EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

Improving Performance in Component Based Distributed Systems

Fahd N. Al-Wesabi[1,*], Huda G. Iskandar[2] and Mokhtar M. Ghilan[3]

[1]Assistant professor, College of Science and Arts in Mahayel Asir, King Khalid University, KSA, Faculty of Computer and IT, Sana'a University, Sana'a, Yemen

[2]Master candidate, Faculty of Computer and IT, Sana'a University, Sana'a, Yemen.
[3]Assistant professor, Faculty of Computer and IT, Sana'a University, Sana'a, Yemen.

Abstract

Assuring high performance is one of the most important factors when building computerized systems. In distributed component-based systems different configurations and workloads are common. Over the years, many approaches were proposed to analyse, predict, measure and evaluate the performance of component-based distributed systems in an attempt to improve its performance. Higher performance results in better utilization of system resources, high throughput and quick response time of user requests. In this paper, we extend the work of the E-Avala approach to improve the overall performance of the proposed approach by adding power processing capabilities.
The proposed approach has been implemented and compared with previous approach using Arena simulation with varying configuration parameters. Comparative results show that the proposed approach has given better performance with different levels of processing powers.

Keywords: Distributed systems, component based distributed systems, performance, availability and delay time.

Received on 20 February 2019, accepted on 23 June 2019, published on 04 July 2019

doi: 10.4108/eai.13-7-2018.159357

*Corresponding author. Email: Fwesabi@gmail.com

1. Introduction

Many new business systems are now developed by configuring off-the-shelf systems. However, some off-the-shelf systems cannot meet all company's requirement. Therefore, a special designed software must be specially developed. When developing new enterprise systems with customized software, component-based software engineering is considered an effective reuse-oriented way.
Component-based software engineering (CBSE) is the successor of object-oriented software development [1,2] and has been supported by commercial component frameworks such as Microsoft's COM, Sun's EJB or CORBA CCM. Software components are units of composition with explicitly defined provided and require interfaces [1].

In today's world, distributed systems have replaced their central ones. This is fairly understandable because distributed systems have higher availability, reliability, and incremental growth. In the case of component-based distributed systems, software components are distributed across many different hosts. Therefore, a decision should be made to locate the platforms in which the components will be deployed [3]. When making such a decision, there are some issues needed to be considered which are:

1. Component requirements of hardware and software. When designed, some components require certain hardware architecture or software systems, so these components should be deployed on a platform that provides their hardware requirements and software support.
2. The availability requirements of the system. in order to satisfy the high availability requirement of some systems, components should be deployed on more than one platform. This means that, a substitutional implementation of the component is available if platform failure occurs.
3. Component communications. If communication level is high between some components, they should be deployed on the same platform or on physically close platforms. This reduces communications latency when sending and receiving messages between components
4. The overall performance of the system. Components should be deployed on platforms with higher processing capabilities to guarantee higher performance for the system.

Improving availability in distributed systems had been given due thought by the researchers in the past. Other researches have proposed approaches to manage redeploying components regarding dependency relations between them. However, these researches have ignored the performance of a distributed system. The implementation and evaluation of an approach that increases the overall performance of the system are complex due to the need of creating virtual tasks processing the components to measure the system's performance. Therefore, it should merit enough attention.

2. Related Works

Over the years, numerous approaches have been proposed to evaluate the performance of component-based software systems such as resource utilization, throughput and response time. Some of these approaches are aimed to predict performance and others to measure performance. The former ones' goal is to avoid performance problems in system implementation by analysing the expected performance of a component-based software design. These problems can lead to redesigning the component-based software architecture with substantial costs if not avoided. The latter ones are aimed to analyse the performance of implemented and running component-based systems to understand their performance properties, to remove performance bottlenecks, determine their maximum capacity, and recognize performance-critical components.

