Volume 5, Issue 4, 2015 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Autonomic Cloud Computing: Autonomic Properties Embedded in Cloud Computing

S Anithakumari, K Chandrasekaran
Department of CSE, National Institute of Technology
Karnataka, India

Abstract— The high emergence of storage, processing and internet technologies together with the effect of virtualization have significantly changed the computing paradigm to distributed, dynamically varying, heterogeneous, scalable and elastic paradigm. This new paradigm, called cloud computing, provides computational power as a service utility. The enhancing dynamism, heterogeneity and interactivity in software, services, applications and networks generated complex and unmanageable systems in a cloud environment. To address this difficulty, researchers tried to feed autonomic properties into cloud computing services, and generated an intelligent and flexible computing service called autonomic cloud computing. Here we present a detailed discussion on the term 'autonomic cloud computing' together with some examples of autonomic properties embedded in cloud services. The paper also covers a brief review of various autonomic cloud engines already implemented and existing in the present literature.

Keywords— Autonomic Clouds, Elastic Services, Autonomic Computing, Autonomic Cloud Engine, Resource Scheduling

I. INTRODUCTION

The swift progress in processing, storage and internet technologies made computing resources more powerful, economical and universally available which enabled the insight of a novel computing model called cloud computing. Cloud computing assures to remodel the method of computing service generated, consumed and distributed by using virtualization and distributed computing techniques. The integrated use of various environments generates high level of heterogeneity in computing and we are now facing huge variations in system abilities, cost patterns and accessibility both on the resource side and on the application side, with varying necessities and constraints. Supervising these dynamic behaviors and imposing adaptations to them using physical techniques becomes impractical in cloud computing. This makes autonomic computing approaches more remarkable in a cloud computing environment.

Autonomic computing is generally viewed as a computing technique with the capability to self manage and dynamically adjust to changes in accordance with changing policies and objectives similar to biological systems in the nature. The essential properties required for an autonomic computing system can be listed as self configuring, self healing, self optimizing and self protecting. Self configuring is the system's ability to configure itself, self healing is the system's ability to recover itself from faults, self optimizing means the ability of the system to perform improvements on its own execution for getting ideal performance and self protecting is the system's ability to defend itself from accidental attacks. All these properties together are called self* properties. Autonomic computing systems reduce the need for human interaction by upgrading the performance and behavior according to computing environments. Self managing environments automatically do their actions based on the observed changes without any initiation from a third party. This is achieved through a control loop, called **MAPE-K** (**M**onitor, **A**nalyze, **P**lan, **E**xecute and **K**nowledge) loop, maintained by the **autonomic manager** in the system. In-built methods are associated with this loop to: *monitor* or gather required information from the system, *analyze* collected information to detect whether it is necessary to take some action, create a *plan* that describes the necessary changes and *execute* the plan to implement these actions.

Autonomic cloud computing is a recently new computing methodology introduced by incorporating autonomic techniques to cloud computing applications. It is developed by attaching self * properties to cloud computing services. Efficient monitoring and management of current cloud systems become impractical without autonomic techniques because the number of systems is increasing day by day. Autonomic clouds are very much useful to form administrative decisions in cases where dependable collaboration among different clouds is a primary concern.

In this article we describe the concept of autonomic cloud computing and its definition and implementation aspects. Different variants of autonomic cloud computing machines and the impact of autonomic computing properties to materialize, self adaptive and self organizing cloud services are also discussed here by reviewing the available literature in this area. Several researchers tried to incorporate autonomic properties like self configuring and self optimizing to cloud computing and come up with various self adaptive cloud services. The remaining part of the paper is organized as: Section 2 details the related works in this area and section 3 describes various definitions and design aspects of autonomic cloud computing. Section 4 contains several mechanisms used for the implementation of autonomic cloud engines. Section 5 discusses the amalgamation of autonomic properties and cloud services for the evolution of autonomic cloud computing and finally section 6 concludes the paper.

II. RELATED WORK

Self-managing computing paradigms like autonomic computing [14] concentrates on simplifying different system administration aspects and their integration. A dedicated self management infrastructure is developed in autonomic systems for performing low-level decisions/tasks and it allows needed system behavior to be specified as high-level policies[18]. Research works in self managing cloud computing [13] focuses on the automation of individual administrative activities such as resource management, service management, etc. These works are purely based on views of modeling resource management using intelligent techniques [5]. Cloud computing uses virtual infrastructure management to organize the dynamic needs in storage and processing requirements of organizations in a successful manner. Cloud infrastructure dynamically deploy virtual machines to configure the resources efficiently. In addition to the already existing frameworks, several generic frameworks are recently evolved in distributed systems[3][15][24] for virtual machine management. Particularly, the hierarchical layer of Snooze[9] is built as per the idea presented in the Hasthi[26] framework that uses a hierarchical self-stabilizing method for the management of large-scale distributed systems.

III. AUTONOMIC CLOUD COMPUTING (ACC)

Researchers have tried several techniques for the design and implementation of autonomic cloud computing. The most succeeded ones are listed below and the details about these implementation techniques are discussed in the following subsections.

1. A-MAPE-K loop based approach
2. Software process based approach
3. Information proxy based approach and
4. Multi agent based approach

A. ACC using A-MAPE-K Loop

This approach uses a control loop based technique for implementing autonomic cloud computing[22]. The control loop includes different phases like: Adaptation(A), Monitor(M), Analyze(A), Plan(P), Execute(E) and Knowledge(K) and is known as A-MAPE-K loop. Adaptation phase is to adapt the cloud infrastructure and applications according to the upcoming variations and to maintain a balance in the virtualization layer.

Planning and execution phases are mainly focused before the application deployment and once the application gets deployed, the consideration is given to Adaptation, Monitoring and Knowledge management. Figure 1 shows the different phases of autonomic cloud management control loop.

