

Predicting Resource Demand in Dynamic Utility Computing Environments

Artur Andrzejak
Computer Science Research

Zuse-Institute Berlin
Berlin, Germany
andrzejak@zib.de

Sven Graupner
Enterprise Software and
Systems Lab (ESSL)

Hewlett-Packard Laboratories
Palo Alto, CA, USA
sven.graupner@hp.com

Stefan Plantikow
Computer Science Research

Zuse-Institute Berlin
Berlin, Germany
plantikow@zib.de

Abstract— We target the problem of predicting resource usage in situations where the modeling data is scarce, non-stationary, or expensive to obtain. This scenario occurs frequently in computing systems and networks, mostly due to the high dynamicity of the underlying processes. Utility computing environments are an important example for such a scenario, as their frequent reconfiguration reduces the amount of training data available for modeling. We propose an approach based on a genetic algorithm and fuzzy logic which allows for creation of robust prediction models even with scarce training data. The method is evaluated on demand usage traces collected from 41 servers in a business data center. The results show in the setting of scarce training data amount our method has a significantly higher prediction accuracy compared to other non-linear techniques such as decision trees or support vector machines.

Keywords - demand prediction, system identification (modeling) techniques, genetic fuzzy controller, utility computing, automated resource allocation

I. INTRODUCTION

The IBM's Autonomic Computing initiative revived the concept of the proactive control of computer systems and networks, and thus spurred interest in application of prediction to system management problems [1], [5]. These applications include job scheduling in utility computing and Grid environments, short- and long-term capacity planning, software rejuvenation, load-balancing in clusters, and other [2].

Hewlett-Packard is developing a management automation platform based on so-called Service Delivery Controllers (SDC). The SDC concept is similar to the controllers, or Autonomic Managers, that are used in Autonomic Computing. The difference is that SDC are specialized for system management tasks and operate on discrete state models representing management states in a managed environment. SDC base their control loop on two distinct models, one model which represents the “desired state”, and one model which represents the “observed state” of an environment. The controller function then automatically aligns a managed environment such that its observed state corresponds to what has been set as desired state for this environment [6]. An essential tool for effective functioning of the controller in respect to resource management (including capacity control and load balancing) is the demand prediction of applications, (virtualized) servers, or virtual server farms.

Most prediction techniques utilized in context of computer systems are based on linear models, such as exponential smoothing and autoregressive methods, mostly ARIMA [5]. More advanced implementations such as Network Weather Service [16] deploy a combination of different models together with dynamic switching between them. These approaches provide a reasonable prediction quality for short-term prediction and in case of simple underlying processes. However, they are unable to model more complex patterns, and mostly do not take into account correlations between signal sources. Contrary to this, predictors based on non-linear models borrowed from data mining are able to capture complex patterns and allow for effortless incorporation of signal correlations [2].

However, these types of predictors usually require a larger amount of historical training data in order to build up the prediction model - the more data, the more complex is the underlying process [14]. The amount of historical data might not be enough for training if the system is complex (thus having a lot of states), or if it changes frequently e.g. due to a distributed nature. This high system dynamicity introduces an inherent conflict: on one hand, such systems require automated, prediction based resource control in order to reduce the management cost. On the other hand, the rate of changes impedes the collection of sufficient data basis to build accurate (non-linear) models.

Fuzzy logic rules have been shown to be an important tool for modeling complex systems [9], especially when classical tools are unsuccessful due to complexity or imprecision [3]. Yet fuzzy logic predictors and controllers require manual design and tuning, which reduces their utility in the scenario of frequent model changes. Especially in dynamic environments of complex or distributed computer systems this causes both increased maintenance costs and possible suboptimal predictor performance. On the other hand, learning schemata such as genetic algorithms have been successfully introduced to automatically derive rules and settings in such tools [3]. The traditional implementations of such genetic fuzzy systems still require fairly large historical data sets, which limits their usage for fully automated demand prediction to a certain type of computing environments.

