Intelligent Automation & Soft Computing
DOI:10.32604/iasc.2020.013835
Article

Improving Availability in Component-Based Distributed Systems

Fahd N. Al-Wesabi*

Department of Computer Science, King Khalid University, KSA & Faculty of Computer and IT, Sana'a University, Sana'a, Yemen
*Corresponding Author: Fahd N. Al-Wesabi. Email: Falwesabi@kku.edu.sa
Received: 23 August 2020; Accepted: 08 October 2020

Abstract: Assuring high availability is an important factor to develop component-based systems, particularly when different workloads and configurations are common. Several methods have been proposed in the literature to redeploy and replicate software components to find the best deployment architecture that guarantees high availability of component-based systems. In this paper, an extended method has been proposed to improve the availability of component-based systems by adding new CPU factors. The proposed method has been implemented by a self-developed program and using a java programming language with Eclipse KEPLER. Several simulations and experiment scenarios have been performed to evaluate the availability with related effectiveness and efficiency of the proposed method. Simulation and experiment results prove the availability, effectiveness, and efficiency of the proposed method using the core five factors and different configuration settings of the component-based system environment. Simulation and experiments result also show the applicability of the proposed method in a different environment and various parameters. The proposed method has been compared with another baseline approach. Comparison results show the proposed method outperforms the baseline approach in terms of availability with related effectiveness and efficiency features with a higher rate of availability. The improvement level of availability accomplished by the approach ranges from 1% to 17% based on the comparison factor and environment.

Keywords: Distributed systems; component-based distributed systems; redeployment; availability

1 Introduction

Nowadays, many new business systems are being developed by configuring off-the-shelf systems. However, some off-the-shelf systems cannot meet all company's requirements [1]. Therefore, specially designed software must be developed. When developing new enterprise systems with customized software, component-based software engineering is considered as an effective reuse-oriented development methodology.

Component-based software engineering (CBSE) is the successor of object-oriented software development [2,3], and it 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 required interfaces [2].

In today's world, distributed systems have been replaced by their central ones. This is fairly understandable because distributed systems have higher availability, reliability, and incremental growth [4]. In the case of component-based distributed systems, software components are distributed across many different hosts and locations. Therefore, a decision should be made to locate the platforms in which the components will be deployed [5]. When making such a decision, some issues that should be considered, which are:

Component requirements: these requirements are related to the required hardware and software to run a component-based distributed system. During the design phase, some components may require certain hardware architecture or software systems, so these components should be deployed on a platform that provides their hardware requirements and software support.

Availability requirements of the system: To satisfy the high availability requirement of some systems, components should be deployed on more than one platform. This means that a substitution implementation of the component is available if a platform failure occurs.

Component communications requirements: If the communication level is high between some components, they should be deployed on the same platform or physically close platforms. This reduces communications potential when sending and receiving services between components [2].

Performance requirements: Components should be deployed on platforms with higher processing capabilities to guarantee higher performance.

In the literature, improving availability in distributed systems has been offered based on some researchers' thoughts. Nowadays, some solutions and approaches have been proposed by researchers to manage redeploying components according to dependency relations between them.

The rest of the paper is organized as follows: In Section 2, the author explains the existing works done so far. Section 3 presents the proposed approach. Section 4 describes the simulation, implementation, and discussion. The results and comparisons are provided in Section 5, and finally the author concludes the article in Section 6.

2 Literature Review

Over the years, many solutions and approaches have been proposed to evaluate the performance of component-based software systems such as resource utilization, throughput, and response time [6]. Some of these approaches were aimed to predict performance and others to measure performance [7]. The former ones' goal was to avoid performance problems in the implementation phase of system development by analyzing the expected performance of component-based software in the design phase. These problems could lead to redesigning the component-based software architecture with substantial costs if not avoided. The latter ones were aimed to analyze the performance of implemented and running component-based systems to understand their performance properties, remove performance bottlenecks, determine their maximum capacity, and recognize performance-critical components.

In Reference [8], the authors proposed an extensible framework, called the deployment improvement framework. The proposed framework aimed 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, and QoS dimensions. The design of the proposed framework and algorithms provides an arbitrary specification of new QoS dimensions and their improvement. They also provide the capability to automatically determine the best algorithm(s) based on system characteristics and execution profiles.

