A Fuzzy Approach to Cloud Admission Control for Safe Overbooking

Carlos Vázquez[1], Luis Tomás[2], Ginés Moreno[1], and Johan Tordsson[2]

[1] Dept. of Computing Systems,
University of Castilla–La Mancha, Spain
{Carlos.Vazquez,Gines.Moreno}@uclm.es
[2] Dept. of Computing Science,
Umeå University, Sweden
{luis,tordsson}@cs.umu.se

Abstract. Cloud computing enables elasticity - rapid provisioning and deprovisioning of computational resources. Elasticity allows cloud users to quickly adapt resource allocation to meet changes in their workloads. For cloud providers, elasticity complicates capacity management as the amount of resources that can be requested by users is unknown and can vary significantly over time. Overbooking techniques allow providers to increase utilization of their data centers. For safe overbooking, cloud providers need admission control mechanisms to handle the tradeoff between increased utilization (and revenue), and risk of exhausting resources, potentially resulting in penalty fees and/or lost customers. We propose a flexible approach (implemented with fuzzy logic programming) to admission control and the associated risk estimation. Our measures exploit different fuzzy logic operators in order to model optimistic, realistic, and pessimistic behaviour under uncertainty. The application has been coded with the MALP language by using the FLOPER system developed in our research group. An experimental evaluation confirm that our fuzzy admission control approach can significantly increase resource utilization while minimizing the risk of exceeding the total available capacity.

Keywords: Cloud Computing, Admission Control, Fuzzy Logic Programming, Resource Utilization, Risk Assessment.

1 Introduction

Cloud computing is a recently emerged paradigm where computational resources are leased over the Internet in a self-service manner under a pay-per use pricing scheme. Organizations and individuals, the cloud users, can thus continuously adjust their cloud resource allocations to their current needs, so called elasticity [21]. The core of cloud infrastructure are data centers, large store-house like facilities hosting hundreds of thousands of servers, along with storage and networking equipment, as well as advanced systems for cooling and power distribution [24]. Through virtualization technologies, these data centers (cloud providers) can provision applications from multiple users on the same physical servers and thus make efficient use of their hardware. In cloud data centers, user applications are packaged as Virtual Machines (VMs) [7], which in essence

F. Masulli, G. Pasi, and R. Yager (Eds.): WILF 2013, LNAI 8256, pp. 212–225, 2013.

are software implementations of servers that are time-shared on the physical hardware. Users can thus at any time, themselves or through automatic elasticity management software, increase or decrease the number of VMs allocated. Consequently, it is common for for cloud providers to require users to specify upper and lower limits to the number of VMs to be used in a *service request* [19], or to simply have predefined rules for all users, e.g., 1-20 VMs per data center for the largest cloud provider, Amazon [6].

For data centers, elasticity results in a long-term capacity allocation problem, as the exact number of VMs to be used at any time by each user is unknown. Running too few VMs in total results in poor data center hardware utilization and lowered incomes from users, whereas having too many VMs may lead to low performance and/or crashes, poor user experience, and may also have financial consequences if Service Level Agreements (SLAs) regarding user performance expectations are violated. To handle this trade off, *admission control* mechanisms [9] can be used by cloud providers to determine whether a new user service request should be admitted into the data center or not. In our previous work [22], we demonstrate how *resource overbooking*, a technique well-known from airline revenue management and network bandwidth multiplexing, can be used to increase provider utilization and revenue, with acceptable risks of running out of hardware capacity. Further examples of previous work in this area includes an algorithmic framework [9] that uses cloud effective demand to estimate the total physical capacity required for performing the overbooking, including probability of launching additional VMs in the future.

However, evaluating risk during admission control with respect to performing resource overbooking actions is far from trivial. Overbooking and the associated scheduling problems are multi-dimensional packing problems, commonly solved using heuristics. It is also not clear in the general case how to balance the short and long term impact when deciding whether to accept a new service. Furthermore, admission control is associated with several uncertainties, include limited knowledge of future workloads, potential side effects from co-locating particular VMs, and exact impact on applications of potential resource shortage. Based on these properties of the admission control problem, we propose a fuzzy approach to admission control. Since its initial development by L. A. Zadeh in the sixties [23], fuzzy logic has become a powerful theoretic tool for reaching elegant solutions to problems in various fields of software, industry, etc. More recently, there exist fuzzy extensions of the classical logic language Prolog, which can be used in a very natural way to solve problems where fuzzy logic plays an important role. A conceptual overview of how our cloud overbooking framework use fuzzy logic during admission control is shown in Figure 1. In must be noted that the risks are calculated for the three capacity dimensions that we consider for each VM: *CPU*, *memory* and *I/O*. For each one of these, the risk is calculated based on predicted information about future available capacity (referred to as *Free* in the rest of the paper), future amount of unrequested capacity (denoted *Unreq*) and the capacity requested by the incoming service (denoted *Req*). *Unreq* is the inverse difference between what users requested and what they really used (*Free*). All these future expected values are predicted by using exponential smoothing functions [22].

The structure of this paper is as follows. In Section 2, a brief introduction to the MALP (*Multi-Adjoint Logic Programming*) language and the FLOPER system is given.

