Rule Based System and Conflicts

[1]Hemant Arun Tumbare and [2]John Singh.K
[1]School of Information Technology and Engineering
VIT University, Vellore-632014, India
[2]Assistant Professor (Selection Grade)
School of Information Technology and Engineering
VIT University, Vellore-632014, India

Abstract

Marking the cloud entities will enable customers to mark the entities with all possible metadata about the entity like its size, location, owner, data center to which it belongs to, the cost center to which it should be charged to, the applications belonging to it, etc, without worrying about the actual inventory view of cloud. It will also enable solutions to use this information to perform tasks/actions based on a combination of these attributes set. To perform the actions, solutions allow users to define rules, based on the attributes. However, if a large number of rules are defined, such rules can conflict and result in the solution performing multiple actions on the same entity without the actual intention of the user, or worse, crashing of the solution. Hence, defining rules needs to be restricted or at least the customer should be warned if creation of a rule can potentially result in conflict. Through this paper, we present a solution to detect conflict among rules. The solution is generic and can be easily integrated with any other solution.

Keywords: Marking, Rules based on marking, Boolean algebra, Quine-McCluskey Method (QM).

INTRODUCTION.

Introduction of Marking cloud infrastructure entities with custom-defined tags will open myriads of possibilities and opportunities for rapid automation. It will start a new era in automation where customers or products can just add some tags on an entity based on its creation parameters and the solutions can simply use these tags to perform complicated operations on the entity and can perform myriads of background tasks on behalf of the user. Complex rules can be created in solutions based on the tags defined on the entities to perform certain actions for all the entities matching the given criteria. These tag-based rules can achieve very high level of automation, which has always been desired in the ever-expanding customer environments.

With this new rule-based automation comes new challenges of matching these rules to various entities quickly, detect changes in the tags on entities and change actions accordingly, detecting entities which do not match any rules or entities which match multiple rules and detecting rules which can potentially lead to conflicts, at the time of application of these rules on the entities. In this paper, we explored the problem of detecting rules which can result in conflicts at the time of application. We present an algorithm to detect such conflicting rules and warn the creator of the rule beforehand.

1. RULE BASED SYSTEM

1.1 Rules

A rule based system consists of a set of rules and each rule consists of a condition and an action. Action part contains what action you want to execute if the condition is satisfied.

For e.g. consider a simple rule where

Condition: (Location = USA) AND (Application = Tomcat)

Action: Power ON the VM.

In this case, if the condition is satisfied then the action part of powering the VM on will be executed. Rule based system may contain number of rules and action related to each rule.

Throughout the paper the following terms are used.

Tag	Tag is metadata associated with cloud entities.
Tag Category	A grouping of related tags.
Simple Expression	Tag category and tag along with a criteria
Criteria	Equals, Startswith, Endswith, Contains
Operator	Operations used to construct conditions part of rules - AND, OR, NOT (similar to the Boolean algebra).
Complex Expression	Simple expressions along with operators form complex expressions.

1.2 Conflicting rules

A system can have any number of rules defined, potentially resulting in conflict. Two rules are said to be in conflict with each other if

(a) their action parts are of conflicting nature and

(b) their condition parts are satisfied by same entity.

If the action parts of two rules are different then there will not be any issue, even if the rules get executed on the same entity. For e.g. if the action part of a rule is – 'power on a VM' and that of other is - 'charge $15' then there will be no conflict even if the two rules get executed on the same VM. But two actions like 'power on' and 'power off' are conflicting because both cannot be executed at the same time. For the purpose of the discussion in this paper, we assume that the action parts are of conflicting nature and we need to detect rule conflict based only on the condition part.

1.3 Conflict Detection

Based on the time of detection we can classify rule conflicts as

1.3.1 Static time conflict

These are the conflicts which can be detected at rule creation time. If a term in a rule's condition is subset or equal to the terms of another rule's condition, then the rules conflict with each other. These types of conflicts can be detected when a new rule is created. For e.g. consider following two rules

(a) If {(Size = small) || (State = powered on)}, then charge $15.

(b) If (Size = small), then charge $10.

If a VM has tag (size = small) set on it, then both the rules become applicable to the VM. However, the system will not be able to apply either rule because the actions are conflicting.

1.3.2 Run time conflict

Run time conflicts cannot be detected at rule creation time. Consider following two rules,

(a) If (Size = small), then charge $15

(b) If (State = powered on), then charge $10

If we use same logic that is used to detect conflict at static time, these two rules are clearly not conflicting. But if there exists a VM with tags {Size = small; State = powered on}, then these two rules have run time conflict. The VM will satisfy both these rules, and then there will be confusion about the action part of which rule should apply. Moreover, it is incorrect to prevent the user from creating such rules which might result in run-time conflict.