In Reference [9], the study described dependency management between system components during dynamic reconfiguration by analyzing and managing static and dynamic dependencies. The proposed study provides consistent reconfiguration of distributed systems and handled nested dependencies.

However, the proposed study did not handle data dependencies and/or its effect during dynamic reconfiguration.

Availability is one of the most important measures that affect the usefulness and efficiency of a distributed system [10]. It depends on how the system's components are deployed on the available hosts. If the components with a high level of communication are located on the same host, the availability will 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 Reference [11], proposed an algorithm called Avala to improve availability in component-based distributed systems via redeployment. The proposed algorithm supports runtime redeployment to increase the software system's availability by monitoring the system, estimating its redeployment architecture, and affecting the estimated redeployment architecture. The proposed algorithm reduced the overall interaction latency and provides a considerable fast-approximate solution compared to the other previous exponentially complex solutions. However, the proposed algorithm did not deal with the constraints in the solution space neither study the dependency relations between components.

In Reference [12], Avala algorithm has been extended and improved to develop E-Avala model to improve availability in distributed systems. Improving E-Avala model depends on providing a dependency relation between the components and implementing a replication mechanism. However, issues such as dealing with functional consistency of components, and including additional system parameters such as components structure, like hierarchical representations of the components, should have been properly addressed.

In many approaches, agent technology has been presented to address some issues in the distributed system i.e. information retrieval, and component integration [13]. In Reference [14], an agent-based solution has been proposed to provide a dynamic mechanism for redeploying or replicating components for both Avala and E-Avala algorithms presented in References [11,12]. As mentioned earlier, these two algorithms aimed to improve availability in component-based distributed systems via redeployment and replication features. However, the proposed agent-based solution decided whether redeploying or replicating is more appropriate or not based on the interaction between the system and components.

Performance and load balancing are considered two of the most important approaches to achieve better performance in distributed systems [15].

In Reference [16], two algorithms have been presented to improve load balancing in distributed systems. The proposed algorithms are simple, adaptive, and based on the hierarchical structure. Both proposed algorithms can run on 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 current load state of the group. Then, they transmitted arrival loads by selecting the group and node with minimum load state. The proposed algorithms present an improvement in terms of drop rate, throughput, and response time for various numbers of nodes and tasks, especially when the system was not fully overloaded.

In Reference [17], a clustered algorithm has been presented to provide dynamic load balancing in distributed systems with their diversity of serving capabilities advantage. Each cluster had three core nodes and a supporting node. The load balancer had a queue in which a load of each cluster's node was stored. Nodes were decided whether they were heavily loaded or not by using a threshold value. If a node was overloaded, the load balancer responsible to identify the most appropriate node to transfer the overload to it. The proposed algorithm reduced communication cost and complexity. However, it is only applicable to a cluster with three nodes.

In Reference [18], authors present a new method to identify the underloaded nodes in distributed systems and transfer the unserved tasks to them. The proposed method assumed n nodes; each node had a

backup node that responsible for each node task in case of a failure in its task. The system preserved 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 was enhanced as the response time was minimized and the nodes of the systems were not overloaded for maximum time.

In Reference [19], a new method has been proposed called MAQ-PRO to assure high performance and availability in multi-tiered component-based applications by reducing an operational cost, by using fewer resources, and revenues increase, by serving more clients. The proposed method involved an algorithm aimed to replicate and allocate components in multi-tiered applications and techniques to develop profile-based analytical models.

In Reference [20], the paper reviewed recent approaches related to the performance of component-based distributed systems with their drawbacks and benefits. As concluded from these researches, the performance models' parameters, such as throughput and responsiveness, were machine centered. Therefore, the research proposes to include the organization parameters in which the end-user perspective of system performance was not ignored or underestimated to satisfy their requirements.

Predicting a system's performance is essential to ensure that the system meets its performance requirements under different configurations and workloads. However, building such models manually takes effort and time. In Reference [21], an automated method has been presented to extract architecture-level performance models of component-based systems by using data collected at run time. The proposed method has been validated in a case study and only 10–20% of error margin resulted between the performance prediction and measurements on the real system.