This paper describes a technique for semi-automatic definition and tuning of fuzzy logic predictors in the scenario of scarce training data. The proposed approach uses as a basis genetic algorithms to generate potential fuzzy logic rule sets. In order to produce adequate and non-overfitted models even with

little data, it allows for incorporating of prior knowledge in a flexible way into the predictor design. The amount of this prior knowledge can be thus chosen accordingly to the amount of training data, availability of a domain expert and the time/cost budget devoted to predictor design. This is achieved by the following measures:

- fine-granular specification of the complexity of predictor family from which potential solutions are chosen,
- flexible specification of the fuzzy rules structure for the sought predictor (and thus opportunity of incorporating prior expert knowledge).

An additional means for dealing with scarce data is the periodical performance comparison of the deployed and several “2nd class” candidate designs in the production scenario, and potential re-selection of the active solution. Since having scarce model data always bears the danger of overfitting or improper design (especially if the family of potential solutions is too large [14]) the proposed approach allows for fine-granular control over of the size of this family via specification of potential number of rules, their structure, and degree of variation.

We discuss an application of this technique for prediction of computation demand of 41 servers from a business data center. We simulate the scenario of scarce training data by evaluating the prediction accuracy of our method for different lengths of the training interval in walk-forward prediction tests (which translated for different amounts of training data). As a reference the accuracy of “traditional” non-linear predictors (J48 and Support Vector Machine SMO [15]) under the same conditions is tested. Even without the re-selection of the active solution, our method shows superior accuracy especially for small amounts of training data, and better accuracy in general. Furthermore, it produces simple, human-readable models, which can be easily checked or improved by a domain expert.

II. RELEVANCE OF PREDICTION FOR UTILITY COMPUTING

Utility computing is a still emerging computing model in which resources are not “owned” (and homed) by a consumer, but are used from a provider over a network. The utility computing model promises several advantages. First, on the consumer side, capital cost for owning resources can be turned into operational cost of using resources, which is preferable for most businesses. Consumers are also relieved of burdens of creating and maintaining an IT infrastructure on their own. Second, on the provider side, providers can pool multiple consumers in their infrastructures achieving higher utilization and returns than in individual deployments. Utility providers can also centralize management and leverage on volume purchases of larger infrastructures. Over all, efficiency of IT infrastructure is expected to increase on both sides with a utility computing model, the consumer and the provider side.

Today, the utility computing model is hardly reality for a variety of reasons. For consumers, there are hurdles in terms of security of critical data and dependencies of their application and data sets on physical IT infrastructure. Since IT is vital to

the business, outsourcing IT tasks (or in-sourcing IT resources) is substantially harder to implement than it is for other business functions. The lowest barrier for utility computing models is for resources that do not carry state, such as networks. On the provider side, the risk for a successful utility business model is also hardly understood. However, large IT providers exploring the utility computing space as a serious alternative to conventional models in IT. Hewlett-Packard, for instance, launched a six-year effort to build a utility computing provider business, which offers consumers access to dedicated or shared computing resources at variable prices. Several large data center facilities are currently being built from where access to basic IT infrastructure is offered, including server, storage, networks, at a price range from 55 cents to \$1.50 per CPU hour based on the configuration [7].

Virtualization of resources is critical for pooling resources among multiple consumers. While consumers still see “their” resources in a virtualized environment, their mapping into underlying physical resources can be adjusted by the utility provider. Adjustments may occur when workload patterns vary between consumers, such as caused by different work times and/or time zones, seasonal patterns, etc. The utility provider can leverage those differences and provide (and sell) more resources than he actually has. The principle is the same that is used in other domains such as in form of virtualized memory where the sum of all allocated virtual memory on a machine can exceed its physical memory. This kind of over-subscribing resources, enabled by virtualized resources, is a major source of efficiency for the utility computing model.

Over-subscribing, however, has a counter force in meeting the commitments the utility provider has made to consumers. It makes use of the fact that consumers will not fully use their committed resource capacity all the time. As long as the aggregate resource consumption of all consumers (the working set) remains below the physically available resource capacity, the scheme of over-subscribing works.

The concept of over-subscribing resources in a utility computing environment depends on predictions of the use of resources by consumers and managing the chance/risk of oversubscriptions to succeed or fail. Instead of over-subscribing resources “in large amounts”, where risks of failure are hard to predict, a more manageable approach is to control the oversubscription degree in dependence of the anticipated resource usage.

