Parametric Design and Performance Analysis of a Decoupled Service-Oriented Prediction Framework based on embedded numerical software

George Kousiouris, Andreas Menychtas, Dimosthenis Kyriazis, Kleopatra Konstanteli, Spyridon Gogouvitis, Gregory Katsaros and Theodora Varvarigou

Dept .of Electrical and Computer Engineering, National Technical University of Athens,

9, Heroon Polytechniou Str, 15773 Athens, Greece

E-mail: {gkousiou, ameny, dimos, kkonst, spyrosg, gkats, doravarv}@mail.ntua.gr

ABSTRACT

In modern utility computing infrastructures, like Grids and Clouds, one of the significant actions of a Service Provider is to predict the resources needed by the services included in its platform in an automated fashion for service provisioning optimization. Furthermore, a variety of software toolkits exist that implement an extended set of algorithms applicable to workload forecasting. However, their automated use as services in the distributed computing paradigm includes a number of design and implementation challenges. In this paper, a decoupled framework is presented, for taking advantage of software like GNU Octave in the process of creating and using prediction models during the service lifecycle of a SOI. A performance analysis of the framework is also conducted. In this context, a methodology for creating parametric or gearbox services with multiple modes of operations based on the execution conditions is portrayed and is applied to transform the aforementioned service framework in order to optimize service performance. A new estimation algorithm is introduced, that creates performance rules of applications as black boxes, through the creation and usage of genetically optimized artificial neural networks. Through this combination, the critical parameters of the networks are decided through an evolutionary iterative process.

Keywords: Service Oriented Infrastructures, Performance Estimation, Quality of Service, Performance analysis, Workload Forecasting, Artificial Neural Networks, Genetic Algorithms

1. INTRODUCTION

In the recent years, modern IT infrastructures have shifted towards a more service-centric paradigm, boosted by technologies such as Service Oriented Architectures (SOAs) [1] and Cloud Computing [2]. In this context, everything tends to be offered as a Service, whether it is infrastructures, platforms or software, following the Cloud SPI (Software-Platform-Infrastructure) model [3].

The operation of these Service Oriented Infrastructures (SOIs) is managed through Service Level Agreements (SLAs [4]) that define the role of the involved parties in the transaction and the clauses of the latter. Such details may be the service that will be offered, the Quality of Service (QoS) parameters of that service as well as legal issues such as compensation in case these QoS levels are not met by the provider. QoS provisioning and management in general is a very important and challenging field of research in distributed environments. In [5] an approach is presented for managing QoS in SOA that focuses on the categorization of the quality characteristics and uses a specific XML-based language for applications and service providers to express QoS requirements and contracts.

One critical task for the service providers prior to signing the specific SLA refers to an estimation of the resources needed to fulfill the user requirements for every application that is offered as a service from their platform. To this direction, workload forecasting and modeling are necessary for the SP in order to determine the required resources for fulfilling application QoS requirements while at the same time maximizing resource utilization, in a private or public Cloud infrastructure. Up to now, a number of different approaches have been implemented in order to tackle with this issue, including graph based ([6],[8] and [9]), probabilistic ([10]) and machine-learning ones ([9][11]). In general, a model is built in order to predict the output from the input, usually by taking into account suitable parameters or by analyzing past historical data. What is lacking is a specific framework for implementing such methods, especially in a service-oriented environment. Useful and widely adopted tools, such as GNU Octave ([24]) make the implementation of such methods much easier than with standard programming languages such as Java or C++. Octave is an open source, community-driven high level numerical computation software, very similar to Matlab. Through this tool, advanced estimation methods can be written in a script format. However the automated adaptation of software like Octave on distributed environments has not reached a suitable maturity level. A very thorough analysis on various schemes of this sort can be found in [21].

The major aim of this paper is to present a flexible, generic and robust service framework for achieving workload forecasting in SOIs. The contribution has been centered around 3 areas:

a.The decoupled framework enabling the incorporation of estimation methods as pluggable octave scripts and the performance analysis of the framework. This framework (which cooperates with the general framework described in [27]) is able to implement whatever estimation method as a pluggable Octave script.

b.The novel application workload forecasting method. An innovative, computationally demanding estimation method is introduced, based on genetically optimized Artificial Neural

Networks (ANNs). This black box approach is completely designed through a Genetic Algorithm for automation and optimization purposes and can reduce the dependency from the knowledge needed for the internal functionality of services.

c.The methodology for transforming typical services to parametric services. Parametric services are defined as the ones capable to change or adjust during runtime the underlying implementation based on the usage conditions of their execution. A 5-step process is introduced, in order to enable the design and implementation of this type of services. To this end, a detailed **performance analysis** is conducted on the behavior of the service framework. **Parametric modes of operation** are investigated, based on the request arrival rates. Depending on the latter, the implementation can change in order to optimize the performance of the server.

The remainder of the paper is structured as follows: Section 2 presents related work in the field of service oriented performance estimation and enablement of mathematical software through Web/Grid services. The next section describes in detail the functionality of the service along with the methodology for creating parametric services whereas Section 4 details the estimation algorithm based on genetically optimized ANNs. In Section 5 we present a case study based on the application of encoding raw video in MPEG4 format. We demonstrate and evaluate the operation of the implemented mechanism. In Section 6 the parametric methodology is applied to the service framework in a stepwise manner. Finally, Section 7 concludes the paper with a discussion on future research and potentials for the current study.

2. RELATED WORK

A number of versatile approaches can be compared to the work described in this paper. They either relate to service oriented performance prediction mechanisms or to offerings of mathematical software in distributed environments. The Network Weather Service [18] includes a set of forecasters and sensors in a distributed environment, in order to retrieve data dynamically and apply them to modules that implement advanced prediction techniques. While ahead of its time, this implementation was heavily based on C language, thus making the implementation of the used techniques more difficult and time consuming.

A framework for incorporating QoS in Grid applications is discussed in [7] .In this paper, a performance model to estimate the response time and a pricing model for determining the price of a job execution are used. In order to determine whether the client's QoS constraints can be fulfilled, for each QoS parameter a corresponding model has to be in place, either derived from analytical modelling or historical data. GridSolve [20] emphasizes on the ease-of-use and includes resource monitoring, scheduling and service-level fault-tolerance. In addition to providing Fortran and C clients, GridSolve enables scientific computing environments (such as Matlab) to be used as clients, so domain scientists can use Grid resources from within their preferred environments. This effort attempts to go the other way round and "gridify" applications such as Matlab and Octave, in order for them to be able to exploit distributed resources. This is also done for a variety of computer algebra packages in [22]. These works may be considered mainly as a successful effort to improve the performance of software like Octave through distributed infrastructures. Other works such as [14] aim mostly at offering licensed software through Cloud infrastructures mainly for licensing purposes and on a pay-per-use basis.

In [19], an architecture very similar to the design presented here is followed, in order to provide Maple-based mathematical web services. While very promising, this approach is not incorporated in a Grid/Cloud environment and focuses mainly on the offering of mathematical services. The authors of [25] demonstrate the use of Octave for creating performance models for network interfaces based on queueing models, while in [26] a limited part of Octave is offered through Web Services interfaces for use of the signal processing functions by mobile phones. A very interesting work is presented in [12]. In this approach, the main attention is given on the differences SOA-based implementations require with regard to the regular software development processes and what modifications must be made to the latter in order to be more effective. One of these aspects is the performance analysis that is required for the service oriented implementation. In [35], an online system based on closed loop feedback from application KPIs is used in order to drive the resource management. This is an ideal case when SLAs are not needed for reserving a priori the resources, but the application can adjust its own reservations. In [36], a performance framework based on benchmarking and statistical inference is introduced, for message-oriented services. A thorough survey on state of the art performance analysis for component systems can be found in [13].

An interesting framework for the performance description of composite services appears in [39]. This work contain contributions in two areas. Initially, a description modeling framework for performance is presented, exploiting P-WSDL and UML models. In our view our work is complementary to this aspect of the framework. Our major goal is to create the actual performance rules/models (e.g. algebraic rules correlating application characteristics and used resources) that may be incorporated in a description modeling solution like P-WSDL and not the description framework itself. Furthermore, the work uses prediction mechanisms in the form of LQN (Layered Queueing Networks). While more precise, any type of analytical modeling like LQN involves more human-driven modeling of each specific service. Thus the application developer (or the performance expert on the SP side) would need more time and internal application knowledge to derive the actual performance model/rule, especially when the corresponding software components do not correlate to existing component templates of the LQN. Other related works ([40],[41]) follow similar methodology, focusing on reliability and performability (combination of performance and reliability metrics) of composite services respectively.