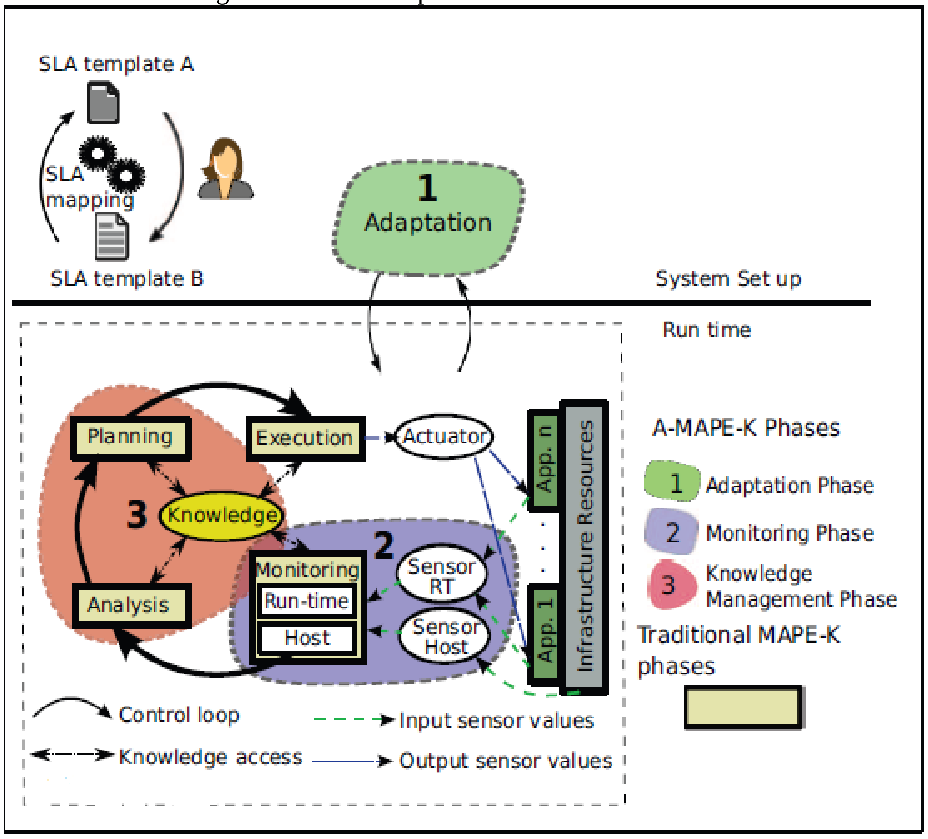


Fig. 1 Autonomic Cloud Manager Control Loop

Adaptation: This phase includes all steps prior to the deployment and initiation of the applications. Template matching of cloud providers and consumers are done in this phase and if some mismatch occurs in measurement units or attribute values, an approach is used for the adaptation of discrepancies in attributes and translation of different attribute values.

Monitor: In the monitoring phase, using two monitoring segments such as run time monitor and host monitor, the low level resource metrics like system up time and down time are monitored. Host monitor is to measure low level resource metrics using some measurement tools and run time monitor is assigned to return system availability using metric mapping, where system down time and up time are mapped to the availability of the system.

Analyze: In the analyze phase the output from the monitoring phase is decoded and analyzed to make the decisions or planning of the system elements.

Plan: Planning phase is to maintain a balance between the autonomic manager and other managed elements of the system. The data obtained from monitor/analyze is used by the plan phase to plan the system properly. The execution strategies required to optimize the system behaviors and operations are fixed here in this phase.

Execute: In execution phase the planned execution strategies are processed to achieve self properties in autonomic computing.

Knowledge Management: Knowledge management represents the intelligent use of monitored data to frame decisions to satisfy requirements while minimizing resource usage and user interactions. The quality of decisions is improved by analyzing the failure or success of earlier decisions and is termed as *learning*. With a knowledge management technique two learning approaches are used: *case based reasoning* and *rule based learning*. Case based reasoning uses previous experiences to solve the problem. It solves a case by looking for similar past cases and by reusing the past solutions to generate the solution. Rule based learning *attempts* to derive decisions from a set of data and rules.

B. Software based ACC system

In [29], the authors describe a software process based development approach for the design of an autonomic cloud computing system. According to this approach a sequence of software steps is followed for the complete design and these different design steps are:

1. Control parameter identification: This is the first phase in the development of an autonomic cloud computing system in which all system parameters that need autonomic management are identified. This includes collecting information from the system and translate this information into equivalent self* functions. In majority of cloud computing systems the parameters need to be controlled are *response time* and *count of virtual machines*/servers and the self* function required is *self optimization*.
2. System model: The next is the decision making phase called system modeling phase, in which the required structure of the system has modeled. The system model describes a representation of the controlled system. The future state prediction and decision making are based on the system model. Control theoretic models for the system are developed by using mathematical equations.
3. System input identification: This phase is to determine the system inputs in continuation to the system model formation. The models will normally contain internal states, input/output values, etc. In the model the input values denote the systems' inputs that must be checked. The inputs normally considered are CPU load, client request arrival rate etc.
4. Model identification: The developed model represents the general structure of the system and the input values have to be initialized to operate the model. Such an initialization is called model identification and is done by executing the system for regular time intervals and collecting data through designated outputs. For getting a good model identification, the same process can be iterated several times.
5. Model update: Due to changes in the managed system an updation of the model is required and a decision regarding this updation, such as how and when the model is updated, is determined in this phase. The model can be updated on every measurement if the time required for update is small and if the measurement and update need more time the model can be rebuilt offline. In the case of offline rebuilding, replace the existing model with the new one which can be done at the time of online processing.
6. System decision type: An autonomic system should be able to make decisions on how and when to adapt changes in the system environment. This decisions can be of two types, such as proactive or reactive, according to the system type and the autonomic systems' requirements.
7. Prediction creation: In the case of a proactive autonomic system, the system must be able to predict the future states and this is done by using an appropriate estimator. According to the type of the system model different estimators are possible, such as, a Karlman filter for a control theoretic model and in a neural network model the prediction is based on training data and the model structure.
8. Coordinator creation: The coordinator works as a central link between different component parts and generate a high level control loop logic. It makes a prediction based decision mechanism or makes a multifaceted control loop where the prediction and model are iteratively modified per sample for minimizing errors.
9. Data measurement: During this step, generally the inputs are measured for controlling the system. In some cases the controlled systems do not give such a direct measurement and it must be computed indirectly using the data given by the controlled resource.
10. Managed system control: The decision regarding how or when to modify the controlled system, must be made to automate the system and the modifications must be purely according to the decision generated at the decision maker. The decision maker should be capable of communicating with the controlled system in a direct or indirect way to implement current variations on it.
11. Autonomic system control: In this phase, the system is able to modify itself during variations in environment and managed system. This last step is to make decisions on how the environment and the changes in the controlled resource influence the autonomic management system.

C. ACC with information proxies

In [7], a mechanism to implement autonomic cloud computing is described with the usage of information proxies. According to D Cenk Erdil's experimental results, information proxies helped a lot to improve decision making in resource scheduling of large scale distributed systems. An information proxy is viewed as a source, or a distributor of packets of information and it distributes the packets at preset time intervals. An information proxy is to distribute any useful information like state of the resource, works that need resources, overall resource utilization, etc. as quantized packets. These quantized packets are stored in remote caches or used fully for improving distributed system services. The efficiency of the information proxy is counted upon several success criteria such as a redundant message ratio, dissemination overhead, etc.

In this approach the autonomic properties such as self configuring or self optimizing of cloud nodes are adjusted by switching on or off remote proxies and by modifying the distribution of information proxies. The information proxies have utilized to incorporate far located nodes if previously established nearer requesters may not be liked to continue with the service. As a formal description, a distant node that collect and distribute information to its range of vicinity is called information proxy, a disseminator is an autonomic cloud node that utilizes such a proxy and a potential proxy is an autonomous cloud node to which a proxy request to be sent. Selection of a potential proxy is done at arbitrary intervals or by using a cloud directory service. Because the nodes that take proxy requests are autonomous in nature the choice of making decision whether to allow or reject a request is purely on the discretion of receiver nodes. Autonomous cloud proxies process the self-optimizing protocol for evaluation and to finalize the acceptance or rejection of any receiving proxy request.

