symmetry

MDPI

Article

A Secure, Scalable and Elastic Autonomic Computing Systems Paradigm: Supporting Dynamic Adaptation of Self-* Services from an Autonomic Cloud

Abdul Jaleel *, Shazia Arshad and Muhammad Shoaib

Department of Computer Science and Engineering, University of Engineering & Technology, Lahore 54890, Pakistan; prayers5@gmail.com (S.A.); shoaib@uet.edu.pk (M.S.)

* Correspondence: abduljaleel.rcet@gmail.com

Received: 25 March 2018; Accepted: 23 April 2018; Published: 2 May 2018

Abstract: Autonomic computing embeds self-management features in software systems using external feedback control loops, i.e., autonomic managers. In existing models of autonomic computing, adaptive behaviors are defined at the design time, autonomic managers are statically configured, and the running system has a fixed set of self-* capabilities. An autonomic computing design should accommodate autonomic capability growth by allowing the dynamic configuration of self-* services, but this causes security and integrity issues. A secure, scalable and elastic autonomic computing system (SSE-ACS) paradigm is proposed to address the runtime inclusion of autonomic managers, ensuring secure communication between autonomic managers and managed resources. Applying the SSE-ACS concept, a layered approach for the dynamic adaptation of self-* services is presented with an online 'Autonomic_Cloud' working as the middleware between Autonomic Managers (offering the self-* services) and Autonomic Computing System (requiring the self-* services). A stock trading and forecasting system is used for simulation purposes. The security impact of the SSE-ACS paradigm is verified by testing possible attack cases over the autonomic computing system with single and multiple autonomic managers running on the same and different machines. The common vulnerability scoring system (CVSS metric) shows a decrease in the vulnerability severity score from high (8.8) for existing ACS to low (3.9) for SSE-ACS. Autonomic managers are introduced into the system at runtime from the Autonomic_Cloud to test the scalability and elasticity. With elastic AMs, the system optimizes the Central Processing Unit (CPU) share resulting in an improved execution time for business logic. For computing systems requiring the continuous support of self-management services, the proposed system achieves a significant improvement in security, scalability, elasticity, autonomic efficiency, and issue resolving time, compared to the state-of-the-art approaches.

Keywords: Autonomic computing; scalable computing; elastic computing; self-management process; self-* services; self-* capabilities as a service (S*SAAS); cloud computing

1. Introduction

Autonomic computing is about software self-management. It aims at the provision of self-* capabilities to computing systems to make them behave like the human autonomous nervous system. The idea is to shift the human task of software controlling to policy and rules definition [1–3]. Self-healing, self-configuring, self-protecting, self-optimizing, self-awareness, context-awareness, openness and anticipation are basic self-* characteristics of autonomic computing systems [1–3]. The introduction of self-* capabilities as a service (S*SAAS) for software self-management is motivated by taking advantage of the powerful processing and storage abilities of cloud computing. With the

integration of autonomic software management and cloud computing, multiple applications can be served by a self-management service at the same time.

The application model of the proposed system is as follows:

- A computing system implements a manageability interface to shares its data with external autonomic managers.
- An autonomic manager registers with the autonomic cloud offering software management services.
- The computing system registers its managed resources with the autonomic cloud to buy the management services.

In an autonomic computing system (ACS), the self-behavior and external environment is continuously monitored to adapt to the changing conditions. The most widely used model for autonomic computing is International Business Machines (IBM)'s reference model [1,3–10]. The two main components of the reference model are the autonomic manager (AM) and managed resource (MR). MAPE-K (monitor, analyse, plan, execute and knowledge base) is the control loop through which an AM controls and manages the system resources (i.e., MRs) [11]. MRs provide touchpoints consisting of sensors, effectors and manageability mechanisms to share data and control with AMs [9–12]. The addition of self-* capabilities [1,2,4,5] into a computing system cause the system to behave just like a human autonomous nervous system [3,6,13].

In IBM's autonomic computing reference model, AMs are statically configured with MRs, producing a fixed set of self-* capabilities. The design of an ACS should support a runtime inclusion/exclusion of the autonomic manager into/out of the computing system to provide a dynamic configuration of more autonomic capabilities in the ACS. The ACS needs to be scalable and elastic, which is possible if a provision has been made for the runtime registration of AMs. Moreover, the ability to learn from experience (like the human nervous system) is possible with the addition of new rules to the knowledge base. An expert can also add or edit the rules in the knowledge base. However, the inclusion of new rules may conflict with the access privileges of an autonomic manager. An AM is allowed to access privileged resources only and not to affect the unprivileged areas of the computing systems. The provision of the runtime configuration of AMs and the addition of conflicting rules compromises the security and integrity of an ACS. Therefore, there should be some authentication and validation when an autonomic manager and a managed resource interact. The lack of authentication is exploitable specifically in case of distributed or decentralized autonomic computing systems.

This research improves the autonomic computing design concept by presenting a secure, scalable and elastic autonomic computing system (SSE-ACS) paradigm. Registration, authentication, and validation steps are introduced to the existing ACS design for this purpose. The SSE-ACS paradigm is based on two algorithms, Registration, Authentication and Validation (RAV) and Self-Management Process (SMP). The RAV algorithm is responsible for the registration, authentication and validation of AMs. It uses ciphering to secure the interaction between MR and AM. Through RAV, the SSE-ACS paradigm allows off-the-shelf AMs to become part of the running system without a breach of integrity. The SMP algorithm implements the self-management process following the Monitor, Analyze, Plan, Execute and Knowledgebase (MAPE-K) loop style.

To offer self-* capabilities as a service (S*SAAS), a layered approach is proposed, utilizing the SSE-ACS design concept. An 'Autonomic_Cloud' is introduced to offer dynamic adaptation of self-* services. The cloud works as the middleware between AMs (offering the self-* services) and ACS (requiring the self-* services). The cloud can also offers its own self-* services. The Autonomic_Cloud comprises an AM_Repository, an MR_Repository and a ConnectivityRecord table. The RAV concept of SSE-ACS is applied to register the AMs and MRs of an ACS with the Autonomic_Cloud. When the ACS selects a required self-* service, the Autonomic_Cloud links its AM with the relevant MRs. The Autonomic_Cloud charges the ACS for the service as a revenue paid to AMs. By enabling such an

integration, self-*capability-as-a-service (S*SAAS) can provide a self-management service to multiple applications at the same time, instead of the current model, which serves a dedicated application.

To evaluate the proposed SSE-ACS paradigm and for performance measures of the Autonomic_Cloud, a stock trading and forecasting system has been developed applying the SSE-ACS paradigm. The RAV algorithm controls the autonomic managers (serverFarm and loadBalancingManager, forecastManager, and intruderManager) as per their access privileges and restricts the managers' access to unauthorized parts of the managed resources (trade-server, trade-database). The security impact of the SSE-ACS paradigm is verified by testing possible attack cases over the autonomic computing system with single and multiple autonomic managers running on the same and different machines. The common vulnerability scoring system (CVSS metric) from First.org [14,15] is used to evaluate the severity of the vulnerability of the existing ACS model and SSE-ACS paradigm. CVSS showed a decrease in the vulnerability severity score from high (8.8) for the existing ACS to low (3.9) for SSE-ACS.

To simulate the scalability and elasticity of the SSE-ACS paradigm, autonomic managers, namely serverAppProtectionManager, databaseConnectivityManager and logbaseManager, were introduced into the working system from the Autonomic_Cloud.

- If the serverApp becomes corrupted due to system-file(s) delete/overwrite/corrupt, the serverAppProtectionManager automatically copies the corrupted/deleted file(s) from the backup and keeps the server working. In an ordinary system (with no protection), the server stays down until files are manually uploaded/reinstalled.
- The databaseConnectivityManager keeps the database-server active so that the connection with the trading-server is live.
- The logbaseManager controls the memory space used by the trade log of clients.

It is experimentally demonstrated that dynamically configured self-* services divide the processing load of the ACS, utilizing the computing resources from the network. Hence, the system optimizes the CPU share using elastic AMs, which results in an improved execution time for the business logic. This produces a better response rate compared to the existing approach. By introducing S*SAAS, the autonomic computing paradigm achieves a significant improvement in security, scalability, elasticity, autonomic efficiency, and issue resolving time, compared to the state-of-the-art approaches (IBM [1,4,7,8], Intelligent Machine Design (IMD) [13], the data stream model and Service Oriented Architecture (SOA) [16], and the policy-based model [17]) that implement static self-management services.

In summary, the following contributions are made by this paper.

- A secure, scalable and elastic autonomic computing system (SSE-ACS) paradigm is presented to improve the autonomic computing design concept.
- An efficient service adaptation scheme is designed to offer self-management capabilities as S*SAAS from an autonomic cloud.
- Experimental evaluations are logged for statically-configured local AMs, dynamically-configured AMs from a server machine, and the runtime registering of AMs from the autonomic cloud. The system optimizes the CPU share using elastic AMs, which resulted in the improved execution time of the business logic.
- For software applications requiring the continuous support of self-management services, the proposed system achieves a significant improvement in security, scalability, elasticity, autonomic efficiency, and issue resolving time, compared to the state-of-the-art approaches.

Related work is presented in Section 2. Section 3 is entitled Materials and Methods, describing the SSE-ACS paradigm and related material. Results and discussion are provided in Section 4, where simulations of stock market data are illustrated to test the applicability of the proposed SSE-ACS paradigm. The research is concluded in Section 5.