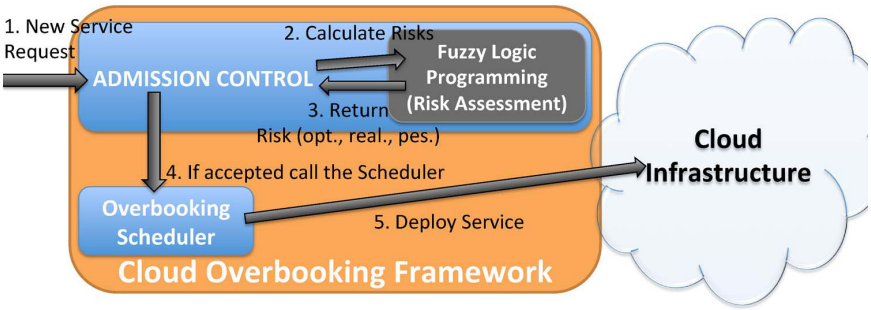


Fig. 1. Conceptual picture of the system

In Section 3 we explain the main features of our implementation based on fuzzy logic programming using MALP and FLOPER. Next, in Section 4, we present our experimental results. Finally, Section 5 concludes the paper and outlines directions for further research.

2 The Multi-adjoint Logic Language and FLOPER

Multi-Adjoint Logic Programming (see [14,11] for a complete formulation of this framework), MALP in brief, can be thought as a fuzzy extension of Prolog and it is based on a first order language, $\mathcal{L}$, containing variables, function/constant symbols, predicate symbols, and several connectives such as implications ($\leftarrow_1, \leftarrow_2, \ldots, \leftarrow_m$), conjunctions ($\&_1, \&_2, \ldots, \&_k$), disjunctions ($\vee_1, \vee_2, \ldots, \vee_l$), and general hybrid operators ("aggregators" $@_1, @_2, \ldots, @_n$), used for combining/propagating truth values through the rules, and thus increasing the language expressiveness. Additionally, our language $\mathcal{L}$ contains the values of a *multi-adjoint lattice* in the form $\langle L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n \rangle$, equipped with a collection of *adjoint pairs* $\langle \leftarrow_i, \&_i \rangle$ where each $\&_i$ is a conjunctor intended to the evaluation of *modus ponens* [20,12,14]. A *rule* is a formula "$A \leftarrow_i \mathcal{B}$ with α", where A is an atomic formula (usually called the *head*), $\mathcal{B}$ (which is called the *body*) is a formula built from atomic formulas $B_1, \ldots, B_n$ ($n \geq 0$), truth values of L and conjunctions, disjunctions and general aggregations, and finally $\alpha \in L$ is the "weight" or *truth degree* of the rule. The set of truth values L may be the carrier of any complete bounded lattice, as for instance occurs with the set of real numbers in the interval $[0, 1]$ with their corresponding ordering $\preceq_R$. Consider, for instance, the following program, $\mathcal{P}$, with associated multi-adjoint lattice $\langle [0, 1], \preceq_R, \leftarrow_{\mathtt{P}}, \&_{\mathtt{P}} \rangle$ (where label $\mathtt{P}$ means for *Product logic* with the following connective definitions for implication and conjunction symbols, respectively: "$\leftarrow_{\mathtt{P}}(x, y) = \min(1, x/y)$", "$\&_{\mathtt{P}}(x, y) = x * y$", as well as "$@_{\mathtt{aver}}(x, y) = (x + y)/2$"):

