ISSN 2047-3338

# A Semi-Automatic Method for Self-Configuration in Distributed Systems Using Hidden Markov Model

Reza Mohamadi Bahram Abadi[1] and Mohsen Jahanshahi[2,*]

[1]Dep. of Computer Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran
[2]Dep. of Computer Engineering, Central Tehran Branch, Islamic Azad University, Tehran, Iran
[1]Mohamadi_re@yahoo.com, [2]Mjahanshahi@iauctb.ac.ir

***Abstract*— The fast, continuous changes in systems require the re-configurations be programmed in such a way that the system can respond to changes efficiently. Self-configuration is the capability of automatic re-configuration of a system in response to such changes. In distributed systems, because of the inconsistent nodes, dynamism of resources and heterogeny of communicative networks, the changes are dramatic. Therefore, the system needs to be monitored consistently and/or periodically to be reconfigured, if required. Upon entry of any user and/or an applied application considering the status *quo*, the system needs to be re-configured. Also in case of system failure, the configuration has to be done automatically in line with the predetermined objectives. This paper presents a semi-automatic method using the hidden Markov model for self-configuration of the system considering infrastructural resources and requests by users.**

***Index Terms*— Self-Configuration, Hidden Markov Model and Distributed System**

## I. INTRODUCTION

SELF-CONFIGURATION is the capability of reconfiguration of a system in an automatic dynamic way in response to the changes made by installing, updating, synthesizing, analyzing and combining system software's. When, a new component is introduced to the system, it has to integrate itself into the rest of components in a hidden way, while other components should adapt to the new component as well [1].

Self-configuration is significantly important to create high level dynamism in performing a series of applied applications in a distributed system. Considering the type of distributed system, there are certain limitations in managing and allocating resources to demands. Also, the time of running an application and required resources for its performance vary drastically and are addressed based on the user's needs [2].

In order to manage and respond to new demands in heterogeneous distributed system, due to the variety and dynamism of resources in access, inconsistency of nodes, dynamism of load distribution, heterogeneity of calculator resources and heterogeneity of communicative infrastructures, network reconfiguration is highly demanded at the time of running. With the continuous changes in such networks, an automatic reconfiguration is critically needed to manage such changes [3].

The system supporting reconfiguration should monitor the influencing factors continually, and upon diagnosis of change, it has to initiate reconfiguration process. In 2007, a basis called "Adapta" presented self-configuration in response to environmental changes in a distributed system based on XML.

The basis includes the followings [4]:

- Monitoring service: it picks up some data from software and hardware resources periodically. E.g. processors' load, amount of main memory in access, network bandwidth.
- Local event service: the data resulting from monitoring is registered locally for software and hardware servers.
- Event processing system: is a distributed system including the analysis of events consisting of different event resources and also distributed nodes, which are influencing in re-configuration.
- Dynamic reconfiguration service: it presents a dynamic adaptable engine which does the reconfiguration of programs considering the event processing results in the processing system in response to the environmental changes.

When there is a new demand in the distributed system, all the required resources must be provided and when the running of a program is over, the resources are released accordingly. Also, with regard to the dynamism of the resources in access and the instability of the nodes, it is possible that the resources in access are on the change anytime. The monitoring service must be able to react properly periodically or at any moment against the change of resources and report changes back [5].

Also, if there is a fault in the system for any reason, the problem has to be solved. First, the system must identify the problem, and then, it must do reconfiguration considering the already diagnosed faults [6]. In this method, data is analyzed using information collected at the time of running and by resource checker, and by deducing from system knowledge and the use of Markov process model, the system begins to reconfigure itself automatically.



[*]*Corresponding Author's Email: Mjahanshahi@iauctb.ac.ir*

## II. Background And Realated Works

With the particular characteristics of distributed systems, ongoing changes are critical to achieve pre-determined goals of the system. Such characteristics may include variety and dynamism of the resources in access, instability of nodes, dynamism in load distribution, heterogeneity of calculator resources, and heterogeneity of communicative infrastructures (i.e., network technology) [7]. Therefore, self-configuration of the system is an important challenge in distributed systems in face of changes and/or threats. Some different architecture has been used in this regard, which will be discussed later in this paper.

