott (2008), in a heterogeneous distributed system. FEM tends to involve a huge number of unknowns, especially in three dimensions. The power of a single computer is often no longer sufficient to solve the resulting equations. In this case, parallel computing with domain decomposition is one of the most successful strategies to solve these equations efficiently. The FEM simulation is partitioned using an overlapping Schwarz method, which belongs to the class of domain decomposition methods, see e.g. Toselli and Widlund (2014). Thinking of a homogeneous blade server this task might include some challenges, but it is more or less straight forward. The mesh that is used for the FEM approach is uniformly divided in as many parts as computational units should be used, see e.g. Nikishkov (2007). This can these days be performed quite efficiently by freely available software, like e.g. METIS, see e.g. Karypis and Kumar (1998), which we use for partitioning the mesh. If the environment is heterogeneous, e.g. motivated by secure cloud simulation as described by Frochte et al. (2014) or by the need to combine different server architectures in a research institute to use the provided equipment as best as possible, the complexity increases rapidly. Therefore, we will consider a distributed system as shown schematically in Figure 1 throughout this paper. The label cloud is motivated by the secure simulation distributed use case and would in other cases just mean different server architectures or clusters. Because the problem consists of distributing a computational task, it makes sense to look for two parameters. One is the waiting time, which is associated with convenience of the user and his wish to achieve results quickly. The other is the CPU time, which in a way represents the energy cost as a second optimization variable into account. This is very important considering aspects of cloud computing and green IT.

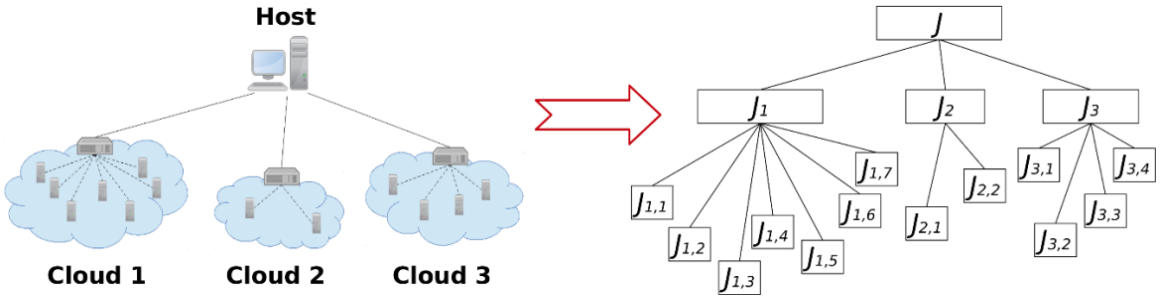

Figure 1. Hierarchical distribution of the job J on a heterogeneous system with three homogeneous sub-systems

As displayed in Figure 1, an important role in the suggested setting plays the host computer, which triggers the task and manages the computation. It submits the relevant subset of the simulation data to a remote master J_i, which again spreads the necessary parts to the computation nodes $J_{i,k}$. The result is a communication cascade, in which one subsystem solves its part of the problem using distributed computing causing network communication between all $J_{i,k}$, with a fixed i, managed by $J_{i.}$. After this is done J_i communicates with J. Because the used solving technology is iterative, this is done multiple times. Most of the communication is done in each subsystem, which makes sense, because the network connection from host J to J_i is generally weaker than the one between the computation nodes. Because of the heterogeneity of the computer system, which means firstly non-equal computation power per computation node across different server-systems and secondly non-equal connection speed between the host computer and the sub-systems, an efficient distribution of the tasks is a non-trivial challenge. Because the problem set shown in Figure 1 is of wide scope, we make some reasonable assumptions. We assume that sub-environments are homogeneous among themselves. This means in every sub-environment, e.g. a cloud architecture or a standard blade server, we have the same connection speed between every node, and every node is equal concerning the computation speed.