The targeted application scenario of the investigation presented in this paper is the assignment of compute-jobs to machines. “Virtualization” is factored into this model by assuming that compute jobs can be deployed on any machine in the system. Each decision of job deployment is based on the prediction of the workloads caused by the (virtualized) compute jobs already deployed on the machine. In this way the risk of a job failure due to the overutilization of the resource is reduced. The scheduling approach is similar to the one presented in [12]. Since this work focuses primarily on

prediction, we leave out the details of this resource management schema.

Virtualization in a utility computing environment enables easy reconfiguration and job migration and thus causes a much higher degree of change in demand behavior as observed in traditional data centers. For example, a physical server might simultaneously host a collection of virtualized servers, each controlled by a utility customer. The latter fact makes it hard for the utility computing operator to collect a consistent long-term demand trace of the virtual server, as its “internals” might change without the knowledge of the operator. A remedy suggested in this paper is to assume an inherently high rate of demand pattern changes, and to find a prediction approach which is robust even in those conditions.

III. DEFINITIONS AND RELATED WORK

Fuzzy logic (FL) is an extension of classical logic which provides an effective conceptual framework for handling decisions under uncertainty and imprecision, especially suitable for a scarce model data scenario. Introduced by Zadeh in 1965 [17], the concept hugely gained popularity when applied by Mamdani and Assilian [9] to design of controllers.

A. Fuzzy Logic Controllers and Predictors

A fuzzy logic predictor/controller (FLC) introduced in [9] is essentially a function interpolator based on if/then rules and fuzzy reasoning. The controller inputs (state variables) and outputs (control variables) are modeled by so-called linguistic variables (LVs), which are essentially one-dimensional real-valued variables. For each LV, its value range is covered by several (named) fuzzy sets, each represented by a membership function. The last species maps a LV value to the interval $[0, \dots, 1]$. Its output is interpreted as the degree of membership of the LV value to the underlying fuzzy set. The “shapes” of the fuzzy sets are commonly triangular, trapezoidal, sigmoidal, or Gaussian. Basically, FL allows us to say that a LV value belongs to interval $[a, b]$ with degree of 90% and to interval $[c, d]$ with degree of 30%, instead of having the “crisp” statement that the LV value is in $[a, b]$ and not in $[c, d]$. Based on these concepts, “fuzzy reasoning” via if/then rules can be implemented [9].

The FLCs have been widely applied in context of complex ill-defined processes, especially those which can be controlled by humans without knowing the explicit model [3]. Examples include cement kiln control, information retrieval systems, navigation systems for automatic cars, operation of trains, feature-definition controllers for robot vision, and more. As a FLCs is essentially a multi-valued function, it can be also applied without changes for prediction tasks, as done in our setting.

B. Genetic Fuzzy Systems

Classical design technique for FLCs consists in manual specification of LVs, fuzzy sets, and rules by a domain expert. This tedious and expensive process can be substituted by learning schemata provided that enough data on system responses to different control variable values are known. Two

approaches are prevalent: neuro fuzzy systems [8] and genetic fuzzy systems [3]. Newer techniques of the fuzzy rule generation (for classification) include fuzzy decision trees [10].

In the genetic fuzzy systems the paradigm of a genetic algorithm (GA) is used to optimize the shapes of fuzzy sets and/or the if/then rules. At each time, a set (generation) of potential solutions (genes) consisting of suitable encodings of FLCs is maintained. These solutions are randomly changed (mutated) and merged (via a so-called cross-over operation), yielding a new set of solutions. Subsequently, the fitness of each new solution is evaluated on historical data or by simulations with a system model. The fitness values decide whether the genes survive into the next generation. This process is stopped after a fixed number of iterations or e.g. if the learning rate drops.

A large number of variations to this technique have been summarized in [3]. As the key component, our approach includes the derivation of FLCs via a genetic algorithm, but also integrates this method into a human-supported process of defining an FLC in a scenario of scarce data. In addition, the novelty of our genetic fuzzy system is the fine-granular control of the structure of the generated FLC and control of the pool’s complexity from which the FLC has been chosen. While the former feature helps to incorporate prior knowledge, the latter prevents creation of overfitted solutions - a likely threat in a scarce data scenario.

