Journal of Technical Science and Technologies; ISSN 2298-0032

Optimization of Autonomic Component Ensembles Resources

Medhat MOUSA*

Abstract

This paper presents a comprehensive study and a new approach for optimization of the Hardware resource, there exists some natural probabilistic models, e.g. Poisson process, Unigram model. The suggested framework allows us to explain the independent events across multiple sites in terms of the Software as a Service (SaaS), Self-healed and self-adapted Autonomic Components using highly optimized, secured, and expressiveness context formal language.

Keywords: Modeling of Cloud Computing Security, Virtualization Technology, and Autonomic Computing Systems

Introduction

Cloud computing as a dynamic computing grid needs some expressiveness and flexible formal language, that could detect the behavioral changes of every component, as explained in (Chakraborty, Samarjit, 2003). An alphabet is the simplest analogy of formal language i.e. a finite set of symbols which are used to form words in a language. An example of an alphabet might be a set like $\{a,b\}$. or it cloud be a set over E i.e. $\{x \in \{a,b\}* \mid x \text{ has equal number of as and bs}\}$ Therefore it is possible to generate a new language by applying some operational semantic rules on its sets or the strings.

In (Chakraborty, 2003)there is an example of generating a new language from some regular expressions by conducting some simple operations–union (denoted by+), concatenation, and the closure operation (denoted by*), to clarify the real meaning of $\{0,1\}$ *i.e.* $\{0\} \cup \{1\}$ the regular expression consider the following language that consisted of , therefore we can conclude some of the regular expression e.g. 0+1.

Related Work and Existing Techniques

In this section we discuss some of the existed techniques that use normalization of the probability distribution q_i. By defining the Operational Semantics of Probabilistic Klaim (pKLAIM) “Discrete Time”, in (Alessandra, 2002), for the discrete time execution models the updated principle simulates a Global Scheduler: At each interval, step one node is selected according to the scheduling probabilities and executed. The result is a Discreet Time Markov Chain (DTMC) (Alfaro, 1997).

In order to update the network configuration, the Global Scheduler has to perform the following tasks:

1. Selecting a node which could initiate a global update, according to the scheduling probabilities,
2. Checking if the selected node can cause a global transition,
3. Executing one of the possible updates, as defined in the local semantics of the node in question.

If the chosen node is unable to perform a local transition an alternative one has to be selected.

pKLAIM Extension

As define in (Lorenzo et, al., 2009) Klaim (Kernel Language for Agents Interaction and Mobility) is an experimental language specifically designed to program the Dynamic and distributed systems consisting of several mobile components that interacts through multiple distributed tuple spaces. Also it is important to mention that pKlaim is based on the original Klaim, a quantitative analysis is important always in case of the security measures at realistic situation and specific tolerance criteria that expressing how much the underlying infrastructure and the hosted system by its applications is vulnerable. In particular we proposed the Discrete Time variant; hence we express the network Architecture at the local level. Meanwhile we can use the probabilistic parallelism and choice operator, in order to present a new adaptive approach “App/VM Live Migration” across Multiple sites due to some accidental changes likewise Virus infections or Resources insufficiency because of (D)DoS attack.

* Ph.Dc, Faculty of Computer Technologies and Engineering, International Black Sea University, Tbilisi, Georgia.
E-mail: mmousa@ibsu.edu.ge

Methodology

➢ Process Syntax of pKlaim

As mentioned in (Alessandra, 2002) the main difference between discrete and continuous time model is the "single step transition probability" which is replaced by "transition rate" in continuous time model. Therefore, transition probabilities could be presented in the function of time, Fig (1) shows the both process syntax in (Discrete & continuous) time model respectively.

P	::=	$\mathbf{nil}$	null process
	|	$a.P$	action prefix
	|	$\|_{i=1}^{n}\ p_i : P_i$	probabilistic parallelism
	|	$+_{i=1}^{n}\ p_i : P_i$	probabilistic choice
	|	X	process variable
	|	$A(P, \ell, e)$	process call
a	::=	$\mathbf{out}(t)@\ell$	sending tuples
	|	$\mathbf{in}(t)@\ell$	receiving tuples
	|	$\mathbf{read}(t)@\ell$	inspecting tuples
	|	$\mathbf{eval}(P)@\ell$	remote evaluation
	|	$\mathbf{newloc}(u)$	new location

Figure 1. *Process syntax*

➢ SCEL Syntax

Furthermore, it is important to integrate our paper's aim with Software Component Ensample Language (SCEL), as Fig(2) shows; the principal idea of a component is illustrated in terms of four ingredients with considering that the Processes are used to build up components that in turn are used to define systems. (Rocco De Nicola1, et al., 2013).