1.3.3 Defining boundary between Static and Runtime conflicts

Conflict detection can ideally be delayed till run-time, when the rules actually get applied on entities and the actions taken. In such a scenario, entities matching multiple rules can be detected easily. However, it is difficult to figure out what actions should be taken by the solution in such a scenario. Hence, run-time detection of conflicts should be avoided as far as possible and it will be safer to detect conflicts at the rule creation time.

Such a static time conflict detection system can't be too restrictive either while detecting rules, as it might warn/stop the users from creating rules which might be theoretically conflicting, but might never occur in the customer environment. For example, consider a scenario where a customer has created two tag categories 'country' and 'state' and is using the country tag category to tag all entities residing on infrastructure outside USA and state tag category to tag all entities residing on infrastructure in USA. Say the admin creates two rules, R1 = if (country is India), backup VM at 2pm PST and R2 = (state = Washington DC), backup VM at 2am PST. Theoretically, the rules can conflict if tags from both the tag categories are set on the same entity. However, this will never happen in the customer environment. Hence, creation of such rules should not be restricted at the time of creation, and such a conflict should be detected at run-time.

To define the boundary of rule conflicts at runtime and at static time, we have assumed the following: If the rule, in its reduced DNF form has a term which is subset of a term in the other rule, the rules are conflicting, otherwise not.

2. IMPLEMENTATION

The implementation of the proposed solution

(a) assumes that the action part of the rules are of conflicting nature and tries to detect whether the condition part can result in conflict,

(b) It concentrates on the detection of static time conflict among the rules. The intention is to detect such conflicts at the time of rule creation and warn the user accordingly to avoid unintended behavior when the system executes actions.

2.1 Representation of rules

The rule conditions are represented in XML. For e.g. figure (1) represents the condition

(Location=India) || {(Location=India) AND (Size=small)}

```
<ComplexExpr operator="or">
   <SimpleExpr name="Location" criteria="equals">
      <Value>India</Value>
   </SimpleExpr>
   <ComplexExpr operator="and">
      <SimpleExpr name="Location" criteria="equals">
         <Value>India</Value>
      </SimpleExpr>
      <SimpleExpr name="Size" criteria="equals">
         <Value>small</Value>
      </SimpleExpr>
   </ComplexExpr>
</ComplexExpr>
```

Fig 1. Sample XML representing rule condition

For ease of visualization, we represent each simple expression by a literal. In further discussion, the terms 'literal' and 'simple expression' are used interchangeably. The above rule condition can then be simplified by assuming

A => (Location = India)

B => (Size = small)

Rule condition: A + AB

In addition we represent simple expressions involving same category but different tags using subscripted literals.

For e.g. if category 'Size' has different values like {'small', 'medium', 'large'} then,

A_1 => (Size = small),

A_2 => (Size = medium) and

A_3 => (Size = large).

2.2 Evaluation of conventional expression reduction techniques

Comparing rules directly, without pre-processing them might result in incorrect rules being detected as conflicting.

For e.g. consider following two rules

R1 = {size=small}, charge $20 and

R2 = {(size=small) OR ((size=small) AND (location=India))} charge $10.

In this case, rule 2 is actually logically equivalent to R3 = {location=India} charge $10. However, if we compare rules directly, R1 and R2 will be reported as conflicting. Hence, it is necessary to reduce rules to their most elementary form without losing any information from the rule and then compare them.

We explored existing literature on expression reduction and came across reduced DNF forms and the methods to achieve this reduction, i.e., K-Map method and QM method. However, conditions in the rules differed from the use of literals in these methods in two important ways:

1) Both the methods use standard Boolean algebra assumptions, some of which are not valid in our case. For example, A + A' = 1 is a valid Boolean algebra expression. However, in case of conditions defined in rules, this is not true. Consider a simple case of a rule, R1: if (Size = small and size ≠ small), charge 10$. At first glance, it might appear that the condition is true for all the entities. However, if the rule is applied as it is, entities which do not have tag small defined on them, will not match the criteria. Hence, the rule cannot be reduced to R2 = if (true), charge 10$.

2) Both the methods expect only two dependent literals, which are compliment of each other, i.e., A and A' are two literals which are independent of all other literals. However, in our case, this is not valid. We can have multiple dependent literals like A_1, A_2 and A_3 cited in 3.1.

To satisfy these requirements, we developed our own expression reduction technique, which will reduce the condition part of the rules to a required format and detect static time conflicts, based on some axioms.

Some of the modified Boolean algebra axioms are as follows:

1) $A + A' \neq 1$.

2) If a tag category has 'n' different tags then, $A_i' = \sum A_j$ where $j \neq i$, $0 < i, j \leq n$.