In [4], the research provided an extensible framework, called Deployment Improvement Framework. This framework's goal is to improve the quality of service QoS of a software-intensive system. The framework determined the best deployment of software components onto hardware hosts according to multiple, possibly conflicting QoS dimensions. The design of the framework model and algorithms allows for arbitrary specification of new QoS dimensions and their improvement. They are also developing the capability to automatically determine the best algorithm(s) based on system characteristics and execution profile.

In [5], the study described dependency management between component during dynamic reconfiguration by analysing and managing static and dynamic dependencies. The proposed work provided consistent reconfiguration of distributed systems and handled nested dependencies. However, the study didn't handle data dependencies and or its effect during dynamic reconfiguration.

Availability is considered to be one of the most important criteria that affect the usefulness and efficiency of a distributed system. It depends on how the system components are deployed on the available hosts. If the components that have high level of communication are located on the same host, the availability will definitely be higher given that all the components are working properly.

In an attempt to increase availability in distributed and mobile environments which can be decreased due to network connectivity losses, the research in [6] proposed an algorithm called Avala to improve availability in component-based distributed systems via redeployment. The proposed study supported runtime redeployment to increase the software system's availability by monitoring the system, estimating its redeployment architecture, and affecting the estimated redeployment architecture. The approach proposed reduced the overall interaction latency in the system and considered to provide a fast-approximate solution compared to the other previous exponentially complex solutions. However, the proposed work did not deal with constraints in the solution space neither study the dependency relations between components.

In [7], the proposed study has presented an extension of the Avala model [6] called E-Avala by providing a dependency relation between the components and implementing replication mechanism. However, issues such as dealing with functional consistency of components and including additional system parameters such components structure (e.g. hierarchical representations of the components must be properly addressed.

In many approaches, an agent technology has been presented to address issues in distributed system information retrieval and in distributed system integration.

In [8], an agent-based monitor provided dynamic mechanism for redeploying or replicating components for the approaches presented in [6] Avala and [7] E-Avala. As mentioned earlier, these two approaches aimed to improve availability in component-based distributed systems via redeployment and replication. Agents make the decision of whether redeploying or replicating is more appropriate based on the interaction between the system and components.

In terms of performance, balancing load is considered one of the most important approaches to achieve better performance in distributed systems.

In [9], the research has presented two algorithms which aimed to balance load in distributed systems. The algorithms are simple, adaptive and based on the hierarchical structure. They

worked in two levels of groups and nodes. The algorithms started by distributing the arrival loads on the groups and nodes with specific biases according to each node and group current load state. Then they transmit arrival loads by selecting the group and node with minimum load state. The proposed algorithms have presented an improvement - especially when the system is not fully overloaded- in terms of drop rate, throughput and response time for various numbers of nodes and tasks.

A clustered algorithm was presented to provide dynamic load balancing in distributed systems with their diversity of serving capabilities advantage [10]. Each cluster has three nodes and a supporting node. The load balancer has a queue in which the load of each cluster node load is stored. Nodes are decided whether they are heavily loaded or not by using threshold value. If a node is overloaded, the load balancer is responsible to identify the most appropriate node to transfer the overload to it. The proposed approach reduced communication cost and complexity. However, it only works for a cluster with three nodes.

The proposed method in [11] assumed n nodes. Each node has a backup node which aimed to identify the underloaded nodes and transfer the unserved tasks to them. It is also responsible for each node tasks in case of a failure in its node. The system preserves a load balanced state due to transferring extra load from overloaded nodes to under-loaded nodes. In this work, the overall performance of the system is enhanced due to minimized response time and the nodes of the systems are not overloaded for maximum time.

3. Factors Influencing Component Performance

In component-based distributed systems the performance of reusable software components is difficult to be determined because there are other factors influencing the provided performance like the context the component is deployed into beside the component implementation. The main factors that have impact on software components performance are mentioned below:

(i) Component implementation: Functionality specified by an interface can be implemented by component developers in different ways. Two components running on the same resources and given the same inputs can provide the same service functionally but exhibit different execution times.