To keep the complexity of such a task distribution manageable for a typical end-user, we use machine learning and data mining approaches in the sense of simulation data mining as it has successfully been applied to support civil engineers at design decisions of a bridge model by Burrows et al. (2011). Of course, our problem set is not related to design aspects but to load balancing issues. Load balancing is a technique to optimize the distribution of sub-problems; for a general insight see e.g. Kameda et al. (1997). Because of the complexity in heterogeneous network environments load balancing is a research field in which heuristic approaches, see e.g. Doulamis et al. (2014), and similar strategies are commonly applied. Our contributions in this paper are summarized as follows: (1) As main contribution we propose a machine learning approach for automating the distribution of FEM simulations in heterogeneous environments. (2) To do this we develop and evaluate a working feature set. We create a unique problem set of interest to both novice and expert users. (3) We devise a machine learning framework for this task including a training corpus and an appropriate performance measure.

The remainder of this paper is organized as follows: In Section 2 we give an overview of the suggested architecture of the approach and provide a more detailed task description. In Section 3 we develop the feature set and the learning approach including the necessary background on partial differential equations and domain decomposition, if needed. In Section 4 we present the results of our method on different example problem sets. Finally in Section 5 we finish with concluding remarks.

2. GENERAL LEARNING APPROACH AND TASK DESCRIPTION

FEM is used for a wide range of problems. Different FEM models like e.g. a model for linear elasticity or one for a laminar flow behaves differently on the same mesh concerning domain decomposition. Problem classes that behave very differently need a different numerical treatment anyway, and thus just problems of the same class can share the same learning data base. The workflow in Figure *2* concentrates as example on linear elasticity problems modeled with the Lamé equation, also known as displacement formulation.

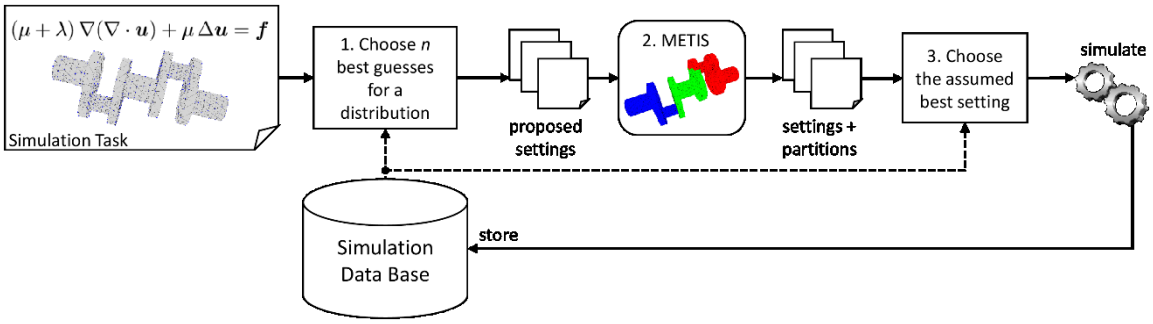

Figure 2. Workflow of the learning method

Very important to understand the used approach is the fact that we have in a way just unlabeled data. Our data base consists of features associated with the waiting time and CPU time of the performed simulation. What our data base does not contain is information about the best choice in a given situation. This is because it is in general a very tricky task to tell whether the setting has been optimal – in most cases there might not be such a thing as a unique optimum. The key idea is to learn the waiting time and the CPU time instead and to use the resulting functions to find good settings.

To achieve this, we argue for a two-stage approach; the first stage uses only a few features, while the second stage uses the full feature set. The reason is that it is quite cheap to evaluate the learned function, but if one needs a partitioning to evaluate the full feature set the call of a partition tool like METIS is not that cheap. To identify the features for the machine learning task in learning workflow we need a more formal description of an exemplary case related to the scenario in Figure 1.