### *A. Review of FOSSII Architecture*

Fig. 1 shows the infrastructure of FOSSII architecture [8]. The figure presents two main components in managing demands in line with the available resources in infrastructure. As the first step, decision is made on controlling resources and applied programs. Then, in the second step, deducing from the knowledge of the first phase, how demands are responded to and how the resources are allocated will be decided.

The infrastructural resource sensors measure them continually while the sensor of applied programs picks up data as to how such programs run in the system. Then, after data from the sensor controllers are analyzed, it analyzes the system as well and responds to the new demands.

### *B. Review of LoM2HiS*

The LoM2HiS" Basis has two controlling components called "controller of local infrastructures" and "time controller" used for applied programs [9]. The first controller is used for controlling low-level resources while the second is used to check data at the time of running in order to prevent the breach of Agreement on Service Level.

The way is to keep details of all agreements in a table to render services within an acceptable limit. In case of any contradiction between the adaptability of the data at the time of running and those in the table, the system is informed [9]. The basis does not present a mechanism for automatic reconfiguration of the system and only suffices to the breach details of the agreement.

### *C. The diagnosis of threats and ways to face them in calculator systems*

Hackers consider different strategies to attack the systems at various levels. An attack plan comes in 3 parts:

- Identification: this includes obtaining data from a machine such as guessing the password.
- Penetrating the system.
- Attack: using a dangerous way to attack others.

All threats influencing the system are kept in an event registered file. To identify the threats, the existing data is first checked in an event registered file. However, for any of the observed threats there are some different reasons, which are not observable by the controller. In this way, while the observed cases in events registered file are used, there is a Markov process model, which is used for identifying the threats in the system [10]. This paper considers the idea for predicting the breach of agreements in the system.

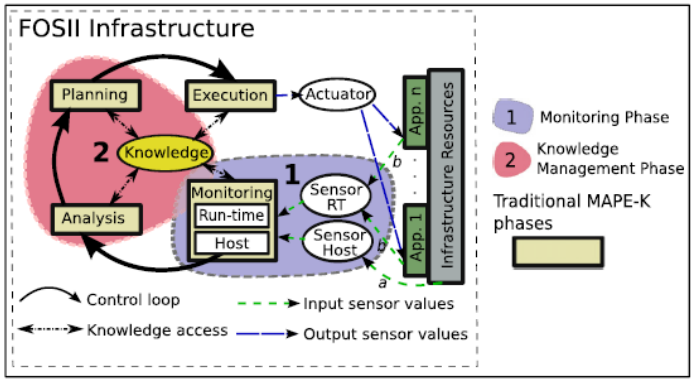


Fig. 1. FFOSSII Architecture for managing the demands

### *D. DeSVI architecture: automatic diagnosis of breach at the service level in Cloud Computing*

Cloud Computing provides suitable grounds to do the comprehensive and measurable calculations in the network. Here, the users expect to receive their services in an acceptable quality and they are willing to pay the costs in line with the service level coming from the service providers [11]. Also, in case of low quality at the service level compared to that of what was already agreed upon, the service provider company must pay the customer for the damages. Therefore, the automatic diagnosis of the agreement breach and identification of the reasons thereof for removing the breaches without user's interference is one of the most important challenges in the area of networks of these kinds.

The architecture DeSVI in Fig. 2 discusses a way to specify the breach of agreement and the way it is managed in the area [11]. At the highest level of this architecture, it is the user (or customer) who uses the services in need through a software link to the service provider in the area. Hence, the service provider does the service request based on talk and service level agreement.

In the second layer, the module application Deployer is next to module Run-Time Monitor. In this layer, the virtual machines and the required resources provide the service and their lodgment in the area. The activity of this module is based on the data obtained from the lower layer. There are some other resources but the virtual machines. However, the architecture DeSVI only stresses the management of the virtual machine.

In the third layer, the module VM Deployer and Configurator for the virtual machines have been planned. The activity of this module is based on data which is received from module Host Monitor in the fourth layer. The host control module measures the existing resources in the physical area of the host and sends the monitored data to the applicant modules.