(ii) Required services: If a component service X invokes required service Y, then the execution time of Y is added up to the execution time of X. each component execution time is calculated by adding up the execution time of all required services.

(iii) Deployment platform: software components may be deployed to different platforms by software architects. In a deployment platform, many software layers can be included such as component container, virtual machine, operating system, etc. and hardware such as processor, storage device, network.

(iv) Usage profile: component services can be invoked by different clients with different input parameters. Values of input parameters can change the execution time of services. In addition to input parameters, component can receive other parameters from the required service which can also impact the total execution time of a service. Furthermore, components can have an internal state from an initialization or former executions, which changes execution times.

(v) Resource contention: Typically, software component does not execute as a single process in isolation on a given platform. The induced waiting times for accessing limited resources add up to the execution time of a software component [1].

As mentioned above, deployment platform is one of the factors influencing component performance whether software or hardware as processor. In this research will include processor as an attempt to improve the performance of the proposed approach.

4. The Proposed Approach

The previous approaches do not take a processor factor into account when ranking the hosts in the system. We consider this to be a very important issue in distributed systems as mentioned previously because in some situations it is difficult to deploy certain components in certain hosts if they have less processing capabilities or if hosts have unequal distribution of workload. In our approach, we employ the notion CPU speed in host ranking formulas (1) and (2) which is defined below. The initial ranking of hardware nodes is performed by calculating, for each hardware node i, the Initial Host Rank (IHRi,) as follows:

$$\boldsymbol{IHR_i}=\sum_{j=1}^{k}\boldsymbol{REL}(\boldsymbol{h_i},\boldsymbol{h_j})+\text{MEM}\ (\boldsymbol{h_i})+\ \text{CPU}\ (\boldsymbol{h_i}) \quad (1)$$

where $\boldsymbol{h_1}, \boldsymbol{h_2}, \ldots, \boldsymbol{h_k}$ $(1 \leq k)$ stands for hosts, $\sum_{j=1}^{k}\boldsymbol{REL}(\boldsymbol{h_i}, \boldsymbol{h_j})$ *is* reliability between $\boldsymbol{h_i}$ *and* $\boldsymbol{h_j}$, $\text{MEM}(\boldsymbol{h_i})$ *is the memory of* $\boldsymbol{h_i}$ *, and CPU(*$\boldsymbol{h_i}$*)* is the processor speed of $\boldsymbol{h_i}$ *.* The next host to be selected is the one with the highest memory capacity, highest CPU speed and highest link quality (i.e., highest value of reliability) with the host(s) already selected. Host Rank (HR) is calculated as follows:

$$\boldsymbol{HR_i}(\boldsymbol{h_i})=\sum_{j=1}^{m}\boldsymbol{REL}(\boldsymbol{h_i}, \text{MH}\ (\boldsymbol{h_j}))+\text{MEM}\ (\boldsymbol{h_i}) + \text{CPU}\ (\boldsymbol{h_i}) \quad (2)$$

Where *m* is the number of hosts that are already selected, $\boldsymbol{REL}\ (\boldsymbol{h_i}, \boldsymbol{MH}\ (\boldsymbol{h_i}))$ is a function that determines the reliability between selected host h and hosts of mapped components.

5. Experimental Setup, Results and Discussion

After we experimented and implemented the algorithm of our approach, we evaluated its performance and compared it with the existing previous approach based on the criteria of delay time, and the number of processes. This subsection introduces results discussion and evaluation of our approach as a contribution to this paper. A methodological description of the process is introduced, as well.

- Evaluating the performance of our proposed approach, different scenarios of transactions on different system configuration parameters were conducted. Then, the performance average of the proposed approach was found.
- Results study of CPU effect on system performance against our approach and previous approach for different scenarios of transactions on different system configuration parameters was conducted. Then, we found the maximum average performance and compared it in the two approaches.

Each step mentioned above is presented in a separate subsection in this paper.