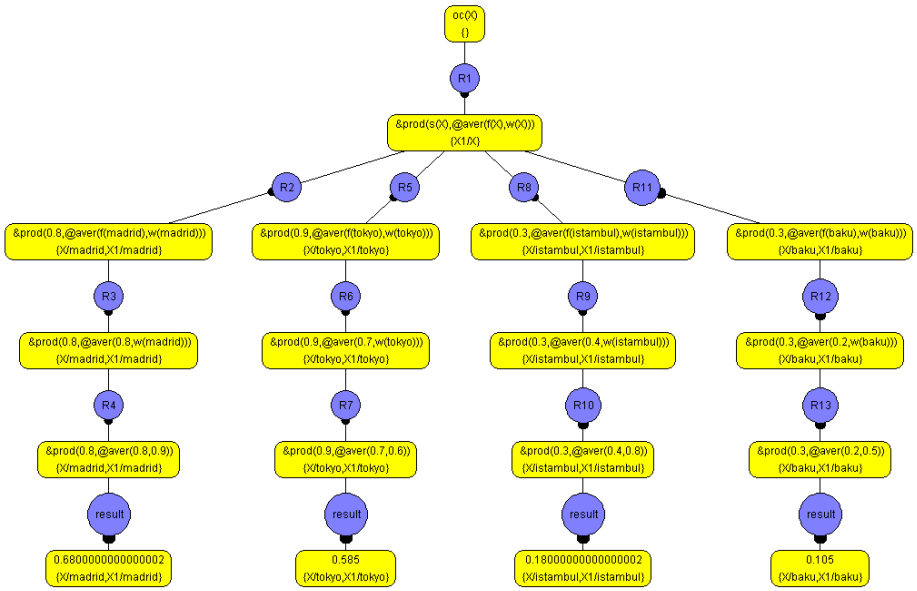


Fig. 2. Execution tree for program $\mathcal{P}$ and goal `oc(X)`

$$
\begin{array}{llllll}
\mathcal{R}_1: & oc(X) & \texttt{<-} & s(X)\ \&prod\ (f(X) & @aver\ w(X))\ with\ 1. \\
\mathcal{R}_2: & s(madrid) & with\ 0.8. & \mathcal{R}_5: & s(tokyo) & with\ 0.9. \\
\mathcal{R}_3: & f(madrid) & with\ 0.8. & \mathcal{R}_6: & f(tokyo) & with\ 0.7. \\
\mathcal{R}_4: & w(madrid) & with\ 0.9. & \mathcal{R}_7: & w(tokyo) & with\ 0.6. \\
\mathcal{R}_8: & s(istambul) & with\ 0.3. & \mathcal{R}_{11}: & s(baku) & with\ 0.3. \\
\mathcal{R}_9: & f(istambul) & with\ 0.4. & \mathcal{R}_{12}: & f(baku) & with\ 0.2. \\
\mathcal{R}_{10}: & w(istambul) & with\ 0.8. & \mathcal{R}_{13}: & w(baku) & with\ 0.5.
\end{array}
$$

This program models, through predicate "`oc`", the chances of a city for being an "olympic city" (i.e., for hosting olympic games). Predicate "`oc`" is defined in rule $\mathcal{R}_1$, whose body collects the information from three other predicates, "`s`", "`f`" and "`w`", modeling, respectively, the *security* level, the *facilities* and the good *weather* of a certain city. These predicates are defined in rules $\mathcal{R}_2$ to $\mathcal{R}_{13}$ for four cities (*Madrid*, *Istambul*, *Tokyo* and *Baku*), in such a way that, for each city, the feature modeled by each predicate is better the greater the truth value of the rule.

In order to run and manage MALP programs, during the last years we have designed the FLOPER (*Fuzzy LOgic Programming Environment for Research*) system [16,15,17,18], which is freely accessible online [10]. The parser of our tool has been implemented by using the classical DCG's (*Definite Clause Grammars*) resource of the Prolog language, since it is a convenient notation for expressing grammar rules. Once

the application is loaded inside a Prolog interpreter, it shows a menu which includes options for loading/compiling, parsing, listing and saving fuzzy programs, as well as for executing/debugging fuzzy goals. These actions are based on the *translation* of the fuzzy code into standard Prolog code: all internal computations (including compiling and executing) are pure Prolog derivations, whereas inputs (fuzzy programs and goals) and outputs (fuzzy computed answers) have always a fuzzy taste, thus producing the illusion on the final user of being working with a purely fuzzy logic programming tool.

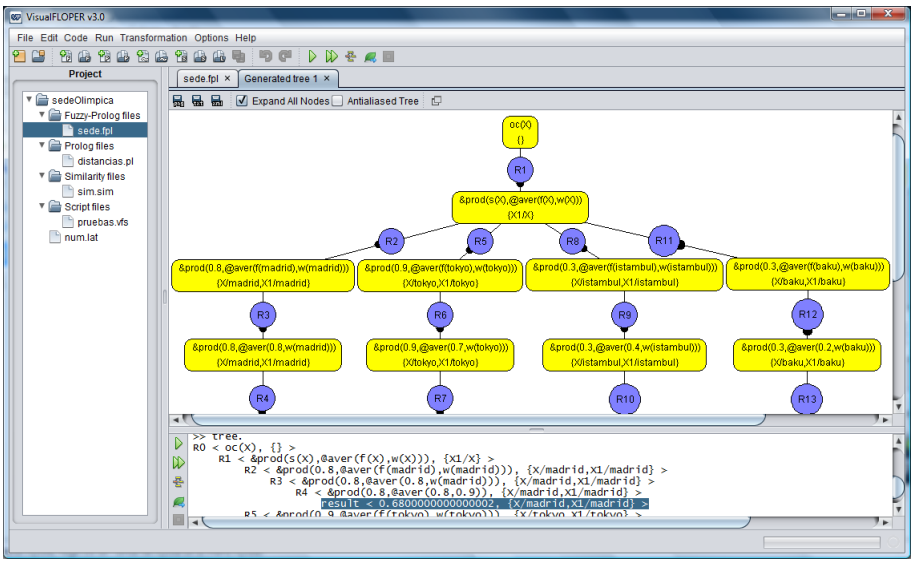


Fig. 3. The FLOPER System showing the execution tree for goal "`oc(X)`"

The FLOPER system is able to manage programs with very different lattices. By using option "`lat`" (and "`show`"), we can associate (and display) a new lattice to a given program. Such lattice must be loaded into the tool as a pure Prolog program. As an example, the following clauses show the program modeling the lattice of the real interval $[0, 1]$ with the usual ordering relation and connectives (conjunction and disjunction of the *Product logic*, as well as the average aggregator) where the meaning of the mandatory predicates "`member`", "`top`", "`bot`" and "`leq`" is obvious:

```
member(X):- number(X), 0=<X, X=<1.          bot(0).
leq(X,Y):- X=<Y.                            top(1).
and_prod(X,Y,Z) :- Z is X*Y.
or_prod(X,Y,Z)  :- U1 is X*Y, U2 is X+Y, Z is U2-U1.
agr_aver(X,Y,Z) :- U1 is X+Y, Z is U1/2.
```