3 Proposed Approach

In component-based distributed systems (CBDS), components present facilitating characteristics to develop complex systems. Availability is one of the importance of these characteristics that affect the usefulness and efficiency of the system. Availability is defined as the ratio of the number of completed interactions in the system to the total number of attempted interactions. The proposed approach aims to improve availability in component-based distributed systems by adding a new system parameter which is host processing capability as an additional factor to rank the best hosts to redeploy components to them accordingly. Availability A, is calculated by Eq. (1).

$$A = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(freq\left(c_i, c_j\right) * rel\left(f\left(c_i\right), f\left(c_j\right)\right)\right)}{\sum_{i=1}^{n}\sum_{j=1}^{n} freq\left(c_i, c_j\right)} \tag{1}$$

The Avala algorithm has been proposed to improve availability in component-based distributed systems via redeployment. The proposed algorithm supports runtime redeployment to increase the software system's availability by monitoring the system, estimating its redeployment architecture, and affecting the estimated redeployment architecture. However, the E-Avala algorithm has been extended and presented by providing a dependency relation between the components and implementing a replication mechanism. On the other hand, the improved approach offers an extension for both Avala and E-Avala algorithms to develop a new approach, and it is briefly explained as follows:

The Avala algorithm ranks the initial host according to Eq. (2) given below:

$$IHR_i = \sum_{j=1}^{k} REL\ (h1, hj) + ME\ (hi) \tag{2}$$

where h_1, h_2, ..., h_k(i ≤ k) stands for hosts, REL denotes the reliability between two hosts $h_i$and$_j$ and MEM (h_i**)** denotes the memory of h_i

While initials software components are ranked according to Eq. (3).

$$ICR_i = d* \sum_{j=1}^{n} FREQ\ (Ci, Cj) + \frac{E}{MEM\ (Ci)} \tag{3}$$

where C_1 , … , C_n (i ≤ n) stands for components, *d indicates host memory*, $FREQ(C_i, C_j)$ denotes the frequency between components C_i and C_j, E indicates event size of interaction between C_i and C_j , and $MEM(C_i)$ denotes the memory of C_i. The next software component to be assigned to h, is the one with the smallest memory requirement and which would maximally contribute to the availability function if it was placed on h. The component rank (CR) is calculated by using Eq. (4).

$$CR(C_i, h) = D_1(C_i, h, n) + D_2(C_i, h) \tag{4}$$

where $D_1(C_i, h, n) = d* \sum_{j=1}^{n} FREQ(C_i, MC_j)*\text{REL}(\text{h}, f(MC_j))$, $D_2(C_i, h)=\frac{E}{MEM(C_i)}$, MC_j indicates mapped components $C_j, f(MC_j)$ is a function that determines the hosts of mapped components, and REL (h, $f(MC_j)$) is a function that determines the reliability between selected host h, and hosts of mapped components.

Next hosts (HR) are ranked using Eq. (5).

$$HR_i(h_i) = \sum_{j=1}^{m} REL(h_i,\ \text{MH}\ (h_i))\ + \text{MEM}\ (h_i) \tag{5}$$

where m denotes hosts that have been selected before.

The improved approach also uses the same equations of initial ranking and distribution of the E-Avala. However, it extends the Avala by presenting two additional functions: RCR, which computes replicate component rank (RCR) without considering data consistency, and Consis-RCR, which computes RCR with consideration for data consistency as shown in Eq. (6) and Eq. (7).

$$RCR(C_i, h, n, nm) = D_3(C_i, h, n) + D_1(C_i, h, nm) \tag{6}$$

where $D_3(C_i, h, n) = \sum_{p=1}^{n} Depend(C_p, C_i) + \frac{1 + 2E}{MEM(C_i)}$

$$Consis - RCR(C_i, h, n, nm) = D_3(C_i, h, n)*(1 - Consis(C_i) + D_1(C_i, h, nm)) \tag{7}$$

Where h is the selected host, l is the level of dependency for system configuration determined by the designer, and nm is the number of mapped components (i.e., already been assigned to selected hosts),

E-Avala considered checking data consistency for a C_i if needed as shown in Eq. (8).

$$Consistency(C_i) = \{1\ if\ C_i\ doesn't\ require\ data\ consistency\ 0 \tag{8}$$

Otherwise, E-Avala also represented a new notion dependency (C_i, C_j) as shown in Eq. (9).

$$\text{Dependency}(C_i, C_j) = \{1\ if\ C_i\ depends\ on\ C_j, -1\ if\ C_j\ depends\ on\ C_i \tag{9}$$

E-Avala runs by comparing the CR components to be redeployed and RCR components to be replicated. One of both CR or RCR with the highest value will be selected in addition to satisfying the constraints of memory, Loc, and Colloc with the current host h and the assigned components. All of this process will be repeated until the host is filled.