When an information proxy model is used in distributed resource matching environment, an autonomous cloud node is termed as the provider or requester of resources, or both. Autonomous cloud nodes build neighborhood nodes, which contain tiny collection of co-located cloud nodes having similar characteristics. So any neighborhood of autonomous cloud nodes cannot span over multiple clouds and is purely inherent to a particular cloud.

D. ACC with AI Techniques

In [5], the authors have described a way of designing autonomic cloud computing with the help of artificial intelligence techniques such as multi agent computing and biologically inspired computing. In this approach, autonomous cloud agents, capable of self monitoring/self correcting resource scheduling activities are implemented with multi agent systems. An agent enabled cloud consists of a mobile agent platform distributed on different virtual machines which are connected by a real or virtual network. A software agent installed on the front-end, acts as a proxy between the interface and agents technology by translating incoming requests into ACL messages (Agent Communication Language). Specialized agents are distributed in the cloud to perform their specific role. An agent platform is composed of JADE containers, which is an agent technology developed in Java.

A set of agent based services that has been conceived to support monitoring and management of cloud resources at the infrastructure level. Agents will be in charge of configuring the system, performing measures, computing statistics, collecting and managing information, detecting critical situation and applying reactions according to the decisions notified by the autonomic engine.

IV. ACC ENGINES

This section introduces various autonomic cloud engines designed so far. Similar to the various design aspects of autonomic cloud computing several approaches are used for the design of autonomic cloud engines. Various autonomic cloud engines available in the literature are: CometCloud[16], CHASE[27], Snooze[9] and Themis[4].

A. CometCloud

The goal of CometCloud[16] is to materialize a virtual computational cloud with resizable computing capability, which integrates local computing environments and public cloud services as per demand, and give mechanisms to support a collection of programming paradigms and applications requirements. Specifically, CometCloud enables policy-based autonomic cloudbridging and cloudbursting. Autonomic cloudbridging enables on-the-fly integration of local computational environments and public cloud services.

CometCloud [16] is based on a decentralized coordination substrate, and supports heterogeneous and dynamic cloud/Grid infrastructures, integration of public/private clouds and cloudbursts. The substrate coordination is used to accommodate a distributed and scalable work space which organizes the task scheduling, submitted by a dynamic set of users, to set of dynamically provisioning workers in private and/or public cloud resources based on their QoS constraints. These QoS constraints together with performance history, policies, and the state of resources are used for selecting the suitable size and mix of the public and private clouds.

A schematic overview [16] of the CometCloud architecture is presented in figure 2. The infrastructure layer uses the self organizing overlay and the Squid information discovery and content-based routing substrates. The routing engine supports flexible content-based routing and complex querying using partial keywords, wildcards, or ranges. It also guarantees that all peer nodes with data elements that match a query will be located. Nodes providing resources in the overlay have different roles and accordingly, different access privileges. The service layer gives a set of supporting services at the programming and application level and also gives a virtually shared abstraction as well as associative access primitives. This layer also supports tuple space coordination model. The programming layer provides the basic framework for the development of application and its management and supports a group of paradigms such as master/worker/BOT. Masters create tasks and workers consume these created tasks. Both the masters and workers can communicate through same virtual shared space or by using a direct connection. Monitoring and scheduling of tasks are controlled by the application framework.

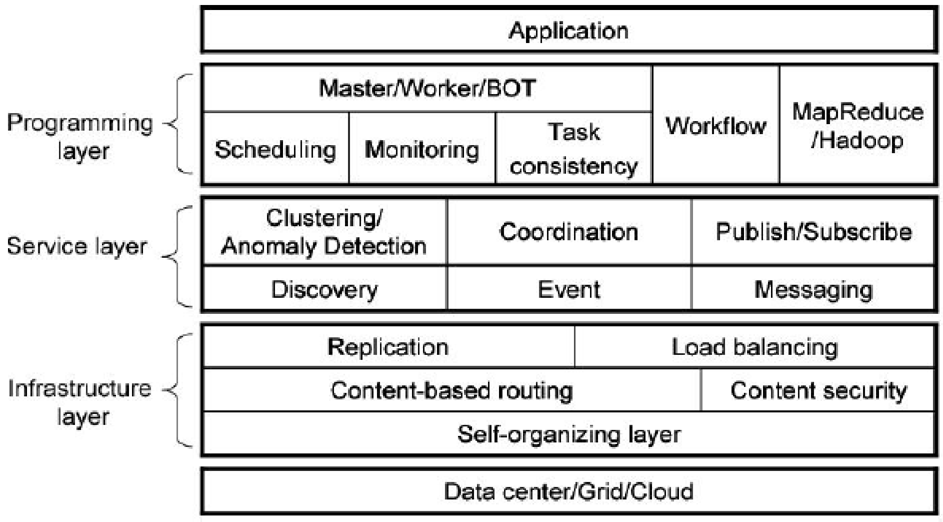
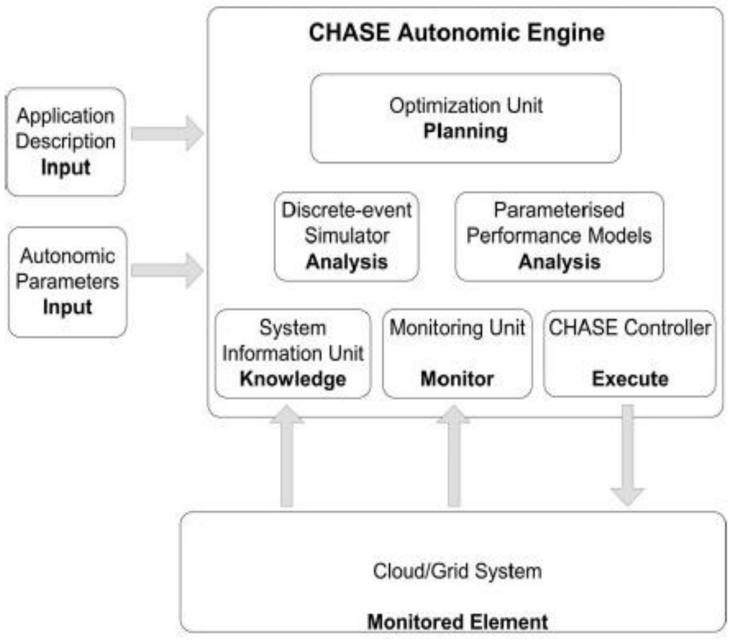


Fig. 2 Architecture of CometCloud

B. Chase

Chase[27] describes a prototype form of an implementation of the autonomic engine developed for addressing the problem of scheduling in Cloud environments by adding self optimization capabilities. Instead of other existing resource managers with *resource-centric* approach, chase engine uses an *application-oriented* approach. CHASE scheduler is operated based on the predictive valuation of the performance of applications in several scenarios in resource assignment, and by exploring the solution space till some user specified constraints are satisfied. Prediction of performance is based on simulations of the application for a given scenario of resource assignment by using a discrete-event simulator.