Let J be a computational task (or job) that can be scheduled in parallel. We want to distribute it on a heterogeneous system with a number of major subsystems, like e.g. cloud hardware provider. For example, we consider a system with three subsystems as displayed in Figure 1. Each of them provides a maximal number of computation nodes: c_1, c_2, c_3. Let n_1, n_2, n_3 be the numbers of computation nodes that we use for the computation and which are limited by the maximal number of computation nodes c_1, c_2, c_3.

The job J can be decomposed into maximal three main-jobs J_1, J_2, J_3 (see Figure 1). The size of these jobs concerning the degrees of freedom might differ. These main-jobs can again be decomposed into n_i sub-jobs in the second step up to the number of available computation nodes c_i $(i=1...3)$ in this system. The result of this approach is a job hierarchy.

At the beginning of the choice n_1, n_2, n_3 stands the question, how many computation nodes, which simply means $n_1+n_2+n_3$, the user in general would like to access. This is limited by the number of computation nodes the system provides him with. Because of the job hierarchy it makes sense in the next step to not directly try to answer the question, how we would like to choose n_1, n_2, n_3, but first to answer, how many main-jobs should be in the first level of the job hierarchy and how they should be dimensioned. The last step is how many sub-jobs of each of these main-jobs should be generated and executed. For a more detailed look we will now continue with the feature selection.

3. FEATURE SELECTION AND LEARNING APPROACH

To start learning our data base must contain information about the hardware, on which the simulation has been carried out. Because our data base may grow slowly, it makes sense to restrict us to as few as possible hardware features. As Figure 1 suggests a minimal description consists of

1. used computation nodes n_1, n_2, n_3.
2. computation node speed (computation power) in every subsystem s_1, s_2, s_3
3. data transfer rate from J (host) to J_i (e.g. cloud provider): d_{h1}, d_{h2}, d_{h3}
4. data transfer rate in homogeneous sub-environment d_{c1}, d_{c2}, d_{c3}.

Therefore, we end up with 12 features for the hardware topology (HF).

The next step is connected to the first task shown in Figure *2*. We have just the model to be simulated without any partitioning. The model consists of the discretized PDE including the right hand side and coefficients, e.g. representing material properties, mesh, the boundary conditions (Dirichlet, Neumann etc.).

As mentioned above the suggested trained system is only suitable for a narrow class of problems. Therefore, the PDE itself is not a feature; consider it as a member of the mentioned problem class for a single learning assistant system. This of course does not cover the right hand side and the coefficients. In Burrows et al. (2013) the aspect of coefficients were discussed. To keep the number of features reasonable we concentrate on the features based on the geometric mesh, which in our case consists of vertices, edges, facets and tetrahedral elements. There are other mesh types for three dimensional problems like hexagonal meshes, but the results and features can easily be transferred to them. As features we use:

1. number of vertices in the mesh n_v
2. volume ratio of minimal and maximal volume of tetrahedrons in the mesh, denoted with r
3. number of boundary vertices n_{bv}

The number of vertices n_v as features correlates with the size of the problem – degrees of freedom – which is the most important aspect. It makes no sense to divide small problems below some limit. The volume ratio r is one of the important aspects, which influence the condition of the problem and its sub problems, see e.g. Brenner and Scott (2008) or Toselli and Widlund (2014). The condition number in turn will influence the behavior of the linear solvers used by the computation nodes. The last feature n_{bv} is related to both. For Dirichlet boundary conditions e.g. for all boundary elements the values are given, so they reduce the degrees of freedom and improve the condition. We denote the selected features as global mesh features (GMF).