The proposed algorithm is illustrated in Alg. 1.

Algorithm 1: Improved E-Avala algorithm

```
Proposed_approach(hosts, comps)
{
numOfMappedComps= 0
  unmappedComps= comps
  h=hosts with max(PerIHR)
  unMappedHosts= hosts-h
  numOfMappedHosts= 1
  RC= component with max (initReplicateCompRank)
  DC= component with max (initDeployeCompRank)
  while (numOfMappedComps < numOfComps and numOfMappedHosts < numOfHosts
  and h <> -1) and
  (h.memory > c.memory and numOfMappedComps < numOfComps and RC<> -1 and DC <> -1 )
     If RC >= DC
        unMappedComps= unMappedComps-RC
        numOfMappedComps= numOfMappedComps+1
        h.memory= h.memory-RC.memory
        replication= replicate (c to h)
        RC= nextReplicateComp ( comps, nnmappedComps, h)
    Else
         unMappedComps= unMappedComps-DC
         numOfMappedComps= numOfMappedComps+1
         h.memory= h.memory-DC.memory
         deployment= deploye ( c to h )
         DC= nextReplicateComp ( comps, nnmappedComps, h)
         h= PerHR (unMappedHosts)
         unMappedHosts= hosts-h
         numOfMappedHosts= numOfMappedHosts+1
   If   numOfMappedComps= numOfComps
     return success
  Else
  No deployment and replication was found
}
```

Both Avala and E-Avala algorithms did not address a processor factor when ranking the hosts in the system. Authors consider this to be a very important issue in distributed systems as mentioned previously because in many cases it is difficult to deploy certain components in certain hosts if they have less processing capabilities. On the other hand if hosts have unequal workload distribution. In the proposed, I employ the notion CPU speed in host ranking as presented in Eq. (2) and Eq. (5), defined above. The initial ranking of hardware nodes is performed by calculating, for each hardware node i, the initial host rank (*IHRi*) as shown in Eq. (10).

$$IHR_i = \sum_{j=1}^{k} REL\ (hi,\ hj) + MEM\ (hi) + CPU\ (hi) \tag{10}$$

where, $h_1, h_2, \ldots, h_k\ (1 \leq \mathrm{k})$ stand for hosts, $\sum_{j=1}^{k} REL\ (hi,\ hj)$ is reliability between h_i and h_j, MEM(h_i) is the memory of h_i, and CPU(h_i) is the processor speed of 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 shown in Eq. (11).

$$HR_i(hi) = \sum_{j=1}^{m} REL\ (hi,\ MH(hj)) + MEM\ (hi) + CPU\ (hi) \tag{11}$$

where, m is the number of hosts that are already selected, and *REL* (*hi*, MH (*hi*)) is a function that determines the reliability between selected host h and hosts of mapped components.

4 Simulation and Implementation

To evaluate the performance and efficiency of the proposed approach, simulation and implementation have been performed using Java programming language with Eclipse KEPLER.

4.1 Simulation Environment

To study the effect of the proposed approach on availability and compare it with the previous approach, the author conducted a series of experiments. The experimental environment was: CPU: Intel Core™i5 M450/ 2.40 GHz, RAM: 4.0 GB, Windows 7, and Java Programming language with Eclipse KEPLER.

4.2 Performance Metrics

Several factors were used to compare availability values resulted from both approaches. These factors are:

Host memory: Represents the storage of the host.

Event size: Captures the average size of data exchanged between a pair of components.

Dependency level: Refers to the case when one component depends on other components.

Processing capabilities: Represents the CPU speed of the hosts in KHz for calculation simplicity.

The number of components.

The first three factors were chosen due to their direct effect on availability rates as shown in Eq. (1) above, and they were also used in the previous approach, whereas the processing capabilities factor was chosen to study the impact of adding this new factor on the availability levels. Some components were used to evaluate the time cost added to the enhanced algorithm of the proposed approach.

4.3 Implementation

To evaluate the effect of the proposed approach on availability values, two steps are accomplished: First, E-Avala algorithm has been re-implemented, and several simulations and experiment scenarios have been performed using specific configuration parameters to measure the values of availability.