2. Related Work

Paul Horn [18] at IBM introduced the concept of autonomic computing with a vision to apply the human nervous system principles of self-regulation and separation of concerns to the design of computer systems. The term autonomic element is used for the combination of AM and MR [19]. Each autonomic element is responsible for its own management (internal state, behavior and interaction with others) driven by the goals prescribed by its designer [19]. In the architectural approach to autonomic computing, an autonomic element is responsible for managing its own behavior, based on defined internal and external interfaces, behaviors, policies, relationships and interaction integrity [2].

Self-management capabilities reduce the cost and complexity of managing system resources and save human time spent managing the computing system [20]. IBM developed a toolkit for the development of autonomic computing systems and described example scenarios in its red book [9]. IBM presented an architectural blueprint for autonomic computing with system resources at the bottom layer and manageability interfaces (touchpoints) encapsulating one or more resources [11]. Touchpoints provide sensors and effectors for interfacing. Touchpoints exploit attributes or properties of the computing systems to sense running behavior. Abuseta [21] termed such attributes and properties as 'context attributes'. Sensors and effectors respectively offer the ability to obtain and set the value of context attributes at runtime [22]. The autonomic manager uses sensor interfaces to retrieve information either via the 'request response' or 'send notification' style. The effector interface is used by the autonomic manager to manage the element action. Effectors have a 'perform operation' and 'solicit response' style of working [12].

The initiative towards a fully autonomic IT infrastructure is an evolutionary process, starting from a basic level and continuing to predictive, managed, adaptive and fully autonomic levels [14]. Different architectures [1,3,7,13], models [4,8,16,17] and design patterns [2,21,23] exist for autonomic computing systems. A great deal of research [1,3,7,9,10] supports IBM's reference model for autonomic computing systems. Some research [15–17] presents models different to IBM's concept. Shuaib et al. [15] extended the intelligent machine design (IMD) architecture to define autonomic computing architecture. Reaction, routine and reflection layers are defined to apply autonomic solutions as either hardwired (direct fixed solving), learned (policies are accessed from working memory and applied according to the context), or derived (policies are defined at runtime using learning and partial reasoning). Technical autonomic maturity indices were presented to align the intelligent machine design architecture with IBM's concepts. Nzekwa et al. [16] suggested a model-driven approach for feedback control loops that is based on a data streaming model and service component architecture. The presented component-based architecture handles feedback control loops inside the architecture instead of implementing it as an external loop. Bazerra et al. [17] presented a policy-based model with autonomic capabilities to manage the quality of the service in computer networks. The autonomic policy-based management model is divided into the information plane (receives data related to the network status and converts it into symptoms using the XML format and considering Service Level Agreement (SLA) conformance, decision plane (finds a solution from the knowledge base that meets the specified SLA), and the execution plane (generates an executable policy through the policy compiler and activator). The research gist and differences of the existing approaches to autonomic computing are summarized in Table 1.

Some research works [4,8] present conceptual models that match IBM's concept. Peer2peer, aggregator–escalator peer, and chain of configurator and configuration manager are different architectural approaches to adaptive computing [4]. These architectural approaches are characterized based on the relationship of sensors to monitors and executors to effectors. Design patterns, behaviors, policies, relationships and interfaces are key aspects when engineering self-adaptive and autonomic systems [2,21,23]. Chen et al. integrated the Internet-of-things and software-as-a-service by presenting a case-study-based model of logistic systems [24]. Kim et al. designed DIAAS (infrastructure as a service with desktop) for the fast distribution of work utilizing distributed computing and storage services [25].

Table 1. Research gist and differences of existing approaches to autonomic computing.

Author-Year	Autonomic Perspective	Control Loop	No of Layers	Building Blocks	Generic	Technique Used	Policy Format	Policy Storage
Mittal et al. 2014 [7]	IBM	Explicit	4	Autonomic Manager + Resources	Yes	Not Defined	Available	Knowledge Base
Ahuja & Dangey 2014 [1]	IBM	Explicit	4	Autonomic Manager + Resources	Yes	Policy Based	Not Available	Knowledge Base
Kumar & Naik 2014 [8]	IBM	Explicit	4	Autonomic Manager + Resources	Not Defined	Rule-Based	Algorithm	Implicit
Shuaib et al. 2011 [13]	IMD	Explicit	3	Layers	Yes	Rule-Based	Rules	MIB
Nzekwa et al. 2010 [16]	Data stream model & SOA	Implicit	1	Agents	No	Data-Oriented Model	Implicit	Implicit
Bazerra et al. 2009 [17]	PB Model	Explicit	3	Modules	Not Defined	Policy-Based	XML-Based	Autonomic Database
Okon & Asagba 2014 [4]	IBM	Explicit	1	Modules	Yes	Policy/Rule	Not Defined	Not Defined

A distributed computing environment requires continuous runtime monitoring to ensure the functioning and security of the system [26]. Authentication, access rights, global credentials and authorization are main security issues in a distributed environment [27–29]. Scalable key management using Public Key Infrastructure (PKI) X.509 ensures Confidentiality, Integrity, Availability (CIA), Authentication and Authorization [30]. Bing Y. et al. presented a scalable and robust approach for the adaptive migration of Virtual Machines (VMs) to cost-effective servers [31]. Masood R. et al. focus on authorization issues in the cloud environment, presenting access control as a service, to restrict the access of confidential data and resources to unauthorized users [32]. Shi and He [33] identified phases (initialization, registration, login-authentication and change password) of security compliance and analyzed an authentication scheme for mutual authentication, anonymity, offline password guessing attacks, privileged insider attacks, stolen-verifier attacks, man-in-the-middle attacks, replay attacks and impersonation attacks.

Lounis et al. presented an innovative architecture to manage a huge amount of highly sensitive data generated by medical sensor networks covering scalability, availability and security issues [34]. Bonanno et al. proposed a cloud-based distributed toolbox to optimally manage the energy dispatch from renewable resources utilizing Graphics Processing Unit (GPU) and Wavelet Recurrent Neural Networks (WRNN) predictors [35]. Yang et al. implemented cloud-based energy serving a multi-agent system using web service techniques with backend information agents involved [36]. Napoli et al. developed a neural-network-driven forecasting setup to manage the power production and dispatch systems of smart grids applying cloud computing [37].

3. Materials and Methods

This section is divided into two parts. Firstly, a modified architectural design for secure, scalable and elastic ACSs is presented in Section 3.1 with the following aspects.

- A layer for initial registration, authentication and validation is added on the resource side.
- A connection controller is defined inside the autonomic manager.
- An algorithm is provided for initial registration, continuous authentication and validation.
- An algorithm is provided for the elastic self-management process.

Secondly, a layered approach for the dynamic adaptation of self-* services is proposed in Section 3.2. The proposed approach utilizes the SSE-ACS design concept and introduces self-management capabilities as a service, i.e., S*SAAS.

3.1. Modified Design for Secure, Scalable and Elastic ACSs

The modified design of the SSE-ACS paradigm consists of two main modules, as shown in Figure 1. The first is the 'system-under-observation', requiring self-* behavior. The system-under-observation consists of the managed resources and touchpoints for each managed resource containing sensors and effectors. A layer for registration, authentication and validation is incorporated above the resource touchpoints to secure their access, labeled as RAV in Figure 1.

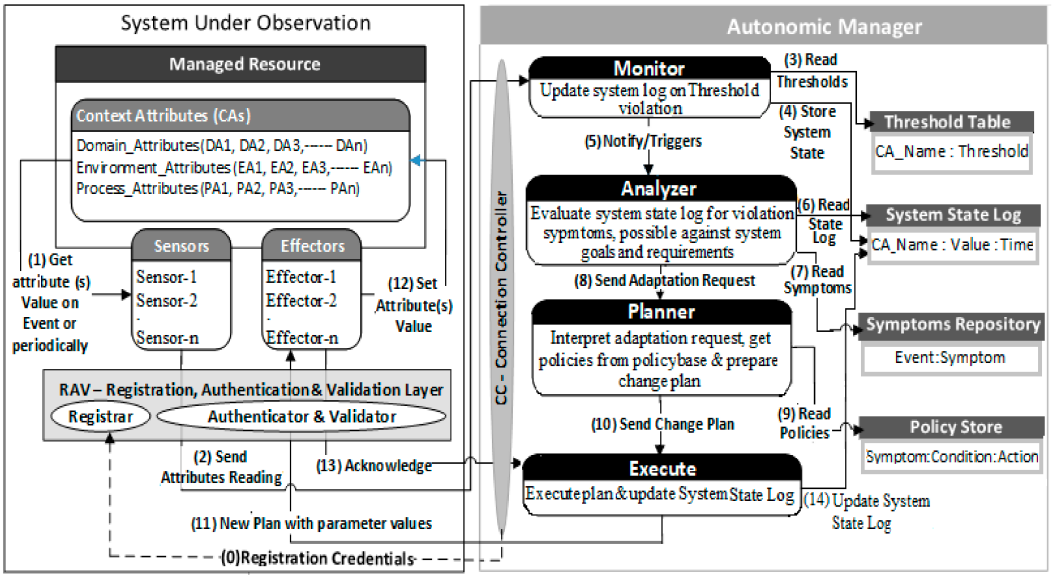

Figure 1. Modified design for SSE-ACSs.

The second module is the 'autonomic manager', which consists of monitor, analyze, plan, execute, and knowledge-base entities (the working of each entity is specified in Figure 1). The autonomic manager is responsible for the management of the ACS. The autonomic manager is equipped with a connection controller, labeled as Connection Controller (CC) in Figure 1, for connecting with the managed resources.

The procedure of the modified design consists of the initial registration, continuous authentication and validation, and a process for self-management. For initial registration, CC inside the autonomic manager generates a subscribing request with credentials for the RAV layer (inside the resource touchpoint). The RAV validates the request and provokes the autonomic manager, thus linking the resource's sensor(s) and effector(s) to the corresponding monitoring and execution of AM. Ciphered data/information flows between RAV and CC. The process is outlined in Figure 2.