FLOPER includes two main ways for evaluating a goal, given a MALP program and its corresponding lattice. Option "`run`" translates the whole program into a pure Prolog program and evaluates the (also translated) goal, thus obtaining a list of fuzzy computed

answers, each one containing the truth-degree and the corresponding variable substitution for each concrete solution. For instance, in our example we can run goal "`oc(X)`" to obtain the following result indicating that the different chances of *Baku*, *Istambul*, *Tokyo* and *Madrid* for being "Olympic cities" are respectively 10.5%, 18%, 58.5% and 68%:

```
>> run.
   [Truth_degree=0.105,X=baku]
   [Truth_degree=0.180,X=istambul]
   [Truth_degree=0.585,X=tokyo]
   [Truth_degree=0.680,X=madrid]
```

On the other hand, option "`tree`" computes and displays the whole execution (or derivation) tree for the intended goal. Moreover, it is possible to select the deepest level to be built (which is obviously mandatory when trees are infinite) via option "`depth`" or even to indicate that only the set of leaves be displayed via option "`leaves`". Coming back again to our example, we can use option "`tree`" to obtain the execution tree for goal "`oc(X)`", which is generated by FLOPER in three different formats. Firstly the tree is displayed in graphical mode, as a PNG file, as shown in Figures 2 and 3. The tree is composed by two kinds of nodes. Yellow nodes represent states reached by FLOPER following the state transition system that describes the operational semantics of MALP [14]. The root node represents the first state (composed by the original goal together with the identity substitution), and subsequent lower nodes are its children states (that is, states reached from the root). A state contains a formula in the upper side and a substitution (obtained after composing all substitutions applied from the original goal to the current state) at the bottom. A final state, if reached, is a fuzzy computed answer whose associated formula is just an element (truth-degree) of the lattice. Blue rounded nodes appearing between a pair of yellow nodes (states) represent program rules; specifically, the program rule that is exploited in order to go from one state (the upper one) to another (the lower state). These rules are named with letter "`R`" plus its position in the program. For example, observe that from the initial state to the next one, the first rule of the program has been exploited, as shown in the blue intermediate node. As an exception, when all atoms have been exploited in (the formula of) a certain state, the following blue node is labeled with word "`result`", informing that the next state contains a fuzzy computed answer.

FLOPER can also generate the execution tree in two textual formats. The first one contains a plain description of the tree, while the second one provides an XML structure to that description. In this XML format, tag "`node`" is used to include all the information of a node, such as the rule performed to reach that state (tag "`rule`"), the formula of the state (tag "`goal`"), the accumulated substitution (tag "`substitution`") and the children nodes in a nested way (tag "`children`"). These XML files can be accurately explored with the *Fuzzy XPath* application we have recently developed in our research group with FLOPER [1,2,4,5], in order to perform some interesting debugging tasks with the same tool, as documented in [3].

$$\&_{\mathtt{P}}(x,y) \triangleq x * y \qquad |_{\mathtt{P}}(x,y) \triangleq x + y - x * y \qquad \leftarrow_{\mathtt{P}}(x,y) \triangleq \min(1, x/y)$$

$$\&_{\mathtt{G}}(x,y) \triangleq \min(x,y) \qquad |_{\mathtt{G}}(x,y) \triangleq \max\{x,y\} \qquad \leftarrow_{\mathtt{G}}(x,y) \triangleq \begin{cases} 1 & \text{if } y \leq x \\ x & \text{otherwise} \end{cases}$$

$$\&_{\mathtt{L}}(x,y) \triangleq \max(0, x+y-1) \qquad |_{\mathtt{L}}(x,y) \triangleq \min\{x+y, 1\} \qquad \leftarrow_{\mathtt{L}}(x,y) \triangleq \min\{x-y+1, 1\}$$

Fig. 4. Fuzzy conjunction, disjunction, and implication connectives from *Łukasiewicz* (pessimistic), *Gödel* (optimistic), and *Product* (realistic) logics, resp., defined in the real unit interval

3 Implementation Based on Fuzzy Logic Programming

On towards fuzzy formulations of the admission control problem, in this section we present a flexible method that has been implemented MALP using FLOPER. As we have just detailed in the previous section, the MALP language represents a fuzzy extension of the popular Prolog language in the field of pure (crisp) logic programming [13]. In this fuzzy declarative framework, each program is accompanied with a lattice for modeling truth-degrees beyond the simpler case of the (crisp) *Boolean* pair $\{true, false\}$. Hence, fuzzy program rules can utilize fuzzy connectives defined on such richer lattices for improving the expressive power of classical Prolog clauses. For instance, some standard connective definitions for conjunctions, disjunctions, and implications in the lattice of real numbers in the unit interval $[0, 1]$ are presented in Figure 4, where labels `L`, `G`, and `P` mean respectively *Łukasiewicz logic*, *Gödel logic*, and *Product logic*, with different capabilities for modeling *pessimistic*, *optimistic*, and *realistic scenarios*, respectively.