Then, an improved algorithm of the proposed approach is implemented, c.

4.4 Experimental Setup

Several experiments scenarios were conducted using the five configuration parameters mentioned above. In each experiment, one parameter is changed and all other parameters are fixed. I have taken the average results for 20 different randomly generated architecture configurations by using the parameters shown in Tab. 1 below.

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

(Continued)

Table 1 (continued).

Input parameter	Value
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 Results Comparison and Discussion

In this section, the author presents several simulations, experiments, and comparison scenarios to study and compare the improvement values of availability for the proposed approach against the baseline E-Avala algorithm presented in Reference [12].

5.1 Comparison Mechanism

The E-Avala model was aimed to increase the availability of DS during disconnection among hosts. It proposed a replication mechanism and dependency relation between components. An agent-based monitor was also proposed to support dynamic component redeployment and component replication mechanisms. Nevertheless, the proposed approach aims to increase availability by adding a new system parameter, which is host processing capability, as an additional factor to rank the best hosts to redeploy components to them accordingly. The effect of adding the processing parameter on availability rates have been studied and compared with availability rates in the E-Avala approach.

5.2 Host Memory Effect Against Availability

In this scenario, all the input parameters are fixed except for the host memory (HM). Tab. 2 shows the availability of host memory values that range from 100 to 500 for both approaches.

Table 2: Host memory effect against availability

Host memory	100	200	300	400
E-Avala approach	0.68	0.91	0.92	0.99
The proposed approach	0.74	0.97	0.97	1.00
Improvement level	8%	6%	5%	1%

Tab. 2 above shows average availability values of E-Avala approach and the approach using different values of host memory. The table also shows the improvement level of availability in the approach against E-Avala approach which was 8% in the best situation.

Fig. 1 shows the average improvement of availability by the proposed approach over the E-Avala, which was 8% in the best situation. But when the host memory was high, the availability presents the same high level since it utilizes the maximum reliability for interactions between components residing on the same host. As results show, the system presents higher availability when having both large memory and high-speed processor. Users' requests are handled faster and resources become available again for new requests.

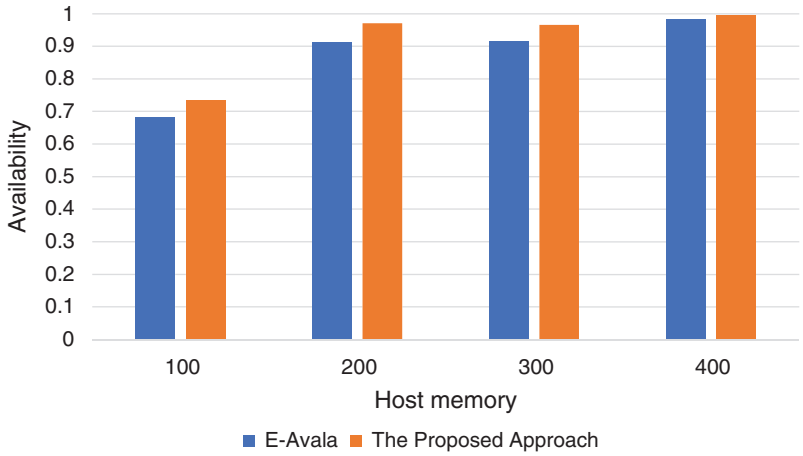

Figure 1: Host memory effect against availability

5.3 Event Size Effect Against Availability

In this scenario, the author changed the event size value and fixed the other input parameters. The comparison results presented in tabular form in Tab. 3 and graphically illustrated in Fig. 2.

Table 3: Event size effect against availability

Event size	10–200	20–300	50–600
E-Avala approach	0.85	0.77	0.87
The proposed approach	0.88	0.87	0.92
Improvement level	3%	12%	6%

As seen from Tab. 3, different ranges of event sizes were used to test availability values in both approaches. The table shows an improvement level of availability in the approach against E-Avala approach, which is 12% in the best situation.

As shown in Fig. 2 above, the average improvement of availability is over 12%. Event size represents the average size of data exchanged between a pair of components. Therefore, having an algorithm that ranks the best host according to its higher processing capabilities in addition to its memory and reliability will increase the availability levels. Handling the exchanged amount of data between components is handled faster by a faster processor.