Assumptions:

1) At any point in time, an entity cannot have more than one tag from same tag category set on it implying $A_1A_2 = 0$.
2) Criterion for matching tags is either 'Equals' or 'Not Equals'.
3) Currently the rule conditions support operations like 'AND', 'OR' and 'NOT'.

We believe the above assumptions will satisfy most of the common use cases.

2.3 Normalize complex rule conditions

Hereafter, an expression refers to the Boolean expression that represents the condition part of the rule. We normalize the condition part of the rules first to 'Negation Normal Form' (NNF) and then to 'Disjunctive Normal Form' (DNF).

2.3.1 Convert any expression to NNF

NNF is a representation where the NOT operator is always present at the literal level. To convert any given expression to NNF we distribute the NOT operators to the level of simple expressions. The Convert NNF to DNF

DNF is a representation where the condition is represented as a disjunction of conjunctive clauses [1].

For example A, (A + B), (AB), (AB + CD) are all in DNF.

Let's define an 'almost DNF' expression as the one whose all sub-expressions are in DNF but it is itself not. for e.g. (C+D)*(A+B). Here the individual sub-expressions (C+D) and (A+B) are in DNF but the expression as a whole is not.

2.4 Reduction filters

Reduction filters are applied on the DNF expression obtained. These filters attempt to simplify DNF expressions by removing redundant conjunctive clauses (also referred as terms) in the DNF.

2.4.1 Complement product filter

This filter eliminates the terms where a literal and its complement are present together. For e.g. (ABCA' + D) => D, since AA' = 0.

2.4.2 Distributive law filter

This filter checks whether a term in the DNF expression is subset or equal to any other term and deletes the superset term.

For e.g. (A + AB + ABC) => A (1 + B + BC) => A.

2.4.3 Product exclusion filter

Product exclusion filter eliminates such terms where subscripted literals co-exist.

For e.g. $A_1A_2 = 0$, so $(A_1A_2C + D) => D$.

2.4.4 Redundancy product filter

Considering the subscripted literals described in 3.1 the redundancy product filter will do a filtering illustrated in the following example.

For e.g. $A_1A_2' => A_1 (A_1 + A_3) => A_1A_1 + A_1A_3 => A_1 + 0 => A_1$.

2.5 Analyse reduced rules for conflict

Given two rules whose condition part has been normalized and reduced as discussed above, figure 2 illustrates the algorithm to find whether the rules have static time conflict or not.

```
for each term1 in rule1's condition {
   for each term2 in rule2's condition {

      1) Compare each literal in term1 with that of term2
      2) If literals are equal i.e. same tag category,
         tag and criteria then rules are conflicting
   }
}
```

Fig 2. Algorithm to detect rule conflict

3. CONCLUSION AND FUTURE WORK

In this paper, we presented an algorithm to detect conflicts in tag-based rules at rule creation time (static time). The algorithm is developed because direct comparison of rules might result in false positives, while reduction of rules using standard Boolean algebra reduction techniques like K-map method and QM method can't be applied. The algorithm is based on some of the Boolean algebra axioms, some modified axioms as per our solutions requirements and some of our assumptions. We created a prototype for the same as well.

Currently, the algorithm is based on the assumption that criterion for matching tag is either 'EQUALS' or NOT_EQUALS'. We will further like to explore as to how the criteria like 'STARTS_WITH', 'ENDS_WITH' and 'LIKE' can be supported.

REFERENCES

[1] Akshay Narayan, Shrisha Rao, Gaurav Ranjan and Kumar Dheenadayalan, "Smart Metering of Cloud Services", In Proc. of IEEE International Conference on Systems, pp. 1-7, 2012.

[2] Ki-Woong Park, Jaesun Han, JaeWoong Chung and Kyu Ho Park, "THEMIS: A Mutually Verifiable Billing System for the Cloud Computing Environment", In Proc. of IEEE International Conference on Cloud Computing, pp. 139-147, 2010.

[3] Stefan Tai, Jens Nimis, Alexander Lenk and Markus Klems, "Cloud Service Engineering", In Proc. of IEEE International Conference on Software Engineering, Vol. 2, pp. 475-476, 2010.

[4] Ibrahim Armac, Michael Kirchhof and Liviana Manolescu, "Modeling and Analysis of Functionality in eHome Systems: Dynamic Rule-based Conflict Detection", In Proc. of IEEE International Conference on Engineering of Computer Based Systems, pp. 218-228, 2006.

[5] Francois Hantry, Mohand-Said Hacid and Romuad Thion, "Detection of conflicting Compliance rules", In Proc. of IEEE International Conference on Enterprise Distributed Object Computing, pp. 419-428, 2011.

[6] LUO Qian, TANG Chang-jie, LI Chuan and YU Er-gai, "Detecting Self-Conflicts for Business Action Rules", In Proc. of IEEE International Conference on Computer Science and Network Technology, Vol. 2, pp.1274-1278, 2011.