In our application we use a refined version of such a lattice, as we try to identify the notion of truth-degree with the one for "*overbooking risk along a time period*". This means that instead of single values, our program manipulates lists of real numbers as truth-degrees[1] after analyzing the behaviour's curves representing "*free, unrequested, and requested (CPU/memory/net) resources*" also expressed as input lists to the tool. For instance, if expression "$\&_{\mathtt{P}}(x,y) \triangleq x * y$" refers to the conjunction of *Product logic* for pairs of values, its extended version coping with pairs of lists of values should look like "$\&_{\mathtt{P}}([x_1,\ldots,x_n],[y_1,\ldots,y_n]) \triangleq [x_1 * y_1,\ldots,x_n * y_n]$". In our application this connective can be recursively defined with the following code:

```
and_prod([],[],[]).
and_prod([X|LX],[Y|LY],[Z|LZ]):- Z is X*Y, and_prod(LX,LY,LZ).
```

In the lattice we have also implemented extended versions managing lists of the remaining connectives seen in Figure 4, as well as other connectives like @*append* (for concatenating two lists of numbers), @*show* (which is described afterwards) and the two connectives @*very* and @*approx* (where $@very(x) = x^2$ and $@approx(x) = \sqrt{x}$) known as *linguistic modifiers*. These are useful for fine-tuning the more pessimistic or optimistic shape of the answers produced by our application under this uncertain scenario.

[1] Sometimes accompanied with annotations like max, avg, $peak$ and so on, for readability reasons.

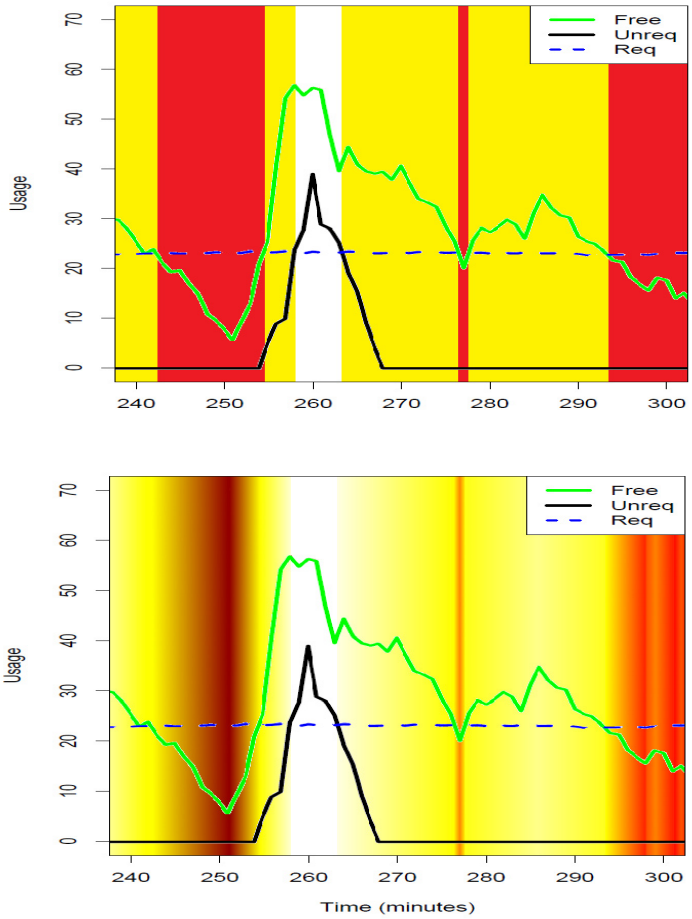


Fig. 5. Graphics showing different choices for estimating risk

Thanks to the high expressive power of the previous lattice, it is possible now to easily design a MALP program composed by a few rules starting with the following one, which receives as input parameters three lists representing the curves associated to free, unrequested and requested values, as well as a fourth argument indicating which resource, or `Field`, (CPU, network or memory) is considered:

```
risk([F|Free],[U|Unreq],[R|Req],Field)<-
            @append(combine(F,U,R), risk(Free,Unreq,Req,Field))
```

This definition of predicate "`risk`" produces a truth degree that is a list of numbers obtained after contrasting the input curves "`Free`", "`Unreq`" and "`Req`". This evaluation is recursively performed by calling predicate "`combine`" with three concrete values each time in order to compare the requested resources with the free and unrequested values.

In Figure 5 we graphically represent two different alternatives to perform this contrast where the background colour moves through white, yellow and red tonalities from the lower to the higher risk found in each instant. A preliminary version for predicate "`combine`" associated to the upper graphic in Figure 5 could be represented by:

```
combine(Free,Unreq,Req) <- (Req>=Free & [1])|(Req>Unreq & [0.5])
```

which, in essence, assigns risk 1 (red band) when the requested resource is over the free value, 0.5 (yellow band) when it is between the free and unrequested values, and 0 (white band) otherwise. Moreover, we have also implemented a more sophisticated version based on linear interpolation (down graphic in Figure 5) according the following formula:

```
inter(Req,Free,Unreq) = (Req-Unreq)/(Free-Unreq)
```

Thus, it can return risk 0 when the requested value is below the unrequested one, a risk in [0,1] (tonalities vary from white through yellow to red as risk grows) if the requested point is between the other two values, and risk above 1 (tonalities vary from red to black as risk grows) if the amount of requested capacity is higher than the free one. When requested is above free, we say that a *peak* emerges, which allows us to improve the evaluation of the final risk by taking into account the performance impact of each peak.