CHASE Autonomic Engine
Application Description Input
Autonomic Parameters Input
Optimization Unit Planning
Discrete-event Simulator Analysis
Parameterised Performance Models Analysis
System Information Unit Knowledge
Monitoring Unit Monitor
CHASE Controller Execute
Cloud/Grid System
Monitored Element
Fig. 3 Chase Autonomic Cloud Engine

The major functional blocks are input unit, planning unit, analysis unit, knowledge unit, monitoring unit and execute unit as depicted in figure 3.

Input Unit: The autonomic prediction is performed in the CHASE engine by using an Application Description and some of Autonomic Performance Parameters. The Application Description represents the behavior of the application utilized to drive the simulations and captures the fundamental operations executed by the application. The application description is generally represented using MetaPL, an XML-based language used for schema modeling, which gives support for the major processing constructs and libraries found in common scientific languages. The Autonomic Performance Parameters are to specify constraints for the resources selection to match with the required performance goals. Currently Two different varieties of performance parameters are defined such as:

- Constraint- the developer of the application imposes constraints for the resource selection. These predefined constraints restrict parameters as the total count of nodes to be used or the count of processes per node or the total count of processes in the system.
- Target- is to define the objective function which identifies whether this has to be maximized or minimized. Parallel efficiency and total execution time are the commonly used target parameters.

Planning Unit: The "smart" component in the architecture is Optimization Unit/Planning Unit. It is possible to limit the number of simulation executions that have to be performed for finding the suitable resource assignment configurations.
Analysis: The performance prediction is implemented with a Discrete Event Simulator, which is a Java-based function prototype for representing the system simulator's heterogeneous evolution and is capable of predicting the execution time for distributed message-passing applications processed in heterogeneous nodes.
Knowledge: In CHASE engine, System Information Unit (SIU) is responsible for gathering information regarding the elementary distributed system. This gathered information purely belongs to two different categories: resource parameters and benchmark results. In the Optimization Unit the resource parameters are used for building the system model and the simulator uses the benchmark results for predicting the time of execution on heterogeneous nodes.
Monitoring: This unit is to interact with the cloud or grid monitoring systems. The responsibility of the monitoring subsystem is to notify all parts, of possible degradations or failures which may lead to violations of performance agreements, so that countermeasures can be planned. CHASE performs performance predictions only at application startup time and once the application has uploaded on autonomically selected nodes, no further optimization is done.
Execute: The developed plan by the Optimization Unit is converted to original cloud resource building instructions by the CHASE Controller. This unit is capable of interacting with the cloud system and process towards the generation or initiation of virtual machines.

Chase utilizes an approach based on simulation for the prediction of application performance behavior, both in terms of response time and usage of resources. Normally applications are represented in a high level language and the applications considered are explicitly parallel HPC codes.

C. Snooze

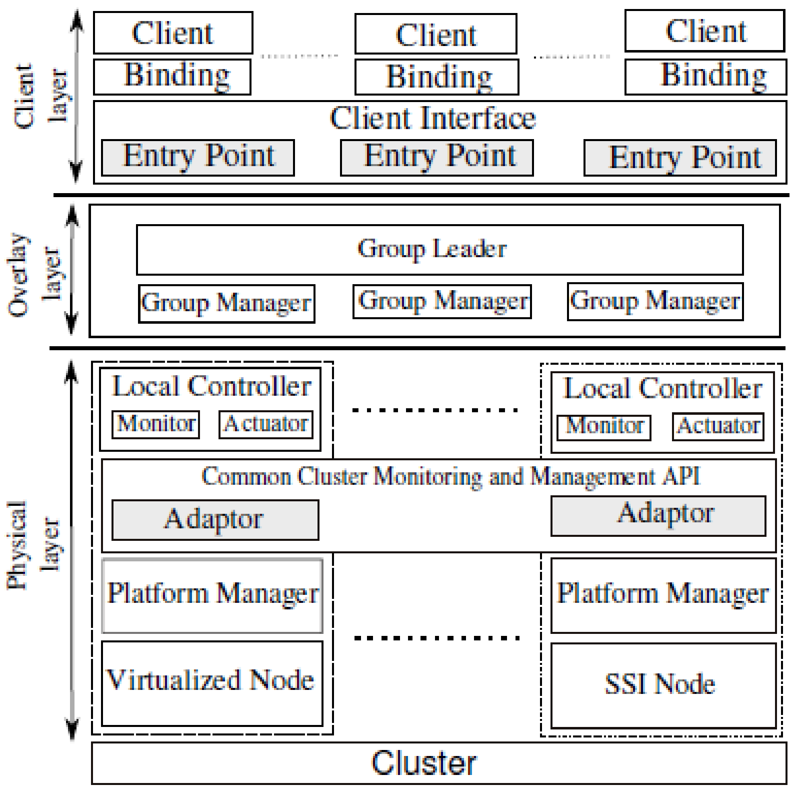


Fig. 4: Snooze Autonomic Cloud Engine

The system overview of Snooze is shown in Figure 4. The architecture is separated into three different layers: physical, hierarchical, and client [9]. At *physical layer* a cluster is formed by arranging machines, where a Local Controller (LC) controls each node. The *hierarchical layer* is to effectively manage the cluster and consist of fault-tolerant component units such as: Group Managers (GMs) and a Group Leaders (GL). Each GM is for managing a subset of LCs, and the GL keeps track of the information about GMs. The *client layer* gives a user interface which is normally built using a predefined set of replicated Entry Points (EPs) and is queried by the clients to discover the current GL. Clients EPs are interacted by using provided set of bindings and to provide a simple, flexible interface all system components are implemented as Java RESTful web services.

D. Themis

This section presents another autonomic cloud engine called Themis [4] which autonomically allocate resources according to the requirement of the system using an economy based approach. Themis uses a general scaling policy by using feedback control loops.

Themis system utilizes a spot market concept for effectively managing resources, thereby meets the Service Level Objectives (SLOs) of applications[4]. Themis system gives: (a) a relative share auction supported scheduler; (b) support for making feedback control loops which adjust the application bid and demand for resources to cope up with application SLOs. The relative share auction assures better utilization of resources and the fine grained job conserving nature optimizes the resource usage to the maximum level. The feedback control loops are basically built on generic scaling policies, meant for focussing particular types of applications.