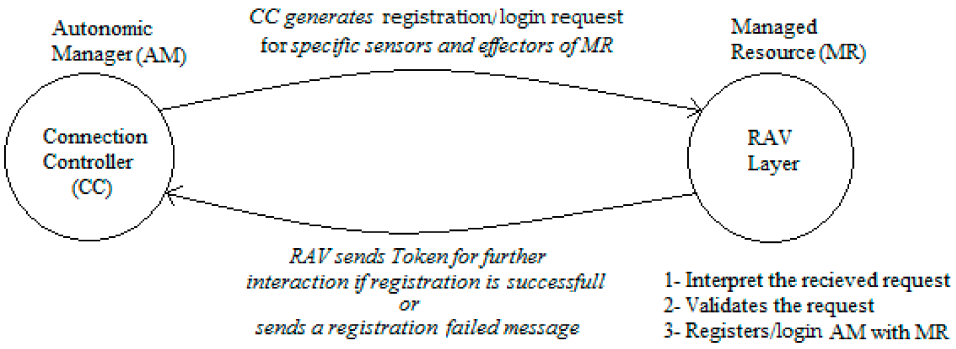

Figure 2. Process of initial registration/login.

After successful registration, the autonomic manager gets control over the resource. On the detection of an anomaly or perturbation, the SMP-algorithm gets triggered to retain stability. A flow graph explaining an SMP process is provided as Figure 3.

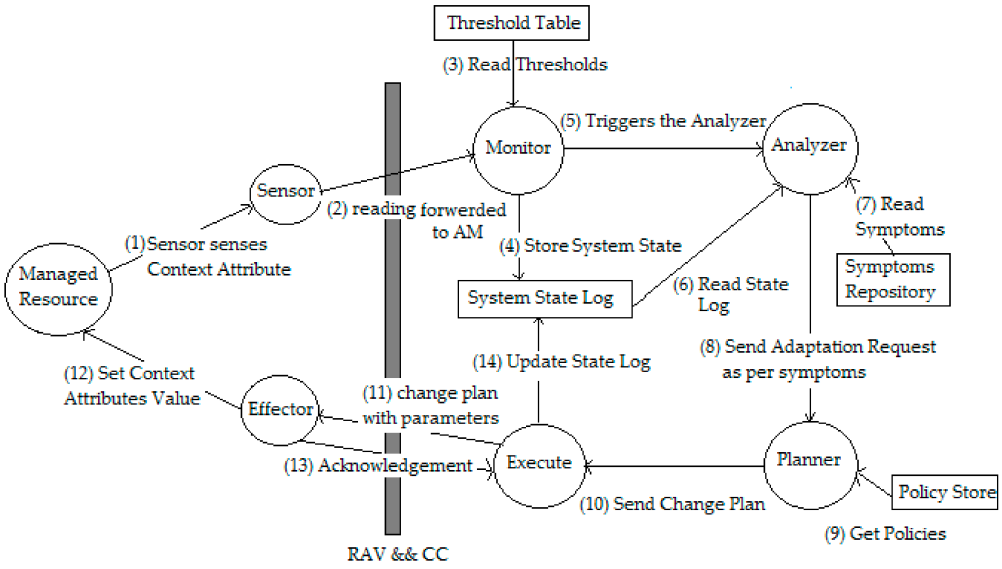

Figure 3. Flow graph of a self-management process (SMP).

Algorithms to elaborate these steps are provided in the following sections:

(a) RAV-Algorithm for Registration, Authentication, and Validation

The RAV-algorithm is used to perform registration, authentication, and validation, which secures the communication between AM and MR. Step(0) in Figure 1 represents the RAV-based initial setup. An AM sends a request to the resource's RAV to subscribe the AM's monitoring and execution with specific sensors and effectors of the resource. The registrar validates the credentials and adds a relationship to the authentication table. The registrar is responsible for registering the AM with an autonomic computing system by entering the AM_Id in the authentication table. It stores the sensorId–monitorId and executorId–effectorId relationships in the authorization table. The authenticator (inside the RAV layer) uses stored credentials to authenticate each incoming request from the AMs. The validator validates the request so that the permitted autonomic manager can access a managed resource. A secure, persistent connection is established on successful validation, and is used for future communications until the connection is closed by either party.

The RAV algorithm is provided below as Table 2.

Table 2. Registration, Authentication, Validation (RAV) algorithm.

Input: A registration/connection/interaction request.
Output: A secure connection for communication between the AM and MR.
Start
1. *Receive AM request at the MR touchpoint*
2. *Identify request type*
    a) If (registration based request)
        Get registration credentials *//Registrar is activated*
        If (credential correct)
            Enter AM_Id in the AuthenticationTable
            Store SensorId-MonitorId and ExecutorId-EffectorId relationships in AuthorizationTable
            Allot login credentials
            Forward request with login credentials to (b)
        Else
            Display Error 'Invalid Registration Credentials'
            End
    b) If (Login-credential-based request)
        Get login credentials *//Authenticator in action*
        If (Authentic credentials)
            Generate token for further communication
            Forward as token-based request to (c)
        Else
            Display Error 'Invalid Login Credentials'
            End
    c) If (Token-based request)
        Get token *//Validator is triggered*
        If (valid token)
            Grant access to the touchpoint
            End
        Else
            Display Error 'Invalid Token'
            End
End

A pictorial representation of the RAV algorithm is provided in Figure 4, to explain the flow of controlled access to resource touchpoints by autonomic managers. When AM sends a registration request to a resource's RAV to subscribe the AM monitors and executors with specific resource sensors and effectors, the RAV interprets the request type to be either a registration, login or a token-based interaction. If it is a registration request, the registrar verifies the registration credentials (involving Public Key Infrastructure (PKI) [38]) and allots login credentials to the AM. The registrar enters AM_Id in the authentication table. The sensorId–monitorId and executorId–effectorId relationships are stored in the authorization table. The request is converted into a login request.

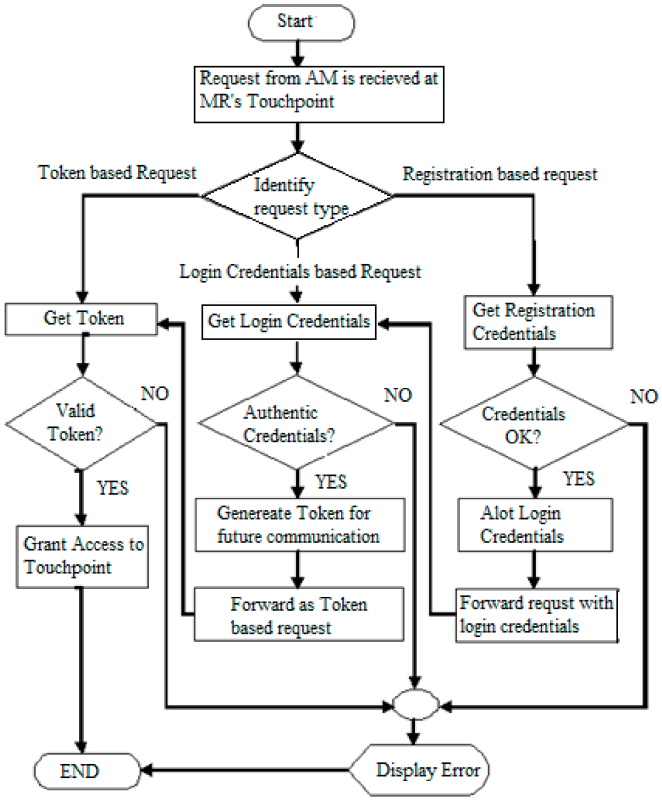

Figure 4. Flow diagram of the controlled access to resources touchpoint (RAV algorithm).

If a login/connect request is received, the authenticator (inside the RAV) uses the credentials stored in the authentication table to authenticate the request from the AM. The authenticator assigns a token to the AM for further communication. The request is converted to a token-based request.

If a token-based interaction request is received, the validator validates the token so that the allowed AM can access a managed resource. A secure, persistent connection is established on successful validation. SMP starts using the established connection. The connection is closed on completion of the SMP process or when the AM is unregistered.

To secure the data over a communication channel, we used Rivest, Shamir and Adleman (RSA) encryption/decryption-based ciphering of data. PKI [38] and scalable key management with rekeying [39] are used for distributing and managing authentication and encryption credentials.

(b) SMP-Algorithm for the Self-Management Process

The SMP algorithm provides a procedure for the self-management process in connection with the RAV-algorithm. The steps of the SMP algorithm are provided below, as Table 3.

Table 3. Self-Management Process (SMP) algorithm.

Input: The value of context attributes from sensors.
Output: System state is readjusted accordingly.
Start
1 - Sensor gets context attribute value on event or periodically
- Sensor sends context attribute's reading to monitor
2 - Monitor reads thresholds for context attributes from threshold table
If (Threshold violation)
Log the violation in system state log
Notify the violation to Analyser
Else
Log the state change in system state log
End
3 - Analyser reads violations from system state log
- Analyser determines the cause of violation by consulting symptoms repository.
If (cause of violation determined)
Send adaptation request to Planner
Else
Display/Log 'violation occurred' on screen/logFile
End
4 - Planner interprets adaptation request
If (policies exist for adaptation request)
Prepare change plan from policies
Send List of actions to Executor
Else
Display/Log adaptation request with a message 'contact system expert'
End
5 - Repeat until all change plans executed by Executer
- Send a change plan with parameter values to concerned Effectors
- Effectors set attributes values in CAs
- Effector send acknowledgement to Executor
End
6 - Executor updates the system state log
End

3.2. Layered Approach for Dynamic Adaptation of Self- Services: Offering Self-* Capabilities as a Service*

In existing models of autonomic computing, the self-* characteristics are defined at the time of design, AMs are statically configured, and the running system has an absolute set of self-management capabilities. To facilitate the dynamic adaptation of self-* services, a layered approach consisting of three layers is proposed (given in Figure 5). First layer is of AMs, which implements: (1) the CC part of the RAV algorithm; and (2) the SMP algorithm. The second layer is an online Autonomic_Cloud that works as a middleware between AMs and MRs. AMs register with the Autonomic_Cloud to offer dynamically adaptable self-* services.

