Resume of "QoS-Aware Autonomic Resource Management in Cloud Computing: A Systematic Review"

(Skipping Chapters 2, 4, 9, 10, and 11 as requested)

Section 1: Introduction and Motivation

Resource management in large, heterogeneous, and distributed cloud environments presents significant challenges due to the inherent characteristics of such environments, including the uncertainty and dispersion of resources, heterogeneity, dynamism, and failures[cite: 1, 2]. Existing resource management techniques often prove insufficient to handle these complexities effectively[cite: 3]. To ensure efficient performance of workloads and applications, these characteristics must be addressed[cite: 4]. This paper focuses on a systematic literature analysis of autonomic resource management in cloud computing, particularly emphasizing Quality of Service (QoS)-aware approaches[cite: 5].

Cloud computing offers services (Infrastructure as a Service - IaaS, Platform as a Service - PaaS, Software as a Service - SaaS) on a pay-per-use basis[cite: 27]. Efficiently monitoring and measuring these services requires QoS, and adherence to Service-Level Agreements (SLAs) is crucial for their delivery[cite: 27]. However, providing dedicated cloud services that meet dynamic user QoS requirements and avoid SLA violations is a major challenge[cite: 28]. Current cloud services are often provisioned based on resource availability without guaranteeing expected performance[cite: 29].

To address this, two important aspects are needed: QoS-awareness and autonomic (or self-) management of cloud services[cite: 31].

- **QoS-aware** implies a service's capacity to be conscious of its own behavior to ensure elasticity, high availability, reliability, and manage cost and time[cite: 32].
- **Autonomic management** means the service can self-manage according to environmental needs[cite: 33]. Autonomic systems, drawing inspiration from biological systems, are designed to maintain stability in unpredictable conditions and adapt to new environmental factors like hardware or software failures, guided by human-defined policies and QoS parameters[cite: 35, 36, 37]. They aim to proactively manage system complexity and optimize resources[cite: 37].

Current challenges include the lack of providers offering comprehensive autonomic services; Amazon Web Services (AWS) provides some integrated autonomic services but with limited customization[cite: 38]. Existing autonomic systems also tend to consider only a few QoS parameters, whereas a comprehensive approach would include availability, security, execution time, SLA violation rate, and more[cite: 39, 40]. The core problems in resource allocation—heterogeneity, dynamism, and failures—necessitate self-management capabilities in cloud systems: self-optimizing, self-healing, self-protecting, and self-configuring[cite: 42, 43, 44, 45, 46]. Autonomic cloud computing aims to manage applications efficiently by fulfilling QoS requirements without human intervention[cite: 48, 50].

1.1 Need for Autonomic Cloud Computing Traditional resource allocation policies struggle to meet QoS requirements and maximize resource efficiency in the dynamic cloud environment[cite: 52]. Autonomic cloud computing offers a solution by effectively allocating resources, fulfilling user QoS, and automatically healing unexpected failures[cite: 53]. Its objectives are:

1. To identify and schedule suitable resources for appropriate workloads in a timely manner to enhance resource utilization. This means using the minimum resources needed for a workload to maintain a

required service quality, or to automatically minimize completion time or maximize throughput[cite: 54, 55].
2. To identify suitable workloads that support the scheduling of multiple workloads concurrently, capable of fulfilling diverse QoS requirements (e.g., CPU utilization, availability, reliability, security) for both heterogeneous and homogenous workloads[cite: 57, 58].

1.3 Our Contributions This research provides:

- A comprehensive investigation of various existing autonomic resource management techniques in cloud computing, covering resource provisioning and scheduling[cite: 63].
- A comparison and categorization of these techniques based on the common characteristics and properties of self-management (self-healing, self-configuring, self-optimizing, and self-protecting) [cite: 64].