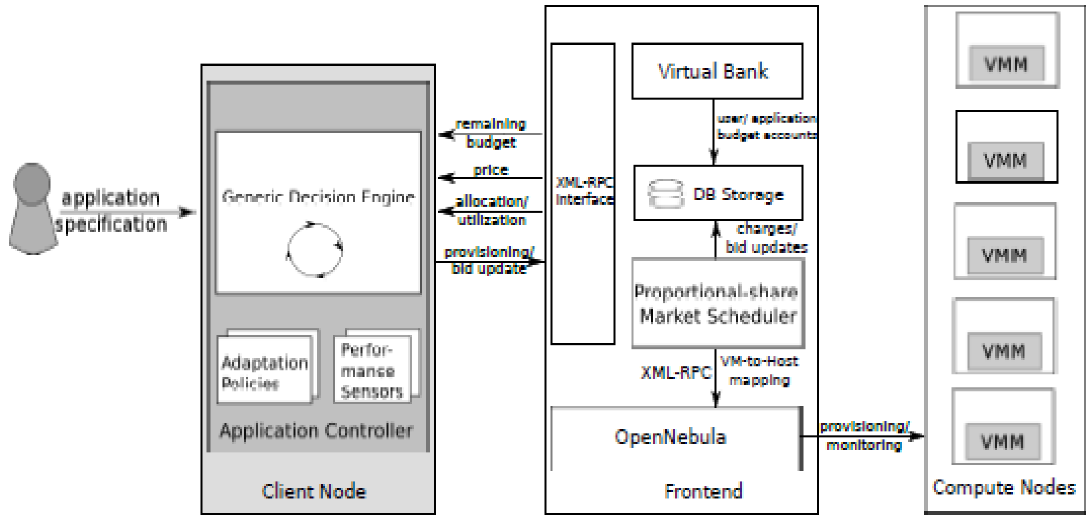


Fig. 5 Themis Architecture

Figure 5 provides an architectural overview of Themis system. This system depends on the usage of a virtual economy in which users get currency from a virtual currency manager called a Virtual Bank, allocated as per some pre-set policies. To execute the applications, users should provision virtual clusters for which they buy resources from a market-based scheduler that located on the infrastructure's front end. To distribute fractional components, of resources on hosts, the scheduler regularly executes a relative share auction [4]. To assure that each virtual machine is getting its paid share for maximizing the utilization of resources, the scheduler also does regular load balancing. The operations corresponding to management of virtual machines, networking and users are done by a Cloud IaaS manager.

For each type of application, there is a particular application controller. The monitoring of application is done by the application controller by using the feedback-based control loops to adjust the application's bid and resource requirements to the variations in price and user SLOs. At the time of execution of an application, the user generates an account and moves an initial budget of currency for it. The scheduling algorithm used by the market-based scheduler operates in two different steps such as:first it calculates the most suitable allocation the virtual machines can get and then does load-balancing to assure the created allocations.The Application controller manages each individual application and the demand for the resources are scaled as per the requirements. The application controller is designed with two types of scaling policies: (i) vertical and (ii) horizontal. The vertical policies are to adjust the given bids to scale the allocations of virtual machines and the horizontal policies are to adjust the provisioned set of virtual machines.

V. ACC: AUTONOMIC PROPERTIES IN CLOUD SERVICES

This section gives a brief description of various autonomic properties implemented in cloud environments and their effects in various cloud service applications. Mainly focussing autonomic properties are self configuring, self organizing, self optimizing and self healing. In addition to these basic self* properties, some researchers have explored autonomic isolation and autonomic testing of cloud services/applications. The following paragraphs describe the full details.

A. Self Configuring Cloud Services

In [21] the authors present an autonomic management framework to implement elasticity of consumption of resources by the cloud services. Elastic resources are very much effective at the service providers' end because they limit the cost of ownership and reduce the upfront capital investment. The service management framework is implemented with a set of mutually interacting agents where the interaction is done through event streams. Each agent follows an identical behavior pattern and performs a relatively simple function. By combining simple agents, management goal graphs are formed to carry out complex management tasks. The management goal graph describes the management task with a set of agents viewed as sensors, actors and effectors. Sensors take a set of events and give a new set of complex events; the actors do planning and decision-making and effectors are to incorporate changes in the managed system by enforcing the actor's plans.

The elastic cloud resources are materialized by self configuring and self organizing properties of autonomic computing by following the MAPE control loop. The sensors do monitoring and analyzing of processing of the Management Goal Graph by taking events and creating new events for the processing of another set of sensors or actors. The actors do planning in resource allocations with respect to the current performance and also forward the needed information to the effectors through events. The effectors are to process actions such as: (i) Modification of the Management Goal Graph like taking or providing an agent or event stream between agents and deploying a new policy to an already existing agent, for accommodating the in between changes in resource allocation, (ii) Modification of the service to incorporate changes in the underlying resource allocations, and (iii) Issue requests to the IaaS for modifying resource allocations for the service.

In [12] the authors present the details regarding a project, called MUSIC (Self adapting applications in a Ubiquitous Computing environment). The preliminary aim of MUSIC is to simplify the design of adaptive applications which will operate in dynamic, open and ubiquitous computational environments and adapt to changes without the need for human interaction. MUSIC has developed a comprehensive software development framework for automating adaptation of the software according to varying user needs at runtime.

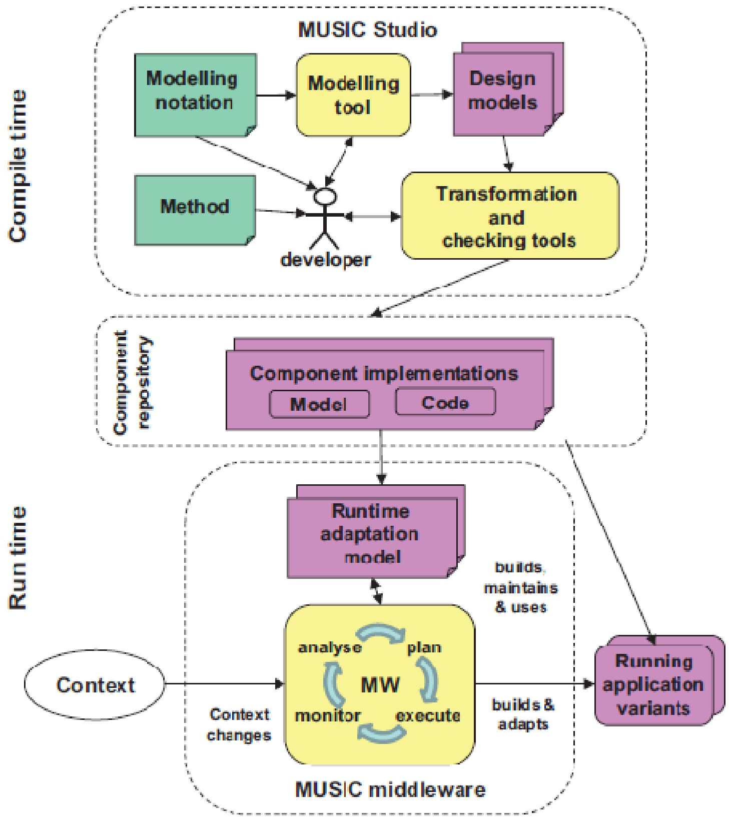


Fig. 6 Framework of MUSIC project

The design and development of autonomic cloud applications is simplified by using the wide support from the MUSIC framework [12], containing: (a) a modelling language for separation of concerns; (b) generic, reusable middleware components that automate context monitoring and management, and adaptation; and (c) tools to support the design and development of models annotated with context and convert them into run-time knowledge accessible to the middleware.

The middleware contains a control loop similar to the MAPE loop in autonomic computing, which monitors the related context sensors, and triggers a planning process at the time of significant changes, to find out whether adaptation is required or not. As a next level the planning process determines a new better configuration suitable for the current context, and then initiates the running application's adaptation. The evaluation of the utility of alternative configurations is also done by the planning process and also selects/adapts the most suitable application for the current context. MUSIC framework is capable of accommodating several adaptation mechanisms in a single adaptation framework such as: parameter setting, component replacement, redeployment, service rebinding etc.