### *E. Management and the gradual removal of the error in the services provided in Cloud area*

In service-based systems, there are a series of services which are in interaction for reaching a particular purpose. The design and the development of these kinds of systems become possible using service-oriented architecture [12].

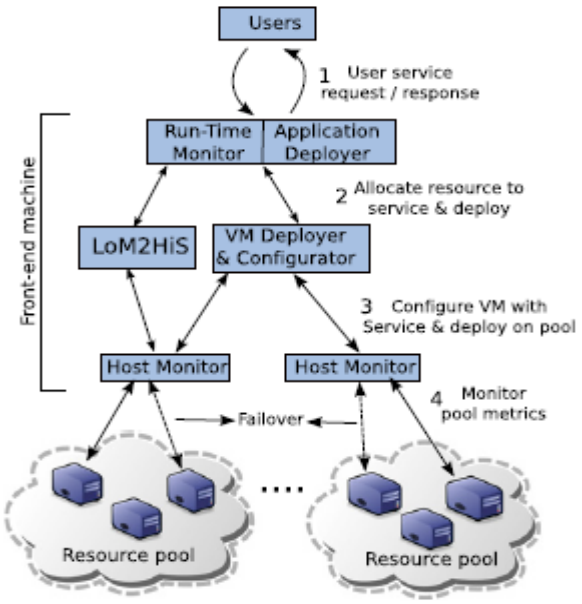


Fig. 2. Architecture DeSVI and components

The applicant of using the service may consider some quality aspects of the service. Therefore, a criterion for evaluation has to be put to attention based on the service, so that if the quality level of the services is lower than expected, the system can be recovered and reconfigured. In 2012, Vincet and his associates presented a new architecture to manage the breach of the agreements consisting of the following components [12]:

- Violation Handling Manager**:** it is used for collecting the quality data of the service, identifying and also determining the breach of quality. The control is done in two steps in this component. The first step includes receiving a series of events and identifying their link with the terms of the agreement breach. The second step deals with any kind of deviation at the service level.
- Impact analysis**:** it is used for determining the influencing areas on the service and its needs, the result of which is influential in the system recovery process.
- Recovery analysis component: it is used for planning a recovery mechanism of the service in the effective area

Violation handling database: Four kinds of data are saved in this base as follows: 1) the diagram of the process and the required time to do it. 2) The service and series of time limitation 3) the data relating to the breach of commitments 4) the data relating to the effective areas.

## III. A Novel Architecture for Self-Configuration In Distributed Systems

In a distributed system, the resources and required services by the user are allocated in line with the type of request. Every user expects to receive the required services in good quality. In case of unacceptability of the service level for the applicants, the existing resources in the network must be reconfigured in a way that the goals of the system are met. To this end, the existing resources in the network must be controlled and judged continually so that in case of any deviation in rendering the service, acts can be taken to remove it [13].

In this section, the infrastructure of the suggested system for self-configuration in distributed system is discussed. The way to meet the needs of the users is managed in the network considering the existing resources. Upon the entry of any new request and/or the finish of every request in the network, the list of available resources is updated. Also the efficiency of important parameters in the system and the running systems are put to control. Considering the best situation and the limit, which has already been determined, if any threat is felt, the system is reconfigured automatically deducing the existing knowledge in the system (Fig. 3).

In the suggested architecture, the steps to perform the action are as follows:

*Step 1:* The existing resources in infrastructure of distributed system are put to control by means of sensors for this end. The obtained data from the resources in access and also allocated resources are saved in a base.

*Step 2:* The requests of the users and/or applied programs stay in the line of the applicants, and in line with the considered priority, one of them is sent to the analysis section for the possibility of execution.

*Step 3:* The administrative task of any request and or applied program is classified. Then, we specify the resources and the required services of each task. Considering the existing data in the base, if it is possible to meet the request, it will be sent to the distributed system to run. For taking this step, we acquire the idea from the architecture DeSVi, which is about the development of applied programs in Cloud calculations. This step is shown in Fig. 4.