In [45], a time series analysis approach is presented, for identifying system changes and event identification that may lead to enhanced management of resources in a high variability context. Our approach mainly focuses on estimating the workload anticipation, as this is derived from the translated high level SLA parameters and can be used in a complementary way with the aforementioned approach, in order to have a failure identification subsystem. Also this approach can be integrated in the service framework proposed in this paper if transformed to a GNU Octave script version. In [46], an approach for behavioural changes detection is presented, which significantly enhances the time needed and computational complexity of the identification method, through successfully transforming the problem to a behavioural classification one. However this approach is mainly for private cloud scenarios, and for any context where the system manager has full control on the resources assigned to different services. In our case, we envisage that the SP that utilizes this framework is not responsible for the infrastructure itself, which belongs to the IP, according to the SPI model entities differentiation. Thus the SP does not have the sufficient level of information needed for this type of management after deploying the application to external public Cloud providers.

The design presented in this paper aims at providing the service oriented framework for performance prediction methods in a generic way. Our goal is three-fold. Initially, to provide a service oriented version of a generic tool like GNU Octave, in which the performance expert of the SP may utilize to write his/her own decoupled algorithm of choice. Furthermore, the designed service can be used in real life automated environments, e.g. during the SLA negotiation process between a Service Provider and the client. The performance analysis results can be used in any similar approach, incorporating specialized software at the bottom of a service-oriented approach. Then we propose a GA-optimized ANN solution, as a method for workload forecasting that requires minimum input of information from all sides, acting as a black box approach. This approach may be combined with LQN or any other approaches for achieving a better trade-off, or for example for areas of the application for which no information is available.

3. Mapping Service- A Performance Estimation and Workload Forecasting Framework

The service oriented framework presented in this paper (Mapping Service) is used in the IRMOS platform, which offers real time execution on service oriented infrastructures. In order for an application to be included in the IRMOS Service Provider's list of enabled services, a number of steps must be implemented with regard to performance estimation in order to adapt it to the framework. The roles that are foreseen in this process include the Application Developer (AD), the Service Provider (SP) and the Service Consumer/Client (Figure 1). The AD must describe the application parameters to the SP without breaking the need for confidentiality regarding the application's inner workings and the SP needs to create performance models based on the AD's descriptions. Then the SP may use these models during runtime, in order to find suitable resources for meeting the customer's (Service Consumer) demands for a specific set of application parameters.

In a nutshell, the responsibilities of the different roles are the following:

- the Application Developer needs to create an XML description of the application, including a set of characteristics for its functional and non-functional requirements, through the tool presented in [47]. This description is called Application Service Component Description (ASCD). This is the only involvement of the Application Developer with regard to performance.
- the Service Provider (which encapsulates the service described in this paper) needs to go through the process of creating a number of mapping rules. These rules/models are created through the invocation by the SP of the service described here and are used for the mapping between user requirements, application characteristics and the resources needed in order to meet the QoS levels that will be exposed in an SLA. The SP needs also to provide historical data of different executions of the service with different parameters, so that the training of the models may be achieved.

The benefits for the involved parties are the following:

- The AD does not have to reveal sensitive information regarding the internal structure of his application. Furthermore, the existence of performance models for their application means that a client's execution may be performed cheaper in comparison to competitive applications for which no models exist.
- The SP may utiize the Mapping Service in order to create black box models for the components, in an automated way, for driving the allocation of resources for a specific service instance requested by a client (SC). Furthermore, the existence of the models means that the specific SP may offer this application (that may also be available through other providers) with better economic terms.

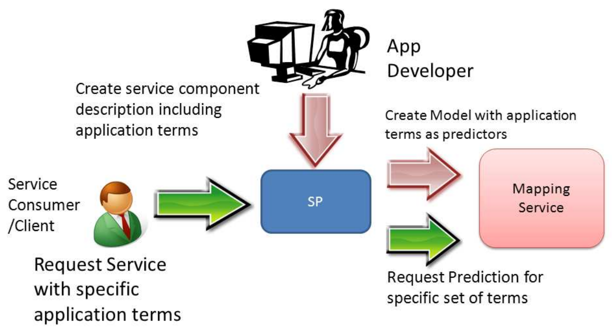
App Developer
Create service component description including application terms
Create Model with application terms as predictors
Service Consumer /Client
SP
Mapping Service
Request Service with specific application terms
Request Prediction for specific set of terms

Figure 1: Roles and Responsibilities

Thus the purpose of the Mapping Service is to create the performance models/rules on behalf of the Service Provider, through exploiting minimal knowledge regarding the application. The description of the overall platform architecture of IRMOS can be found in [27]. The high level functions that must be provided by such a framework are a) **model creation** based on the estimation method b) **model exposure** for the online estimation process. Furthermore, these processes must be fully **automatic** in order to fit in the service oriented infrastructure.

With regard to the architecture of the framework adopted to support the proposed methodology the decoupled layered approach presented in [28] has been extended in this work. In detail, the service has been extended with regard to:

- Web Services layer: responsible for receiving requests and forwarding them to the backend of the service. With regard to the previous work, a RESTful implementation has also been created and directly compared with the initial SOAP-based one, in terms of performance and robustness
- GNU Octave-based model creation and usage scripts. These are based on a novel algorithm for creating GA-optimized ANN models for the actual estimation process. The basic features of the latter are its black

box nature, which is necessary due to the different roles in the value chain and the needs for confidentiality, and its optimization of the models on a per case basis. Octave was used since it is easier to implement advanced estimation methods as a plug-in. In the initial work a simple linear regression model was used.

- Detailed performance and timing analysis for the investigation of different and parametric implementations of the service, which can result in optimized service behavior during runtime, in the context of the OPTIMIS project[16].

3.1 Model Creation and Usage

For a model to be created by the SP, the Mapping Service (MS) needs to be invoked (Figure 2-Model Creation). During this process the MS utilizes the ASCD in order to extract the inputs and outputs of the application, mainly the inputs or characteristics that influence application performance and are regarded as predictors and the outputs that can be considered as the QoS metrics. The schema used for the creation of this description can be found in [23]. This information, along with the necessary dataset (acquired through benchmarking or historical data) is used internally in order to create the models.

The models are intended to be used for direct workload estimation based on application parameters during the online negotiation of an SLA. For this reason the SP invokes the requestPrediction of the MS including in the arguments the application inputs for the specific SLA. The MS retrieves the models from the repository and runs them with the specific input values/predictors. The output of the model is retrieved and passed back to the SP. The sequence diagram of this phase appears in Figure 2(Model Usage).

3.2 Multimodal Analysis- Parametric Design

With regard to the service presented in this paper, it consists of 4 different layers (WS layer, Java/XML processing, OS and numerical software). The initial standard implementation of this service (one independent thread for each request) was analyzed in terms of its performance on the model usage phase (the demanding one from a performance point of view). From this analysis it was evident that the major bottlenecks refer to the startup and concurrent execution of the numerous octave threads for the executing requests. Thus if one needed to improve the performance of this service, an alternative design needed to be pursued, that would have the goal to minimize the number of the concurrently executing octave threads.

In order to assert therefore an alternative design, it was considered waiting for a period of T_{WAIT} time, merging different requests and launching only one Octave thread that would run the actual models for all pending requests (Figure 3). The critical parameter of such an implementation is T_{WAIT} , which is also dependent on other factors such as the request arrival rate. In conclusion, during runtime, the service framework should be able to set different waiting times, based on the current arrival rates, so that the behavior of the service is improved (Figure 2- Periodic Daemon). For generalizing this process, the methodology for building such parametric services is defined:

As parametric or gearbox services we denote the services that are capable to change or adjust the underlying implementation based on the usage conditions of their execution. These services may incorporate different or parametric implementations that have the same functional outcome but different performance characteristics, similarly to the operation of an automatic gearbox transmission system in automobiles. The different phases that are necessary to transform a regular service in a parametric one are the following:

1. Timing analysis of the delays inserted by the different layers of the implemented service
2. Identification of the most severe bottlenecks
3. Identification of an alternative design pattern that may alleviate from these bottlenecks
4. Identification of the critical parameters that influence the performance of the alternative design
5. Investigation and analysis of the critical boundaries of these parameters based on which dynamic adjustment of the used mode of operation may be performed during runtime.

The complete and thorough investigation of the parametric steps and the timing analysis on the prototype service is included in Section 6, as a show case.