The program is invoked by calling predicate "`main`" with appropriate parameters:

```
main(Free,Unreq,Req,Field) <- @show(risk(Free,Unreq,Req,Field))
```

This rule makes use of connective "`@show`", which receives the truth degree (i.e., a list of numbers) produced by "`risk`" and returns a new truth degree as a list with the following shape:

```
[   avg(n1), min(n2), max(n3),
    over([peak(h1,l1,a1),...,peak(hi,li,ai)]),
    opt(n4), real(n5), pes(n6)                  ]
```

Here, labels "`avg`", "`min`", and "`max`" contain the average (n_1), minimum (n_2), and maximum (n_3) values, respectively, of the input list; "`over`" gives the list of peaks (each one is represented by its maximum height (h_j), length (l_j), and area (a_j)) and finally, "`opt`", "`real`", and "`pes`" labels provide an optimistic (n_4), realistic (n_5), and pessimistic (n_6) estimation -based on the previous elements- about the risk of accepting the requested task. These estimations are produced by combining the average measure (appropriately modulated with the $@approx$ and $@very$ connectives, for referring to the pessimistic and optimistic cases, respectively) together with the disjunctions of all the peaks by using different versions of the disjunction operators. This is modeled according to Łukasiewicz, Product, and Gödel fuzzy logics, as shown in the table of Figure 4, where it is easy to see that $\forall x, y \in [0,1], x|_{\mathtt{L}}y \geq x|_{\mathtt{P}}y \geq x|_{\mathtt{G}}z$. This justifies once again the power of fuzzy logic and the strong expressive resources of MALP for managing pessimistic, realistic and optimistic scenarios. For instance, when we introduce the following goal into FLOPER:

```
main([50,20,40,73,99],[25,10,2,51,40],[20,23,45,60,49],cpu)
```

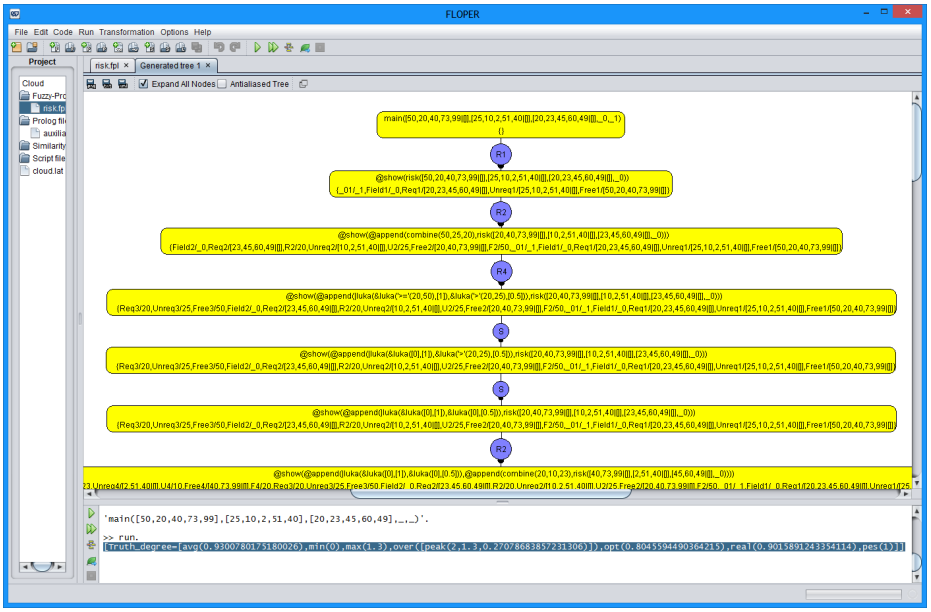


Fig. 6. FLOPER executing our application

The system solves it by generating a list representing the final truth degree associated to the query, with the following shape:

```
[  avg(0.9300780175180026), min(0), max(1.3),
   over([peak(2,1.3,0.27078683857231306)]),
   opt(0.8045594490364215), real(0.9015891243354114), pes(1)]
```

In Figure 6 we show a screen-shot of FLOPER when executing the previous goal. In the main window, we can observe (the initial portion of) the derivation tree for this goal which, in essence, consists in a transition system where each state is coloured in yellow and transitions appear as blue circles, so the initial state is just the original goal appearing in the root of the tree, and the final state (not explicitly displayed in the figure) contains the final truth degree associated to the query. In our case, this solution corresponds to the text darkened in blue in the box at the bottom of the screen.

4 Experiments

To evaluate our proposal, the fuzzy risk assessment is included into the framework presented in [22], which only included a simple admission control technique. This way, the admission control now uses this information to take the decisions about service acceptance or rejection when performing resource overbooking. In that previous work, a simulator to test the development was implemented which is reused here to simulate the cloud infrastructure and emulate the workload.