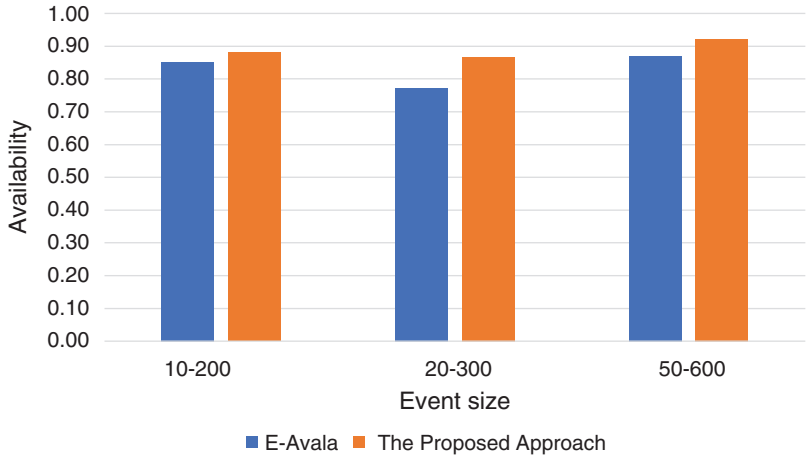

Figure 2: Event size effect against availability

5.4 Dependency Level Effect Against Availability

In this test, the dependency level value was changed and the other input parameters were fixed. The comparison results presented in tabular form in Tab. 4 and graphically illustrated in Fig. 3.

Table 4: Dependency level effect against availability

Dependency level	2	4	6	8	10
E-Avala approach	0.71	0.64	0.59	0.68	0.64
The proposed approach	0.75	0.73	0.69	0.77	0.7
Improvement level	6%	14%	17%	13%	9%

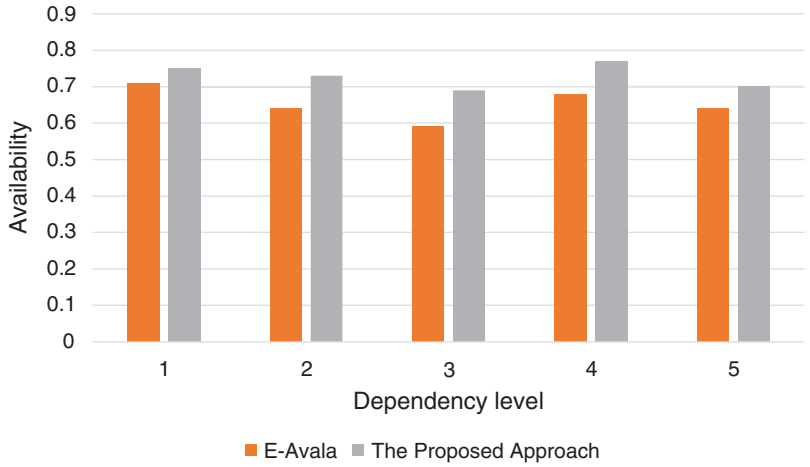

Figure 3: Dependency level effect against availability

As seen from Tab. 4 above, availability values of both approaches were evaluated using different levels of component dependency. The table also shows the improvement level in the approach against the E-Avala approach. The results show 17% improvement in the best situation.

As shown in Fig. 3 above, the average improvement of availability by the proposed approach over the E-Avala was 17% in the best situation. This improvement can be justified due to considering the host processing capabilities in ranking the hosts in which the best host with the highest processing capabilities can transfer data between components with faster dependency relation.

5.5 Processing Capabilities Effect Against Availability

In this test, the processing speed value was changed and the other input parameters were fixed, the comparison results presented in tabular form in Tab. 5 and graphically illustrated in Fig. 4.

Table 5: Processing capabilities effect against availability

Processing capabilities	Low	Mid	High
E-Avala approach	0.77	0.75	0.75
The proposed approach	0.81	0.82	0.84
Improvement level	5%	9%	11%