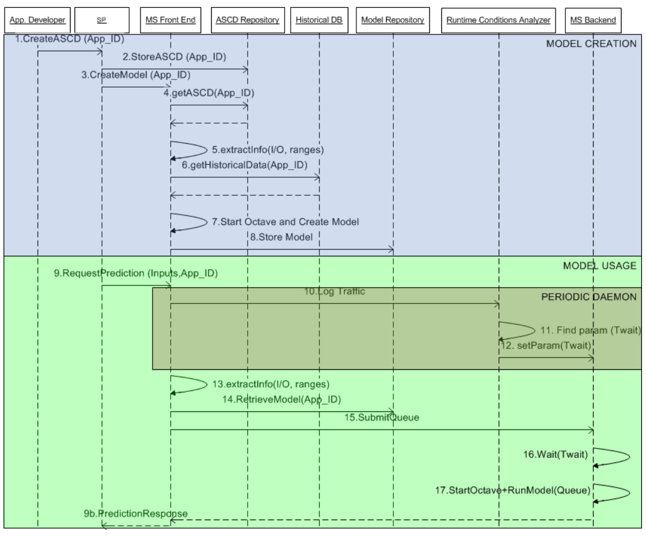

Figure 2: Sequence Diagram for CreateModel and Parametric RequestPrediction Operations

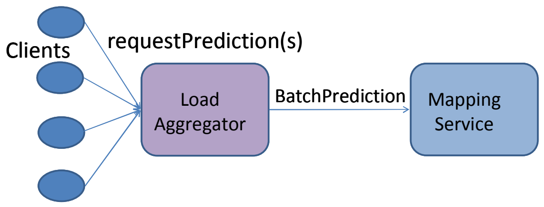

Figure 3: Addition of Load Aggregator in order to gather requests in a queue and launch Octave environment once for all cases (batch Octave processing)

4. Estimation Algorithm

In the initial implementation of this approach ([28]), a simple multivariate linear regression method was used in order to validate the experiments. In this work, we have replaced this method with an enhanced one based on Artificial Neural Networks (ANNs), that are the basis of the model for each ASC. The reason for choosing ANNs is that they are able to capture both linear and non-linear dependencies (as also identified in [29]) between the input and the output that influence the performance of the software component. Furthermore, the ANN models are flexible and can map directly application characteristics, that may be used as workload forecasting predictors, hardware parameters and Key Performance Indicator levels in a black box fashion, thus meeting the confidentiality requirements expressed in Section 3. However, in order to optimize the model creation and create a specific and optimized ANN **for each ASC**, a Genetic Algorithm (GA [30]) is used. The GA selects the parameters of the ANNs and launches the training script, which also measures the ability of the networks to generalize their predictions on a validation set. The best implementation proved to be the usage of this intermediate validation ability as the metric for producing and saving the best networks. The evolutionary metric for the GA in this case was the mean square error on the training set. More details regarding the application of the GA on the ANN design and the variations of the algorithm examined follow.

4.1 GA-based ANN optimization

The purpose of the GA is to investigate the search space and identify the fittest characteristics of the ANN that will lead to an **optimized structure** for the latter. In this process, the most important parameters of the network are decided, such as the number of hidden layers, the neurons per layer and the transfer functions per layer. In general, this process in the past was based on human experience and exhaustive testing. In our approach, this has been **automated** through the use of the GAs, so that it can be used in a service-oriented environment. It is necessary to stress that our approach does not enhance the actual GA process, but rather shows how it can be applied to the design and optimization of the ANN structure.

The first step in order to apply a GA to any optimization problem is to transform the investigated parameters to a format that can be manipulated by the former. Each candidate solution is coded like a chromosome in a bit-string format which represents the possible values for the input parameters of the function to be optimized. Afterwards the operators of the GA (mutation, elitism, crossover) evaluate the fitness of each solution and alter the members of the population in order to find a better combination. The logical mapping from the bit-string to the problem parameters appears in Figure 4 where N_L is the number of layers used for each solution (activated), TF_i is the transfer function and N/L_i the neurons per layer of the i_{th} layer.

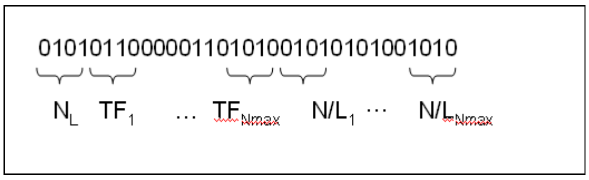

Figure 4: Chromosome encoding for ANN parameters

Due to the fact that the chromosome length is static and it is unknown in advance how many layers will be created in each candidate solution, TFs and N/Ls are created for the maximum number of layers and then the parts of the bit-string that are used according to the first parameter (number of layers) are taken. This of course creates an extra burden for the GA. The reason is that conclusions will be drawn for the fitness of the unused cases from the overall returned network performance that may confuse the convergence process. However from the experiments it was concluded that even with this drawback the algorithm converged rather quickly. After the creation of the population, each network is dynamically constructed and assigned a different combination of the parameters according to the chromosome, is trained and its ability to reach the target goal is returned to the GA, as a means to measure performance. One could argue about the use of the GA in order to solve the ANN architecture problem. This process could also be performed in a random fashion, in a search for a better network. However there have been a number of surveys ([37],[38]) indicating that the GA approach is faster and that with this methodology, randomness (which in a GA is depicted in the selection of which bit to mutate or which parts to crossover) is combined with the natural way of evolution in order to converge to a solution more rapidly. In this paper, a comparison is also made between the random and the GA-based approach that confirms the selection of the latter.

A known problem of ANNs is the loss of generalization ability due to over-fitting to the training data. In order to tackle with this situation, the models from each generation that have the best performance in terms of the regular error criterion (MSE on the training set) are saved and an additional generalization test is performed on them on a data set that has not been used during training. Then the network with the least mean generalization error is chosen as the final candidate. The conceptual diagram of the approach appears in Figure 5 and the according pseudo-code in Figure 6.

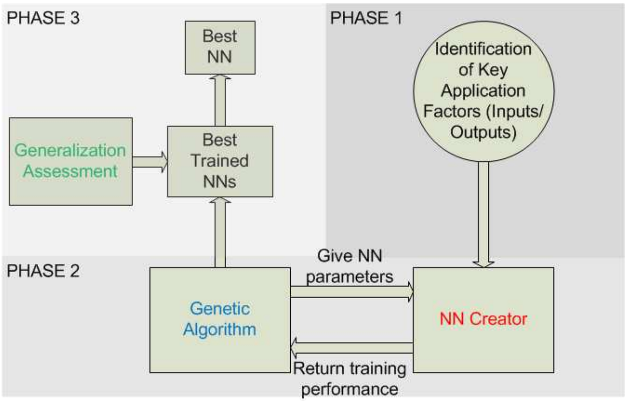
PHASE 3
PHASE 1
Best NN
Identification of Key Application Factors (Inputs/ Outputs)
Generalization Assessment
Best Trained NNs
PHASE 2
Give NN parameters
Genetic Algorithm
NN Creator
Return training performance

Figure 5: Conceptual diagram of GA-based ANN design

In summary the phases that have been identified and described are the following:

a. Phase 1: The application inputs and outputs are extracted from the XML description (provided by the Application Developer)

b. Phase 2: The data set (including the inputs, outputs and training data, produced from historical data or benchmarking) is passed to the ANN, which receives the rest of the parameters (referring to the network design) from the GA population and returns the fitness criterion (MSE after the training process). In each generation, all the networks that meet a specific “save” criterion are stored in a Best Networks Pool.

c. Phase 3: The final candidate (best ANN) is selected from the Best Networks Pool with the use of the Generalization Assessment module.

```
1.  Create initial random population
2.  For i=1 to Max generations
3.      For k=1 to Max population members
4.          Take chromosome and check the first
            parameter (number of layers
            activated)
5.          For j= 1 to Max number of activated
            layers
6.              Extract the parts of the
                bit-string that are
                activated
7.          End for
8.          Create ANN based on chromosome
            parameters
9.          Train network
10.         If Save_Criterion<global limit
            (arbitrary value defined at the
            beginning)
11.             Save net in global best
                (for generalization check)
12.         Return performance criterion
            (training or intermediate
            validation) to the GA
13.     End for
14.     Transform population with GA operators and
        create i+1 generation
15. End for
16. For every network inside global best
17.     Check error in independent final
        validation test
18.     If error< up to now best
19.         This network is the best
20. End for
```

Figure 6: Pseudocode for the model creation

Another variation that was investigated is the inclusion of an intermediate step for generalization measurement, that can serve as a better metric for the GA. This metric (Mean Absolute Error of prediction on an intermediate validation set) can be used in order to save the best networks from each generation that exhibit this generalization capability (line 10 in Figure 6) but it can also be regarded as the evolutionary (fitness) metric, meaning that a network's ability to interpolate its estimations accurately is the basic performance characteristic of the solution, rather than the adaptation to the training set (line 12 in Figure 6). For this approach to be implemented, the data set needs to be divided into 3 subsets, one for training, one for intermediate generalization measurement and one for the final generalization measurement, which is the ultimate metric for the selection of a network. Given that the size of the final validation set did not change, this means that the training set must decrease in comparison to the initial approach. The trade-off between this decrement and the new metric is investigated in Section 5.