This leads to 12 hardware related features and 3 model related features for the first step. Our learning approach proposes to predict the computational cost t_c and the waiting time t_w by a learned function. To do this we suggest multilayer feedforward artificial neural networks (ANN). A design decision in this context is the question to use one ANN with two outputs, one for computational cost and one for waiting time, or to use two divided but then in general smaller ANNs. In our test it turned out that we achieved better results using two divided networks. This might be related to the number of data sets in the training set and may change, if a huge amount of data is available. Our solution can be denoted as follows:

$$t_w = \mathrm{ANN}_1^{waiting}(\mathrm{HF}, \mathrm{GMF})$$
$$t_c = \mathrm{ANN}_1^{computation}(\mathrm{HF}, \mathrm{GMF})$$

By choosing a weight α the user can fix the relation of t_w and t_c for the optimization problem. The goal is to find the minimum of

$$f(n_1, n_2, n_3) = \alpha \frac{t_w}{c_{wait}} + (1-\alpha)\frac{t_c}{c_{comp}}.$$

This is a problem, which just depends on the chosen number of used nodes, because in our scenario the rest of the features are fixed. In our example with three systems it is a three-dimensional problem. Which technique is used for optimization depends on the size of the search space defined by the total number of used nodes. In many applications it is possible to evaluate all possible combinations and choose the best one; otherwise techniques like the Hill climbing should be used. The scaling coefficients c_{wait} and c_{comp} can be chosen to balance waiting and computation time because in general for distributed problems the computation time is much bigger than the waiting time.

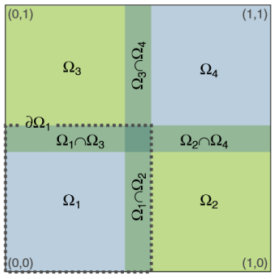

Figure 3. Overlapping Partitions

Using the approach described above we end up with some candidates, this means not just to pick the minimum but a small subset of the best candidates. If we evaluate all variations, we choose the m smallest

ones; if one uses Hill climbing, one may choose a whole region near the minimum or a union of results of Hill climbing iterations started from different initial points and maybe stopped in local minima.

So, for the next step we build a partition of the mesh for the resulting m combinations using METIS. Because of the hierarchical structure of the problem set this is performed in two steps: In the first step three weights, w_1, w_2, w_3 are chosen. These control how many percent of the mesh every partition should consist of. Therefore, these weights represent the sizes of the main-jobs J_i in the first hierarchy level. However, computation is mainly performed on the computation nodes in the second level of the job hierarchy. Hence, the main-jobs are further partitioned into sub-jobs $J_{i,k}$. The corresponding sub-partitions below a main-partition are equally weighted, since all used nodes within one cluster are equal. $w_i = 0$ is interpreted as turn off the specific subsystem. As Figure *2* suggests for this second step, there are additional features available, because now we know more details about how this job will be partitioned. A new aspect is the size of the overlaps illustrated in Figure 3. This is very important for the convergent speed of the domain decomposition methods. To improve convergence and stability, we let the host J and the local master J_i perform a few iterations of a GMRES algorithm, see e.g. Greenbaum (1987), which leads to less network traffic, particularly between the J and J_i, which is the most expensive one. Nevertheless, on the one hand bigger overlap leads to fewer iterations. On the other hand more parts of the problem are solved redundantly and increase the problem size artificially. Beyond this, we have the effect that concerning the network traffic small overlaps are desirable, because this is the data that has to be transferred in every iteration step. Thus, we have here a non-linear effect that is mainly associated with the number of vertices, which are exchanged between neighbor domains. Therefore, we need to modify the global features used for the last step, which leads to the following feature set per main partition, which means on the first level for all i partitions:

1. np_i, number of vertices in the partition
2. rp_i, volume ratio of minimal and maximal volume of tetrahedrons in the partition
3. op_i, ratio of number of overlapping vertices to the partition size.