(Skipping Section 2: History of Autonomic Computing)

Section 3: Autonomic Cloud Computing: The Background

Autonomic systems, guided by QoS parameters, are inspired by biological systems capable of handling uncertainty, heterogeneity, dynamism, and faults[cite: 108]. Their goal is to execute applications within deadlines, fulfilling user-described QoS requirements with minimal complexity[cite: 109]. These systems function much like the human Autonomic Nervous System (ANS), which manages bodily functions dynamically and without conscious effort[cite: 110, 111, 112]. Similarly, Autonomic Cloud Computing Systems (ACCSs) control cloud-based systems and applications without human involvement, exhibiting properties like self-healing, self-protecting, self-configuring, and self-optimizing[cite: 112, 113].

3.1 Overview of Autonomic Cloud Computing ACCSs are often based on IBM's autonomic reference model, which includes a control loop with four steps: Monitor, Analyze, Plan, and Execute (MAPE)[cite: 114, 115]. This model (Figure 2 in the paper) also involves two interfaces for environmental interaction (Sensors and Effectors) and a Knowledge Base to store rules and policies[cite: 116, 117].

- ACCSs are composed of Autonomic Elements (AEs), where an Autonomic Manager (AM) acts as an intelligent agent. The AM interacts with the environment via manageability interfaces (Sensors and Effectors) and takes actions based on input from sensors and rules defined in the knowledge base[cite: 118]. The AM is configured by an administrator using high-level policies[cite: 119].
- **MAPE Loop Phases:**
  - **Monitor:** Continuously collects information (e.g., on QoS parameters) from sensors[cite: 120].
  - **Analyze and Plan:** These modules analyze the monitored information and devise a plan for appropriate actions in response to system-generated alerts[cite: 121].
  - **Executor:** Implements the plan after analysis, aiming to maintain QoS parameter values by tracking changes and acting according to rules in the knowledge base[cite: 122, 123, 124].
  - **Effector:** Used to apply changes to the managed resources or disseminate new policies, rules, and alerts within the autonomic system[cite: 125].

3.2 Self-Management Properties of Autonomic Cloud Computing Cloud computing self-management encompasses four generic properties (Figure 3 and Table I in the paper)[cite: 126]:

- **Self-healing:** The ability to automatically detect, diagnose, and recover from faults or failures, thereby improving performance through fault tolerance[cite: 133].

- **Self-configuring:** The capability to adapt dynamically to changes in the operational environment, such as by automatically installing new components or updating existing ones based on high-level policies[cite: 133].
- **Self-optimizing:** The ability to continuously seek opportunities to improve performance and efficiency, for example, by reallocating resources to complete workloads or reducing resource overloading/underloading[cite: 133].
- **Self-protecting:** The capacity to anticipate, detect, identify, and protect against various threats and attacks to maintain system security and integrity[cite: 133].

3.3 Evolution of Autonomic Cloud Computing This section (illustrated by Figure 4 in the paper) outlines the progression of research in autonomic cloud computing from 2009 to 2015, highlighting key techniques and their focused QoS parameters each year:

- **2009:** Focus on workload provisioning (e.g., clustering-based pattern detection) and self-optimization techniques (e.g., using utility theory for SLA adherence, scalability, availability)[cite: 129, 130, 135].
- **2010:** Introduction of techniques for anomaly detection (e.g., Bayesian Networks), SLA awareness (considering cost, availability), and managing workloads based on deadlines and budgets (e.g., Cloud Auto Scaling - CAS by Mao et al. for VM instance management)[cite: 137, 138, 139, 142, 144].
- **2011:** Development of approaches for fault tolerance (e.g., Adaptive Fault Tolerance in Real-Time Cloud - AFRTC), market-based resource allocation (e.g., ARAS-M using genetic algorithms), hybrid bio-inspired mechanisms for peak load handling, and SLA-aware virtualization focusing on service brokering and agreement negotiation[cite: 152, 153, 155, 163, 167, 168].
- **2012:** Emergence of techniques focused on self-healing through monitoring and recovery, self-configuration via power-aware adaptive resource management, and energy-based optimization using methods like dynamic voltage frequency scaling[cite: 170, 171, 172, 174].
- **2013:** Introduction of adaptive SLA-based techniques (e.g., Case-Based Reasoning - CBR), self-protecting systems (e.g., Architecture-Based Self-Protection - ABSP using patterns for threat detection), and cost-based query services (e.g., COCCUS with a centralized architecture for managing scheduling policies and budget information)[cite: 175, 176, 177, 181, 183].
- **2014:** Proposal of dynamic workload distribution techniques like Autonomic Workload Manager (AWM), which uses dynamic scalability and modified auto-scaling algorithms for queuing network models[cite: 188, 189].
- **2015:** Focus on resource contention-aware scheduling, exemplified by the Autonomic Resource Contention Scheduling (ARCS) technique for distributed systems[cite: 194, 195].