The models created through the aforementioned steps may be used also in a more dynamic scenario, in which the SP enquiries repeatedly the MS for the predicted QoS levels, in case for example the conditions of the execution change. This may happen either due to application/user responsibility (new values for the input characteristics) or infrastructure management considerations..

4.2 Statistical Combination of Best Model Pool

As it was described in the previous sections, the algorithm stores progressively a number of suitable networks according to the "save" criterion chosen, thus creating a pool of the best models, before choosing from them the final candidate based on the final generalization test during Phase 3. At this point it was investigated whether it would be worthwhile to statistically combine these models and not choose one overall candidate, in order to create a more robust approach. This combination was performed through different statistical metrics such as the mean and median values.

If N models are available, then their mean estimated output can be calculated. The actual value of the estimated output is denoted as T, and the error of the i_{th} model for a single estimation is e_i. When a number of models are used, for which there is no absolute guarantee that one of them performs better than the others, by choosing only one, in the worst case scenario we can have that this model will produce its maximum error in this prediction. However, if the mean value of all the models is used, then:

$$\frac{1}{N}\sum_{i=1}^{N}(T+e_i)=\frac{1}{N}\sum_{i=1}^{N}T+\frac{1}{N}\sum_{i=1}^{N}e_i=T+\frac{1}{N}\sum_{i=1}^{N}e_i$$

which means that the error now is the mean error of all the used models: $\left|\frac{1}{N}\sum_{i=1}^{N}e_i\right|<\max_i(|e_i|)$

This is especially helpful when there is no guaranteed way to know which network produces the best estimation. In many cases the best one may have a minor difference in the error produced from the second best.

Another aspect that was investigated was the removal of outliers. For this metric, we acquired the prediction from all the stored models and removed the values that had a deviation of more than three times than the standard deviation. Then the new mean prediction was calculated. A third approach was also based on a reduced window of the estimated outputs, starting from the middle of the sorted array of all outputs and investigating all windows of values above and beneath the middle value. This removal was extended from 0 (mean) to k/2 (median).

The evaluation of the framework is centered around three aspects. Initially the performance of the estimation algorithm and its variations is investigated in Section 5. Afterwards, a performance analysis of the framework in the two operations described in Section 3 is presented in Section 6, along with the enforcement of the parametric design.

5. Evaluation of the estimation algorithm

The application used in order to measure the ability of the estimation algorithm was the encoding of raw video in MPEG4 format with the FFMPEG encoder [15]. As predictors, specific characteristics of the application were chosen. These were the duration of the video, the frames per second (FPS) used in the capture, the resolution of the images and an index of movement inside the video (0 for still images, 0.5 for mediocre movement and 1 for intense movement).The identification of the predictors lies on the latter, but they do not require extensive knowledge of the internal code of the application, only experience gained from using the component. The developer does not need to be an expert on the application or on workload forecasting to identify them. The predicted output was the execution time of the encoder (which is regarded as the QoS level offered in the SLA), that is the completion time until a specific video file is converted to the compressed format. For describing the hardware capabilities of the physical node of execution, two parameters were considered, the number of cores of the CPU and its speed. From the available data set, about 70% was used for the training (50% for **direct training** and 20% for **intermediate validation** of the generalization capability) of the models and the remaining 30% was used for the validation of each model's accuracy, as an **independent validation set.** This set was not used during training of the ANN.

With relation to the original version with the multivariate regression (described in [28]), we have also included **hardware** descriptors in the model, in order to observe their effect on the QoS levels of the application, for a given configuration. The testbed included 4 different machines with different number of cores and CPU speed. Each execution of the dataset was conducted 5 times, from which the mean time was extracted. It is recognized that these hardware metrics are quite crude however extensive research has been performed in the field in order to identify a more suitable characteristic, with no concrete outcome. Some indicative approaches are the Matlab benchmark tests [32], SPEC [33], Berkeley Dwarfs [34] etc. Due to the flexibility of the ANN approach and their black box nature, the hardware metrics can be replaced at any time with other more detailed ones.

80 different configurations were performed, by varying the model input parameters that have been described above. Overall, 8 different versions of the proposed approach were examined and compared to a random network design process. The latter created 600 different networks in a random fashion (same number as the maximum produced through the GA), The variations of the algorithm depended on what GA metric was returned as a performance metric for the network, if intermediate validation was used and what condition for saving networks throughout the generations applied. These versions appear in detail in

Table 1, where MSE is the Mean Square Error of the networks in the training data set and IVE is the Intermediate Validation Error which is the Mean Absolute Error (MAE, Equation 1) in the intermediate validation data set.

$$MAE = \frac{\sum_{i=1}^{K} [| f(x_i) - Y_i |]}{K} * 100$$

Equation 1: MAE definition: K number of samples, f estimated output (completion time), Y real output

Table 1: Different Variations of the Algorithm

Version	Intermediate Validation	Returned GA performance metric	Save Criterion
1	No	MSE	MSE<10^{-27}
2	No	MSE	MSE<10^{-28}
3	No	MSE	MSE<10^{-29}
4	Yes	MSE	IVE <15%
5	Yes	IVE	IVE<15%
6	Yes	IVE	IVE<8%
7	Yes	IVE	IVE<15% and MSE <10^{-27}
8	Yes	MSE	IVE<15% and MSE <10^{-27}

For all these cases, the statistical analysis that was presented in Chapter 4 was also tested. The MAE of the mean, the median, the removal of outliers and the best window were compared with the overall best model MAE produced from each version examined and are presented in Appendix B. These errors refer only to the independent final validation set. In some plots the outliers are the same as the mean, due to no outlier detection. Table 2 contains the comparison between the top 5 different approaches and variations investigated above, along with the random selection, for comparison purposes. While the best one (Version 4-Best model) did not have the best Mean Absolute Error, however its superior performance in the maximum error along with the minor difference in the MAE makes it a more suitable candidate for the final selection. From this table it is evident that the approaches presented in this paper outperform the random design method. Furthermore, despite the fact that a strict metric is used, like the Mean Absolute Error, the results are reliable and are able to predict with accuracy the output, based on a per execution basis. One additional benefit of Version 4 is the metric for saving the networks. In the cases where the MSE on the training set is used as the save criterion, this is an arbitrary value that may change from application to application and from one data set to another. However the need to have an absolute validation error of e.g. <15% is generic.

From this analysis, it can be concluded that the statistical combination of the models did not significantly help in improving the overall quality of the estimations. Three out of top 5 were the best models and the only statistical combinations that reached the top-5 regarded the overall best version (Version 4). This is indicative that all (or most of) the saved networks produced from this version were the most accurate and that this factor aided the statistical combinations to reach an increased level of accuracy. Conclusively, Version 4 is the resulting fittest approach. One significant outcome is also that despite the fact that in Version 4, the training set was reduced from 70% to 50% of the original data set in order to perform the intermediate validation (in comparison to Versions 1-3 that did not use this step), this decrease actually benefited the network design process, since it guided it towards characteristics that express better generalization capabilities.

Furthermore, the robustness of the approach is evident due to the fact that even the statistical combination of numerous models is worse than the overall best candidate. In Figure 7, the progress of the saved networks per generation is portrayed. This is another indication of the convergence of the GA. The lack of stability in this process is due to the inherent randomness of the GA operators such as mutation and crossover.

Table 2: Best versions for the FFMPEG execution time prediction (MAE on the independent validation set)

Version	Neurons / layer	MATLAB Transfer Functions per layer	Mean Absolute Error (%)	Maximum Absolute Error (%)
Version 4 (Best model)	6-13-1	Tansig-Softmax-Tansig	8.21	22.45
Version 5 (Best model)	6-10-13-1	Softmax-Satlins-Satlin-Satlins	7.68	39.96
Version 4 (Best window)	-	-	8.04	30.58
Version 4 (Median)	-	-	8.07	29.48
Version 1 (Best model)	6-28-2-1	Tansig-Radbas-Tansig-Tansig	8.68	23.76
Random Best	6-10-1	Satlins-Satlin-Purelin	10.20	36.95

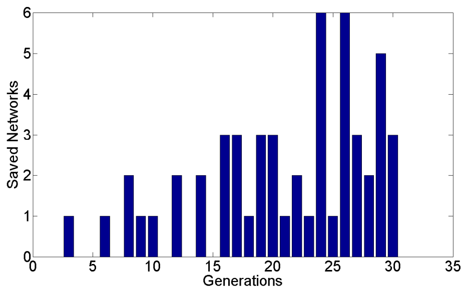

Figure 7: Saved Networks over generations for Version 4

At this point it must be stressed that the framework may be used at any time by the SP or a more specialized component like an evaluator, for acquiring runtime predictions on the fly. For example, due to the flexibility of the predictors, the SP may perform ad hoc enquiries each time a new file is submitted for encoding, and based on the specific characteristics of the file, thus achieving a more dynamic management of the application. New predictions may be acquired also in case the SP needs to reallocate the application to a new hardware node, for internal management purposes, and thus needs a new estimation for the output of the model when executed on the new assignment node.