The work of the MUSIC project has provided a coherent set of solutions which ease the task of developing context aware, self adaptive systems for mobile and ubiquitous computing scenarios. The current version of the MUSIC middleware offers only basic support for security because the project intentionally focused on the described adaptation aspects. A very promising aspect of the MUSIC technology is its ability to support "systems of systems" with the coordinated adaptation of sets of applications collaborating by discovering, providing and using services between them. This is based on state of-the-art service discovery and service level negotiation mechanisms. Thus, the MUSIC framework also provides a foundation for enabling applications to flexibly use services provided in the cloud, automatically adapting the usage of cloud-based services depending on quality of service and application context considerations.

In [3] the authors present the details about the autonomic management of cloud-based virtual networks (CVNs). A CVN is a network for interconnecting virtual computing resources such as virtual machines and a CVN customer can dynamically create, modify, and delete its CVN and can interconnect its CVN with its own existing on-premise networks. With CVNs, cloud customers can build new sites by effectively expanding their enterprise networks into the cloud and can avail the full benefits of cloud computing easily. For the effective utilization of CVN technology, some of its core objectives require autonomic management. These core objectives are: (a) Reachability isolation-VMs in a particular CVN could see the data from other VMs on the same CVN only; (b) Performance isolation: One CVNs networking activities must be immune to other CVNs activities; (c) Scalability; (d) Self-configuration and automated network bootstrapping and management; (e) Support various policies; and (f) Performance.

For the creation and management of CVNs several interactions between cloud customer and cloud-service providers are required and CVN Management Interface (CMI) provides a well-defined interface for this. A customer can easily and dynamically expand its network into the cloud with CVN and can build an enterprise network composed of multiple sites. Through the CMI a CVN provider receives users' requests and returns the results and a CVN user manages its own CVNs.

Specifically CMI exposes the following core functions [3] to CVN customers.

Life-cycle management: Create and deploy a new CVN. Modifying an existing CVN, and Delete an existing CVN.

Operation management: Start or stop a CVN.

Connect to existing network: Establish a protected connectivity (e.g., VPNs) between CVNs or between a CVN and a customers own networks.

Property update: Modify various properties of a CVN, such as size, addresses, performance specification, and policies, etc.

Monitoring: Retrieve various kinds of operational information about a CVN such as CPU utilization, memory usage, storage usage, a number of active VM instances, VM network interface statistics, CVN flow numbers, etc.

B. Self Optimizing Cloud Services

In [30] the authors present a study of an Application Layer Network on the basis of “Catallaxy” concept of F.A. Von Hayek. Catallaxy explains a "free market” concept of self organization approach for implementing electronic services and business resources. This is done by realizing resource allocation in Application Layer Networks (ALNs) where participants request and offer computing resources and application services of different complexity and value, thereby creates interdependent markets. The complex interdependencies are split up into two different related markets such as: a service market and a resource market. Service market is to trade application services and resource market is to trade computing and data resources, such as processors, memory, virtual machines and storage space. The functional difference between resource and service made, the hosting of different instances of the same service on different resources and the corresponding pricing mechanisms, possible.

The self organizing goals [30] of ALN can be realized like:

Adaptivity. Agents can be trained from their own experiences and from previous agreements and there is genetic recombination, mutation and selection. Agents are reactive and being able to adjust the goals to evolving environments.

IT Autonomy and initiative. The task to be processed is handled by complex service and if it finds an opportunity, the agent delegates specific task and objective to bring out in any way. The agent will find its way based on its own adaptation and learning, its own local knowledge, its own competence and reasoning. The effect of negotiation analysis reveals the successful application of the Catallactic strategy to Application Layer Networks.

Distribution and decentralization. Multi-agent systems used in Application Layer Networks are open and distributed. As the results of the simulations runs evidence, it is neither established, nor predictable, which agents will be involved in trades.

Hayek's Catallaxy could become an important concept to help understanding and designing such complex and dynamic agent communities. Applying the strategy to Application Layer Networks and particularly to cloud computing scenarios, offers the possibility to generate an open market for virtualised resources and standardized services.

In [20], the authors explain the probability of a decentralized approach for optimum allocation of resources in distributed systems, such as server farms structured as clouds of computing resources provided by different administrative domains. The approach proposed here relies on local autonomic features, embedded in computing resources, and their cooperation through gossiping on overlay networks. Self-optimization addresses load balancing and power saving issues. In a heterogeneous pervasive computing environment, a possible scenario is the creation of dynamic clouds of computing resources provided by multiple actors, including end-users, interconnected through the Internet, as proposed in [4]. These resources join and leave the cloud in an unplanned way, as peers in a P2P application, e.g., due to hardware/software/network failures. Such computing pervasive clouds must be enriched with supervision features to obtain decentralized server farms, but their characteristics (e.g., distribution, dynamicity, resource churn rate) prevent to adopt logically fully centralized solutions for introducing them: (a) a fully centralized solutions can not get an effective picture of whole system state, due to distribution and churn rate of resources and failures in the networks; (b) the elaboration of a global optimized solution could be useless, as the computed solution has become obsolete due to the dynamic changes in the cloud; (c) resources under different administrative domains would like to maintain independence in the decisions.

C. Self Healing Cloud Services

Recent developments in information technology made the resource sharing and remote collaboration of distributed systems easier and made the convergence of related approaches to the evolution of a single computing paradigm called Future Internet or Interclouds. Resource sharing like information dissemination and matchmaking are the essential focus on this convergence process. In [8], the authors have implemented the resource sharing and information dissemination using information proxies. Information proxies together with self-managing autonomous systems can be used to reduce the difficulty in administrative boundaries of clouds and interclouds on the way to converge different distributed systems. The purpose of an information proxy is to disseminate any required information like resource state, overall utilization, etc. as quantized packets of information.

Information dissemination using Proxy-based distribution is not a direct form and the actual source gives more perfect coverage with the freshest information. The intended benefits of proxy-based dissemination is of two-fold such as: it helps increase dissemination coverage with less overhead than is incurred with regular information dissemination, and it decreases the packet overhead by reducing the count of intermediate nodes that disseminates among the source and destination. Here the authors considered an autonomic network with self-optimizing, self- healing and self-configuring nodes, which dynamically adjust the information dissemination behavior by using the basic methods like: (i) switching on or off far located information proxies, or (ii) altering the distribution of information proxies. These information proxies will be utilized for attracting far located nodes when prior selected requesters are not interested to the providers.

Here a remote node that receives and disseminates information to its vicinity is called an information proxy, and the autonomous cloud node that utilizes such a proxy is called the disseminator and a candidate autonomous cloud node to which a proxy request may be sent is called a potential proxy. In this resource sharing approach, a potential proxy is selected at random from the cloud service directory. The nodes which collect proxy requests are autonomous and so whether to process or reject a coming request is decided as per the discretion of the receiving node. Autonomous cloud proxies also decide whether to reject or accept the incoming proxy request by running the self-optimizing protocol evaluation. In this environment, an autonomous cloud node can be viewed as a resource requester, a resource provider, or both.