Because the number of features would increase dramatically, the equally-sized sub-partitions are not considered, and nevertheless their effect is quite deterministic and static using METIS, if one does not optimize the overlapping size as done by Burrows et al. (2013) but just keep it fixed as we do it here. If one looks at the features on the second level, one may argue, if concerning some aspects there is less information compared to the first level. One has to keep in mind that n_v correlates with np_i and op_i. The global information n_{bv} is really missing but for the solution of the sub problems of less importance. With a fixed rule, how to dimension the overlap, the degrees of freedom on the boundaries of the partitions correlate with op_i . For dynamic chosen overlap sizes this information might play a bigger role. We denote the set of all of these features for all main partition as partition mesh features (PMF). Using this together with the hardware features (HF) this leads us to the following approach for the next two ANNs, which are again multilayer feedforward artificial neural networks:

$$t_w = ANN_2^{waiting}(HF, PMF)$$
$$t_c = ANN_2^{computation}(HF, PMF)$$

This network ANN_2 has the potential to give a more accurate answer compared to ANN_1 but at the prize of an increased number of features – in the case of three main-partitions we have 9 PMF instead of 3 GMF – and it is preceded by a partitioning procedure to extract the features from the main-partitions. The first aspect leads to the demand of more data for the training and the last makes it more expensive. To emphasize this aspect: this is the main reason for using a two-stage approach. The features for ANN_1 are available just by analyzing the FEM model geometry and the evaluation is cheap. Therefore, it is the first filter selecting a few candidates. For these candidates the partitioning of the main-jobs J_i is performed by METIS, which leads to much higher additional costs but only for these few new candidates. With this information for these candidates the ANN_2 picks up the most promising distribution. The simulation is then performed, distributed accordingly. When the simulation has finished, we can use the corresponding features with the measured times and add it to our data base to improve the neuronal networks. Because the approach is not based on labeled data, this leads to a system improving itself in an unsupervised way.

4. RESULTS

As test problem we choose the linear elasticity model using the Lamé equation with a homogeneous isotropic material parameter and Dirichlet boundary conditions. For training purpose this has been simulated for four three-dimensional objects (geometries), shown in Figure 4, with a mesh size from 250,000 vertices up to 400,000. Because the Lamé equation computes the displacement vector, this means up to 1,200,000 degrees of freedom. The four objects refer to prototypes, like a full convex geometry (sphere), a geometry with a hole (Torus), one with a re-entrant angle (L-shape) and the simple cube.

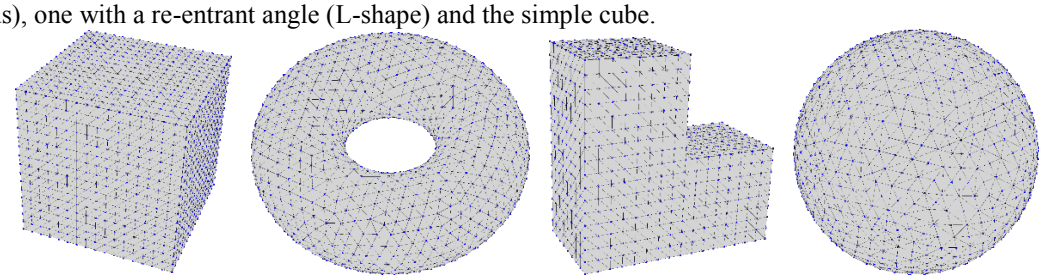

Figure 4. Training objects (with coarse meshes for the visualization)

We provided 564 FEM simulations for different hardware topologies with two and three server systems to prepare the training set. They contain different CPUs, in this case 1) Xeon E5645 @ 2.4GHz, 6-Core, 2) Xeon E5-2660 @2,2GHz 8-Core, 3) Xeon E5-2640 @ 2,5GHz 6-Core. These are constant as well as the connection speed between the nodes in one server system, which differ from server system to server system between fast Ethernet (100 Mbit/s) and Gigabit Ethernet (10 Gbit/s). The connection speed from the host server to each of the server systems has been varied between 10 Mbit/s, 100 Mbit/s and 1 Gbit/s. In this test problem we assume that the user constantly wishes to use 16 computational nodes and at least two server systems, which means clouds in the scenario discussed by Frochte et al. (2014). With the constraint to 16 computational nodes we can reduce the number of features by one, because we have $n_3 = 16 - n_1 - n_2$.