3.4 QoS-Aware Cloud The agreement between cloud consumers and providers centers on parameters like price, time, and other QoS aspects[cite: 198]. QoS-aware cloud architectures aim to satisfy application QoS requirements through efficient resource management and dynamic configuration to avoid issues like over-provisioning[cite: 200, 201, 202]. Frameworks like "Q-Cloud" were proposed to manage resource allocation and mitigate workload interference by introducing QoS states (Q-States), offering users flexibility in defining application-specific QoS levels[cite: 204, 205, 206, 207]. Other approaches involve two-level architectures that separate application QoS specifications from the actual allocation and provisioning of resources, using application-oriented local decision models[cite: 210, 211, 212, 213].

3.5 SLA and QoS: Intertwined in the Autonomic Cloud Cloud services often comprise components from different providers, making End-to-End (E2E) QoS management crucial, as degradation in one component can affect the global service[cite: 217, 218, 219]. SLAs are key agreements formalizing QoS expectations between clients and providers, necessitating continuous supervision of QoS parameters due to the cloud's dynamic nature (Figure 5 in the paper illustrates user-provider SLA negotiation for various QoS parameters) [cite: 222, 223, 221]. Unstable user demands can lead to SLA violations if QoS degrades[cite: 229, 230]. Therefore, QoS aspects must be considered from the design phase of cloud components[cite: 232, 233]. A self-management approach is essential to guarantee the E2E behavior of the global service, with autonomic systems playing a key role in managing virtual machines and application QoS requirements[cite: 234, 235, 246].

3.6 QoS-Based Grid and Cloud A comparison between QoS-based Grid and Cloud computing (Table II in the paper) highlights differences in service management (on-demand vs. preinstalled), QoS management focus (critical vs. noncritical applications), and SLA fulfillment (autonomic self-adaptation vs. request/response models)[cite: 250, 251, 252, 239]. QoS-based cloud services tend to use less storage, can execute urgent applications more effectively, and leverage autonomic systems for service delivery[cite: 239]. Research issues in QoS-based Grid computing include variation in resource availability, challenges in parallel processing, and decentralized control (Table III)[cite: 253, 241]. Key research issues in QoS-based Cloud computing involve managing increasing resource costs (due to workload fluctuations), workload execution time (due to resource uncertainty and system characteristics), and power consumption (Table IV)[cite: 258, 257].

(Skipping Section 4: Review Technique)

Section 5: Extraction Outcomes

This section presents the results of the systematic literature review covering 110 research articles on autonomic cloud computing published between 2009 and 2015[cite: 286, 290].

- **Publication Venues:** Most research articles were published in journals (48%) and conferences (35%), with smaller percentages in workshops (5%) and symposiums (12%)[cite: 291, 292].
- **Focus of Self-Management Properties (Figure 7):** The distribution of research focus among the four self-management properties was:
  - Self-Optimizing: 44% [cite: 293]
  - Self-Configuring: 31% [cite: 293]
  - Self-Healing: 16% [cite: 293]
  - Self-Protecting: 9% [cite: 293]
- **QoS Parameters Considered (Figure 8):** The most frequently addressed QoS parameter in the reviewed papers was:
  - Cost: 20% [cite: 294]
  - Other prominent parameters included Resource Utilization (17%), Execution Time (17%), Response Time (16%), and Energy (15%)[cite: 299].
  - Scalability and Security were the least frequently considered, each at 2%[cite: 294].
- **Trends Over Time (Figure 9):** A time-based count illustrates the research emphasis on different self-management areas from 2009 to 2015[cite: 300].