Autonomous cloud nodes create neighborhoods consisting of several sets of co-located nodes having identical properties. The concept of neighborhood is specific to a specific cloud and it cannot spread over several clouds. The neighborhood leader is again treated as an information proxy and is capable of distributing information to the neighborhood. The neighborhood leader is for collecting the neighborhood resource information, record keeping, disseminating aggregated information into the neighborhood and outside of the neighborhood, proxy selection, and coordinating reservation requests. Here the information proxies are proposed as the main mechanism for enabling convergence of clouds and grids to form the next generation of distributed systems because, proxies have the ability to bypass administrative barriers and improve dissemination performance with respect to different parameters. The autonomic properties like self configuring and self healing properties are focused more in this implementation of intercloud federation.

D. Autonomic Isolation as a service

In [31] the authors present the autonomic isolation of IaaS resources using an autonomic management framework. Here IaaS infrastructure is viewed as a layered resource having three main layers: physical layer, hypervisor (OS) layer and application (VM) layer. The basic assumptions made are: (i)Cloud resources provide well-managed interfaces for capturing its state (detection phase), and to do processing actions on it (reaction phase), from both network and system perspectives; (ii) decision-making components in each layer (autonomic managers), individually give the needed interface to realize collective management behavior (horizontal orchestration); and (iii) each layer offers the needed management interfaces to perform synchronization between layers (vertical orchestration).

Two types of orchestrators guarantee overall consistency of self-management of cloud resource isolation. **Vertical Orchestrator** synchronizes the management of overall cloud resource isolation between different layers, by enforcing some specific rules on required layers, as per administrator decided policies. **Horizontal Orchestrators** are to synchronize each layer's cloud resource isolation behavior between computing and networking views of infrastructure resources. Each view forwards collected information to the orchestrator, it consolidates the knowledge and gives to Vertical Orchestrator. At the time of modification of a local layer isolation policy, individual views are modified by the Horizontal Orchestrator using specified interfaces. Thus, orchestrators implement a systematic, layered and hierarchical model of self-management to cloud resource isolation.

The sample autonomic cloud resource isolation framework is implemented on an IaaS infrastructure where the physical resources are provided by dedicated network equipments. Network traffic is separated by a firewall and switch through VLAN tables and routing tables and are controlled by autonomic loop. Through the horizontal and vertical orchestration of various autonomic loops the framework automates administration and overcome fragmentation of security components. A preliminary use case is: if a virus is initiated on a VM it is caught at the detection engine and the vertical orchestrator then forwards a group of requests such as: disable connection corresponding to this VM at the hypervisor level, do VM migration to a special physical machine, clean the VM, and do one more migration to move the VM to its initial machine.

E. Autonomic Testing as a service

King et al.[17] gives a brief mechanism of test managers with a complete description of the autonomic self-testing approach In [18] they have developed an automated test harness for a cloud service, which is to provide test support as-a-service (TSaaS). The authors have implemented an approach called autonomic self-testing (AST) by utilizing autonomic computing techniques to runtime testing of distributed applications in dynamically adaptive systems. AST is framed with two general strategies such as: (1) replication with validation and (2) safe adaptation with validation. The first strategy adaptively tests changes in the managed resources where copies are preserved for the purpose of validation, and the second strategy directly tests in-place adaptive changes on managed resources. Test actions for the service generate a standard model for testing, that identifies the preconditions, application of inputs, and assertions that are to be considered in the form of probable outcomes. The preconditions are interpreted to code scripts for automatic testing tools. Autonomic managers designed for runtime testing actions use this test model to materialize autonomic self testing.

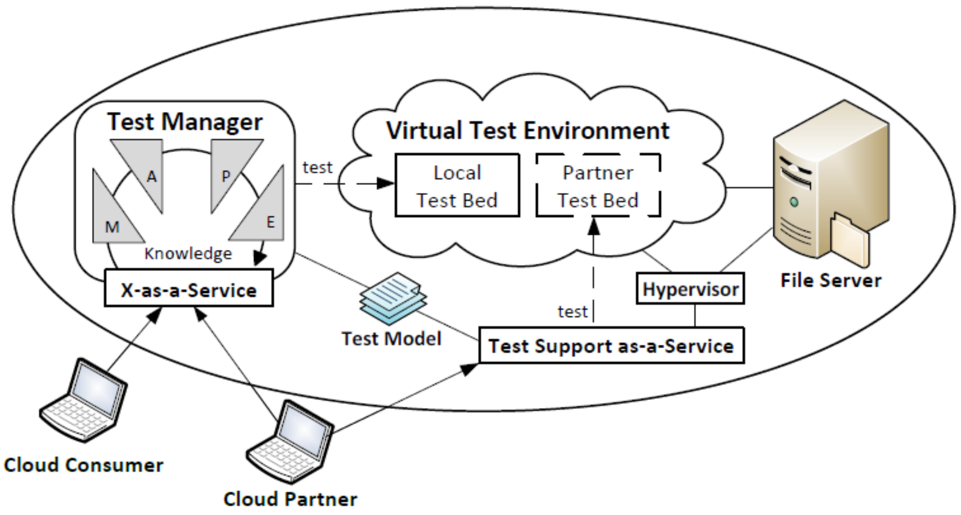


Figure 7: Test Support as-a-Service: Autonomic Self-Testing in Cloud

Test harness development. During the harness development process, the automated test sub-procedures are framed as operations and are implemented on a separate test bed. The operations are then interfaced through an API for test support for parties developing cloud services and applications to extend the hosted service. Pre-deployment testing of cloud services includes validating them for accurate functionality, interoperability, operational integrity, performance, security, fault tolerance, among others by writing test cases that check the software against the said quality attributes, and by creating the test executions. Figure 7 shows the structure of a cloud provider with this approach.

Each cloud provider hosts some X-as-a-Service and that is given to cloud consumers over the Internet. When the hosted service dynamically adapts or updates, a test manager (top-left) intercepts the change and validates it on a local test bed. The local test bed provides the hardware, software, and networking components necessary to validate the change using a copy of the hosted service.

F. Summary of Incorporated Autonomic Properties

The autonomic cloud services described in previous sections implemented by incorporating self managing properties to cloud computing affects various measurements of the system quality and thus became a solution to satisfy cloud computing needs in different aspects such as performance and security. Table 1 presents a consolidation of the self* properties incorporated, applied computing resource and the effect in terms of system quality/performance.