*Step 4:* The given services and also important system parameters are monitored by means of sensors, which have been used for this purpose (For example, the amount of use from the processor load, the amount of main memory in access, bandwidth of the network and delay in communications).

*Step 5:* data of the services and the figures of the ex-parameters are compared deducing the existing knowledge in the service base and the parameters containing data at the proper level. In case of unsuitability, warning message appears. The warning is checked to seek the cause in the next phase. The message is also saved in a log file (Fig. 5).

*Step 6:* Using the data obtained from the event registered file relating to the warnings produced as a result of N number of observations from the ex-system and also knowing about the warnings base and their creating factors, a hidden Markov model will be presented the use of which is the most likeliest situation for creating the warning.

As it is possible to have a number of factors for every critical situation in the system, it is very important to seek to an influencing factor for the reconfiguration of the system for the removal of the critical situation. However; the system controller only observes the critical situation the causing factor of which is hidden from the observer. This situation can well be explained and removed by using the hidden Markov model. For example if the amount of the main memory becomes less than that of accepted, some factors such as the damage of nodes for rendering the service (X1), the use of this space by some applicants (X2) and the inefficiency of the executive program (X3) may be influencing (Fig. 6).

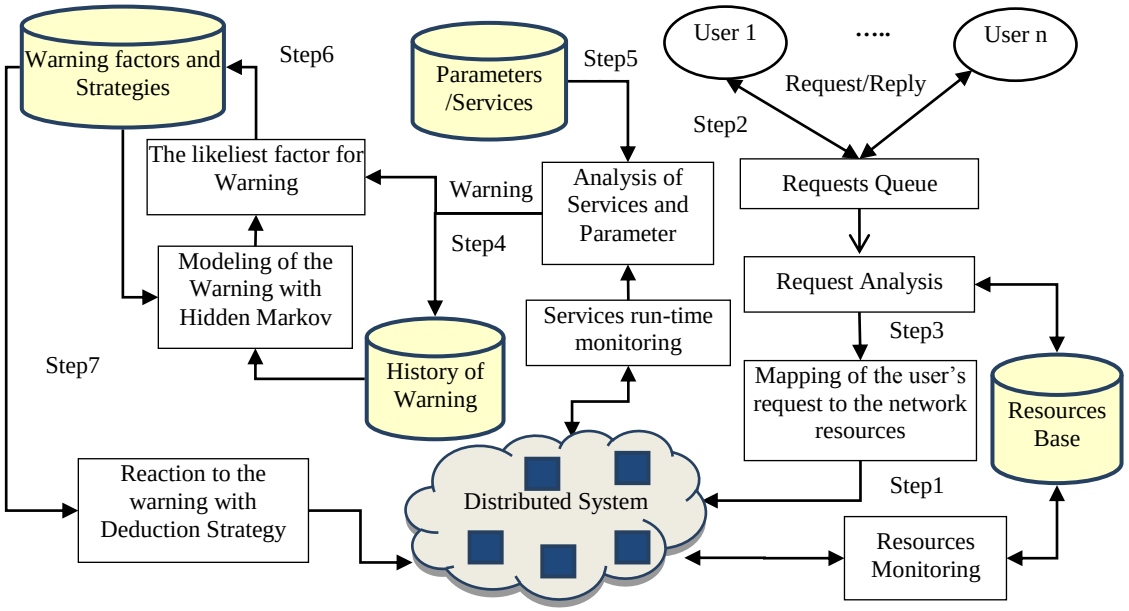


Fig. 3. Architecture of self-configuration system

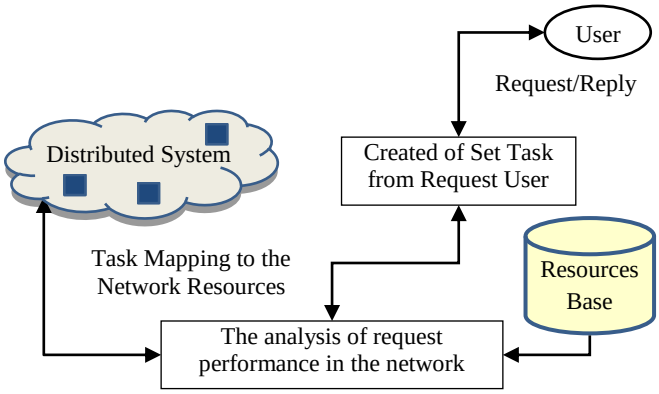


Fig. 4. Mapping of the new users' request to the distributed system resources

| Threshold | Desirable threshold | Parameters/Services |
|---|---|---|
| > 1024 | >2048 | Available Memory |
| > 12 Mbit/s | >15 Mbit/s | Bandwidth |
| > 75% | >25% | CPU |

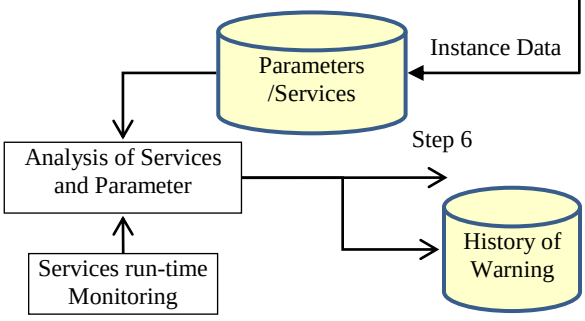


Fig. 5. Finding a critical case for the system

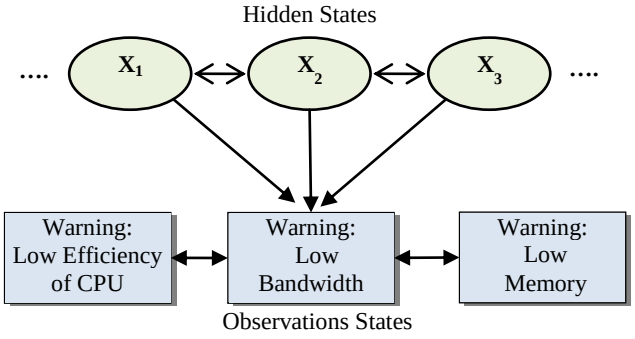


Fig. 6. Warnings observation and factors relating to Markov model

Considering the fact that the data relating to the ex-warnings are saved in an event registered file and also factors to cause each of the warnings saved in a base therefore, using a hidden Markov model and Baum-Welch and Viterbi algorithm helps us get an idea of last observances and we can find a hidden Markov model for the likeliest cause of a warning message. Each time with a new warning from the system, one can possibly explain the cause of the warning with the help of the presented model (Fig. 7).

*Step 7:* in the base of warnings, a strategy has been presented for the causes of warnings and the system acts for the automatic reconfiguration of the system using these information (Fig. 8).