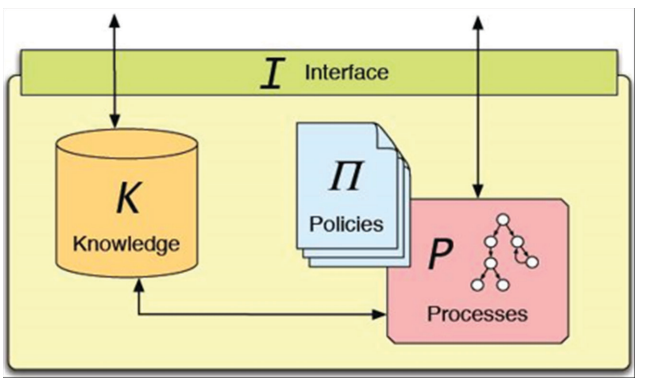

Figure 2. *Shows four ingredients of the component*

1. An interface $\mathcal{I}$ publishing and making available structural and behavioral information about the component itself in the form of attributes. Among them, attribute *id* is mandatory and is bound to the name of the component. Notably, component names are not required to be unique; this would allow us to easily model replicated service components.

2. A knowledge repository $\mathcal{K}$ managing both application data and awareness data, together with the specific handling mechanism. The knowledge repository of a component stores also the whole information provided by its interface, which therefore can be dynamically manipulated by means of the operations provided by the knowledge repositorie's handling mechanisms.

3. A set of policies Π regulating the interaction between the different internal parts of the component and the interaction of the component with the others.

4. A process **P** together with a set of process definitions that can be dynamically activated. Some of the processes in **P** perform local computation, while others may coordinate processes interaction with the knowledge repository and deal with therelated issues to adaptation.

Finally, SYSTEMS aggregate COMPONENTS through the composition operator.

Problem Statement

Despite the Importance of the query action in any dynamic distributed grid, especially in Cloud Computing based infrastructure, nevertheless, the consumed bandwidth playing very important role, Fig(3), shows the configured topology for Datacenter with Five VMs with their Applications.

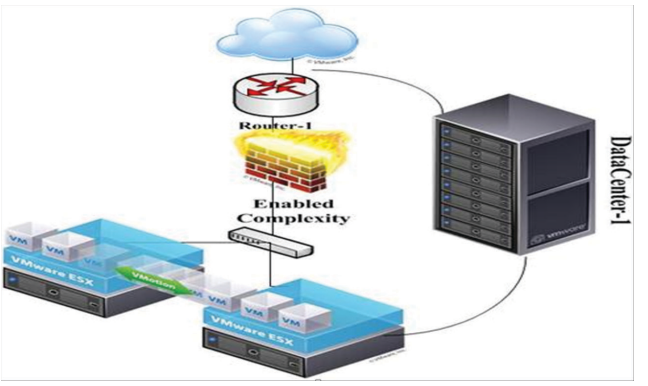

Figure 3. *Shows Application movements across 2 Servers*

In the Software Access Control Policy Language (SACPL) environment, query process causing significant delay especially if it is consisting of a set of sites and VMs, as well as the installed Applications Platform as a Service "PaaS", e.g. in the proposed topology, five running Virtual Machines under one Enabled Complexity "EC", to measure the potential threats at both sites and the VM level.

Journal of Technical Science and Technologies; ISSN 2298-0032

In the mentioned above scenario each SCEL "VM" component has to perform and initiate a query action at a periodic interval or after certain trigger, to update its knowledge repository $\mathcal{K}$ as well as updating the rest of the SCEL's knowledge repository $\mathcal{K}$. I.e. 25 **query** per certain trigger.

Some considerations have been taken into account in order to reduce the **query** in the SCEL environment:

1. Each VM has to be connected by the enabled eomplexity,
2. Unreachable VMs will be taken offilne "Isolated" just in case of:
3. The VM running "y" service, it will be eliminated after moving its application to another VM with respect to the current topology as well as the health state of this VM, otherwise there will be a new VM will be created to host the received application accordingly.
4. If there is no service at this VM, it will be eliminated automatically, in order to reclaim the underlying Hardware.
5. EC is aware of the whole infrastructure regarding Hardware availability at each host and the threat levels.

If EC(subject:Self$_3$>0.3) ^ $P_{CPULoad}$(subject: Self$_3$>9)

^ P_{Mem}(subject:Self$_3$> 8).

Then Qry(EC:?x<0.3)^$P_{CPULoad}$ (?x: Self$_3$<9) ^ P_{Mem} (?x: Self$_3$< 8).

Permit Ser$_Y$@ Self$_3$ to Migrate to ?x// Put("Ser$_Y$",True)@?x.

Then Get(subject: Self$_3$, True)@self.nill.