We used the standard approach, see e.g. Marsland (2015), Chapter 2.2, to control the training of the ANNs by dividing our data into three subsets: training, validation and test, as Figure 5 shows.

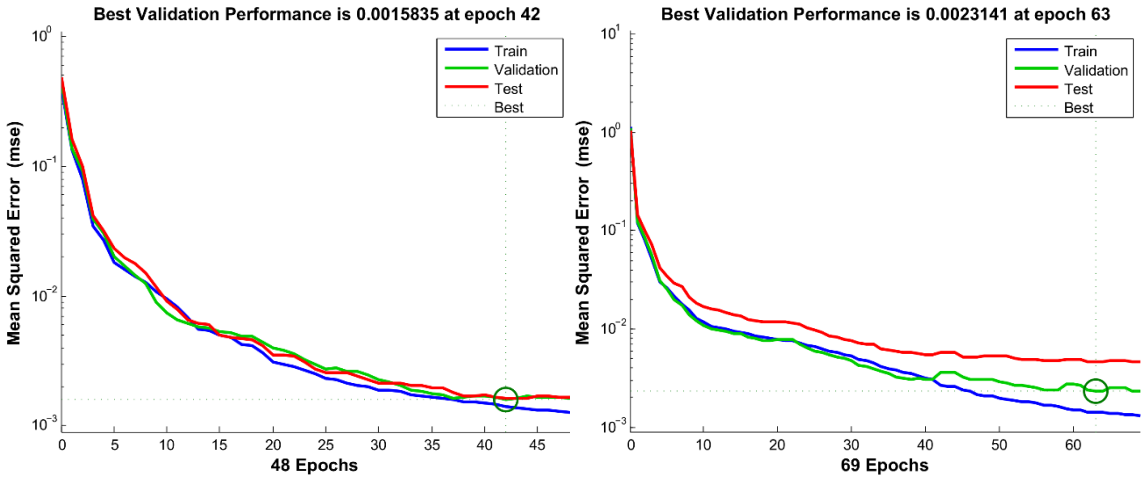

Figure 5. Training of ANN_1 (left) and ANN_2 (right)

As mentioned above the ANNs differ concerning their features and as expected they differ concerning their size as well. In our experiments it turned out that the following choices produce reasonable results: ANN_1 for the computation time has 23 neurons and 36 for the waiting time. ANN_2 has for the waiting time 75 neurons and 45 for the computation time. One can sum up that the approximation of the waiting time is more complex than the computation time and that ANN_2 is more complex than ANN_1.

While the difference between ANN_1 and ANN_2 is expected, because ANN_2 has more information to consider, the difference between waiting and computational time might need a further discussion. The communication between host and server system is performed in a network system, which is used

unexclusively. This means traffic caused by other users tends to add some additional statistical errors to our measurement.

All the results above, e.g. shown in Figure 5, were achieved on a validation and test set, that is not used for the training. In this case the suggested approach works very well, but the data set just contains geometries, that are used for training. The meshes may differ because of global mesh refinements, but they represent the same geometries with more or less the same ratio between boundaries and inner vertices etc.

For testing the load balancing approach we additionally used three new objects, see Figure 6, which were not included in the initial training, test or validation set above. Ellipsoid is related but not equal to sphere, T-Shape is in a way related to L-Shape and the cube. The hardest geometry is the crankshaft, which is totally different to the training geometries.