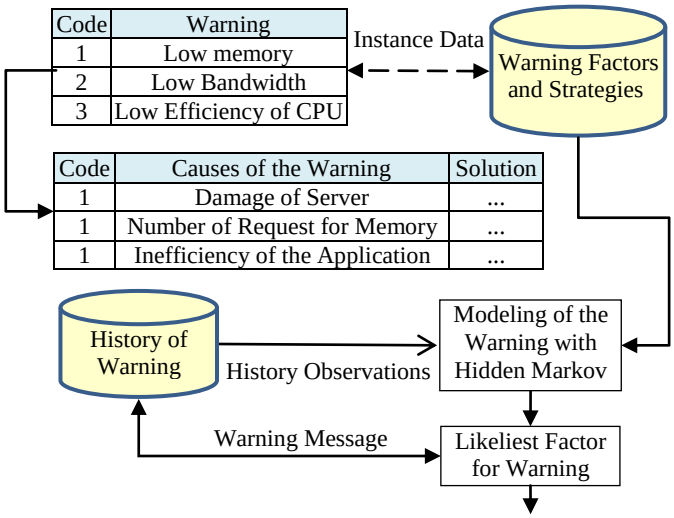


Fig. 7. Modeling the observed warnings in hidden Markov model

*Instance Data in the repository of warning factors and strategies*

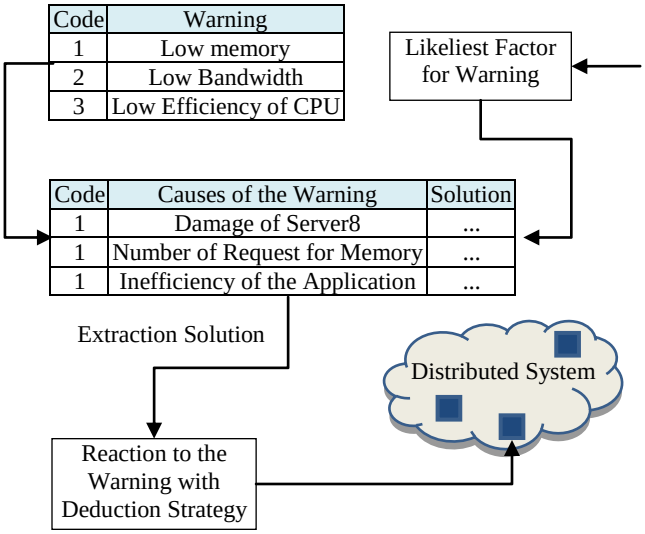


Fig. 8. Strategy extraction for reaction against system warning message

TABLE I.
EVALUATION PARAMETERS OF SERVICE QUALITY IN A DISTRIBUTED STORAGE SYSTEM

| Performance Parameters | | |
|---|---|---|
| ***SLA Parameter name*** | ***Unit*** | ***Description*** |
| SNCurrentReadTransferRate | MB/s | Current load of the SN for read transfers. SNs with lower values are preferred. |
| SNMaxReadTransferRate | MB/s | Maximal SN throughput. SNs with higher values are preferred. |
| SNTapeReadRate | MB/s | Read transfer rate for tape drive. SNs with higher values are preferred. |
| SNTapeLatency | s | The average latency of tape drives. The latency includes the time necessary to load a tape and |
| SNisFileCached | n/a | This parameter tells if the given file resides in cache. 0 means that the file is in cache or that |
| SNLoad | % | SN load. Shows how much of the potential storage bandwidth is currently used. SNs with lower |
| SNIOps | IO/s | Number of I/O operations per second. SNs with lower values are preferred. |

## IV. Performance Evalution

To review the efficiency of the suggested architecture, we evaluate its performance on a distributed storage system. With regard to the agreement of the service level that is made between the service givers and users some parameters are set for the evaluation of SLA. The purpose of the suggested architecture is to observe the agreed threshold for the SLA parameters in the system.

In a traditional storage system, the way to access the users is made with the best effort. This ordinary approach in applied programs, which must be guaranteed in the quality of the service, (QoS) has limitations. In line with the services given and the expectations of the service receivers from the way they are presented, we can define some different indexes for evaluating QoS of the services rendered [14]. In Table I, there is a series of parameters defined as a sample for the evaluation of efficiency in a distributed storage system.

In each parameter, those indexes are influencing the degree of which can be calculated. Also, the degree of some parameters can be calculated using the existing parameters in Table I. For example, for calculating load, Readpreference and WritePreference of a storage node, the following formulae are used.

$$\textbf{SNLoad} = \frac{\text{SNCurrentReadTransferRate}}{\text{SNMaxReadTransferRate}} + \frac{\text{SNCurrentWriteTransferRate}}{\text{SNMaxWriteTransferRate}} \quad (1)$$

$$\textbf{SNReadPreference} = \left(k_3 \times \frac{\text{filesize}}{\text{SNTapeReadRate}} + k_4 \times \text{SNTapeLatency}\right) \times k_5 \times \text{SNisFileCached} - k_6 \times \text{SNLoad} - k_7 \times \text{SNIOps} - k_1 \times \text{SNMaxReadTransferRate} \times \text{filesize} - k_2 \times \text{filesize} \times \text{SNCurrentReadTransferRate} \quad (2)$$

$$\textbf{SNWritePreference} = k_1 \times \text{SNMaxWriteTransferRate} \times \text{filesize} - k_6 \times \text{SNLoad} - k_2 \times \text{SNCurrentWriteTransferRate} \times \text{filesize} - k_6 \times \text{SNIOps} \quad (3)$$

To check the suggested architectural performance, we consider the parameters of Load ReadPreference and WritePreference as the evaluation parameters of QoS in a distributed storage system. With regard to what is expected from the service providers for each of the parameters, we define a threshold. These agreed figures are considered as one SLA for each parameter.