Table 1: Consolidation of embedded autonomic properties

Autonomic Properties	**Applied Resource**	**Quality Metrics Authors**	**Authors**
Self configuring /Self optimizing	Application layer networks	Usability, Maintainability, Efficiency and Functionality	Streitberger et.al.[30]
Self configuring	Software modules	Usability, Maintainability, and Functionality	Hallsteinsen et. al.[12]
Self configuring	VM resources	Usability, Maintainability, and Functionality	Martin et.al.[21]
Self Testing	Cloud services and applications	Efficiency and Functionality	King et. al.[17],[18]
Autonomic Isolation/ Self configuring	Computing and Networking resources in cloud	Usability, Maintainability, and Functionality	Wailly et.al.[31]
Self optimizing	Computing resources in server clusters	Maintainability, Efficiency and	Streitberger et.al.[30],

		Functionality	Manzalini et.al.[20]
Self configuring /self healing	Cloud services and applications	Maintainability, Efficiency, Reliability and Functionality	Erdil et.al.[8]
Self configuring	Cloud based virtual networks	Usability, Maintainability, Efficiency and Functionality	Choi et.al.[3]

VI. CONCLUSIONS AND FUTURE WORKS

Autonomic techniques stimulated by biological systems, enabled the development of self managing properties in computing systems and applications which are very well helpful to the dynamic behaviors of cloud computing environments. This paper reviewed various aspects of research in autonomic cloud computing. Here we discussed the details regarding the design and implementation of autonomic cloud computing. Attempts made by various researchers to incorporate autonomic properties, like self configuring, self healing, self optimizing and self protecting, in cloud services and corresponding autonomic cloud variants are also discussed here. The mostly tried autonomic properties are self configuring and self optimizing properties and the remaining self healing and self protecting properties require some more exploration in the field.

REFERENCES

[1] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crumme, et al. *The grads project: Software support for high-level grid application development*. International Journal of High Performance Computing Applications, 15(4):327–344, 2001.

[2] N. Bonvin, T. G. Papaioannou, and K. Aberer. Autonomic sla-driven provisioning for cloud applications. In Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on, pages 434–443. IEEE, 2011.

[3] T. Choi, N. Kodirov, T.-H. Lee, D. Kim, and J. Lee. Autonomic management framework for cloud-based virtual networks. In Network Operations and Management Symposium (APNOMS), 2011 13th Asia-Pacific, pages 1–7. IEEE, 2011.

[4] S. V. Costache, N. Parlavantzas, C. Morin, S. Kortas, et al. Themis: A spot-market based automatic resource scaling framework. In HPDC 2012 Poster Session, 2012.

[5] A. Cuomo, M. Rak, S. Venticinque, and U. Villano. Enhancing an autonomic cloud architecture with mobile agents. In Euro-Par 2011: Parallel Processing Workshops, pages 94–103. Springer, 2012.

[6] P. T. Endo, D. Sadok, and J. Kelner. Autonomic cloud computing: giving intelligence to simpleton nodes. In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pages 502–505. IEEE, 2011.

[7] D. C. Erdil. Dependable autonomic cloud computing with information proxies. In Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, pages 1518– 1524. IEEE, 2011.

[8] D. C. Erdil. Autonomic cloud resource sharing for intercloud federations. Future Generation Computer Systems, 2012.

[9] E. Feller and C. Morin. Autonomous and energy-aware management of large-scale cloud infrastructures. In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, pages 2542–2545. IEEE, 2012.

[10] E. Feller, C. Rohr, D. Margery, and C. Morin. Energy management in iaas clouds: A holistic approach. In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, pages 204–212. IEEE, 2012.

[11] A. Goscinski and M. Brock. Toward dynamic and attribute based publication, discovery and selection for cloud computing. Future Generation Computer Systems, 26(7):947–970, 2010.

[12] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G. A. Papadopoulos. A development framework and methodology for self-adapting applications in ubiquitous computing environments. Journal of Systems and Software, 2012.

[13] M. Holze and N. Ritter. System models for goal-driven self-management in autonomic databases. Data & Knowledge Engineering, 70(8):685– 01, 2011.

[14] J. O. Kephart and D. M. Chess. The vision of autonomic computing. Computer, 36(1):41–50, 2003.

[15] H. Kim, Y. El-Khamra, I. Rodero, S. Jha, and M. Parashar. Autonomic management of application workflows on hybrid computing infrastructure. Scientific Programming, 19(2):75–89, 2011.

[16] H. Kim and M. Parashar. Cometcloud: An autonomic cloud engine. Cloud Computing: Principles and Paradigms, pages 275–297, 2011.

[17] T. M. King and A. S. Ganti. Migrating autonomic self-testing to the cloud. In Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on, pages 438–443. IEEE, 2010.

[18] T. M. King, A. E. Ramirez, R. Cruz, and P. J. Clarke. An integrated self testing framework for autonomic computing systems. Journal of Computers, 2(9):37–49, 2007.

[19] H. Liu, V. Bhat, M. Parashar, and S. Klasky. An autonomic service architecture for self-managing grid applications. In Grid Computing, 2005. The 6th IEEE/ACM International Workshop on, pages 8–pp. IEEE, 2005.

[20] A. Manzalini and C. Moiso. Self-optimization of resource allocation in decentralised server farms. In Intelligence in Next Generation Networks (ICIN), 2011 15th International Conference on, pages 219–224. IEEE, 2011.

[21] P. Martin, A. Brown, W. Powley, and J. L. Vazquez-Poletti. Autonomic management of elastic services in the cloud. In Computers and Communications (ISCC), 2011 IEEE Symposium on, pages 135–140. IEEE, 2011.

[22] M. Maurer, I. Breskovic, V. C. Emeakaroha, and I. Brandic. Revealing the mape loop for the autonomic management of cloud infrastructures. In Computers and Communications (ISCC), 2011 IEEE Symposium on, pages 147–152. IEEE, 2011.

[23] H. Mearns, J. Leaney, A. Parakhine, J. Debenham, and D. Verchere. An autonomic open marketplace for inter-cloud service management. In Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on, pages 186–193. IEEE, 2011.

[24] G. Papuzzo and G. Spezzano. Autonomic management of workflows on hybrid grid-cloud infrastructure. In Proceedings of the 7th International Conference on Network and Services Management, pages 230–233. International Federation for Information Processing, 2011.

[25] A. Paschke and M. Bichler. Knowledge representation concepts for automated sla management. Decision Support Systems, 46(1):187–205, 2008.

[26] S. Perera and D. Gannon. Enforcing user-defined management logic in large scale systems. In Services-I, 2009 World Conference on, pages 243– 250. IEEE, 2009.

[27] M. Rak, A. Cuomo, and U. Villano. Chase: an autonomic service engine for cloud environments. In Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on, pages 116–121. IEEE, 2011.

[28] A. S´anchez, J. Montes, M. S. P´erez, and T. Cortes. An autonomic framework for enhancing the quality of data grid services. Future Generation Computer Systems, 28(7):1005–1016, 2012.

[29] B. Solomon, D. Ionescu, M. Litoiu, and G. Iszlai. Designing autonomic management systems for cloud computing. In Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference on, pages 631–636. IEEE, 2010.

[30] W. Streitberger and T. Eymann. A simulation of an economic, self organising resource allocation approach for application layer networks. Computer Networks, 53(10):1760–1770, 2009.

[31] A. Wailly, M. Lacoste, and H. Debar. Towards multi-layer autonomic isolation of cloud computing and networking resources. In Network and Information Systems Security (SAR-SSI), 2011 Conference on, pages 1–9. IEEE, 2011.