The third layer comprises the MRs of ACSs requiring self-management services. An MR implements the RAV algorithm and shares internal data in the form of CAs to be used by AMs for surveillance and management purposes. The ACS registers itself and its MR(s) with the MR_Repository inside the Autonomic_Cloud to obtain self-management services from live AMs.

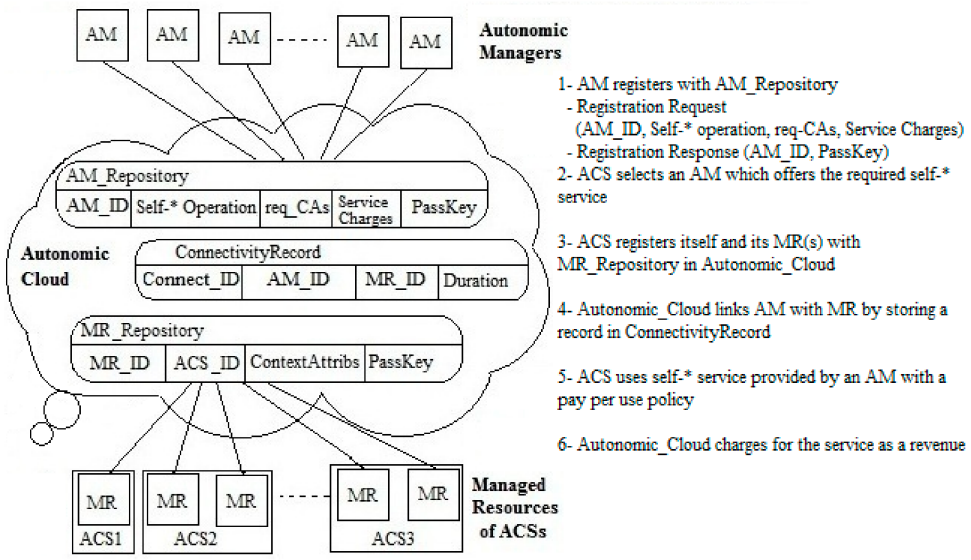

Figure 5. Autonomic cloud offering self-* capabilities as a service to ACSs (the S*SAAS).

The process for the dynamic adaptation of self-* services is provided in Table 4.

Table 4. Process for the dynamic adaptation of self-* services.

(1) AM registers itself with the AM_Repository in the Autonomic_Cloud by sending a registration request.
(2) ACS asks for available self-management services from the Autonomic_Cloud and selects a required service.
(3) ACS registers itself and its MR(s) with the MR_Repository in the Autonomic_Cloud by a sending registration request.
(4) Autonomic_Cloud links the AM with the MR of the ACS by storing a record in the ConnectvityRecord table.
(5) ACS uses the self-* service provided by the AM with a pay-per-use policy.
(6) Autonomic_Cloud keeps track of the services provided by AM and used by the ACS to facilitate its pay-per-use policy.

The RAV algorithm, given in Section 3.1(a), addresses the registration of autonomic managers with ACS resources. The RAV also ensures secure communication between the AM and MR through authentication and validation. The SMP algorithm (provided in Section 3.1(b)), implements the self-management process. The Autonomic_Cloud charges the ACS according to a pay-per-use policy, as a revenue paid to the AMs.

4. Results and Discussion

To evaluate the proposed SSE-ACS paradigm and the performance measures of the Autonomic_Cloud, a stock trading and forecasting system was implemented. It facilitates buyers, brokers and shareholders in bargaining and trading via an e-stock exchange.

4.1. Experimental Setup

The Scripting Languages, Hypertext Preprocessor (PHP ver 5.4) and JAVASCRIPT, were used for the development of the stock trading and forecasting system, which runs over a serverFarm. Structured Query Language (MYSQL ver 5.6) was used for the management of the stock trading database. VMware ESXi (4.1.0, VMWare Inc., Palo Alto, CA, USA) [40] was used for the serverFarm set-up in a virtualized environment. A Dell PowerEdge (1950 MKII, Dell Inc., Round Rock, TX, USA) was used to host the hypervisors. Five server virtual machines (VMs) (1 vCPU, 2GB RAM, 20GB SCSI datastore, and Windows Server 2008 R2 Standard) were created with an idle state. VMware vSphere Client (4.1.0 VMWare Inc., Palo Alto, CA, USA) was installed on the VM for performance measures

with the stock trading system on board. The experimental deployment is provided below as Figure 6, describing the interaction of the relevant hardware (hypervisors manager and 100 clients) and software (RAV algorithm, SMP algorithm, and Psutil-based monitoring component [41]).

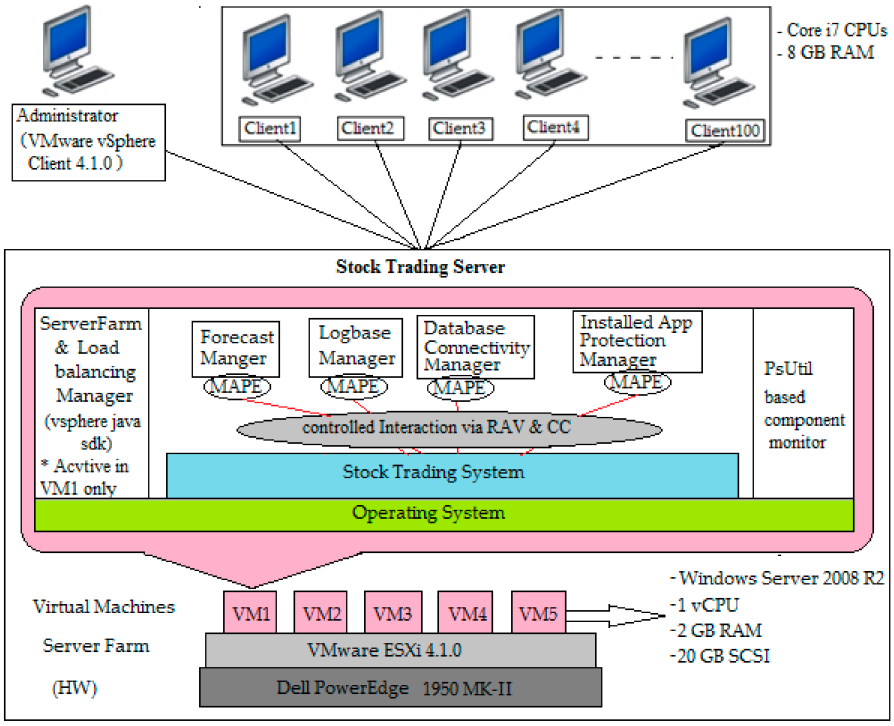

Figure 6. Experimental deployment to demonstrate self-* capabilities.

Experimental evaluations of the self-* capabilities were demonstrated by following the following sequence:

Initial State: VM1 and VM2 are ON but the client's requests are processed by VM1 only. The ServerFarm&Load Manager runs on VM1 to activate/deactivate the VMs according to the increase/decrease of server load as a result of an increase/decrease in client requests.
Activated Self-* services: The self-* services provided below were executed for the stock trading system.

- **Stock trade forecasting service:** Australian stock market information [42] is retrieved for leading companies and the forecasting service is executed using this data to predict stock trends.
- **Server-farm management and server-load balancing service:** Five VMs are created to handle the client's requests. The ServerFarm&Load manager activates/deactivates VMs with the increase/decrease of the processing load.
- **Installed application protection service:** Server program files are manually deleted while the server is active. This service recovers the deleted files from the backup to keep the system running.
- **Database connectivity control service:** The Database (DB)-service is manually reduced to cause an interruption. This service restarts the DB-service immediately to keep the DB live.
- **Log-base memory management service:** Continuous client requests cause the log memory to fill. This service deletes old files to clear the required space.

These self-* services implement SMP-algorithms to incorporate self-management capabilities inside the trading system.

Client Setup: A setup of 100 computers (corei7 CPU, 8 GB RAM) are prepared, which connect as client machines to the trading server.

Increase/Decrease of Server Load: The clients are connected in a random sequence to increase and decrease the server processing load.

- The load balancer checks the load for VM1. If it exceeds the set limit, it shifts the extra load to VM2 and boots VM3 to idle state.
- If the load in VM2 is too high, it shifts the extra load to VM3 and boots VM4, and so on.
- On the decrease of the processing load, the load balancer turn off idle VMs, keeping at least two VMs always active.

Performance Monitoring: The Psutil (version 5.4.2) library of Python [41] was utilized to develop a monitoring program to measure the use of the CPU, memory and disk for the running of processes and system utilization.

Securing the interaction between AMs and the Trading System: The RAV algorithm was applied to maintain secure the interaction between the AMs offering self-* services and the trading system. The security impact of the SSE-ACS paradigm was verified by testing possible attack cases over the autonomic computing system with single and multiple autonomic managers running on the same and different machines.