5.1. Experimental Parameters and Setup

In order to test the proposed approach and compare it with other approaches, we conducted a series of simulation experiments. The experimental environment (CPU: Intel Core™i5 M450/2.40 GHz, RAM: 4.0 GB, Windows 7, and Java Programming language with Eclipse KEPLER, and Arena simulation.) is explained below:

First, we implemented the proposed approach and analysis the performance of the proposed approach against the previous approach.

Then the effect of adding CPU factor has been studied in our approach with the existing previous approach and compared their results.

The experiments parameters were categorized according to the CPU speed into three classes: low, mid and high sampled as A, B, and C respectively.

To measure the performance of our proposed approach and compare it with the current developed approach which has been developed under a similar environment and criteria, we used the delay time and number of processes.

We evaluate the efficiency and feasibility of the proposed approach by conducting a series of experiments based on various configuration parameters of the proposed approach and previous approach with different level of CPU speed.

5.2. Experimental Results

In this subsection, we present the evaluation for the performance of our approach against CPU speed of different scenarios of transactions. First, we present the study results and analysis performance for our approach and previous approach against different CPU speed. Then, we find the best performance average of which approach that has the best performance. Finally, we assess the effect of our approach on the previous approach. In each class, we have taken the average results for 20 different randomly generated architecture configurations by using the parameters presented below in Table 1.

Table 1. System Input Parameters

Input Parameter	Value
No of components	100
No of hosts	20
Min component memory (in KB)	2
Max component memory (in KB)	8
Min host memory (in KB)	50
Max host memory (in KB)	100
Min host CPU speed (in kHz)	1.1
Max host CPU speed (in kHz)	4.1
Level of dependency	3
Min component frequency (in events/s)	0
Max component frequency (in events/s)	10
Min host reliability	0
Max host reliability	1
Min component event size (in KB)	1
Max component event size (in KB)	10
Min host bandwidth (in KB/S)	30
Max host bandwidth (in KB/S)	1000

5.2.1. Class A (Low)

In computer systems, CPU speed varies which results in a different level of performance. In this class, CPU speed is ranging between 1.1 and 2.1 kHz.

Figures 1 and 2 show the performance rates of our proposed approach against all configuration parameters in terms of delay time and number of processes.

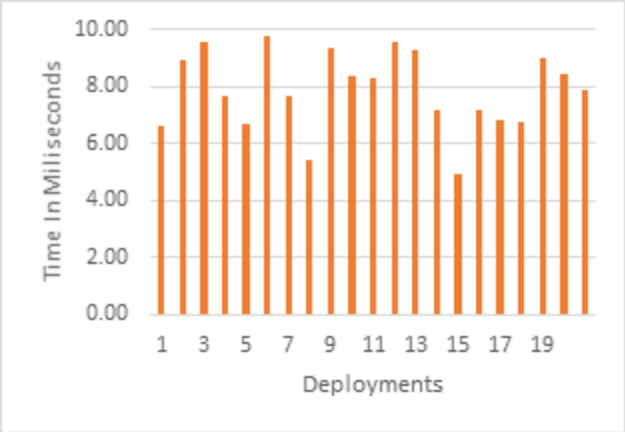

Figure 1. Delay Time of Proposed Approach in Class A

From Figure 1, we can see that our proposed approach gives the average of delay time with value of 7.88 millisecond (ms).

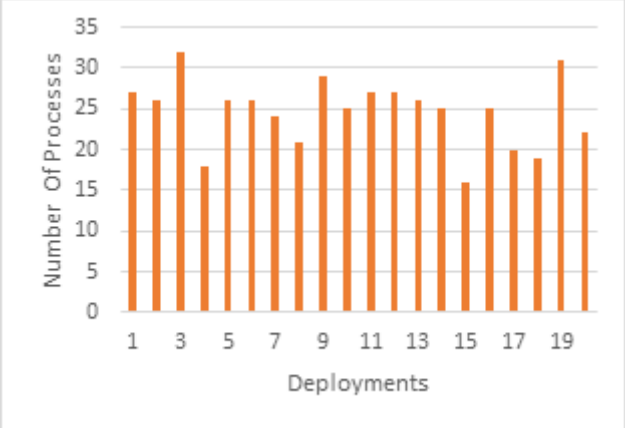

Figure 2. Number of Processes in Class A

As shown by figure 2 above, the maximum number of processes produced by our approach is 32.