Section 6: Phases of Autonomic Resource Management in Cloud

Autonomic resource management in the cloud is categorized into six interconnected phases (Figure 10 in the paper)[cite: 301]:

6.1 Design of Application (Figure 11 taxonomy) The design of cloud applications influences resource needs and management strategies.

- **Type of Application:** Cloud applications consist of tasks. They are divided into:
    - **Workload:** An abstraction of work to be performed (e.g., running a web service, Hadoop data node). Can be homogenous or heterogeneous[cite: 310, 311].
    - **Workflow:** A set of interrelated tasks and their distribution, often represented using Directed Acyclic Graphs (DAGs) where nodes are tasks and edges are relationships[cite: 310, 312, 313].
- **Domain of Application:** Includes scientific applications (e.g., business, High-Throughput Computing (HTC), High-Performance Computing (HPC)), healthcare (requiring high throughput, availability, scalability), biological applications (managing large datasets with extensive I/O), and geoscience applications (processing geospatial and nonspatial data, e.g., GIS)[cite: 315, 316, 317, 318, 319, 320, 321, 322, 323, 326, 327].
- **Application Presentation:** Applications are often designed using data definition languages like XML (Extensible Markup Language) or GUI-based tools (e.g., Petri Nets - PNs for visualization) for easier presentation and management[cite: 328, 329, 330, 331, 332].
- **I/O Data Requirements:** Considers different types of data (input, intermediate, output) and their volume (large or small). Some applications require sequential data input, others parallel[cite: 332, 333, 334, 335, 336, 337].

6.2 Workload Scheduling (Figure 12 taxonomy) This phase involves assigning appropriate resources to workloads and mapping workloads to resources based on QoS.

- **Architecture:**
    - **Centralized:** A single central controller makes all scheduling decisions. Scalability can be an issue[cite: 343, 344].
    - **Hierarchical:** Multiple levels of schedulers, with higher-level schedulers controlling lower-level ones. Failure of central control can affect the system[cite: 343, 346, 349, 350].
    - **Decentralized:** Resources are assigned based on individual workload requirements, offering scalability but often more suited for homogenous workloads and resources[cite: 343, 351, 352].
- **Objective:** The scheduler performs matchmaking (mapping workloads to available resources) based on QoS requirements. The aim is to optimize by minimizing cost and time while maximizing resource utilization, including load balancing[cite: 353, 354, 355, 356, 357, 358, 359].
- **Decision:**
    - **Static Scheduling:** Resource mapping is done before execution based on user requirements[cite: 361, 362].
    - **Dynamic Scheduling:** Resources are mapped at runtime, providing better scalability and performance by reducing resource wastage[cite: 361, 362, 363].
- **Integration:** Involves combining the results of different execution units or individual task schedulers, often facilitated by a broker that tracks execution status, especially in Service-Oriented Architectures (SOA). Can be separate or combined[cite: 364, 365, 366, 367, 368].

6.3 Allocation (Figure 13 taxonomy) This is the process of assigning provisioned resources, typically managed by a Cloud Resource Manager (CRM).

- **Decision Criteria:** How decisions for resource scheduling and mapping are made.
    - **Independent Decisions:** Each scheduler acts independently without considering overall resource utilization status[cite: 384, 385].
    - **Mutual Decisions:** Decisions are made with coordination among different schedulers, reducing resource contention[cite: 384, 386, 387].
- **Coordination Mechanism:**

  - **Group-Based:** Resources are shared among groups of cloud users with similar requirements. SLAs are established between groups. This approach can improve performance and fault tolerance[cite: 389, 390, 391, 392].
  - **Market-Based:** Uses negotiation (via SLAs) for resource provision between interested economic entities (users buying, providers selling computing resources)[cite: 389, 393, 394, 395].
- **Intercommunication Protocol:**
  - **One-to-One:** One consumer interacts with one provider based on a negotiated SLA[cite: 395, 396].
  - **One-to-Many:** One cloud provider serves multiple cloud users, which can lead to high network bandwidth consumption[cite: 395, 396].