Comparison with State-of-the-Art Approaches: Experimental results are logged for statically configured local AMs, dynamically configured AMs from a server machine, and runtime registering of AMs from the Autonomic_Cloud. To increase the requirement of self-management services, the system achieves a significant improvement in the scalability, elasticity, autonomic efficiency, and issue resolving time, compared to state-of-the-art approaches like IBM [1,4,7,8], IMD [13], the data stream model and SOA [16], and the policy-base model [17]. These approaches implement static self-management services, whereas the proposed approach brings dynamism. The system optimizes CPU sharing using elastic AMs.

4.2. Experimental Results

A set of AMs are implemented to provide self-* services to the stock trading system. The experimental process and relevant results for AMs offering self-* services are provided below.

4.2.1. Autonomic Managers Delivering Self-* Services

The detailed working and relevant results of each autonomic manager are provided in sections (a) through (e).

(a) Stock Trade Forecast Manager

The forecastManager predicts the future value of a company's stock and financial trades. It helps users in deciding whether to sell or buy the shares. The context attributes chosen to obtain stock market data for prediction generation are provided in Table 5.

Table 5. Context attributes (CAs) for stock trade prediction.

Given period's close-ups and close-downs
Most recent closing price
Lowest and highest of previous 14 trading sessions (L14, H14)
High, low and closing stock prices
Volume traded

The technical analysis method [43] is used to estimate and predict the next day's trade. The data received for CAs is used by sensors to calculate the values of technical indicators. The technical indicators selected to be used by the forecastManager for trade prediction are provided in Table 6.

Table 6. Technical indicators for stock trade prediction [44].

Relative Strength Index (RSI)
Stochastic Oscillator (SO)
William's % R
Money Flow Index (MFI)
Moving Average Convergence Divergence (MACD)

Sensors send the values of technical indicators to the forecastManager. The thresholds for the technical indicators, symptoms possible and relevant decisions were defined as the rule base of the forecastManager, shown in Table 7.

Table 7. Rule-base of the forecasting manager.

Condition	Symptom	Action
RSI < 30	Indicates stock oversold, showing buy signal	Buy shares
SO-%D < 20	Indicates oversold, price will increase in near future	
MFI < 20	Shares oversold	
%R < −80	Buy signal	
MACD above the signal line	Indicates buy signal	
RSI > 70	Indicates stock overbought, showing sell signal	Sell shares
SO-%D > 80	Indicates overbought, price will decrease in near future	
%R > −20	Sell signal	
MFI > 80	Shares overbought	
MACD below the signal line	Indicates sell signal	

Australian stock market information [42] was retrieved for leading companies to validate the accuracy of the prediction forecaster. The Yahoo finance API [45] and Yahoo Query Language (YQL) [46] was used to retrieve the required data. Figure 7 shows a comparative relation of actual trade volume and technical indicator prediction results for Yahoo for a period of 30 days starting from 01/19/17. The continuous line represents the percentage change in actual trade volume. Grey bars represent the number of technical indicators generating a buy signal. Black bars show the number of technical indicators generating a sell signal. The prediction results are based on a larger number of indicators predicting an outcome. The comparative relation shows that the forecastManager mostly made accurate predictions.

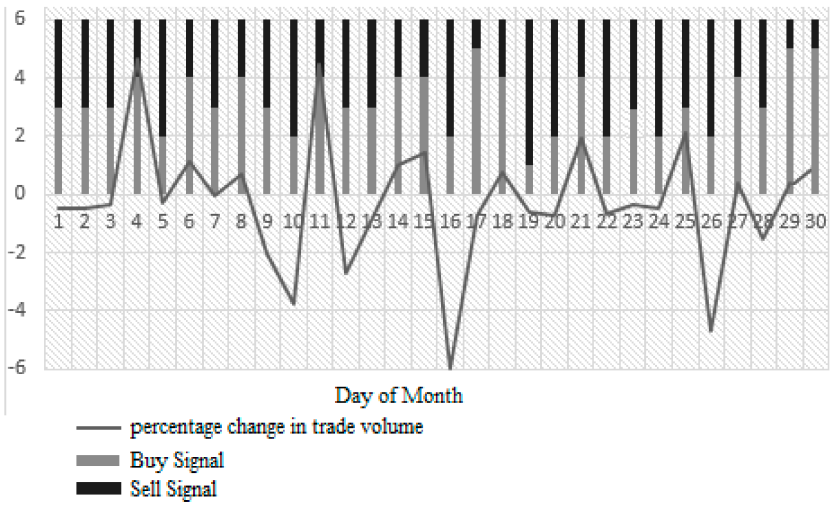

Figure 7. Actual stock trading vs. technical indicator prediction.

(b) serverFarm&loadBalancing_Manager

The serverFarm&loadBalancingManager manages the serverFarm and controls the processing load for sever instances to effectively respond to client requests. CPU usage, number of queued processes and the turnaround time for client requests were taken as context attributes. Server load was measured base on CPU usage and the number of queued process waiting for the CPU. The results of the simulation are demonstrated in Figure 8.

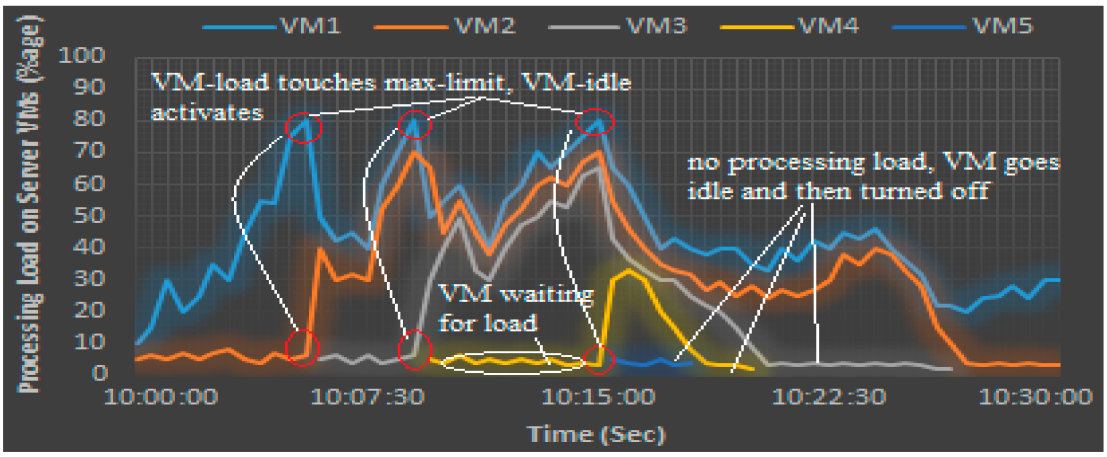

Figure 8. Experimental results (Server Virtual Machines (serverVMs) management against increase/decrease of processing load).

The serverFarm&loadBalancing_Manager transfers client requests to serverVMs balancing the processing load for the maximum utilization of server-VMs. The manager adds or removes a serverVM into/from a serverFarm whenever the cumulative processing load increases above 80% or decreases below 10%, respectively. Whenever the latency of a client request approaches 10 s, a new serverVM is added. A serverVM is shut down when the latency decreases to 1 s and the load on any serverVM is less than 80%. The relevant rules are defined in the policy base; the upper and lower limits are defined for active servers in the threshold table.

(c) Server Application Protection Manager (SAPM)

When a file is deleted/corrupted, the server application process crashes. In an ordinary system, the server stays down until the file is uploaded/reinstalled. The SAPM automatically copies the corrupted/deleted files from the backup and the system starts working again in a much shorter time. The SAPM compares the application file's count and hash value with the application backup. If a difference is found, the files are updated from the backup. This keeps the server application secure from corruption and recovers the application process if it crashes.

The dotted line in Figure 9 shows the percent of CPU used by the trading server. It reaches zero if the server process crashes when a file is deleted/corrupted. The continuous line in Figure 9 shows the percent of CPU usage by the trading server when the SAPM heals the affected file and the trading server process starts working within a shorter time.

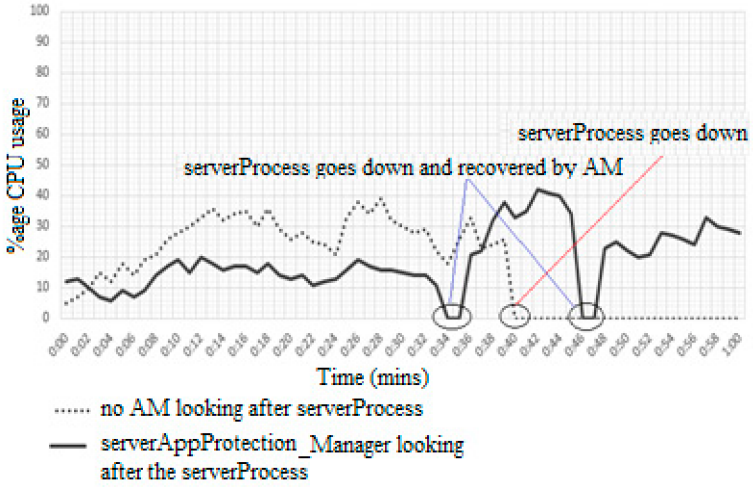

Figure 9. Simulation results for the ServerAppProtection_Manager.

(d) Database Connectivity Manager (DCM)

The trading server continuously uses data from the DB_server. If the DB_server fails, the trading server is halted. The DCM is designed to keep the database server connected to the trading server. On database disconnection, the trading server informs the DCM. The DCM asks the application-server to delay its database usage and wait until the DB-server responds. DCM sends a restart command to the DB_server. Database restart is communicated to the application server. The application server resumes its connection with the database.

The dotted line in Figure 10 shows the number of requests/second processed by the database server. It hits zero when the DB_server goes down until the server process is manually restarted, which takes a significant amount of time. The continuous line in Figure 10 shows the number of requests/second processed by the database server with the DCM restarting the DB_server in a shorter time, with the server process suffering less delay.