IV. THE APPROACH

In this section we describe the encoding of an FLC, the details of our GA learning scheme and conclude with the overall process of deriving such a FLC in the scenario of scarce model data.

A. The Structure of the FLC and Encoding Prior Knowledge

The structure of our FLC is based on an open source fuzzy engine implementation by Sazonov [13]. This fairly standard engine allows for multiple clauses on the left hand side of rules, for hedges, and for weighting of rules. The evaluation of each rule yields a firing level, which is saved together with its fuzzy set (fixed at the right hand side of the rule). The final membership function for a particular output LV is a linear combination of the saved fuzzy sets weighted by the firing levels. The defuzzification is done by the centroid computation. The fuzzy sets for each LV are trapezoids, each specified by four real numbers.

The simplified form of a fuzzy rule is the following one:

if C_1 **and|or** C_2 ... **then** C_{out}

where C_1, \dots, C_k , are clauses of the form

$C = (C_a$ **and|or** $C_b)$ with C_a, C_b clauses,

or

$C = LV\text{-label}$ **is** *hedge-list* *fuzzy-set*,

and C_{out} can only take the form in the above line.

Here the *fuzzy-set* is one of the fuzzy sets previously defined for the LV *LV-label*, and *hedge-list* is an

optional list of hedges (modifiers of the membership function values) coming defined as:

not: $X_{\text{not}} = 1 - X$
very: $X_{\text{very}} = X^2$
somewhat: $X_{\text{somewhat}} = \text{sqrt}(X)$.

An exemplary rule is the following one:

if distance **is** negLarge **and** (angle **is** negLarge **or** angle **is** posLarge) **then** power **is** negMedium,

with distance and angle being input LVs, and power an output LV.

We have extended this rule description schema for the purpose of describing the *families of FLCs* in the following way:

- each rule is substituted by a named *set of rule patterns* S_i ,
- each rule pattern has same syntax as in the original engine, but instead of concrete labels of the LVs and of their fuzzy sets, we use *references to named collections of LVs* and *labels of collections of fuzzy sets*, respectively,
- analogously, the concrete lists of hedges are substituted by *labels of collections of composite hedges* (i.e. “final” lists of hedges),
- each LV can be either “raw” controller input value, or a *function* of the input values, with arbitrary number of parameters and their bounds; we assume that the input and the function values are normalized to 0..1 (example: normalized moving average of an input with the averaging length as a parameter),
- optionally, we allow a specification of the equidistant trapezoidal fuzzy sets by specifying their number, width, and min/max values of the covered interval.

An example should illustrate the principle. Assume that we have as a starting point four input LVs A, B, C , and D , in each case with fuzzy sets *low, mid, high*, and an output LV Z with fuzzy sets *neg* and *pos*. We want to specify a family of FLCs with two rules each. For the first rule, we specify as the set S_1 of rule patterns:

- **if** refa **is** rim **then** refz **is** one
- **if** refb **is** last **then** refz **is** one
- **if** refc **is** hedgeSetA rim **and** (refb **is** one **or** refb **is** two) **then** refz **is** one.

For the second rule, the set S_2 of rule patterns is used:

- **if** refa **is** rim **then** refz **is** nHedge one
- **if** refb **is** one **then** refz **is** last
- **if** refc **is** all **or** refb **is** two **then** refz **is** last.

Hereby *refa* is a reference to a LV from the set $\{A, B\}$, *refb* is a reference to a LV from the set $\{C, D\}$, *refc* is a reference to a LV from $\{A, B, C, D\}$, and finally *refz* is a reference to an LV from $\{Z\}$, i.e. always to Z . The collections of fuzzy sets are

derived as follows (assuming some order on the fuzzy sets of each LV):

1. *one, two, last* denote the first, second, and the last fuzzy set, respectively,
2. *rim* denotes one among the first or the last fuzzy set, and *all* represents any possible fuzzy set.

The label *hedgeSetA* denotes one of the atomic or composite hedges from the set $\{\text{somewhat}, \text{very}, \text{not_very}\}$, with *not_very* being the composite hedge “not very”. The label *nHedge* refers to the singleton $\{\text{not}\}$.