The same ANN optimization method has also been applied for the creation of the models in [17], in a differentiated application type. In that paper two ANN models were created in order to predict the mean delay and the standard deviation respectively of an e-learning server's responses. The models in that case also showed extensive accuracy, with errors of 2.51% and 2.75% respectively. The predictors that were used as inputs to the models were the number of users, the real-time scheduling configuration and the characteristics of the physical server on which the application was deployed.

6. Parametric Analysis

In this section, the detailed steps of the parametric methodology are enforced on the case study service, the layered service estimation framework described in Section 3. A detailed performance analysis is made on the simple version of the service (one thread raised for each incoming request) and based on the findings the parametric form of the service described in 3.2 is concluded.

6.1 RequestPrediction Method

6.1.1 Step 1: Detailed Timing Analysis

In order to measure and stress the system's performance, a number of different scenarios have been tried out. Overall, measurements have been performed, for both phases that are described in Section 3.1. More emphasis has been given to the requestPrediction method, which is more critical in terms of time performance. This is due to the fact that it is used during the **online negotiation stage**, where the Service Consumers are also involved.

In detail, the following cases have been taken under consideration:

- Standalone execution of the Octave model, in order to investigate what is the baseline time needed for Octave to calculate the estimation and for different numbers of concurrent executions. This corresponds to Step 17 of Figure 2 and is denoted as T_{OCTAVE} .
- Service execution of the Octave model (requestPrediction), in order to measure the overall time T_{NL} of steps 9,13,14,15,16,,17, 9b of Figure 2 when T_{WAIT} is zero and the internal service processing time (T_{INT}). The latter can be deconstructed to preprocessing time (T_{PRE}-steps 13,14) and octave time (T_{OCTAVE}-step 17)
- Usage of clients on the same and on different machines, to observe the overhead of the clients to the service performance. This is critical for cases where services are merged on the same physical host.
- Batch execution for multiple predictions. This was based on the assumption that most of the delay was due to the Octave environment initialization (T_{OS}). Thus, starting the latter once and performing multiple predictions in the same instance was also considered in order to observe the pure processing time for one prediction (T_{OP}) inside the Octave environment.
- Change of the WS layer, to replace the SOAP-based technology with the REST protocol and measurements of the inner response times. This was the time elapsed from the call reception to the call end. By subtracting the inner response times (T_{INT}) from the overall response times (T_{NL}) we can observe the delay inserted by the different service protocols (T_{SC}-steps 9 and 9b). By comparing the standalone Octave executions (T_{OCTAVE}) to the inner response times of the service versions, we can observe the delay inserted by the processing (T_{PRE}) that is

needed inside the service and has to do with XML processing, ASCD repository access etc.

In order to ensure that the desired number of concurrent calls would be performed, the according number of clients was launched. Each client performed a requestPrediction call for 500 times in a sequential way. Each client also documented the response times of the requests. In Figure 8 and Figure 9, the results from the experiments are portrayed. Detailed delays are included for all the scenarios mentioned in the previous paragraph, in order to observe the effect of the various parameters on the times. The main conclusion is that all times appear to have a linear dependency on the number of concurrent calls. Another critical conclusion is the division of the delays in the previously mentioned steps. In detail, the breakdown of the time delays in the various stages is 4% for the WS layer (T_{SC}), 28% for the XML processing (T_{PRE}), 66% for the Octave startup (T_{OS}) and 2% for the internal Octave model calculations (T_{OP}). Hence we can conclude that is that the Octave startup time is the main reason for the aforementioned delay.

This is also evident by the far improved per estimation times for the batch requests (100 estimations compressed in one call). When having batch estimations, this startup time happens only once for the service call and only one Octave thread is serving the request. The range of this startup time is around 1 second (for 1 client), as can be roughly seen from the differences between single and batch execution in Figure 9. After that, for each estimation the time needed by Octave in order to apply the inputs to the model and acquire the output is in the range of 20 milliseconds for 1 client.

The overhead of calls when these are performed from the same machine (Figure 8) due to the extra strain on the server is insignificant. This can be helpful in cases where it is needed to concentrate all the services in one physical node for consolidation purposes. Calls on the other hand that originate from different machines have larger deviations due to network delays but slightly improved delays. However this conclusion may be **service-specific**. In this case, the internal processing times for Octave (T_{OCTAVE}) are significantly higher than the service delays (T_{SC}), which helps in hiding the effect of the specific parameters.

6.1.2 *Step 2: Identification of Bottlenecks*

The performance analysis of the service behavior that was presented in the previous section is very beneficial if it used as **feedback** in the design of the service. Through this analysis, the weak and strong points of the implementation have been identified. The vast majority of the delay during the requestPrediction operation is owed to Octave initialization and concurrently active Octave threads. The pure processing time for running the models is very low. Thus concentrating requests in a single call to Octave and performing batch estimations is very beneficial, since we can gain the delay that is due to Octave startup and minimize the concurrently running Octave threads.

6.1.3 *Step 3: Alternative Design/implementation patterns: Load Aggregation as a preprocessing step*

The addition of a Load Aggregator component (Figure 3) as a front-end for the service can serve the purpose of gathering incoming requests from individual clients and reduce the number of Octave startups, as already described in Section 3. These requests can be merged into a queue until a criterion is met (maximum number of requests or time interval) and then the process can continue (Octave startup and internal Octave processing) as a batch process. This way we can avoid the Octave startup time, which as seen from the measurements corresponds to almost 66% of the delay. However an analysis must be conducted in order to investigate whether this implementation actually benefits response times and what are the fittest parameters that must be selected (e.g. **waiting time** before launching Octave in order to serve the request queue).

This design does not assume that the arrival rate of requests is synchronized. The requests may arrive in an asynchronous manner. The purpose of the waiting time is to gather and queue these asynchronous requests into one batch request towards the Octave environment, which as seen from the timing analysis is very beneficial for the performance of the service framework.

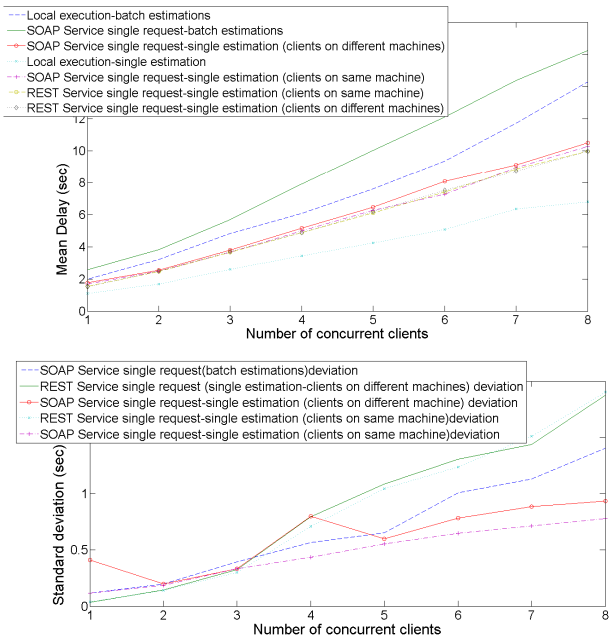

Figure 8: Overall times and deviations for various deployments and number of concurrent requests

6.1.4 *Steps 4-5: Identification of critical parameters and their boundaries for dynamic adjustment*

Let's denote as *R(t)* the request arrival. As seen in the measurements, the delay times are directly related with the **number *n* of concurrent requests** made towards the server and can be calculated during runtime by the service. The overall delay for one request T_{NL} (without Load Aggregation) is then given by Equation 2, if we consider the mean values that are extracted from the measurements.

$$T_{NL} = T_{PRE}(n) + T_{OS}(n) + T_{OP}(n)$$

Equation 2: Delay for serving *n* concurrent requests without the Load Aggregator

Some indicative values for these relationships for the specific implementation are depicted in Equation 3, after fitting a line to the according graph, with a norm of residuals equal to 0.5687:

$$T_{PRE} = 0.394 * n + 0.0086$$

$$T_{OCTAVE}(n) = T_{OS}(n) + T_{OP}(n) = 0.85 * n + 0.081$$

$$T_{NL}(n) = 1.244 * n + 0.0896$$

Equation 3: Extracted values from experimental results for single estimation per service call without Load Aggregation