Table 1. Performance Summary (Figure 7)

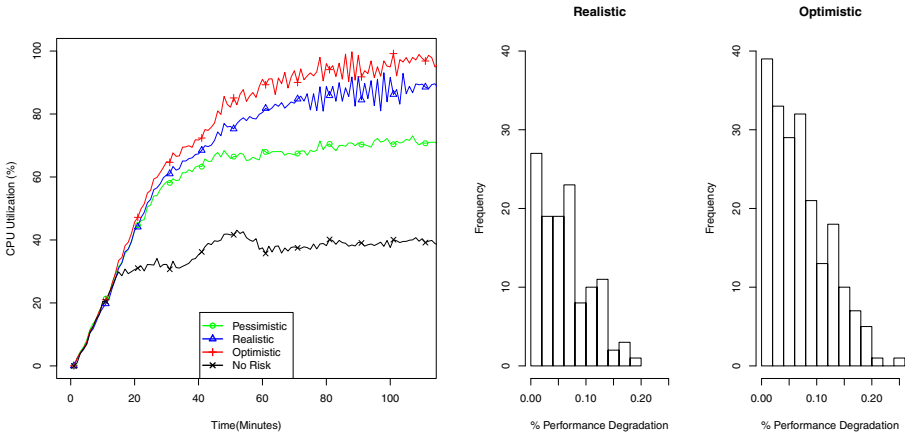
	Average utilization	Node capacity overpassed (%)	Aggregated node capacity overpassed (%)
No Risk	38.9 % (1)	0	0
Pessimistic	69.1 % (1.78)	0	0
Realistic	84.6 % (2.17)	6.99	0.43
Optimistic	92.5 % (2.38)	11.88	0.84

The cloud infrastructure simulated for testing the different risk evaluators consists of 16 nodes where each one of them has 32 cores. We consider four different types of VMs (S, M, L and XL), similar to Amazon's model [6], where each one doubles the capacity of the previous one, starting from the S VM (1 CPU and 1.7GB of memory). Those VMs simulate the execution of a dynamic workload made of different kind of applications (some of them with steady behavior and others with bursty one), profiled by using monitoring tools after running the real applications. The workload is a mixture of applications, following a Poisson distribution for submission rates. See [22] for more details about the testbed and workload generation. With that workload, the performance evaluation has been carried out by generating service requests according to that Poisson distribution. Then, the accepted requests (by the admission control) are scheduled and run on the 16 nodes. During this execution, we measure the *utilization* and *resource shortage*.

Our evaluation is centered on measuring the impact of accurately evaluating the risks taken by the admission control when performing resource overbooking within data centers. The different risk values provided by the fuzzy logic engine are compared against each other and also against a base case where no overbooking is performed – no risks being taken. Those risk assessments from least risky to most are labeled as *"Pessimistic"*, *"Realistic"*, and *"Optimistic"* – mapping them to the respective values calculated by the fuzzy logic engine with those names. The base case is labeled *"No Risk"*.

Figure 7 (a) shows the resource utilization achieved by using the different risk values at the admission control. Clearly, the more risks we take, the higher utilization is achieved. However, this may have a negative impact regarding running out of resources if total capacity is overpassed, not only regarding the whole data center utilization but also regarding every single node into the system. Owing to that fact, Figure 7 (b) shows a histograms over how many times one of the nodes has overpassed its total capacity, and how large the impact on the performance is – performance degradation that may end up in resource SLA violations. The x-axis represents the performance degradation experienced when total capacity in (at least) one of the nodes is overpassed. So, the smaller the bars are, the better (less frequent risk situations) and it is desired that they remain as close to 0 as possible - fewer performance degradation and greater possibilities of resolving these. Notably, as shown in Figure 7 (a), the total infrastructure capacity is not overpassed. This means that *VM migration* can be used to decrease the risks by moving VMs from the overloaded nodes to the ones that still have enough available capacity. This way certain overload situations can be avoided, as has been proposed by Beloglazov et al. [8].

Finally, Table 1 highlights the improvement obtained thanks to performing resource overbooking (up to 2.38 times) and the cost that this entails. Pessimistic has the lowest improvement but without any performance degradation, while the other two techniques



(a) Data center utilization.

(b) Capacity overpassing (node level).

Fig. 7. Resource utilization and risk assessment comparison

present higher utilization rates but at expense of higher performance degradation that may result in running out of resources. For the realistic and optimistic cases, the total capacity of a single node has been overpassed around 6 % and 12 % of the time, respectively. Despite this, the total impact on the final performance is not remarkable (below 1%) – calculated as the percent of time the capacity is overpassed at a node, weighted by the amount of overpassed capacity.

5 Conclusions

In this paper we have used the FLOPER programming environment developed in our research group for implementing with the fuzzy logic language MALP a real-world application in the field of cloud computing.

Admission control techniques that apply overbooking actions are a promising solution for low data center resource utilization, a problem that arises from the elastic nature of cloud applications. However, overbooking actions may lead to performance degradation if not planned carefully.

We propose an admission control that bases its acceptance or rejection decisions on the information about the risks being taken. A fuzzy logic engine provides the information that allows the admission control to estimate the long-term risks of accepting the incoming request. That risk assessment is a combination of several parameter regarding the relationship between available capacity and requested one, such as the difference between these and the information about the peaks when insufficient capacity is expected, providing different degrees of risk that leads to more (or less) aggressive decisions regarding job acceptance.

The evaluation shows significant increases in resource utilization obtained by our risk-aware fuzzy admission control methods. Even for the most optimistic estimates,

available resources are exhausted as little as 0.84% of the time, while increasing utilization by 138%. Thus, our fuzzy methods are a promising approach to help the admission control to evaluate the risks associated with accepting a new service.

Further direction include to extend our work by taking the risk assessment into account together with the SLA information. One such extension could be to specify different costs depending on the risk to be taken or using the different risk values depending on the penalty that is to be paid in case of SLA violation, i.e., the greater the penalty the more pessimistic the admission control should be.