To construct a concrete FLC from this family, we have to take the following decisions:

1. Select a rule pattern from each of S_1, S_2 .
2. For each of the references *refa, ..., refz*, select a LV from the appropriate collection. This selection is done once for the whole FLC, i.e. each occurrence of a reference has the *same* LV binding.
3. For each *occurrence* of a labelled collection of fuzzy sets, select an allowed fuzzy set, i.e. two different occurrences of the collection *rim* might have different selected fuzzy sets.
4. Do the analogous step as in 3. for the labelled collections of composite hedges.
5. For each used LV which is a function of the input, select the values of parameters in the allowed bounds.
6. Specify the shapes of fuzzy sets for each LV either by “hard coding” them, or by defining their number and width for a normalized value interval.

An exemplary FLC contained in the above family is the rule set

- **if** A **is** high **then** Z **is** neg
- **if** C **is** mid **or** D **is** mid **then** Z **is** pos.

A family of FLCs defined in this way represents all possible solutions of a genetic algorithm for learning an FLC. Simultaneously, the above process of selecting a family member translates almost directly into an encoding of a gene: it contains the information about the choices made in the above described decision process. This gene encoding is significantly different from the proposed encodings of the FLCs as genes [3], with our approach having finer control on the possible final forms of each rule. Simultaneously, many other encoding approaches are generalized, as we can generate almost any rule (by specifying all “reasonable” combinations of clauses as S_i using references to sets of *all* LVs, and making no restrictions on the hedge collections or fuzzy sets etc.). Finally, note that by fixing all settings except the function parameter values our encoding directly translates to a “traditional” GA-based optimization of purely numerical values. The disadvantages of our approach are the fixed number of final rules, the need for manual (yet one-time) generation of object collections (e.g. LVs, hedges), and need for possible adjusting the rule patterns between different rule pattern collections.

The fine granular control of the rule sets is intended to encode prior knowledge in a flexible way, ranging from fixed rules to

huge families of rules with all intermittent levels. This helps an expert to incorporate his knowledge without unnecessarily restricting possibilities for rules where she lacks deeper knowledge. In the scenario of scarce data, it is advisable to keep the complexity of FLCs small in order to avoid overfitting by restricting the number of rules, their size, and the sets of LVs.

B. The Genetic Algorithm Learning Scheme

The encoding of a single gene reflects the structure of the FLC family description:

- for each set of rule patterns, we specify which rule pattern is active,
- each reference to a LV is mapped to a concrete LV,
- each occurrence of a label of a fuzzy set collection is mapped to a particular fuzzy set; analogous applies to hedges,
- the function parameter values, if any, are attached to the LV representing this function.

In the current implementation we do not change the number and the shapes of the fuzzy sets of each LV during the GA run. To avoid problems with the ranges of the inputs or functions, we normalize their values into the interval $[0..1]$ using the SoftMax method [11].

The initial population of the GA consists of randomly chosen members of the FLC family. The mutation is done by random changes of every encoded part of the gene. For example, we can substitute a rule pattern from S_i by another one (with possible initialization of references to LVs with random choices etc.); a less material mutation changes one of the fuzzy sets in a rule, or the parameter values of a function. For each of these mutation types, separate probability values can be set.

The crossover operates for each rule position i in the parent genes in the following way:

- for different rule patterns (at “rule position” i), the rule patterns are swapped with certain probability (and LV references new to this gene are initialized to defaults),
- in the other case, only the “subobjects” (references to LVs, hedge and fuzzy set instances) of the rule pattern are swapped with certain probability between the identical rule patterns of the parent genes.

The gene evaluation metric is application dependent (see next section), yet evaluates the whole FLC without differentiating individual rules. In the current implementation, we have fixed number of generations while monitoring the learning rate. In the future, we intend to use convergence-based stopping criteria.

C. The Process of Obtaining Fuzzy Logic Predictors

The complete process of obtaining FLCs from scarce data is outlined by the following steps:

1. Encoding of prior knowledge and FLC family size in form of sets of rule patterns, sets of LVs, and the lists of membership functions for each LV.