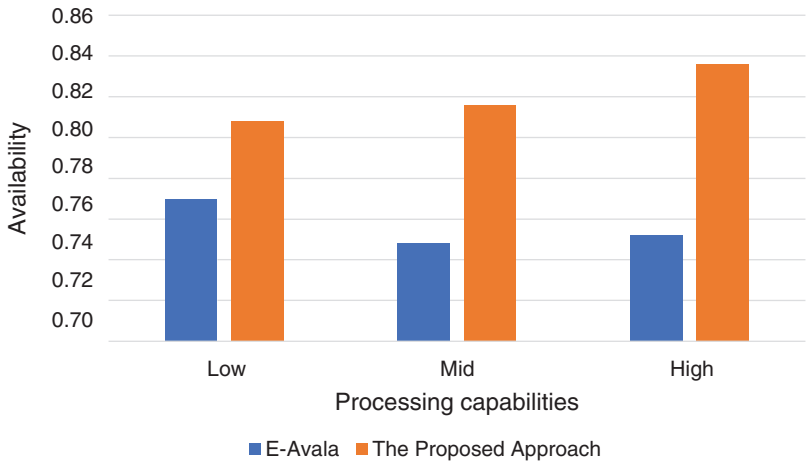

Figure 4: Processing capabilities effect against availability

Tab. 5 above shows the average values of the availability of both approaches using different CPU classes. Improvement levels show higher values of availability using the proposed approach in all classes.

As shown in Fig. 4 above, the proposed approach gives a higher level of availabilities than E-Avala with an average improvement of 8%. Hosts with higher processing capabilities handle requests faster so software and hardware resources become available in less waiting time compared to the E-Avala approach.

5.6 Number of Components Effect Against Time Cost

In this scenario, all the input parameters are fixed except the number of components to find out how much the proposed approach requires additional running time in comparison with E-Avala. The comparison results presented in tabular form in Tab. 6 and graphically illustrated in Fig. 5.

Tab. 6 above shows the time consumed in milliseconds by each approach having a different number of software components. As seen from the improvement level, the approach requires additional time that does not exceed 7% with 300 software components.

Table 6: Time Consumed of E-Avala against the proposed approach

Number of components	100	150	200	250	300
E-Avala approach	844	4341	1478	4314	6002
The proposed approach	860	4432	1569	4620	6400
Improvement level	2%	2%	6%	7%	7%

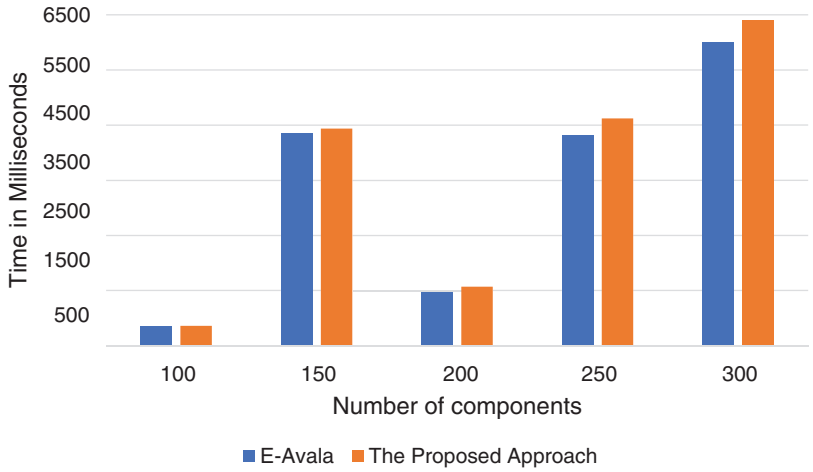

Figure 5: Time consumed by E-Avala *vs.* the proposed approach

Fig. 5 above shows the results of running time for the values of the components 100, 150, 200, 250, 300. The proposed approach requires at most 7% increase in time, which is the time spent in calculating each host CPU speed in the formulas.

6 Conclusions

Distributed systems provide higher availability and higher performance when compared to the central ones. Both factors should be equally considered when developing distributed systems. Many approaches have been proposed to enhance availability in component-based distributed systems. In this paper, the author presents an improved approach as an extension of E-Avala to enhance host ranking by adding a new system factor (CPU speed) to increase system availability. A self-developed program, several simulations, and experimental scenarios have been performed to evaluate the availability feature of the proposed approach. The experiment results show good improvement of the system availability using various configuration parameters. Comparison results of the proposed approach with the baseline E-Avala approach proved the availability and applicability of the proposed approach. For future work, some issues need to be considered such as functional consistency between components and adding additional system parameters.

Funding Statement: The author expresses his appreciation to the Deanship of Scientific Research at King Khalid University for funding this work through Research Groups under Grant Number (R. G. P. 2/55/40 /2019).