Using the influencing indexes, each parameter is calculated. If the calculated figure for one parameter is less than threshold, there is a breach in SLA. The reason of it should be searched for among the influencing parameters.

The hidden Markov model has been used in suggested architecture for the prediction of the most efficient index in the breach of SLA, To clarify the point, we can deduce the

fact that the calculated figures for the parameters as “observed states” and "indexes as the “hidden states’ are considered. From the beginning of the system working, the time output relating to the breach of SLA and the influencing index in the breach of SLA for parameters are saved in a log file. With the analysis of the recent observations in log file, the likelihood of influencing the indexes in the breach of parameters' SLA is specified over the time. Once the parameters and indexes were observed in a hidden Markov model, with occurrence of the SLA breach for one parameter, the most likelihood index for the breach of SLA is identified and introduced. Then, we amend the index for securing the parameter SLA.

TABLE II.
THRESHOLD FOR THE SAMPLE PARAMETERS IN DISTRIBUTED STORAGE SYSTEM

| ID | SLA Parameter | Threshold |
|---|---|---|
| 1 | SN Load | ≥12 Mbit/s |
| 2 | SNReadPreference | ≥15 Mbit/s |
| 3 | SN WritePreference | ≥10 Mbit/s |

TABLE III.
INFLUENCING LIKELIHOOD ANY OF THE INDEXES IN SLA BREACH

| ID Code | SLA Parameter | Occurrence Probability |
|---|---|---|
| 1 | SN Current Read Transfer Rate | 40% |
| 1 | SN Max Read Transfer Rate | 20% |
| 1 | SN Current Write Transfer Rate | 30% |
| 1 | SN Max Write Transfer Rate | 10% |
| 2 | SNMaxReadTransferRate | 18% |
| 2 | SNCurrentReadTransferRate | 15% |
| 2 | SNTapeReadRate | 9% |
| 2 | SNTapeLatency | 23% |
| 2 | SNisFileCashed | 16% |
| 2 | SNLoad | 10% |
| 2 | SNIOps | 4% |
| 2 | FileSize | 5% |
| 3 | SNMaxWriteTransferRate | 20% |
| 3 | SNCurrentWriteTransferRate | 23% |
| 3 | SNLoad | 43% |
| 3 | SNIOps | 17% |
| 3 | Filesize | 12% |

TABLE IV.
RESULT OF ARCHITECTURAL TESTS FOR PARAMETER SNALOAD

| ID Code | TP | FN | FP |
|---|---|---|---|
| SN Load | 11 | 2 | 3 |
| SNReadPreference | 64 | 6 | 21 |
| SN WritePreference | 43 | 4 | 16 |
| SUM | 118 | 12 | 40 |

### *A. Review of Test Result*

To check the results of the tests, we use the suggested architecture on the university database including information about finance, administration, research and training. The data is saved on a distributed storage system by means of applied software and is recovered in case of need. Degree of system QoS is evaluated based on the calculation results of three parameters of Load, Readpreference and WritePreference and its comparison with corresponding thresholds. The degree of threshold for each parameter is shown in Table II.

The existing data of university in Log File was considered over the past three years and for the indexes of which one parameter is created, the likelihood of influencing each index for the breach of parameter SLA in intervals has been calculated. The results have been shown in Table III.

Then, the system was put to monitoring for six months, and the figures SNLoad, ReadPreference and WritePreference were calculated periodically. In each calculation , if the calculated parameter is not within the limits of the threshold, it will be a breach of SLA and based on the suggested architecture, with the use of the hidden Markov model, the most likelihood of the effective index was predicted the results of which are shown in Table III.