2. Running the genetic algorithm in order to obtain a ranked list of potential solutions (rule sets).
3. A selection of the best solution according to the fitness values of the solutions.
4. Deployment of the best rule set in the “production” environment and monitoring its performance together with the potential performance of the “2nd class” solutions.
5. Periodical comparison of the performance results, with potential re-selection of the active rule set from the list of all monitored solutions (i.e. the deployed one and the “2nd class” solutions).

This process contains the (infrequent) manual step 1., all others being automated. In step 1. a domain expert must define functions of the input (equivalent to LVs), group them into sets, and perform an analogous step for the rule patterns. Optionally, in step 3., a human expert has a chance to modify manually the subsequently deployed solutions. This allows for using the solutions proposed by the GA as mere “inspirational basis” while final solutions are created by the expert.

V. EVALUATION SCENARIO

To evaluate our approach we predict computational demand of each of 41 servers of a business data center. The prediction is done 30 minutes “into the future”; other lag values yielded similar results. We do not predict a numerical value of the demand, but only the anticipated level of the demand among the following five: 0%-20%, 20%-40%, 40%-60%, 60%-80%, and 80%-100% (per cent of the server capacity). A finer degree of quantization is not necessary for the application scenario (and the prediction errors are likely to overwhelm the level size). The data has been collected using HP MeasureWare (Openview Performance Agent) between July 29th, 2001 and September 2nd, 2001 by sampling in 5 minute intervals. For more details see [1] and [12].

In our experiment we created sets of fuzzy rules which attempted to predict computational demand of a server based on the past values of this demand, as well as on the historical memory and disk usage values. The target variable (computational demand) has been normalized by the SoftMax method [11] and then discretized into the levels listed above. Also all the other data has been normalized into the interval $[0,..,1]$ prior to further processing. Subsequently, a collection of normalized smoothing functions (simple moving averages with different lengths) of this data has been computed. These functions constituted the collection of the LVs, together with some calendar functions, e.g. hour of the day, day of the week etc. To each LV we attached five trapezoidal fuzzy sets which cumulatively covered the range $[0,..,1]$. As the collections of the rule patterns we used the sets S_1 , S_2 described above, and some other simple rule patterns. There are exactly four rules per predictor, which allows only for simple models and avoids overfitting when training data is scarce. With 60 initial (random) genes and at least 10 genes per generation, the best solutions have been usually found well before the maximal generation (set to 20).

We run all experiments in the walk-forward mode. In this iterative process data from an interval of a fixed size (training interval size s) is used for training, and the model predictions

are evaluated against the true values on the subsequent test interval. In the next iteration, both the training and test intervals are moved “forward” by the size of the test interval, and the process is repeated until no more data is available. We use a fixed test interval size (2 days, or 576 samples), and change the training interval sizes. The effects of scarce training data are simulated by reducing the training interval size s (and thus the amount of training data). The values for s are: 1h, 3h, 6h, 12h, 24h, 36h, 3 days (3x24h), 6 days, 12 days and 124 days. Note that we do not need to know whether the used data is dynamic or rather stationary: a predictor demanding a lot of training data is likely to fail for a small values of s in any case. On the other hand, a predictor requiring only little training data should also work in the non-stationary case. The evaluation criterion for the accuracy is the mean squared error (MSE) of the prediction.

Figure 1 shows the dependence of the MSE (smaller values are better) from the training interval size for our method (GaFuzzy, left), and the Support Vector Machine SMO (right) for four typical servers. As for the GaFuzzy, smallest errors occur in the 12h - 36h range of the training interval length, and seem to increase with larger interval sizes. We explain this by the fact that with increased training data sizes several demand

patterns occur, and the simple model of GaFuzzy (only four rules) is not able to convey their whole complexity. Here a manual correction of the model complexity could improve the predictor. The SMO predictions tend to become better with increasing amount of training data, and capture best the cases with most training data. However, in exception of one trace they perform worse than GaFuzzy.

The latter phenomenon is also shown in Figure 2. Here we show the ratio of the MSE for SMO (left) and a decision algorithm J48 (right) to the MSE of GaFuzzy (higher values are in favor of our method) for a fixed training interval of 3 days (all 41 servers). The decision tree algorithm performs better than SMO, however still in most cases (blue columns) GaFuzzy gives more accurate predictions.