Conflicts of Interest: The author declares that he has no conflicts of interest to report regarding the present study.

References

[1] V. Steen and A. Tanenbaum, "A brief introduction to distributed systems," *Computing*, vol. 98, no. 10, pp. 967–1009, 2016.

[2] I. Sommerville, "Software engineering," 10th ed., Boston: Pearson, 2018. [Online]. Available: http://iansommerville.com/software-engineering-book/.

[3] J. Patni and M. Aswal, "Distributed approach of load balancing in dynamic grid computing environment," *International Journal of Communication Networks and Distributed Systems*, vol. 19, no. 1, pp. 1–8, 2017.

[4] D. Ford, F. Labelle, F. Popovici, M. Stokely, V. Truong *et al.,* "*Availability in globally distributed storage systems*. USENIX," 2010. [Online]. Available: http://www.usenix.org/events/osdi10/tech/full_papers/Ford.pdf.

[5] M. Kumari and R. Kumar, "A comparative study of various load balancing algorithm in parallel and distributed multiprocessor system," *International Journal of Computer Applications*, vol. 169, no. 10, pp. 31–35, 2017.

[6] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Elsevier Performance Evaluation*, vol. 67, no. 8, pp. 634–658, 2010.

[7] S. Becker, L. Grunske, R. Mirandola and S. Overhage, "Performance prediction of component-based systems," *Springer Architecting Systems with Trustworthy Components*, vol. 3938, pp. 169–192, 2016.

[8] S. Malek, N. Medvidovic and M. Mikic-Rakic, "An extensible framework for improving a distributed software system's deployment architecture," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 73–100, 2012.

[9] X. Chen, "Dependence management for dynamic reconfiguration of component-based distributed systems," in *Proc. IEEE*, pp. 279–284, 2002.

[10] G. Pepermans, J. Driesen, D. Haeseldonckx, R. Belmans and W. D'haeseleer, "Distributed generation: Definition, benefits and issues," *Elsevier Energy Policy*, vol. 33, no. 6, pp. 787–798, 2003.

[11] M. Mikic-Rakic, S. Malek and N. Medvidovic, "Improving availability in large, distributed component-based systems via redeployment, " in *Proc*. Berlin: Springer, pp. 83–98, 2005.

[12] S. Al-Areqi, A. Hudaib and N. Obeid, *Improving Availability in Distributed Component-Based Systems via Replication*, vol. 351. Berlin: Springer, 43–52, 2011.

[13] M. Klusch, *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*. Springer Science & Business Media, 2002. [Online]. Available: https://www.springer.com/gp/book/9783642642234.

[14] N. Obeid and S. Al-Areqi, "Using agents for dynamic components redeployment and replication in distributed systems," *Springer Applied Intelligence*, vol. 489, pp. 19–25, 2013.

[15] S. Weil, S. Brandt, E. Miller, D. Long and C. Maltzahn, "A scalable, high-performance distributed file system," in *Proc. USENIX Association*, pp. 307–320, 2006.

[16] I. Barazandeh, S. Mortazavi and A. Rahmani, "Two new biasing load balancing algorithms in distributed systems," in *Proc. IEEE, AH-ICI 2009. First Asian Himalayas International Conf.*, pp. 1–5, 2009.

[17] R. Jadhav and S. Kamlapur, "Performance evaluation in distributed system using dynamic load balancing," *International Journal of Applied Information Systems*, vol. 2, no. 7, pp. 36–41, 2012.

[18] S. Rajani and N. Garg, "A clustered approach for load balancing in distributed systems," *International Journal of Mobile Computing & Application*, vol. 1, no. 3, pp. 1–6, 2015.

[19] N. Roy, A. Dubey, A. Gokhale and L. Dowdy, "A capacity planning process for performance assurance of component-based distributed systems," *ACM SIGSOFT Software Engineering Notes*, vol. 39, no. 3, pp. 259–270, 2011.

[20] U. Kaur and S. Sharma, "Performance evaluation using various models in distributed component based systems," *International Journal of Computer Applications*, vol. 98, no. 1, pp. 24–33, 2014.

[21] F. Brosig, N. Huber and S. Kounev, "Automated extraction of architecture-level performance models of distributed component-based systems," in *Proc. IEEE Computer Society*, pp. 183–192, 2011.