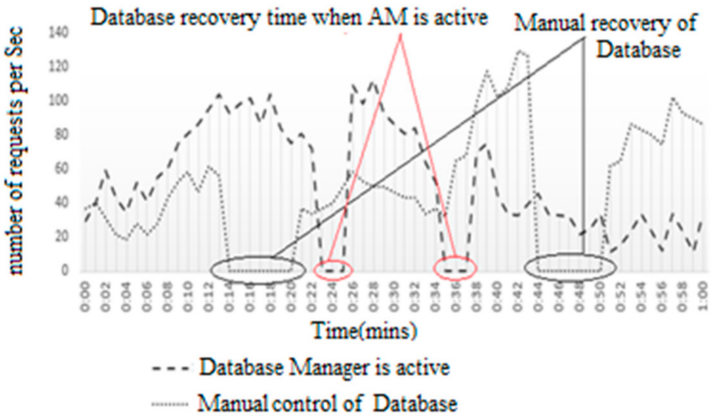

Figure 10. Simulation results for the database connectivity manager.

(e) LogBase_Manager (LBM)

LBM controls the memory space used by the trading-log of clients. For simulation purposes, we set the available log memory size to 1000 KB, and the client's logfile size was set to maximum 10 KB. Client log files are created, updated and deleted as they signup, login, logout and leave. If the size of a logfile reaches 10 KB, the oldest contents are deleted to store information on the new

transactions. The LBM keeps the client log files sorted by date modified. When the total available memory is consumed, the LBM deletes the old log files. The graph in Figure 11 shows the results of the simulation. Whenever the log memory size reaches 1000 KB limit, a minimum number of oldest logfiles is deleted to fulfill the need at hand.

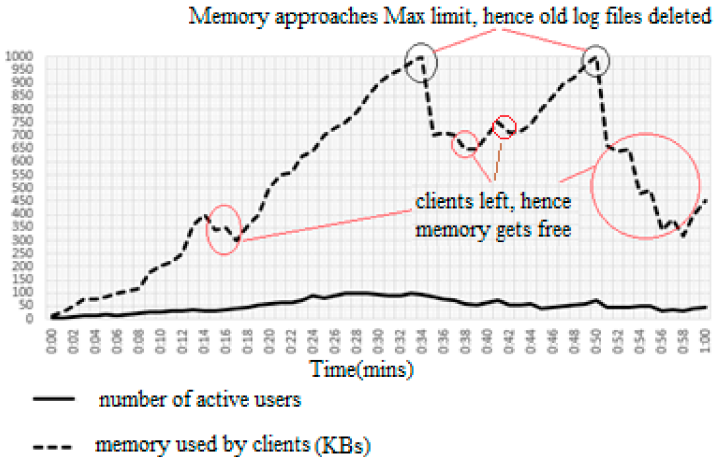

Figure 11. Simulation results for the logbase manager.

4.2.2. The Security Impact of the SSE-ACS Paradigm

In existing autonomic computing architectures, autonomic managers—namely serverFarm&loadBalancing_Manager, forecastManager and intruderManager—can interfere in the vicinity of each other. A security issue arises in the following cases:

Vulnerability-i: The serverFarm&loadBalancingManager accesses information related to the forecastManager, which is an access privilege concern.

Vulnerability-ii: The forecastManager accesses the serverFarm and turns a server ON/OFF in the serverFarm. It transfers the server's load without having privileges.

Vulnerability-iii: An intruder autonomic manager uses the sensors/effectors of the forecastManager to manipulate stock trade data.

We tested the IBM-ACS-based implementation of the stock trading and forecasting system for possible attacks with the following implementation fashions.

Implementation-1: A single AM controls the server and performs the tasks of serverFarm&loadBalancingManager and forecastManager. AM and MR run on the same machine.

Implementation-2: The AM of case1 runs on another machine and controls the server.

Implementation-3: The serverFarm&loadBalancingManager and forecastingManager are implemented as separate managers. The AMs and MRs run on the same machine.

Implementation-4: The AMs of case 3 and MR (server) run on separate machines.

The results from these implementation fashions are compared with the results of the SSE-ACS-based implementations. The CIA triad-based security parameters, possible attack cases, vulnerabilities involved and their effect on the distinctive styles of implementation are compared in Table 8. The SSE-ACS paradigm avoids the attacks by validating the access privileges defined in the ACS authentication and validation tables.

Table 8. Evaluation of the security impact of the SSE-ACS paradigm.

Security Parameters	Attack Cases	Vulnerability Involved	ACS with Single AM (AM and MR Running on Same Machine)	ACS with Single AM (AM and MR Running on Different Machines)	ACS with Multiple AM (AM and MR Running on Same Machine)	ACS with Multiple AM (AM and MR Running on Different Machines)	SSE-ACS Paradigm Based on the ACS System
Access Control (Authorization, Authentication)	Wrongly pretending to be someone else	*Vulnerability-i*	X	⦸	X	⦸	☑
Confidentiality	Unauthorized Access	*Vulnerability-i* *Vulnerability-ii*	XX	XX	⦸ ⦸	⦸ ⦸	☑☑
	Eavesdropping	*Vulnerability-iii*	X	⦸	X	⦸	☑
	Snooping	*Vulnerability-iii*	X	⦸	X	⦸	☑
Integrity	Denial of Receipt	*Vulnerability-i* *Vulnerability-ii*	XX	XX	⦸ ⦸	⦸ ⦸	✓
	Repudiation	*Vulnerability-i* *Vulnerability-ii*	XX	XX	⦸ ⦸	-, ⦸ ⦸ ⦸	✓
	Spoofing	*Vulnerability-iii*	X	⦸	X	⦸	☑
	Modification	*Vulnerability-iii*	X	⦸	X	⦸	☑☑
	Unauthorized Access	*Vulnerability-i* *Vulnerability-ii*	XX	XX	⦸ ⦸	⦸ ⦸	☑☑☑
	CPU overload	*Vulnerability-iii*	XX	XX	⦸ ⦸	⦸ ⦸ ⦸	☑
	Communication Errors	*Vulnerability-iii*	X	⦸	⦸ ⦸	⦸	☑
Availability	Denial of Receipt	*Vulnerability-iii*	XX	-	X	-	✓ ☑☑☑
	Delay	*Vulnerability-iii*	X	⦸	X	⦸	
	Repudiation	*Vulnerability-i* *Vulnerability-ii*	XX	XX	- ⦸	-, ⦸ ⦸ ⦸	

Note: Symbol X denotes, "Attack not applicable, as not exposed to network"; Symbol XX denotes, "Attack not applicable, as single manager handling ServerFarm, ServerLoad and Forecasting Process"; Symbol ⦸ denotes, "No authentication, authorization & encryption employed, hence attack succeeds"; Symbol ⦸ ⦸ denotes, "Managers can interfere in the vicinity of each other"; Symbol - denotes, "No proof of delivery"; Symbol ☑ denotes, "Authentication, authorization & encryption via RAV solves the issue"; Symbol ☑☑ denotes, "Resolved by maintaining ProcessID, SocketNo, PassKey and SessionID"; Symbol ✓ denotes, "Issue solved via proof of origin/delivery".

The common vulnerability scoring system (CVSS metric) from First.org [14,15] was used to evaluate the severity of the vulnerability of IBM's ACS model and the SSE-ACS paradigm. CVSS captures principal characteristics of a vulnerability and produces a numerical score to reflect the severity. The results of the test applied to IBM's model of ACS and the SSE-ACS paradigm are shown in Table 9. The CVSS shows a decrease in the vulnerability severity score from high (8.8) for the existing model of ACS to low (3.9) for the SSE-ACS based system.

Table 9. CVSS metric evaluation (IBM's ACS model vs. SSE-ACS paradigm).

Parameter	Without Security (IBM-ACS)	With Security (SSE-ACS)	Comments
Attack Vector	Local	Local	The attacked point is a resource touchpoint and the vulnerability is exploited by autonomic managers. In the case of an existing model, the unauthorized AM succeeds. In SSE-ACS, the attack fails.
Attack Complexity	Low	High	With no authentication, specialized access condition does not exist. The attacker obtains repeatable success against the vulnerable component. With an authentication layer, the attack is unsuccessful.
Privileges Required	Low	High	In the existing model, an attacker presents itself as a privileged component and exploits the system. Authentication layer and privilege maintenance in SSE-ACS makes the attack unsuccessful.
User Interaction	None	None	The vulnerability is exploited by an AM. User interaction is not required for the attack.
Scope	Changed	Changed	The vulnerability exploitation is initiated by an unauthorized autonomic manager. The under-attack component is a touchpoint and the affected component is the resource. In the existing model, the attack succeeds, whereas the SSE-ACS prevents the attack.
Confidentiality Impact	High	Low	In the existing model, access to restricted information is obtained from the attacked resource. The disclosed information may present a serious impact. In SSE-ACS, authentication restricts the attack and hence the confidentiality impact is low.
Integrity Impact	High	None	In the existing model, the absence of authentication causes the loss of confidentiality, as all resources under the touchpoint are disclosed to the attacker. In SSE-ACS, the authentication layer controls unauthorized access and hence the integrity impact is cleared.
Availability Impact	High	Low	In the existing model, the attacker AM denies access to resources under the impacted touchpoint. The RAV controls access of a resource and hence the availability impact is low in SSE-ACS
Attack vector without authentication: CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:C /C:H/I:H/A:H CVSS Score:8.8			Attack vector with authentication: CVSS:3.0/AV:L/AC:H/PR:H/UI:N/S:C/C:L/I:N/A:L CVSS Score:3.9

4.2.3. Scalability and Elasticity Impact of SSE-ACS Paradigm

To simulate the scalability and elasticity of the SSE-ACS paradigm, the first experiment was carried out with five statically-configured AMs (the working of each AM is discussed above) running on the same machine. The CPU usage of the AM processes was logged within time slices of 5 msec for 1 h. The process was repeated five times at different times of the day. The experimental evaluation found 33.47% average CPU usage and 4820 KBs of memory usage by the AMs, as shown in Table 10, whereas the average CPU usage of the stock trading system was logged as 30.16%.