Acknowledgment. This work was supported in part by the Swedish Research Council under grant number 2012-5908. Carlos Vázquez and Ginés Moreno received grants for International mobility from the University of Castilla-La Mancha (CYTEMA project and "Vicerrectorado de Profesorado").

References

1. Almendros-Jiménez, J.M., Luna, A., Moreno, G.: Fuzzy logic programming for implementing a flexible xpath-based query language. Electronic Notes in Theoretical Computer Science 282, 3–18 (2012)
2. Almendros-Jiménez, J.M., Luna, A., Moreno, G.: A xpath debugger based on fuzzy chance degrees. In: Herrero, P., Panetto, H., Meersman, R., Dillon, T. (eds.) OTM-WS 2012. LNCS, vol. 7567, pp. 669–672. Springer, Heidelberg (2012)
3. Almendros-Jiménez, J.M., Luna, A., Moreno, G., Vázquez, C.: Analyzing fuzzy logic computations with fuzzy xpath. In: Fredlund, A. (ed.) Proc. of XIII Spanish Conference on Programming and Languages, PROLE 2013, Madrid, Spain, September 18-20, p. 15. ECEASST (to appear, 2013)
4. Almendros-Jiménez, J.M., Luna, A., Moreno, G.: A Flexible XPath-based Query Language Implemented with Fuzzy Logic Programming. In: Bassiliades, N., Governatori, G., Paschke, A. (eds.) RuleML 2011 - Europe. LNCS, vol. 6826, pp. 186–193. Springer, Heidelberg (2011)
5. Almendros-Jiménez, J.M., Luna, A., Moreno, G.: Annotating Fuzzy Chance Degrees when Debugging Xpath Queries. In: Rojas, I., Joya, G., Cabestany, J. (eds.) IWANN 2013, Part II. LNCS, vol. 7903, pp. 300–311. Springer, Heidelberg (2013)
6. Amazon Elastic Compute Cloud (Amazon EC2), `http://aws.amazon.com/ec2/` (visited July 30, 2013)
7. Barham, P., Dragovic, B., et al.: Xen and the art of virtualization. SIGOPS Oper. Syst. Rev. 37(5), 164–177 (2003)
8. Beloglazov, A., Buyya, R.: Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. IEEE Transactions on Parallel and Distributed Systems 24(7), 1366–1379 (2013)
9. Breitgand, D., Dubitzky, Z., Epstein, A., Glikson, A., Shapira, I.: SLA-aware resource overcommit in an IaaS cloud. In: Proc. of the 8th Intl. Conference on Network and Service Management (CNSM), pp. 73–81 (2012)
10. FLOPER - A Fuzzy LOgic Programming Environment for Research, `http://dectau.uclm.es/floper/` (Visited June 7, 2013)
11. Julián, P., Moreno, G., Penabad, J.: Operational/Interpretive Unfolding of Multi-adjoint Logic Programs. Journal of Universal Computer Science 12(11), 1679–1699 (2006)

12. Klement, E.P., Mesiar, R., Pap, E.: Triangular Norms. Trends in logic, Studia logica library. Springer (2000)
13. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Berlin (1987)
14. Medina, J., Ojeda-Aciego, M., Vojtáš, P.: Similarity-based Unification: A multi-adjoint approach. Fuzzy Sets and Systems 146, 43–62 (2004)
15. Morcillo, P.J., Moreno, G.: Programming with fuzzy logic rules by using the FLOPER tool. In: Bassiliades, N., Governatori, G., Paschke, A. (eds.) RuleML 2008. LNCS, vol. 5321, pp. 119–126. Springer, Heidelberg (2008)
16. Morcillo, P.J., Moreno, G.: Modeling interpretive steps in fuzzy logic computations. In: Di Gesù, V., Pal, S.K., Petrosino, A. (eds.) WILF 2009. LNCS (LNAI), vol. 5571, pp. 44–51. Springer, Heidelberg (2009)
17. Morcillo, P.J., Moreno, G., Penabad, J., Vázquez, C.: A Practical Management of Fuzzy Truth Degrees using FLOPER. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 20–34. Springer, Heidelberg (2010)
18. Morcillo, P.J., Moreno, G., Penabad, J., Vázquez, C.: Fuzzy Computed Answers Collecting Proof Information. In: Cabestany, J., Rojas, I., Joya, G. (eds.) IWANN 2011, Part II. LNCS, vol. 6692, pp. 445–452. Springer, Heidelberg (2011)
19. Rochwerger, B., Breitgand, D., et al.: The Reservoir model and architecture for open federated cloud computing. IBM J. Res. Dev. 53(4), 535–545 (2009)
20. Schweizer, B., Sklar, A.: Probabilistic Metric Spaces. Courier Dover Publ. (1983)
21. The NIST Definition of Cloud Computing, `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf` (visited July 30, 2013)
22. Tomás, L., Tordsson, J.: Improving Cloud Infrastructure Utilization through Overbooking. In: Proc. of the ACM Cloud and Autonomic Computing Conference, CAC (to appear, 2013)
23. Zadeh, L.A.: Fuzzy Sets. Information and Control 8(3), 338–353 (1965)
24. Zaharia, M., Hindman, B., et al.: The datacenter needs an operating system. In: Proc. of the 3rd USENIX Conference on Hot Topics in Cloud Computing, p. 17 (2011)