Finally, Figure 3 shows a direct comparison of the MSE between GaFuzzy, J48 and SMO for all training interval lengths (two selected servers). For server A, the GaFuzzy algorithm performs best, while for server B this is only the case for training interval sizes smaller than 36 hours: for 36 hours and more, the J48 algorithm is most accurate.

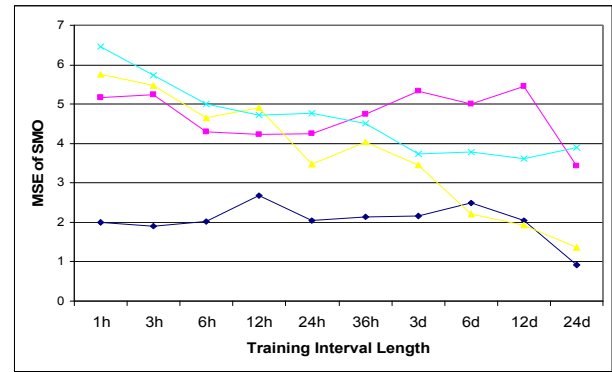
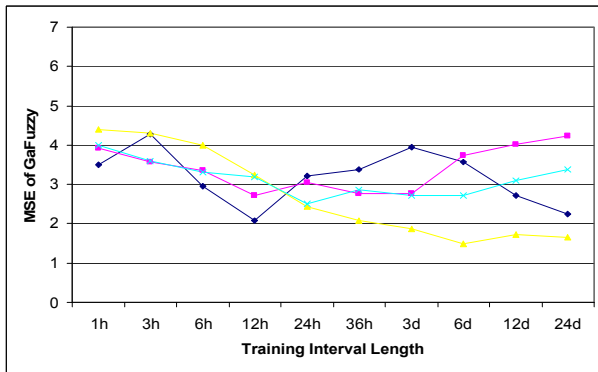


Figure 1: MSE of the GaFuzzy (left) and MSE of the SMO (right) for different training size (four selected servers)

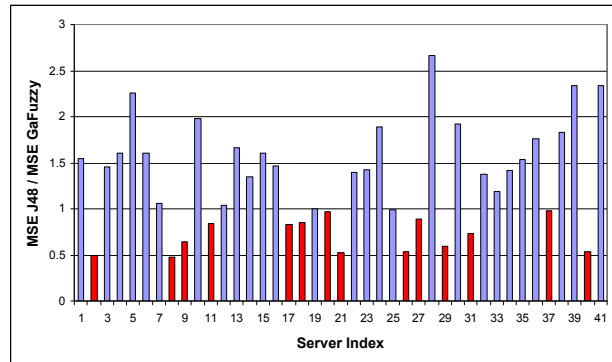
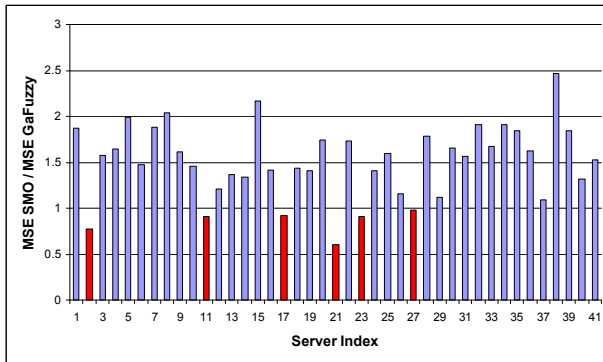


Figure 2: Ratio of the SMO error (left) or the J48 error (right) to the GaFuzzy error for training interval size of 3 days