Algorithm 1. *Query Reduction at SCEL environemnt*

Query at Work

In the tradition SCEL environment each component has to update its knowledge regularly, here we assume that most important security criteria including Hardware (CPU, Memory) e.g. (in range 1:10)/ each, and health state of the VM that is determined by enabled complexity, e.g. (0:1) therefore:

- The un-threatened "healthy" VM, "enabled complexity" indicates a less possibility of infection by Virus, Malware, etc. Besides the hardware availability counters that show normal usage of the recourse's utilization, i.e. it will not be obligated to initiate any query process,
- The **infected** VM will propagate its status among the other SCELs via Query action,
- Enabled Complexity will propose the fittest VM, i.e. (in terms of the security level and HW availability) to the infected VM,
- The infected VM will put/migrate its App to the elected Healthy VM,

In (**I.Rodonaia et al., 2013**), the "below code" it shows that other than ID, the interfaces $\mathcal{J}$ and $\mathcal{I}$ provide the attributes "ComplexityLevel","CPULoad" and "Hardware". Stores a context information, updated by the underlying infrastructure (usually, from the firewalls, gateways or special probes) and are `sensed' by the Managed Element (**ME**).

The CPi where the application is running is the SCEL component: $\mathcal{I}[\mathcal{K}, \Pi, AM[ME]]$

AM≜ $P_{ComplexityMonitor}$ [$P_{CPULoad}$]

$P_{ComplexityMonitor}$ ≜ **qry**("ComplexityLevel", "high") @ self.

get("ComplexityHigh", false) @self.

put(("ComplexityHigh", true) @self.

qry("ComplexityLevel","low")@self.

get("ComplexityHigh",true)@self.

put("ComplexityHigh",false)@self. $P_{ComplexityMonitor}$

$P_{CPULoad}$ ≜ **qry**("CPUloadLevel","low")@ self.

get("CPULow", false) @self.

put("CPULow", true) @self.

qry("CPUloadLevel","high")@self.

get("CPULow",true)@self.

put("CPULow",false)@self. P_{CULoad}

$P_{MigrateCPi}$ ≜ **qry**("required_functionality_id", ?X) @ self.

/* retrieving from the knowledge repository the process implementing a required functionality ID and bounding it to a process variable X */

get("required_functionality_id_args", ?y,?z) @self.

qry("CPiId", ?c)@Ω. /* searching an item *c* among components belonging to the ensemble identified by predicate Ω */ **fresh**(n).

/* fresh name n is used for coordination purposes */

put("required_functionality_id_params",n,y,z)@c

/* storing actual parameters of theprocess to be executed in the found component c : moving from VM7 toVM5 */

get("required_functionality_id","terminated",n)@self.

/* removing the process from the knowledge repository of 'old' CPi */

get("required_functionality_id", X) @self.nil

/* eliminating the process in 'old' CPi */

Here the predicate Ω is determined as follows:

$\Omega(\mathcal{I})$=($\mathcal{I}$.ComplexityLevel="low") ^ ($\mathcal{I}$. CPULoad<75) ^ ($\mathcal{I}$.Hardware>=5))

And is used for group-oriented communication in the action qry ("CPild",?c) @ Ω. This predicate defines the ensemble of components, which publish in their interface's attributes Complexity Level, CPU Load and Hardware along with relevant values. We assume that these attributes are provided by the interface of each component and obtain dynamically updated values from corresponding probes (sensors) as a result of constant monitoring (sensing) of the computing environment.

Conclusion

In this paper, we have suggested a new approach that can be adopted in the Cloud Computing based distributed dynamic environments, also we have presented the traditional Query ,its mechanism, and how it could be developed using our algorithm in order to make the SCEL components more productive.

References

Alessandra Di Pierro, Chris Hankin. (2002). Probabilistic KLAIM. p.1

Alfaro, L. d. (1997). formal verification of probabilistic systems phd .

Chakraborty, S. (2003). Formal Languages and Automata Theory. Computer Engineering and Networks Laboratory,P. 3.

Chakraborty, S. (2003). Formal Languages and Automata Theory. P. 2.

Chakraborty, Samarjit. (2003). Formal Languages and Automata Theory. Computer Engineering and Networks Laboratory,P. 2.

Irakli Rodonaia, Alexander Milinikov, Medhat Mousa, Vakhtange Rodnaia. (2013). A technique of formal security modeling in autonomic cloud computing environment.P. 4.

Lorenzo Bettini, Viviana Bono, Rocco De Nicola, Gianluigi Ferrari,Daniele Gorla. (n.d.). The Klaim Project:Theory and Practice. Global Computing initiative,P. 1.

Peter Mell, Timothy Grance. (2011). The NIST Definition of Cloud . Recommendations of the National Institute ,P. 2.

Rocco De Nicola1, ea l. (2013, January 28). SCEL: a Language for Autonomic Computing. P. 7.