5.2.2. Class B (Mid)

Performance was examined with 20 tests with CPU speed ranging between 2.1 and 3.5 kHz. Figures 3 and 4 show the performance rates of our proposed approach in terms of delay time and number of processes.

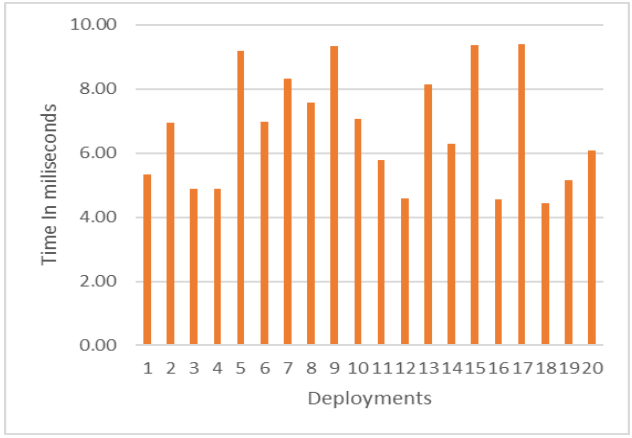

Figure 3. Delay Time of Proposed Approach in Class B

From Figure 3, we can see that our proposed approach gives the average of delay time with value of 6.72 millisecond (ms).

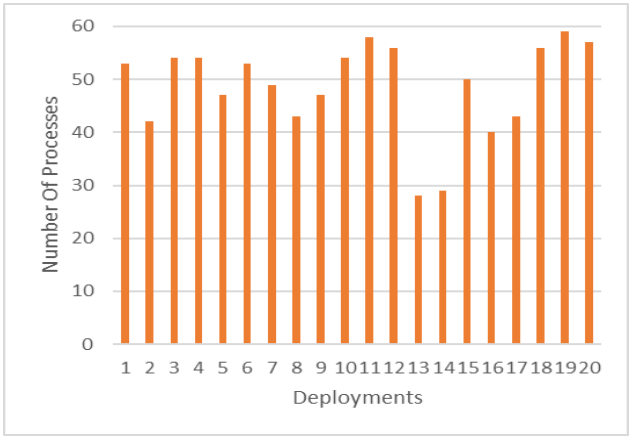

Figure 4. Number of Processes in Class B

As shown by figure 4 above, our approach's maximum number of processes is 57.

5.2.3. Class C (High)

In this class performance was examined with 20 tests of CPU speed between 3.5 to 4.1 KHz. Figures 5 and 6 show the performance rates of our proposed approach against all possible CPU values.

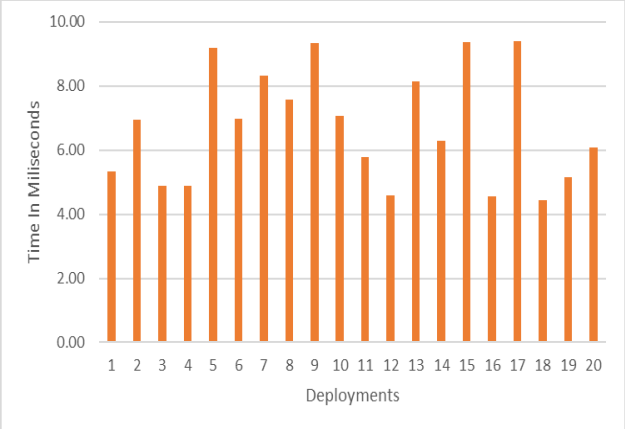

Figure 5. Delay Time of Proposed Approach in Class C

From Figure 5, we can see that the proposed approach gives the average of delay time with value of 6.72 millisecond (ms).