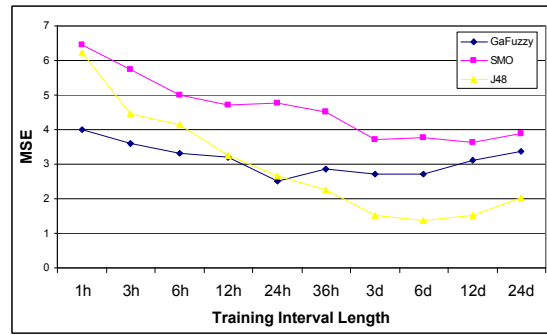
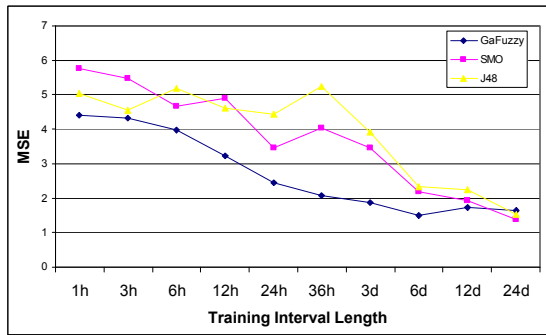


Figure 3: Comparison of the MSE for GaFuzzy, J48 and SMO (server A: left, server B: right)

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a novel technique for semi-automatic generation of predictors based on genetic fuzzy systems in a scenario of scarce data. The benefits of this approach include a flexible encoding of the prior knowledge, human readable and compact predictor

representation, and robust prediction even with small training data sets. The latter aspect makes our approach especially suitable for dynamic computer systems and networks such as utility computing environments - a scenario which motivated this development.

Future and ongoing work will be the refinement of our gene evaluation method, and introducing optimization of the fuzzy

REFERENCES

- [1] A. Andrzejak and M. Ceyran, "Characterizing and Predicting Resource Demand by Periodicity Mining", *J. Network and System Management*, Vol. 13, No. 2, June 2005.
- [2] A. Andrzejak, P. Domingues, and L. Silva, "Predicting Machine Availabilities in Desktop Pools", accepted to NOMS 2006.
- [3] O. Cordón, F. Herrera, "General Study on Genetic Fuzzy Systems", in *Genetic Algorithms in Engineering and Computer Science*, G. Winter, J. Periaux, M. Galan, P. Cuesta (Eds.), John Wiley and Sons, 1995, 33-57.
- [4] Y. Diao, J. L. Hellerstein, and S. Parekh, "Using fuzzy control to maximize profits in service level management", *IBM Systems Journal*, Volume 41, Number 3, 2002.
- [5] P. A. Dinda, "A Prediction-Based Real-Time Scheduling Advisor". *IPDPS 2002*.
- [6] S. Graupner, N. Cook, D. Coleman, T. Nitzsche, "Platform for Delivering IT Management Services", *COMSWARE 2006*, New Delhi, India, January 8-12, 2006.
- [7] Hewlett-Packard, "HP Launches Formal Utility Computing Service", *Information Week*, Nov 2005.
- [8] A. Joshi et.al, "On Neurobiological, Neurofuzzy, Machine Learning and Pattern Recognition Techniques", *IEEE Transaction on Neural Networks*, vol. 8, no.1, pp. 18-31, Jan.1997.
- [9] E. H. Mamdani, S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller", *Int. Journal of Man-Machine Studies* 7(1): 1-13 (1975).
- [10] C. Olaru, and L. Wehenkel, "A complete fuzzy decision tree technique", *Fuzzy Sets Syst.* 138, 2 (Sep. 2003), 221-254.
- [11] D. Pyle, "Data preparation for data mining", Morgan Kaufmann Publishers Inc., 1999.
- [12] J. Rolia, A. Andrzejak and M. Arlitt, "Automating Enterprise Application Placement in Resource Utilities", 14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2003), Heidelberg, October 2003.
- [13] E. S. Sazonov, "Open source fuzzy inference engine for Java", <http://www.intelligent-systems.info/FuzzyEngine.htm>, 2002.
- [14] V. Vapnik, "Statistical Learning Theory", Wiley, New York, 1998.
- [15] I. H. Witten and F. Eibe, "Data Mining - Practical machine learning tools with Java implementations", Morgan Kaufmann Publishers, San Francisco, 2000.
- [16] R. Wolski, "Forecasting network performance to support dynamic scheduling using the Network Weather Service". *Proc. of HPDC 1997*, pages 316-325, 1997.
- [17] L. A. Zadeh, "Fuzzy Sets", *Information and Control*, 8(1965) 338-353.