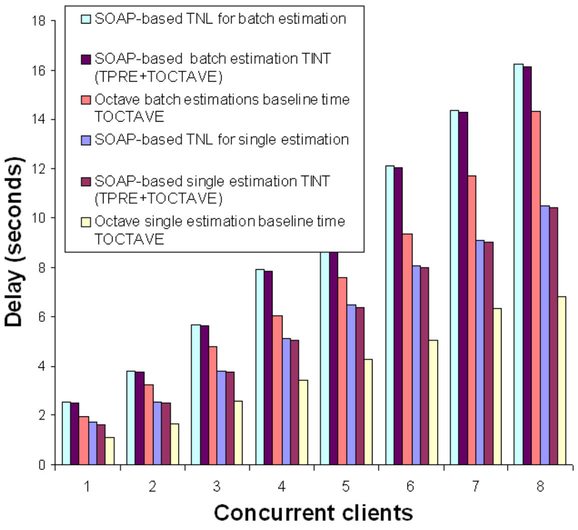

Figure 9: Comparison of delays for the three different steps (T_{NL}, T_{INT}=T_{PRE}+T_{OP},T_{OP}) for single and batch estimation

For a given period of time, the number *n* of concurrent calls can be found if the overall time is measured (from the mean of the individual response times) and we solve Equation 3. Now let's consider waiting for T_W amount of time before launching the Octave environment. During this time, all the requests that arrive and finish the preprocessing layer (service call and XML processing) are entered in a queue. At the end of T_W Octave is launched and all the requests in the queue are processed. The overall requests N_R that have arrived during this interval (0 to T_W) are given by Equation 4, if we consider a **constant** rate *C* of arrivals:

$$N_R = \int_0^{T_W} R(t)dt = C * T_W$$

Equation 4: Arrived requests during interval T_W

These are not exactly the requests that need to be served because they may not have been preprocessed (in the XML processing part) in time in order to enter the queue. However, given that the same happens for the previous time interval, the remaining requests from that slot are served during the current one. Thus the overall requests are indeed given by Equation 4, if we consider a constant rate for *R(t)=C*t*. The delay interval *DI* before all requests are served is given by Equation 5, for the worst case scenario which is depicted in Figure 12. In this case, the preprocessing ends just after the T_W of the previous interval expires. Thus there is no overlap between T_W and T_{PRE} and they are both added in full in the *DI*, along with the processing time needed by Octave. T_{OCTAVE} is the time needed by Octave to calculate the answer, which is analyzed in the time needed to start the environment (for 1 user) and the time needed for one estimation, multiplied by the number of **pending** requests gathered during T_W. T_{PRE} is the preprocessing time for each request, which is not affected by the number of concurrent calls if $T_{PRE}<1/C$.

$$DI = T_{PRE} + T_W + T_{OCTAVE} = T_{PRE}(1) + T_W + T_{OS}(1) + N_R * T_{OP}(1)$$

Equation 5: Overall Delay Interval for providing the answer. The number of concurrent requests is now n=1, because all the requests have been merged through the Load Aggregator

Then to count the overall mean delay per request T_L we can sum the individual delays of each request (Equation 6).

$$T_L = \frac{\sum_{k=0}^{N_R-1}(DI - k * \frac{dR(t)}{dt})}{N_R}$$

Equation 6: Average delay per request for Load Aggregation

Now the condition for enabling the Load Aggregator mode is when T_{NL} is greater than T_L. In order to find the waiting time T_W we can follow Equation 7. A simple sanity check indicates that if 1/C is very high (indicating sparse requests) then C is very low, in which case T_W will have a negative value, indicating that Load Aggregation should not be enabled. This is expected since when requests are sparse, n in Equation 2 would be always 1.

$$\frac{\sum_{k=0}^{N_R-1}(DI - \frac{k}{C})}{N_R} < T_{NL} \Rightarrow T_W < \frac{2*C*(T_{NL} - T_{PRE} - T_{OS}(1)) - 1}{C + 2*C^2*T_{OP}(1)}$$

Equation 7: Condition for determining T_W based on value of *C*

Furthermore, we can bind T_W to be greater than 1/C (request inter-arrival time), because for T_W<1/C the Load Aggregation phase is degenerated to the no aggregation phase. Each request is served as standalone and not through batch estimation. So we need that

$$\frac{1}{C} < \frac{2*C*(T_{NL} - T_{PRE} - T_{OS}(1)) - 1}{C + 2*C^2*T_{OP}(1)}$$

which after solving is reduced to:

$$\frac{1}{C} < T_{NL} - T_{PRE} - T_{OS}(1) - T_{OP}(1)$$

Equation 8: Limit of C for applying the Load Aggregation

In conclusion, if Equation 8 is satisfied during runtime, then we can enable the Load Aggregation, with a waiting time that is according to Equation 7. Otherwise no load aggregation is enabled. At this point it must be stressed that we used the assumption of constant rate of requests in order to simplify the analysis. In real life situations one can expect a more probabilistic form of request arrivals. In order to simulate these arrivals with the constant rate, we considered C as *the mean value of the inter-arrival times, when these are given by a probabilistic distribution*. For identifying the fittest types, numerous interesting works exist (indicatively [42],[43] and [44]), from which however no single distribution can be identified. In general, the type depends on various features, such as the examined dataset, the type of metric observed, the type of the service/website etc. However, the 4 most prominent were Weibull, Pareto, Gamma and Poisson. For these, we have considered 3 different mean values (0.1, 0.5 and 1 seconds) for the inter-arrival times. For each configuration, 10000 values were generated through these distributions for each mean value. The mean response time of the service for the calls following the produced inter-arrival times was documented and compared against the same metric for the service when called by requests with the same static inter-arrival rate as the mean value (0.1, 0.5 and 1 seconds). The results (which appear in the Appendix A) show that indeed the constant rate can in most cases be used, given that it depicts very similar mean response times compared to distributions with the same mean value.

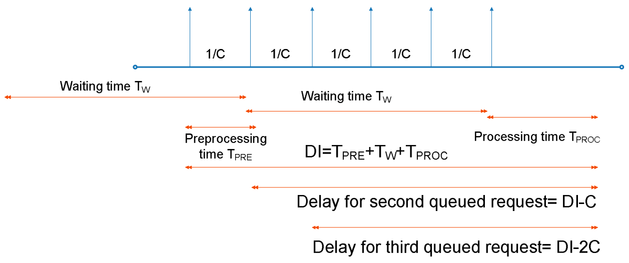
1/C
1/C
1/C
1/C
1/C
Waiting time T_W
Waiting time T_W
Preprocessing time T_{PRE}
Processing time T_{PROC}
$Dl=T_{PRE}+T_W+T_{PROC}$
Delay for second queued request= Dl-C
Delay for third queued request= Dl-2C

Figure 10: Representation of Equation 5

Comparison between RESTful and SOAP implementations

An interesting conclusion from the timing analysis is that overall the RESTful implementation is marginally faster than the SOAP based one, as depicted in Figure 9. The distributions of the individual response times are depicted in Figure 11. It must be noted that these are the overall times. However, as seen from the standard deviations in Figure9b, these are significantly increased for the REST cases. So overall, the mean times are very similar, REST is faster, however the distribution of the values shows that the SOAP protocol is more smooth. The diversity of the response times for values that are close to zero (~0.2 seconds) for the REST implementation is indicative that the call **failed**. This can be safely assumed since the Octave environment needs around 1 second at least to start. This failure is experienced for 128 cases of the 4008 samples gathered, thus around 3.2%. This does not happen for the SOAP implementation and can also explain the increased standard deviation values that are portrayed by REST.

However, this marginal difference of REST is on the overall response times. After subtracting the inner times from the overall service times, removing the values for which REST failed and comparing the mean values for 8 concurrent clients, the mean REST call time was found to be **6.68** milliseconds (with standard deviation of 2.1 milliseconds) while the SOAP call time was **79.81** milliseconds (with standard deviation of 514 milliseconds). The large value of the deviation is due to three values in the dataset (out of 4000 samples) that exhibit a very high delay (specifically 8, 17 and 26 seconds). If these values are removed the mean SOAP delay is reduced to 67 milliseconds and the standard deviation to 57 milliseconds. Thus there is **one order of magnitude** difference between the two protocols.

It is acknowledged that in the examined operation, this difference (~60 msecs) is not significant given that the overall service times are close to 1.2 seconds for the single client case. However in services where the inner processing is more lightweight (in the range of a couple of hundred milliseconds) it is expected to have much greater effect. But despite the speed difference (which favors REST), the call failure difference supports the usage of SOAP in cases where call failure is unacceptable. This call failure seems to linearly increase in the REST case when workload increases.