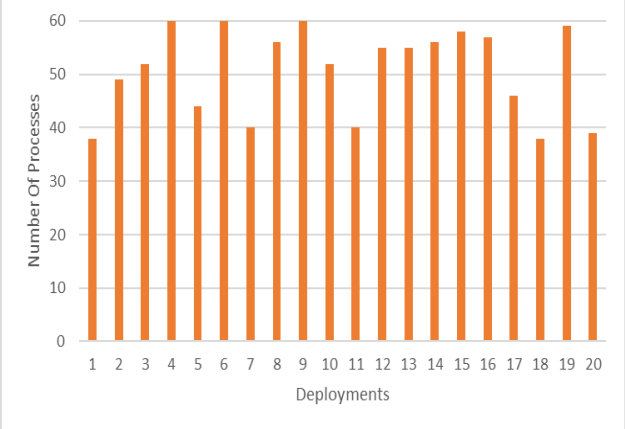

Figure 6. Number of Processes in Class C

As shown by figure 6 above, the maximum number of processes given by our approach is 60.

5.3. Comparative Results

In this subsection, we present an evaluation of the CPU effects of all classes on system performance against our approach and previous approach for different configuration parameters.

Figures 7 and 8 shows the maximum of maximum values of delay time and number of processes for our approach and previous approach against the effects of sixty tests categorized as [low-speed CPU, medium speed CPU, and high-speed CPU].

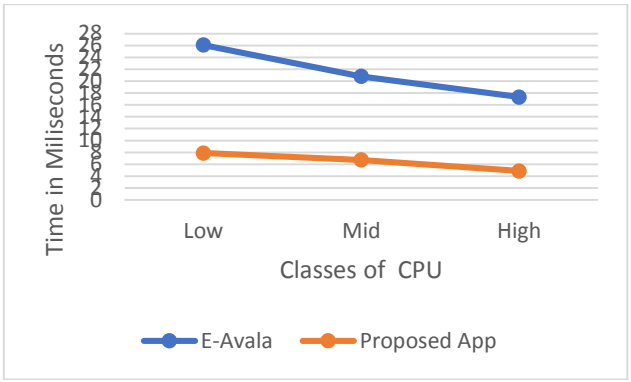

Figure 7. Delay Time Enhancement of Our Approach With E-Avala

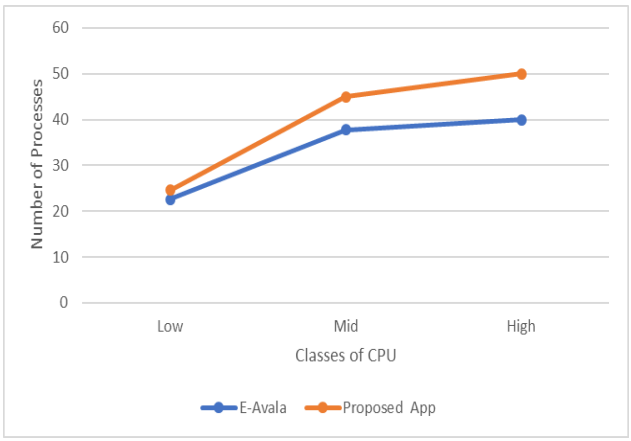

Figure 8. Number of Processes Enhancement of Our Approach With E-Avala

Inviting comparisons with CPU speed effect in terms of the best performance as figures 7 and 8 shows, our approach has given the lowest delay time and the highest number of processes value under all classes of CPU speed, and under various configuration parameters. As can be seen from the figures above and its values, the proposed approach is more applicable in all three classes. It shows improvement in terms of delay time in classes A, B, and C with improvement values of 79%, 81%, and 84% respectively. In terms of number of processes, the proposed approach provides higher number with improvement level of 9%, 19%, and 25% in classes A, B, and C respectively. This improvement is due to considering the processing capabilities in addition to reliability and memory size of hosts when ranking them as best hosts to redeploy or replicate software components to them. The equations of ranking hosts examine the processing capabilities of each host in which they are ranked to provide the best host it terms of memory size and processing power. The systems deployed using the proposed approach provide less delay time so that number of processes executed are more. As can be seen, the improvement level is increasing and has the highest values in class C which is axiomatic due to