- TP: number of cases that the influencing index on the breach of SLA has distinguished correctly.
- FN: number of cases no index of which is influencing on the breach except for the influencing cases.
- FP: number of cases the influencing index of which on the breach SLA has been identified by mistake.

To evaluate Recall and Precision of the results of the suggested architecture, we use the data in Table IV (FP, FN, and TP) in the following formulae:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{118}{118 + 12} \cong 0.91$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{118}{118 + 40} \cong 0.75$$

As the results of the suggested algorithm are evaluated based on Log File, the more time duration of Log File, the more extensive base of knowledge, and following that, results of evaluation become more exact.

After the most likelihood of index was identified by the suggested architecture in the breach of SLA, the best would be made to recover the index in line of SLA considering the already existing strategies in the system for this challenge.

As the suggested architecture has been checked in the system randomly, the results of the evaluation will be different by a change in the situation, storage system and applied program based on which they are saved and recovered. Also, one cannot judge the performance of the suggested architecture in decisive terms. In fact, we can see the suggested architecture as a decision support system that helps the administrator in making decisions against the breach of SLA.

## REFERENCES


[1] Mishra, Arun, and A. K. Misra. "Component assessment and proactive model for support of dynamic integration in self adaptive system." *ACM SIGSOFT Software Engineering Notes* 34, no. 4 (2009): 1-9.

[2] Brejová, Brona, Daniel G. Brown, and Tomaˇs Vinaˇr. "ADVANCES IN HIDDEN MARKOV MODELS FOR SEQUENCE." *Bioinformatics Algorithms: Techniques and Applications* 3 (2008): 55.

[3] Lember, Jüri, and Alexey Koloydenko. "The adjusted Viterbi training for hidden Markov models." *Bernoulli* 14, no. 1 (2008): 180-206.

[4] Sallem, Marcio Augusto Sekeff, and F. J. da Silva e Silva. "The Adapta Framework for Building Self-Adaptive Distributed Applications." In *Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on*, pp. 46-46. IEEE, 2007.

[5] Koloydenko, Alexey, and Jüri Lember. "Infinite Viterbi alignments in the two state hidden Markov models." *Acta Comment. Univ. Tartu. Math* 12 (2008): 109-124.

[6] Lee, Geunho, Yosuke Hanada, and Nak Young Chong. "Decentralized Self-configuration of a Swarm of Robots." In *Proc. of 7th SICE System Integration Division Annual Conference*. 2006.

[7] Lember, Jüri, and Alexey Alexandrovich Koloydenko. "A constructive proof of the existence of Viterbi processes." *Information Theory, IEEE Transactions on* 56, no. 4 (2010): 2017-2033.

[8] Emeakaroha, Vincent C., Marco AS Netto, Rodrigo N. Calheiros, Ivona Brandic, Rajkumar Buyya, and César AF De Rose. "Towards autonomic detection of SLA violations in Cloud infrastructures." *Future Generation Computer Systems* 28, no. 7 (2012): 1017-1029.

[9] Gandhi, Anshul, Yuan Chen, Daniel Gmach, Martin Arlitt, and Manish Marwah. "Hybrid resource provisioning for minimizing data center SLA violations and power consumption." *Sustainable Computing: Informatics and Systems* 2, no. 2 (2012): 91-104.

[10] Lember, Jüri. "On approximation of smoothing probabilities for hidden markov model" *statistics & probability letters* 81, no. 2 (2011): 310-316

[11] Lember, Jüri, and Alexey Koloydenko. "The adjusted Viterbi training for hidden Markov models." *Bernoulli* 14, no. 1 (2008): 180-206.

[12] Kuljus, Kristi, and Jüri Lember. "Asymptotic risks of Viterbi segmentation." *Stochastic processes and their applications* 122, no. 9 (2012): 3312-3341.

[13] Ismail, Azlan, Jun Yan, and Jun Shen. "Incremental service level agreements violation handling with time impact analysis." *Journal of Systems and Software* 86, no. 6 (2013): 1530-1544.

[14] Nikolowa D, Słotaa R, Polaka S. "Model of QoS Management in a Distributed Data Sharing an Archiving System". *International Conference on Computational Science 18*, 2013: 100-109.