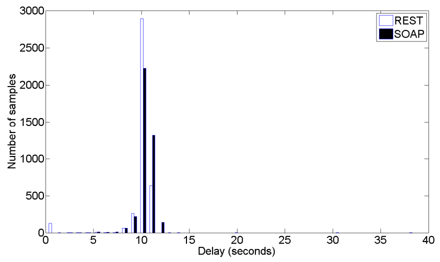

Figure 11: Distribution of Server Response Times for the REST and SOAP protocols and 8 concurrent clients

6.2 CreateModel Operation

6.2.1 Step 1: Detailed Timing Analysis

For the CreateModel operation, the results appear in Figure 12. Again the delays seem to be linear compared to the number of concurrent requests made to the server. One anticipated outcome is that this operation takes much more time then the requestPrediction, because of the nature of the algorithm described in Section 4. However given that this time is random for each different execution, due to the randomness of the parameters that are selected through the GA, we have also plotted the standard deviation of the runs. For each data point, 30 different executions were performed and the mean value from these was considered as the delay. The standard deviation of the measurements is within expected values.

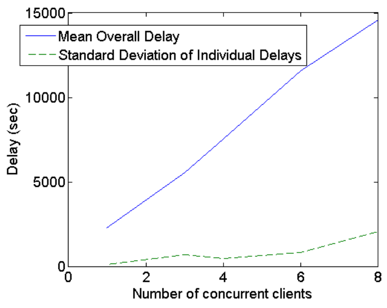

Figure 12: Delay of createModel operation

6.2.2 Steps 2- 3: Identification of Bottlenecks and Alternative Design/implementation patterns

The createModel operation times shows linear increase which leads to unacceptable delays. Thus a solution should be found to mitigate this effect. Given that the processing time for the particular method selected is high (in the range of thousands of seconds) and that it should scale compared to the number of requests, the aforementioned implementation does not meet this requirement.

Ideally, in order to be able to meet the elastic demand, the service should be elastically expanded (bursted) to a Cloud provider. This means that a new instance of the server should be created on the Cloud for every new request for model creation. This way, the performance of the createModel operation would be stable, scalable and similar to the delay portrayed for one call. The difference would be located in the additional delay inserted in order to activate the Cloud Provider's API and boot the VM that contains the server and in the additional cost. This was in the range of 40-60 seconds.

6.2.3 Steps 4-5: Identification and dynamic adjustment of critical parameters boundaries:

A boundary based on price for starting and utilizing new VMs in the Cloud can also exist, in order to achieve a trade-off between performance and cost. If C_{MAX} is the maximum desired cost, C_{CUR} the current cost, T_{CUR} the current mean delay of the responses and T_{TARGET} the desired value, then the decision process may follow the pseudo-code in Figure 13.

```
1.  for each incoming createModel request
2.          if $C_{CUR}<C_{MAX}$ and $T_{CUR}>T_{TARGET}$
3.                  call Cloud API (addVM)
4.                  wait until VM is booted
5.                  forward request to VM
6.                  get response (model stored)
7.                  forward response to
                       original caller
8.                  terminate VM
9.  end
```

Figure 13: Pseudo-code for bursting into the Cloud

7. Conclusions

In this paper we have presented a service oriented framework for implementing Octave based workload forecasting methods. This framework can be applied in a real life production environment, in order to aid in the SLA negotiation process between consumers and service providers in SOIs. In this context the proposed framework is generic and loosely coupled, so that at any time specific layers can be removed and replaced by more advanced ones. The flexibility of the approach has been demonstrated through the use of different WS protocols.

From the measurements it has been seen that the Octave startup time is the major bottleneck of the service (~1 second for 1 client or 66% of the delay). Differences in WS protocols can improve the pure service call times by an order of magnitude, but in this specific implementation this does not translate to a significant overall performance gain (~4%) due to the increased internal processing times. REST-based implementation also suffers from a 1.5-3% call failure depending on the number of the concurrent calls. The pure estimation time inside the GNU Octave tool is around 20 milliseconds (for 1 client). Required processing for service purposes is around 0.4 seconds or 30% of the delay (for 1 client). All the delays show a linear dependency from the number of concurrent requests being processed by the server. Furthermore, a methodology for creating parametric services with optimized behavior has been highlighted and has been demonstrated on the implemented service.

For the future, one goal to pursue is to extend the usage of this framework for solving optimization problems, in the context of the EU-funded FP7 project OPTIMIS. Our main objective is to replace the estimation framework based on GNU Octave with an optimization framework based on the GAMS tool [31], in order to use it during admission control of services in Cloud infrastructures. The conclusions and design aspects that were demonstrated in this paper can be extended to any similar service that utilizes specialized embedded software in the bottom layer.

8. ACKNOWLEDGMENTS

This work has been supported by the IRMOS and OPTIMIS projects and has been partly funded by the European Commission's IST activity of the 7th Framework Programme under contract number 214777 and 257115

9. REFERENCES

[1] T. Erl, “Service-oriented Architecture: Concepts, Technology, and Design”, Upper Saddle River: Prentice Hall PTR. ISBN 0-13-185858-0, 2005.

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A view of cloud computing. Commun. ACM 53, 4 (April 2010), 50-58.

[3] The NIST definition of Cloud Computing. Available at: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[4] Debusmann, M.; Keller, A., "SLA-driven management of distributed systems using the common information model," Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on , vol., no., pp. 563-576, 24-28 March 2003.

[5] G. Wang, A. Chen, C. Wang, C. Fung, and S. Uczekaj, “Integrated Quality of Service (QoS) Management in Service-Oriented Enterprise Architectures”, In Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC), Monterey, California, September 2004.

[6] Zhengting He, Cheng Peng, Aloysius Mok, "A Performance Estimation Tool for Video Applications," rtas, pp. 267-276, 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06), 2006

[7] Siegfried Benkner, Gerhard Engelbrecht, "A Generic QoS Infrastructure for Grid Web Services," aict-iciw, p. 141, Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW'06), 2006

[8] Peer Hasselmeyer, Bastian Koller, Lutz Schubert, Philipp Wieder: Towards SLA-Supported Resource Management. HPCC 2006: 743-752

[9] Jarvis, S. A., Spooner, D. P., Keung, H. N., Cao, J., Saini, S., and Nudd, G. R. 2006.Performance prediction and its use in parallel and distributed computing systems.Future Gener. Comput. Syst. 22, 7 (Aug. 2006), 745-754.

[10] Oana Florescu, Menno de Hoon, Jeroen Voeten, and Henk Corporaal. Probabilistic modelling and evaluation of soft real-time embedded systems. In Proceedings of SAMOS VI, LNCS 4017,2006.

[11] Singh, K., İpek, E., McKee, S. A., de Supinski, B. R., Schulz, M., and Caruana, R. 2007. Predicting parallel application performance via machine learning approaches: Research Articles. Concurr. Comput. : Pract. Exper. 19, 17 (Dec. 2007), 2219-2235.

[12] Marc N. Haines and Marcus A. Rothenberger. 2010. How a service-oriented architecture may change the software development process. *Commun. ACM* 53, 8 (August 2010), 135-140.

[13] Heiko Koziolek, Performance evaluation of component-based software systems: A survey, Performance Evaluation, Volume 67, Issue 8, Special Issue on Software and Performance.

[14] Using GAMS on the Amazon Elastic Compute Cloud. Available at:http://support.gams-software.com/doku.php?id=platform:aws

[15] Fabrice Bellard, FFMPEG multimedia system. Available at: http://www.ffmpeg.org, 2005.

[16] A.J. Ferrer, F. Hernandez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R.M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S.K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgo, T. Sharif, and C. Sheridan "OPTIMIS: a Holistic Approach to Cloud Service Provisioning, , Future Generation Computer Systems, Elsevier, Vol. 28, No. 1, pp. 66-77, 2012.

[17] Tommaso Cucinotta, Fabio Checconi, George Kousiouris, Kleopatra Konstanteli, Spyridon V. Gogouvitis, Dimosthenis Kyriazis, Theodora A. Varvarigou, Alessandro Mazzetti, Zlatko Zlatev, Juri Papay, Michael Boniface, Sören Berger, Dominik Lamp, Thomas Voith, Manuel Stein: Virtualised e-Learning on the IRMOS real-time Cloud. Service Oriented Computing and Applications 6(2): 151-166 (2012)

[18] Rich Wolski, Neil T Spring, Jim Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, Future Generation Computer Systems, Volume 15, Issues 5–6, October 1999, Pages 757-768,

[19] E. Smirnova, C.M. So, S.M. Watt, Providing mathematical Web services using Maple in the MONET architecture. In Procs. MONET Workshop (2004)

[20] YarKhan, A., Dongarra J., Seymour K., 2007, in IFIP International Federation for Information Processing, Volume 239. Grid-Based Problem Solving Environments, eds. Gaffney, P. W.. Pool, J.C.T., (Boston: Springer), pp. 215–224.