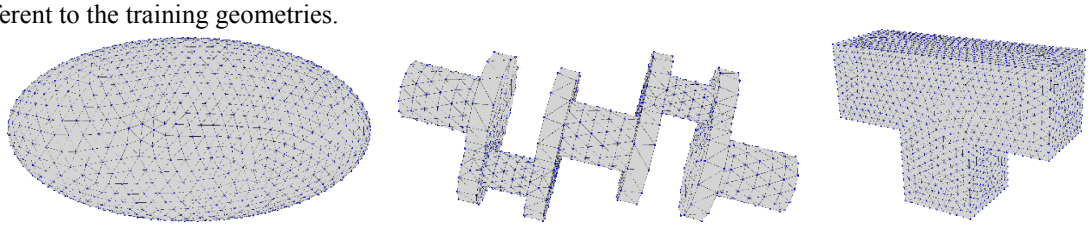

Figure 6. Test objects (with coarse meshes for the visualization)

We tested the quality of predictions of ANNs for foreign objects and geometries on these three. As one can see from Figure 5 the ANN_1 works quite well on our initial data base with known objects. If ANN_1 works so well, does it really make sense to apply a second more expensive step using ANN_2 afterwards? Yes, it makes sense, because ANN_1 loses some of its quality on foreign objects. In our approach the main job of ANN_1 is to work as a filter before using ANN_2.

To do this ANN_1 must mainly keep the relation between the results. If a configuration is better than another in real life, the learned function should keep this relationship. The absolute values are less important. To illustrate effects we picked the shaft as the most demanding geometry. Figure 7 shows the results of 5 randomly picked data base entries with three different values for α. So, the first plot is the use case for optimization of computation time only, the last for waiting time only, and $\alpha = 0.5$ is a mixed scenario. The data is ordered by the value of ANN_1. A theoretical optimal result would be that both graphs are monotonically increasing. Again this figure illustrates that the computation time is easier to predict than the waiting time. For $\alpha = 0$ both are monotonically increasing, in the two other scenarios it is not. So, it turned out in our tests that ANN_1 is able to work as a reasonable filter, but as proposed it is necessary to attach a second step with ANN_2 to increase the probability to pick the best one of the set collected by ANN_1.

The results for the three test objects of Figure 6 are shown in the regression plot in Figure 8. They illustrate that the learned functions correlate with the real world measurements we performed for comparison. Again we see, that it is easier for us to predict the computation time than the waiting time. We assume, that with a bigger data base the statistical effects concerning the waiting time will be eliminated more and more. Concerning the waiting time the results suggest that for new objects not always the best solution would be picked up. This corresponds with our tests, in which always a reasonable approach is suggested by the assistance system, but sometimes with expertise knowledge and experience we could provide a better solution.

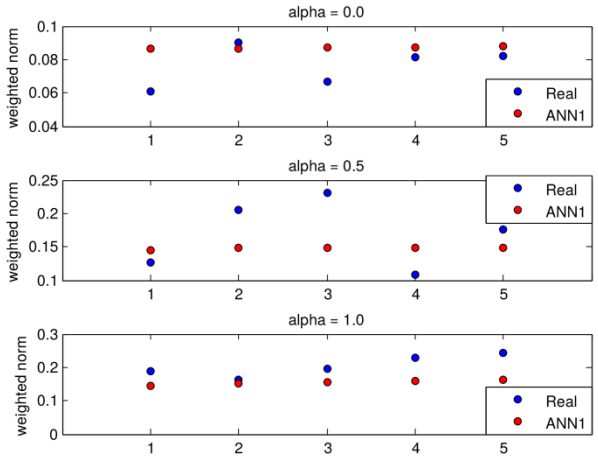

Figure 7. Performance of ANN_1 keeping relation the crankshaft geometry

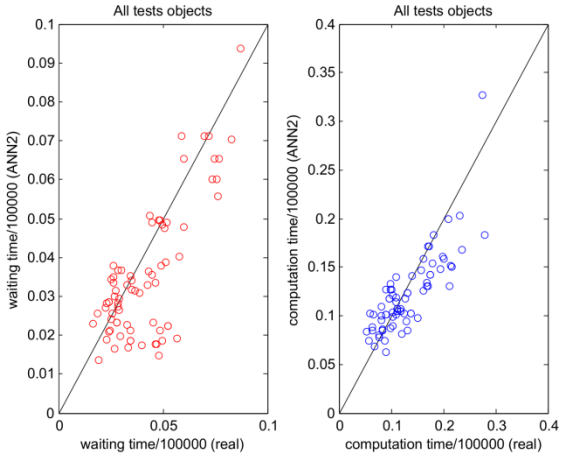

Figure 8. Comparison of ANN_2 predictions vs. real on simulation times (scaled) for foreign test objects