Table 10. Percentage CPU and memory usage by AMs.

Process Name	No. of Times Process Was Activated in One Hour	Process Action Time (AVG)	AVG CPU %Age Usage	AVG Memory Usage (KBs)
serverFarm & loadBalancingManager	8	5 s	4.42	916
Forecast Manager	5	2 s	1.31	773
serverApp Protection Manager	2	90 s	8.55	596
Database Connectivity Manager	2	110 s	14.65	926
Logbase Manager	12	20 s	4.54	1609
Total			33.47	4820

The second experiment was carried out with two number statically-configured AMs (serverFarm&loadBalancingManager, Forecast Manager) and three AMs (serverAppProtectionManager, DatabaseConnectivityManager, LogbaseManager) dynamically registered in the Autonomic_Cloud. The CPU usage of the autonomic stock trading system was again logged with time slices of 5 msec for 1 h. The experiment was repeated five times. The average CPU usage of 6.47% for the two static AMs and 30.16% for the stock trading system was logged on the local machine. The average CPU usage of 29.56% was measured on the network machine where the Autonomic_Cloud with three dynamically-configurable AMs were running. The results show that the processing load of the autonomic computing was distributed in the Autonomic_Cloud. Hence, the system optimized the CPU share with elastic AMs and an improved execution time was achieved.

4.2.4. S*SAAS Performance Measures

For comparative performance evaluation of the autonomic service styles, we defined the average response quality (ARQ) metrics, as in the following equation:

$$ARQ = \frac{\sum_{i=1}^{ServiceCount} (ServiceConfigurationTime + \sum_{j=0}^{serviceRequestsCount} IssueResolvingTime)}{TotalServiceRequestsCount} \quad (1)$$

The ARQ evaluates the autonomic efficiency of a set of self-* services for an increasing number of self-management requests. A lesser value of ARQ means that the autonomic efficiency is better.

An experimental evaluation was logged by generating a request to the pool of five AMs in three scenarios: (1) statically configured local AMs; (2) dynamically configured AMs from a server machine; and (3) runtime registered AMs from the Autonomic_Cloud. The evaluation results of ARQ using Equation (1) are provided as Figure 12. It shows that autonomic efficiency degraded with the increasing number of self-management requests for the static AMs. The dynamic AMs utilizing the CPU share from the server machine produced a better response. The S*SAAS responded best to the increased number of self-management requests.

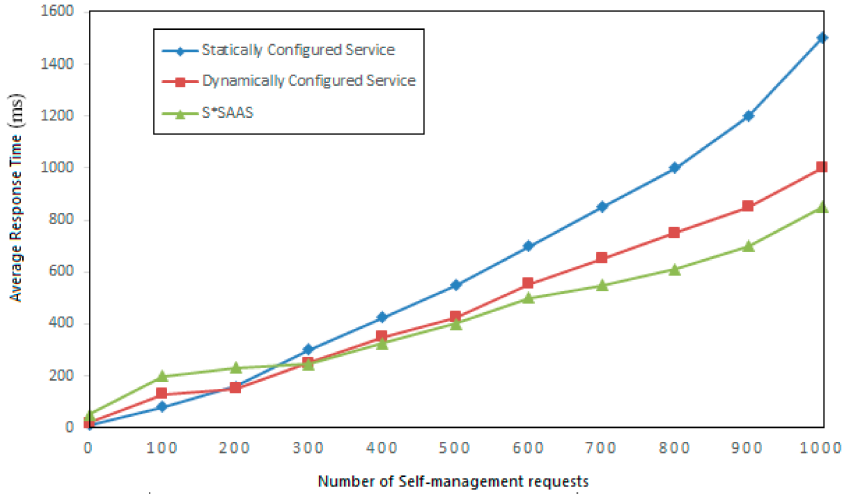

Figure 12. Average response quality for static, dynamic and S*SAAS services.

Thus, for software applications requiring continuous support from self-management services, the proposed system achieves a significant improvement in the autonomic efficiency and issue resolution time, compared to the state-of-the-art approaches.

In addition to the experimentally-demonstrated self-* services from the Autonomic_Cloud, ideas from different research works can be offered as self-* services from the Autonomic_Cloud. For example:

- The management of a huge amount of highly sensitive data generated by medical sensor networks covering scalability, availability and security issues [34].
- A cloud-based distributed toolbox to optimally manage the energy dispatch from renewable resources utilizing the GPU and WRNN predictors [35].
- Cloud-based energy serving a multi-agent system using web service techniques with backend information agents involved [36].
- A neural-network-driven forecasting setup to manage the power production and dispatch in a smart grid [37]
- A user wants to run their own application over the cloud to process their own data utilizing encrypted computation [47]

4.3. Socio-Economic Benefits of the Proposed Research

The socio-economic benefits of the proposed research include but are not limited to data centers, big data analysis, and energy management, etc. Applications include electric, defense, computer programs, networks and data centers. Interruption due to varying conditions of these processes and structures can occur before the system operators find out about the damage and remove its cause. The design of control systems that integrate and configure self-* services from the cloud offer the dynamic incorporation of self-management capabilities. This facilitates the control systems for the detection of anomalies of large-scale distributed systems and to alleviate the process interference, and humans receive better software services as the system works intelligently to manage its perturbations.

4.4. Complexity Analysis

The complexity of the RAV process is linear. The registration time is constant, whereas the search for a monitor–sensor and executor–effector entry in the authentication table depends on the table size (for example, 'n'). Hence, O(n) drops to log(n) if binary searching is used. The complexity of the SMP process is linear. It depends on the number of context attributes, the size of the threshold table, the size of the symptoms repository, and the size of the rules base.

5. Conclusions

This research improves the autonomic computing design concept in two aspects. Firstly, a modified architectural design for a secure, scalable and elastic autonomic computing system was presented. The SSE-ACS paradigm was defined, consisting of a self-management process (SMP) and a registration, authentication and validation process (RAV). The RAV secures the use of the sensors and effectors of a resource from unauthorized autonomic managers. The RAV layer inside the touchpoint allows the addition and removal of autonomic managers at the runtime without any unauthorized usage concerns. The potential to accommodate the autonomic system's growth at runtime renders SSE-ACS scalable and elastic.

Secondly, self-* capabilities as a service (S*SAAS) are offered by the introduction of an Autonomic_Cloud. A layered approach for the dynamic adaptation of self-* services was proposed for this purpose. With a pay-per-use policy imitating the cloud computing style, the proposed approach allows the continual growth of self-* capabilities with less investment. The system optimizes the CPU share using elastic AMs resulting in an improved execution time of the business logic. For software applications requiring the continuous support of self-management services, the proposed system achieved a significant improvement in autonomic efficiency and issue resolving time, compared to the state-of-the-art approaches.

The structure breakdown of the autonomic manager and knowledge base into sub-entities provided high cohesion, low coupling, and allowed system implementation in a distributed environment. This is useful in cases of high-performance systems and thread-based systems.

Authors Contribution

The authors contributed equally to the design of ideas, analysis of results, and writing of the article.

Acknowledgments: We would like to acknowledge my family, friends and colleagues for their support, suggestions and reviews while conducting this study. I am also thankful to them for the successful completion of this part of the project.

Conflicts of Interest: The authors declare no conflict of interest.

Nomenclature

ACS	Autonomic Computing System
AM	Autonomic Manager
CC	Connection Controller
CIA	Confidentiality, Integrity, Availability
CPU	Central Processing Unit
CVSS	Common Vulnerability Scoring System
MR	Managed Resource
PC	Personal Computer
PKI	Public Key Infrastructure
RAV	Registration, Authentication and Validation
RSA	Rivest, Shamir & Adleman (public key encryption technology)
RSI	Relative Strength Index
SSE-ACS	Secure, Scalable and Elastic Autonomic Computing System
SMP	Self-Management Process
SO	Stochastic Oscillator
William's % R	William's % Relative Strength
MFI	Money Flow Index
MACD	Moving Average Convergence Divergence
L14	Lowest of the previous 14 trading sessions
H14	Highest of the previous 14 trading session

References

1. Ahuja, K.; Dangey, H. Autonomic Computing: An emerging perspective and issues. In Proceedings of the IEEE International Conference on Issues and Challenges in Intelligent Computing Techniques, Ghaziabad, India, 7–8 February 2014; pp. 471–475. [CrossRef]
2. White, S.R.; Hanson, J.E.; Whalley, I.; Chess, D.M.; Kephart, J.O. An architectural approach to autonomic computing. In Proceedings of the IEEE International Conference on Autonomic Computing, New York, NY, USA, 17–18 May 2004; pp. 2–9. [CrossRef]
3. Hariri, S.; Khargharia, B.; Chen, H.; Yang, J.; Zhang, Y.; Parashar, M.; Liu, H. The autonomic computing paradigm. *Clust. Comput.* **2006**, *9*, 5–17. [CrossRef]
4. Okon, S.C.; Asagba, P.O. Self-Organization and Self-Healing: Rationale and Strategies for Designing and Developing a Dependable Software System. *IJIRCCE* **2014**, *2*, 3687–3698.
5. Raibulet, C. Hints on Quality Evaluation of Self-Systems. In Proceedings of the IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems (SASO), London, UK, 8–12 September 2014; pp. 185–186. [CrossRef]
6. Wolf, T.D.; Holvoet, T. A Taxonomy for Self-* Properties in Decentralised Autonomic Computing. In *Autonomic Computing: Concepts, Infrastructure and Applications*; Parashar, M., Hariri, S., Eds.; CRC Press/Taylor & Francis: Boca Raton, FL, USA, 2006; Chapter 1; pp. 3–24.
7. Mittal, P.; Singhal, A.; Bansal, A. A Study on Architecture of Autonomic Computing-Self Managed Systems. *IJCA* **2014**, *92*, 6–9. [CrossRef]
8. Kumar, K.P.; Naik, N.S. Self-Healing model for software application. In Proceedings of the IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE, 2014), Jaipur, India, 9–11 May 2014; pp. 1–6. [CrossRef]