6.4 Monitoring (Figure 14 taxonomy) Essential for performance optimization by tracking resource utilization and system status. Information is gathered by sensors and changes implemented by effectors.

- **Execution Monitoring:** Tracking the status of tasks/workloads (e.g., started, queued, completed, failed)[cite: 406, 407].
  - **Active Monitoring:** The agent continuously checks execution and modifies procedures based on information from other schedulers, often sending status updates automatically[cite: 407, 408, 409].
  - **Passive Monitoring:** A local agent monitors execution status against user-defined QoS requirements in SLAs[cite: 407, 410].
- **Status Monitoring:** Autonomic Elements (AEs) monitor system performance, resource utilization, memory usage, network usage, and SLA deviations to prevent violations[cite: 411, 412, 413].
- **Service Monitoring:** Collects information about free resources, resource utilization status, and processor load[cite: 414, 415].
  - **Centralized Service:** Monitored data stored in a central repository; may not scale well[cite: 417, 418, 421].
  - **Decentralized Service:** Load is balanced, and fault tolerance is provided more efficiently[cite: 417, 422].
- **Resource Usage Monitoring:** Measures resource utilization (e.g., memory, processor) to avoid underloading and overloading. It supports both consumer goals (minimum cost/time, SLA adherence) and provider goals (minimum resources for execution)[cite: 423, 424, 425, 426, 427, 428, 429].

6.5 Self-Management (Figure 15 taxonomy) Focuses on the four key properties of autonomic systems.

- **Self-Healing:** The system's ability to automatically identify, analyze, and recover from various faults (e.g., configuration changes, resource unavailability, overloading, memory shortages, network failures)[cite: 430, 431, 432]. This improves performance through fault tolerance. Techniques include:
  - **Checkpointing:** Transferring a failed workload to other available resources to resume execution from the point of failure[cite: 433, 434].
  - **Failure Forecasting:** Predicting future resource requirements to prevent execution failures[cite: 433, 435].
  - **Replication:** Executing a workload on more than one resource to increase the chances of successful completion[cite: 433, 436, 437].
- **Self-Configuring:** The system's capability to adapt to changes in its environment, such as by automatically installing missing or outdated components or reinstalling components based on alerts, without human intervention[cite: 437, 438, 440, 441].

- **Self-Optimizing:** The system's ability to improve its performance[cite: 442]. This is often achieved through:
    - **Dynamic Scheduling:** Continuously checking execution status and improving system performance based on feedback from autonomic elements[cite: 443, 444].
    - **Adaptive Scheduling:** Used for data-intensive applications, allowing easy adaptation to changing environmental conditions[cite: 445, 446].
- **Self-Protecting:** The system's capacity to defend against intrusions and threats to maintain security and integrity[cite: 446, 447]. This involves:
    - **Secure Scheduling Policies:** For both provider and user sides[cite: 448].
    - **Trust Management:** Detecting malicious attackers through behavioral auditing[cite: 449, 450].
    - **Intrusion Detection:** Continuously monitoring and analyzing attacks to prevent future occurrences[cite: 449, 451, 452].

6.6 QoS Parameters (Figure 16 taxonomy) Various QoS parameters are considered in designing autonomic cloud computing systems. The paper identifies and defines eight key types[cite: 453, 454, 455]:

- **Scalability:** Maintaining performance as user numbers or resource usage increases[cite: 455, 456].
- **Availability:** Ensuring data and services are accessible with desired performance, excluding scheduled downtime[cite: 455, 457].
- **Reliability:** Consistently performing according to predefined objectives[cite: 455, 458].
- **Security:** Protecting stored data, often through encryption and access controls[cite: 455, 459].
- **Cost:** Amount spent (e.g., per hour) for workload execution[cite: 455, 462].
- **Execution Time:** Time required for complete workload execution[cite: 455, 461].
- **Energy:** Amount of energy consumed by resources to execute a workload[cite: 455, 460].
- **Resource Utilization:** Ratio of actual resource execution time to total resource uptime for a single resource[cite: 455, 463].
- **SLA Violation:** The likelihood of breaching the Service-Level Agreement[cite: 455, 464].