5. CONCLUSION

For a long time, see e.g. Stein and Curatolo (1998), it became clear that a key challenge in complex scenarios in simulation requires assistant systems for end-users like engineers. Assistant systems based on a static expert system cannot keep up with today's demands, and self-learning and self-improving systems like e.g. S. Burrows et al, (2011) are more promising. In this paper, we proposed a machine learning method, which is able to provide assistance in a highly non-linear and complex load balancing task for distributed computing of simulations. To achieve this, our method introduced a novel feature set with a two-stage architecture to minimize additional computational costs. The presented work shows several avenues of future work that we expect to offer further improvements for this learning assistant system. An important aspect will be the combination with features concerning material properties and the right hand side of the PDE.

ACKNOWLEDGEMENT

The authors would like to thank Patrick Bouillon for the support & administration of the server systems. This work is supported by the BMBF (Federal Ministry of Education and Research, Germany) under grant 03FH01812. The homepage of the project is located under www.simcloud.eu.

REFERENCES

I. Bernst et al, 2014, An approach for load balancing for simulation in heterogeneous distributed systems using simulation data mining. *Proceedings of the 11th Intern. Conference Applied Computing.* Porto, Portugal, pp. 254–258.

S. Burrows et al, 2011, Simulation data mining for supporting bridge design. *Proceedings of the Ninth Australasian Data Mining Conference.* Australia, pp. 163-170.

S. Burrows et al, 2013. Learning overlap optimization for domain decomposition methods. *7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 13)*, pp. 438–449.

T. F. Brady and E. Yellig, 2005, Simulation data mining: A new form of computer simulation output. *Proceedings of the Thirty-Seventh Winter Simulation Conference*. Orlando, Florida.

S. Brenner and L. Scott, 2008, *The mathematical theory of finite element method, 3rd ed.* Springer-Verlag, New York-Berlin-Heidelberg.

N. Doulamis et al., 2014, Resource Selection for Tasks with Time Requirements Using Spectral Clustering, *IEEE Trans. on Computers*, Vol. 63, No.2

J. Frochte et al, 2014, An approach for secure cloud computing for FEM simulation. *Proceedings of the 11th International Conference Applied Computing*. Porto, Portugal, pp. 234–238.

Anne Greenbaum, 1987, *Iterative Methods for Solving Linear Systems*, 1 edition, SIAM

H. Kameda et al, 1997. *Optimal Load Balancing in Distributed Computer Systems*. Springer-Verlag, London, UK.

G. Karypis and V. Kumar, 1998, A fast and high quality multilevel scheme for partitioning irregular graphs. *Journal on Scientific Computing.* Vol. 20, No. 1, pp. 359–392.

S. Marsland, 2015, *Machine learning - An Algorithmic Perspective*, CRC Press a Chapman& Hall book, Boca Raton.

G. P. Nikishkov, 2007, *Basics of the domain decomposition method for finite element analysis*. Saxe-Coburg Publications, Kippen, UK.

Stein, B. and Curatolo, D., 1998. Selection of Numerical Methods in Specific Simulation Applications, *Proceedings of the Eleventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems.* Castellon, Spain, 918-927

A. Toselli and O. Widlund, 2004, *Domain Decomposition Methods – Algorithms and Theory, 1st ed.* Springer-Verlag, Leipzig, Germany.