[21] Petcu, D. 2006. BetweenWeb and Grid-based Mathematical Services. In Proceedings of the international Multi-Conference on Computing in the Global information Technology (August 01 - 03, 2006). ICCGI. IEEE Computer Society, Washington, DC.

[22] A. Al Zain, K. Hammond, P.W. Trinder, S.A. Linton, H.-W. Loidl, and M. Costanti. SymGrid-Par: Designing a Framework for Executing Computational Algebra Systems on Computational Grids. In Proc. International Conference on Computer Science (Workshop on Practical Aspects of High-level Parallel Programming), Beijing, China, May 27-30, 2007.

[23] IRMOS Project: "Irmos Application Blueprint", DTO and partners, December 2009

[24] GNU Octave Tool. Available at:http://www.gnu.org/software/octave/

[25] Paola Laface, Damiano Carra and Renato Lo Cigno, "A Performance Model for Multimedia Services Provisioning on Network Interfaces", in Lecture Notes in Computer Science, Volume 3375/2005, Springer Berlin / Heidelberg

[26] Roger J. Castaldo Michael A. McKay Vladimir Tosic, "Exposing GNU Octave Signal Processing Functions as Extensible Markup Language (XML) Web Services", in Canadian Conference on Electrical and Computer Engineering, 2006. CCECE '06.

[27] IRMOS Project D3.1.2: "IRMOS Overall Architecture", NTUA and other partners, February 2009

[28] George Kousiouris, Dimosthenis Kyriazis, Kleopatra Konstanteli, Spyridon Gogouvitis, Gregory Katsaros, Theodora Varvarigou, "A Service-Oriented Framework for GNU Octave-Based Performance Prediction," scc, pp.114-121, 2010 IEEE International Conference on Services Computing, 2010

[29] Lianzhang Zhu; Xiaowing Liu; , "Technical Target Setting in QFD for Web Service Systems Using an Artificial Neural Network," Services Computing, IEEE Transactions on , vol.3, no.4, pp.338-352.

[30] Holland, J. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press.

[31] GAMS Modeling System. Available at:http://www.gams.com/

[32] Matlab Benchmarks description. Available at:http://www.mathworks.com/help/techdoc/ref/bench.html

[33] SPEC Benchmarks. Available at:http://www.spec.org/benchmarks.html

[34] Berkeley Dwarfs. Available at:http://view.eecs.berkeley.edu/wiki/Dwarfs

[35] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. 2010. Q-clouds: managing performance interference effects for QoS-aware clouds. In Proceedings of the 5th European conference on Computer systems (EuroSys '10). ACM, New York, NY, USA, 237-250.

[36] J. Happe, D. Westermann, K. Sachs, and L. Kapova, "Statistical inference of software performance models for parametric performance completions," in Lecture Notes in Computer Science, 2010, Volume 6093/2010, 20-35.

[37] S. Areibi, M. Moussa, H. Abdullah, A comparison of genetic/memetic algorithms and other heuristic search techniques, in: ICAI 2001, Las Vegas, Nevada, 2001

[38] C. J. Fourie andW. J. Perold, “Comparison of genetic algorithms to other optimization techniques for raising circuit yield in superconducting digital circuits,” IEEE Trans. Appl. Supercond., vol. 13, no. 2, pp. 511–514, Jun. 2003

[39] D’Ambrogio, A., Bocciarelli, P.: A Model-driven Approach to Describe and Predict the Performance of Composite Services. In: Proceedings of the 6th Int. Workshop on Software and Performance (WOSP), Buenos Aires, Argentina (2007)

[40] P. Bocciarelli, A. D'Ambrogio. “Model-Driven Performability Analysis of Composite Web Services.” SPEC Intl Perf. Evaluation Workshop 2008: 228-246.

[41] P. Bocciarelli and A. D’Ambrogio, “A model-driven method for describing and predicting the reliability of composite services”, Software and Systems Modeling, Springer Berlin / Heidelberg, pp. 265-280, Volume 10, Issue 2, 2011.

[42] Martin F. Arlitt and Carey L. Williamson. 1997. Internet Web servers: workload characterization and performance implications. IEEE/ACM Trans. Netw. 5, 5 (October 1997), 631-645.

[43] Zhen Liu, Nicolas Niclausse, César Jalpa-Villanueva, Traffic model and performance evaluation of Web servers, Performance Evaluation, Volume 46, Issues 2–3, October 2001, Pages 77-100, ISSN 0166-5316.

[44] Paul Barford and Mark Crovella. 1998. Generating representative Web workloads for network and server performance evaluation. SIGMETRICS Perform. Eval. Rev. 26, 1 (June 1998), 151-160.

[45] Casolari, S.; Colajanni, M.; Tosi, S.; Presti, F.L.; , "Real-time models supporting resource management decisions in highly variable systems," Performance Computing and Communications Conference (IPCCC), 2010 IEEE 29th International , vol., no., pp.247-254, 9-11 Dec. 2010

[46] Casolari, S.; Colajanni, M.; Tosi, S.; , "Detecting Behavioral Variations in System Resources of Large Data Centers," Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on , vol., no., pp.371-378, Aug. 31 2011-Sept. 2 2011

[47] Luís Costa, “Modeling Interactive Real-time Applications on Service Oriented Infrastructures”, 3rd IRMOS Public Seminar, available at: http://irmosproject.eu/Files/02_IRMOS_Oslo_Seminar_Modelling-PartI.pdf

Dr. George Kousiouris received his diploma in Electrical and Computer Engineering from the University of Patras, Greece in 2005 and his Ph.D. in Grid and Cloud computing from of the Dept. of Electrical and Computer Engineering of the National Technical University of Athens in March 2012. Currently he is a researcher for the Institute of Communication and Computer Systems (ICCS). His interests are mainly computational intelligence, optimization, artificial intelligence and web services.

Dr. Andreas Menychtas graduated from the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA) in 2004. In 2009, he received his Ph.D. in area of Distributed Computing from the School of Electrical and Computer Engineering of the National Technical University of Athens. His research interests include Distributed Systems, Web Services, Object Oriented Programming, Service Oriented Architectures, Security and Information Engineering. Currently, he works as research engineer in the Institute of Communication and Computer Systems (ICCS) of National Technical University of Athens.

Dr. Dimosthenis Kyriazis received his diploma from the school of Electrical and Computer Engineering of the National Technical University of Athens (NTUA) in 2001 and his MSc degree in “Techno-economics” in 2004. Since 2007, he holds a PhD in the area of Service Oriented Architectures with a focus on quality aspects and workflow management from the school of Electrical and Computer Engineering Department of NTUA. His expertise lies with service-based, distributed and heterogeneous systems, software and service engineering. He is a Senior Research Engineer of the Institute of Communication and Computer Systems (ICCS) of NTUA

Dr Kleopatra Konstanteli received her diploma in Electrical and Computer Engineering in 2004 by the National Technical University of Athens (NTUA). In 2007 she received a Master in Techno-economical Systems and her Ph.D. in the area of distributed computing from the school of Electrical and Computer Engineers of NTUA in 2011Her research interests are mainly focused in the field of distributed computing and optimization.

Spyridon V. Gogouvitis received the Dipl. -Ing. from the School of Electrical and Computer Engineering of the National Technical University of Athens (NTUA) in 2006. He is currently pursuing his PhD in the Telecommunication Laboratory of Electrical and Computer Engineering School of NTUA. He is also working as a research associate in the Institute of Communication and Computer Systems (ICCS). His research interests include Cloud Computing, Web Services and Service Oriented Architectures, Mobile and Ubiquitous Computing.

Gregory Katsaros received the diploma from the Dept. of Electrical and Computer Engineering of the National Technical University of Athens (NTUA, Greece) in 2006 and the MS degree in Techno-Economic Systems (MBA) in 2008. He has been working as an Associate Researcher in the Telecommunications Laboratory of the Institute of Communications and Computer Systems (ICCS) of NTUA from 2006 His research interests include Cloud management, service engineering and monitoring infrastructures.

Prof. Theodora A. Varvarigou received the B. Tech degree from the National Technical University of Athens, Athens, Greece in 1988, the MS degrees in Electrical Engineering (1989) and in Computer Science (1991) from Stanford University, Stanford, California in 1989 and the Ph.D. degree from Stanford University as well in 1991. Since 2007 she is a Professor at the National Technical University of Athens. Prof. Varvarigou has great experience in the area of semantic web technologies, scheduling over distributed platforms, embedded systems and grid computing. In this area, she has published more than 150 papers in leading journals and conferences.