9. Jacob, B.; Lanyon-Hogg, R.; Nadgir, D.K.; Yassin, A.F. A practical guide to the IBM autonomic computing toolkit. In *IBM Redbooks*; IBM Corp. International Technical Support Organization: North Castle, NY, USA, 2004.
10. Manoel, E.; Nielson, M.J.; Salahshour, A.; KVL, S.S.; Sudarshanan, S. Problem determination using self-managing autonomic technology. In *IBM Redbooks*; IBM International Technical Support Organization: North Castle, NY, USA, 2005.
11. IBM, Coorporation. *An Architectural Blueprint for Autonomic Computing*; IBM White Paper; IBM: Hawthorne, NY, USA, 2006; Volume 31, pp. 1–6.
12. Miller, B. The Autonomic Computing Edge: Keeping in Touch with Touchpoints. Article Series on Autonomic Architecture. 2005. Available online: http://www.ibm.com/developerworks/autonomic/library/ac-edge5 (accessed on 2 January 2016).
13. Shuaib, H.; Anthony, R.J.; Pelc, M. A framework for certifying autonomic computing systems. In Proceedings of the IARIA 7th International Conference on Autonomic and Autonomous Systems, Venice, Italy, 22–27 May 2011; ICAS: Venice, Italy; Mestre, Italy, 2011; pp. 122–127.
14. First.org. Common Vulnerability Score System v3.0. first.org. 2015. Available online: https://www.first.org/cvss/user-guide (accessed on 5 March 2017).
15. Wang, J.A.; Wang, H.; Guo, M.; Xia, M. Security metrics for software systems. In Proceedings of the ACM 47th Annual Southeast Regional Conference, Clemson, South Carolina, 19–21 March 2009. [CrossRef]
16. Nzekwa, R.A.; Rouvoy, R.; Seinturier, L. Modelling feedback control loops for self- adaptive systems. In Proceedings of the Third International DisCoTec Workshop on Context-Aware Adaptation Mechanisms for Pervasive and Ubiquitous Services, Amsterdam, The Netherlands, 10 June 2010; pp. 1–6.
17. Bezerra, D.S.; Martins, R.; Martins, J.S.B. A Policy-Based Autonomic Model Suitable for Quality of Service Management. *J. Netw.* **2009**, *4*, 495–504. [CrossRef]
18. Horn, P. *Autonomic Computing: IBM's Perspective on the State of Information Technology*; IBM Corp.: North Castle, NY, USA, 2001.
19. Kephart, J.O.; Chess, D.M. The vision of autonomic computing. *Computer* **2003**, *36*, 41–50. [CrossRef]
20. McCann, J.; Huebscher, M. Evaluation issues in autonomic computing. In Proceedings of the Grid and Cooperative Computing Workshops, Wuhan, China, 21–24 October 2004; Lecture Notes in Computer Science. Springer: Berlin/Heidelberg, Germany, 2004; Volume 3252, pp. 597–608. [CrossRef]
21. Abuseta, Y.; Swesi, K. Design patterns for self-adaptive systems engineering. *IJSEA* **2015**, *6*, 11–28. [CrossRef]
22. Gandrille, E.; Hamon, C.; Lalanda, P. Linking Reference and Runtime Architectures in Autonomic Systems. In Proceedings of the IST-115 Symposium on Architecture Definition and Evaluation, Toulouse, France, 13–14 May 2013; pp. 1–8.
23. Ramirez, A.J.; Cheng, B.H. Design patterns for developing dynamically adaptive systems. In Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, Cape Town, South Africa, 3–4 May 2010; pp. 49–58. [CrossRef]
24. Chen, S.L.; Chen, Y.Y.; Hsu, C. A new approach to integrate internet-of-things and software-as-a-service model for logistic systems: A case study. *Sensors* **2014**, *14*, 6144–6164. [CrossRef] [PubMed]
25. Kim, H.W.; Han, J.; Park, J.H.; Jeong, Y.S. DIaaS: Resource Management System for the Intra-Cloud with On-Premise Desktops. *Symmetry* **2017**, *9*, 1–17. [CrossRef]
26. Atighetchi, M.; Ishakian, V.; Loyall, J.; Pal, P.; Sinclair, A.; Grant, R. Metrinome: Continuous Monitoring and Security Validation of Distributed Systems. *CSIAC* **2014**, *2*, 20–26.
27. El-Kabbany, G.F.; Rasslan, M. Security Issues in Distributed Computing System Models. In *Security Solutions for Hyperconnectivity and the Internet of Things, by Mohamed Eltayeb, Maurice Dawson Marwan Omar*; IGI Global: Hershey, PA, USA, 2016; Chapter 9; pp. 211–259. [CrossRef]
28. Mishra, K.S.; Tripathi, A.K. Some Issues, Challenges and Problems of Distributed Software System. *IJCSIT* **2014**, *5*, 4922–4925.
29. Kumar, M.; Agrawal, N. Analysis of Different Security Issues and Attacks in Distributed System A-Review. *IJARCSSE* **2013**, *3*, 232–237.
30. Berket, K.; Essiari, A.; Muratas, A. PKI-based security for peer-to-peer information sharing. In Proceedings of the IEEE Fourth International Conference on Peer-to-Peer Computing, Zurich, Switzerland, 27 August 2004; pp. 45–52. [CrossRef]
31. Bing, Y.; Yanni, H.; Hanning, Y.; Zhou, X.; Xu, Z. A cost-effective scheme supporting adaptive service migration in cloud data center. *Front. Comput. Sci.* **2015**, *9*, 875–886.

32. Masood, R.; Shibli, M.A.; Ghazi, Y.; Kanwal, A.; Ali, A. Cloud authorization: Exploring techniques and approach towards effective access control framework. *Front. Comput. Sci.* **2015**, *9*, 297–321. [CrossRef]
33. Shi, W.; He, D. A security enhanced mutual authentication scheme based on nonce and smart cards. *JCIE* **2014**, *37*, 1090–1095. [CrossRef]
34. Lounis, A.; Hadjidj, A.; Bouabdallah, A.; Challal, Y. Secure and scalable cloud-based architecture for e-health wireless sensor networks. In Proceedings of the IEEE 21 international conference on Computer communications and networks (ICCCN), Munich, Germany, 30 July–2 August 2012; pp. 1–7.
35. Bonanno, F.; Capizzi, G.; Sciuto, G.L.; Napoli, C.; Pappalardo, G.; Tramontana, E. A novel cloud-distributed toolbox for optimal energy dispatch management from renewables in igss by using wrnn predictors and gpu parallel solutions. In Proceedings of the IEEE International Symposium on Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM), Ischia, Italy, 18–20 June 2014; pp. 1077–1084.
36. Yang, S.Y. A novel cloud information agent system with Web service techniques: Example of an energy-saving multi-agent system. *Expert Syst. Appl.* **2013**, *40*, 1758–1785. [CrossRef]
37. Napoli, C.; Pappalardo, G.; Tina, G.M.; Tramontana, E. Cooperative strategy for optimal management of smart grids by wavelet rnns and cloud computing. *IEEE Trans. Neural Netw. Learn. Syst.* **2016**, *27*, 1672–1685. [CrossRef] [PubMed]
38. TechTarget. X.509 Certificate of Public Key Infrastructure (PKI). Available online: http://searchsecurity.techtarget.com/definition/X509-certificate (accessed on 4 January 2016).
39. Tseng, Y.M.; Yu, C.H. Towards Scalable Key Management for Secure Multicast Communication. *Inf. Technol. Control* **2012**, *41*, 173–182. [CrossRef]
40. vmware.com. Available online: https://docs.vmware.com/en/VMware-vSphere/6.5/vsphere-esxi-vcenter-server-65-installation-setup-guide.pdf (accessed on 4 January 2016).
41. Python. psutil 5.4.2, Pyhton. Available online: https://pypi.python.org/pypi/psutil (accessed on 1 September 2017).
42. MarketIndex.com. Available online: https://www.marketindex.com.au/ (accessed on 1 February 2016).
43. Lo, A.W.; Mamaysky, H.; Wang, J. Foundations of technical analysis: Computational algorithms, statistical inference and empirical implementation. *J. Financ.* **2000**, *55*, 1705–1765. [CrossRef]
44. StockCharts.com. Available online: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:introduction_to_technical_indicators_and_oscillators (accessed on 6 January 2016).
45. Yahoo Finance(marketindex), Yahoo Finance API. Available online: https://www.marketindex.com.au/yahoo-finance-api (accessed on 5 February 2016).
46. Yahoo.com, Yahoo Query Language. Available online: https://developer.yahoo.com/yql/ (accessed on 1 February 2016).
47. Fletcher, C.W.; Dijk, M.V.; Devadas, S. A secure processor architecture for encrypted computation on untrusted programs. In Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing, Raleigh, NC, USA, 15 October 2012; pp. 3–8.

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).