Section 7: QoS-Aware Autonomic Resource Management Techniques in the Cloud

This section conducts a survey of selected QoS-aware autonomic resource management techniques from the literature[cite: 467]. It maps these techniques (examples include CBR, ASP, COCCUS, CAS, AWM, etc., as listed in Table VII) to the taxonomy developed in Section 6, comparing them based on their characteristics related to application design, workload scheduling, allocation, monitoring, self-management properties, and the QoS parameters they address (detailed in Tables VIII through XIII)[cite: 468, 469, 470]. The aim is to identify how these techniques implement different aspects of autonomic resource management and to find gaps in existing research[cite: 468]. For instance, the Case-Based Reasoning (CBR) technique uses human-based interaction for SLAs, considering resource utilization and scalability, and employs a control loop and knowledge base for resource configuration[cite: 471, 472, 473, 479]. Another example, Application Service Provider (ASP), uses WSDL and HTTP to design proactive/reactive heuristic policies for optimal solutions based on SLAs, focusing on load balancing and VM allocation[cite: 480, 481, 482, 489, 490]. The section essentially provides a comparative analysis by applying the paper's classification framework to concrete examples.

Section 8: QoS-Aware Autonomic Resource Management in the Cloud: A Perspective Model

This section emphasizes that self-management aims to fulfill user QoS requirements by effectively managing workload and resource utilization, often through utility functions[cite: 558, 559, 563]. A key research issue is the current lack of comprehensive autonomic service offerings by most providers, with AWS being an exception that provides integrated autonomic services, albeit with limited customization for end-users[cite: 565, 566, 567, 568]. Figure 17 and 18 illustrate AWS's environment and user roles (administrator, power user, end-user) with varying degrees of customization[cite: 569, 571, 572, 574, 575]. The limited end-user customization can hinder fulfilling specific QoS requirements[cite: 577, 578, 579]. To address this, a more effective QoS-aware autonomic resource management technique is needed[cite: 580]. The authors suggest potential improvements for AWS-like systems, such as GUI-based service requirement specification, higher customization (selecting database, platform, OS, etc.), flexible pay-per-use, enhanced reliability through a global computing infrastructure, improved scalability with tools like auto-scaling, and better security via an E2E approach[cite: 581, 582, 583, 584, 585, 586].

The paper then proposes a **QoS-aware autonomic resource management model (Figure 19)** that operates on global and local levels[cite: 587, 588].

- **Global Level:** Cloud consumers submit workloads with QoS requirements (defined in SLAs). An Autonomic Manager (AM), following the MAPE loop, interacts with the system. It uses Sensors for Resource Discovery (finding adequate resources based on QoS) and a Knowledge Base for rules and policies[cite: 589, 594, 595, 596, 597, 598, 600].
- **Local Level:** Workload execution is divided into subtasks handled by local instances. Each local instance performs:
  - **Resource Monitoring:** Ensures policy conditions are fulfilled and resources are provided[cite: 604].
  - **QoS Monitor:** Verifies if all QoS attributes in the SLA are met. If not, it generates alerts triggering **Adaptation** (maintaining effective execution despite QoS changes)[cite: 606, 607, 608].
  - **Composition:** Determines the best resource/workload pair for execution[cite: 603].
  - **Resource Provisioning:** Provides resources to the workload based on QoS and system policy, with information sent to users for verification[cite: 609].
  - **Resource Scheduling:** The Autonomic Element (AE) allocates resources to workload(s) after user verification[cite: 610].
  - **Executor:** Performs the final resource execution to meet specified deadlines[cite: 611].

This model aims to provide a comprehensive framework for managing cloud resources autonomically while strictly adhering to QoS requirements.