increasing the processing capabilities in this class. The proposed approach is predicted to provide better performance with more complex systems with a greater number of hosts and components.

6. Conclusions

Aval and E-Avala approaches have been proposed to enhance availability in component-based distributed systems. However, they have ignored the performance factor. In this paper, we have presented our approach as an extension of E-Avala to enhance host ranking by adding a new factor (CPU speed) in an attempt to improve system performance. The experiments showed that the proposed approach had better performance in terms of delay time and number of processes.

References

[1] Duncan S. Component software: Beyond object-oriented programming. Software Quality Professional. 2003 Sep 1; 5(4):42.

[2] Heineman GT, Councill WT. Component-based software engineering. Putting the pieces together, addison-westley. 2001 May:5.

[3] Sommerville I. Software engineering 9th Edition. ISBN-10. 2011;137035152.

[4] Malek S, Medvidovic N, Mikic-Rakic M. An extensible framework for improving a distributed software system's deployment architecture. IEEE Transactions on Software Engineering. 2011 Jan 6; 38(1):73-100.

[5] Chen X. Dependence management for dynamic reconfiguration of component-based distributed systems. In Proceedings 17th IEEE International Conference on Automated Software Engineering, 2002 (pp. 279-284). IEEE.

[6] Mikic-Rakic M, Malek S, Medvidovic N. Improving availability in large, distributed component-based systems via redeployment. In International Working Conference on Component Deployment 2005 Nov 28 (pp. 83-98). Springer, Berlin, Heidelberg.

[7] Al-Areqi S, Hudaib A, Obeid N. Improving Availability in Distributed Component-Based Systems via Replication. In New Challenges for Intelligent Information and Database Systems 2011 (pp. 43-52). Springer, Berlin, Heidelberg.

[8] Obeid N, Al-Areqi S. Using Agents for Dynamic Components Redeployment and Replication in Distributed Systems. In Contemporary Challenges and Solutions in Applied Artificial Intelligence 2013 (pp. 19-25). Springer, Heidelberg.

[9] Barazandeh I, Mortazavi SS, Rahmani AM. Two new biasing load balancing algorithms in distributed systems. In2009 First Asian Himalayas International Conference on Internet 2009 Nov 3 (pp. 1-5). IEEE.

[10] Rajani S, Garg N. A clustered approach for load balancing in distributed systems. SSRG International Journal of Mobile Computing & Application (SSRG-IJMCA). 2015; 2(1).

[11] Jadhav R, Kamlapur S, Priyadarshini I. Performance evaluation in distributed system using dynamic load balancing. Performance Evaluation. 2012 Feb; 2(7).

[12] Roy N, Dubey A, Gokhale A, Dowdy L. A capacity planning process for performance assurance of component-based distributed systems. ACM SIGSOFT Software Engineering Notes. 2011 Sep 30; 36(5):41-.

[13] Kaur U, Sharma S. Performance Evaluation using Various Models in Distributed Component based Systems. International Journal of Computer Applications. 2014 Jan 1; 98(1).

[14] Brosig F, Huber N, Kounev S. Automated extraction of architecture-level performance models of distributed component-based systems. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering 2011 Nov 6 (pp. 183-192). IEEE Computer Society.

[15] Kaur N, Singh A. Component complexity metrics: A survey. International Journal of Advanced Research in Computer Science and Software Engineering. 2013 Jun; 3(6).