An Efficient Replication Approach based on Trust for Distributed Self-healing Systems

Nizar Msadek and Theo Ungerer
Institute of Computer Science, University of Augsburg, Universitätsstr. 6a, 86135 Augsburg, Germany

Keywords: Autonomous Systems, Open Distributed System, Trust, Replication, High-performance Computing.

Abstract: Replication typically occurs in a wide range of open distributed systems, especially in self-healing systems (i.e., mainly for fault-tolerance purposes) and in high-performance computing (i.e., mainly for fast response times). All these systems face a common issue: how can replicas automatically and efficiently be managed in a system despite changing requirements of their environment? One way to overcome this issue is *trust*. The contribution of this paper is a novel approach based on trust that provides a good management of replicas — especially for those of important services — despite uncertainties in the behavior of nodes. Depending on the importance level of a service possessing the replicas and the assessment of the trustworthiness of a node, we can optimize the trust distribution of replicas at runtime. For evaluation purposes we applied our approach to an evaluator based on the TEM middleware. In this testbed, the usage of trust reduced the replication overhead by about 14% while providing a much better placement of important replicas than without trust.

1 INTRODUCTION

Implementing efficient and dependable replication mechanisms for open distributed systems is a non-trivial task (Halsall, 1996; Reif et al., 2016). This is due to the fact that such systems are becoming increasingly complex in their organizational structures, especially when unknown heterogeneous entities might arbitrarily enter and leave the network at any time. Therefore, new ways have to be found to develop and manage them. One way to overcome this issue is trust (Msadek and Ungerer, 2016). Using appropriate trust mechanisms, entities in the system can have a clue about which entities to cooperate with. This is very important to improve the robustness of such systems, which depend on a cooperation of autonomous entities (Msadek, 2016). In this paper, we primarily focus on self-healing and note that our goal is to develop an autonomous replication mechanism based on trust that works in a distributed manner and also ensures global optimality. The mechanism should provide a good replica management and placement — especially for those of important services — despite non-benevolent behavior of nodes in the network. Therefore, the contribution of this paper leads to a methodology that offers the following major points:

(i) an overview of the system model and the considered problem to indicate the specific purpose of our research (see Sections 2 and 3),

(ii) a mechanism for specifying the importance level of services based on the number of requests as well as a mechanism to monitor the trust behavior of nodes at runtime (see Section 4.1), and

(iii) a replication model considering the above introduced points to manage the amount of replicas for better trustworthiness of important services (see Section 4.2).

All aspects are evaluated and discussed with respect to a toolkit based on the TEM (Anders et al., 2013), a trust-enabling middleware for building real-world distributed Organic Computing systems. Section 5 provides evaluation results of the proposed mechanism and Section 6 aligns the trust-based approach in the context of state-of-the-art systems. Finally the paper is closed with a conclusion and future work in Section 7.

2 SYSTEM MODEL AND ASSUMPTIONS

We target a distributed system consisting of a finite set of nodes $\mathcal{N} = \{n_1, n_2.., n_n\}$, representing machines

Msadek, N. and Ungerer, T.
An Efficient Replication Approach based on Trust for Distributed Self-healing Systems.
DOI: 10.5220/0006382601990206
In *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2017) - Volume 1*, pages 199-206
ISBN: 978-989-758-263-9

which can interact with each other through a set of messages. The i-th node is denoted by n_i, or alternatively by i if it is not ambiguous. These nodes are heterogeneous in terms of storage resources. Thus, every node provides a storage space with a free capacity $\texttt{Capa}_i$ to offer services in the system a place for storing their data. The set of services is denoted by $\mathcal{S} = \{s_1, s_2.., s_k\}$ and the data of a service s_j is expressed by $\texttt{Data}_j$. This data is replicated among the nodes of the network. Thus, a $\texttt{Data}_j$ is partitioned into a set of owner and replicas with $\texttt{Data}_j = \{\texttt{Owner}_j\} \cup \{\texttt{Repl}_{j,1}, \texttt{Repl}_{j,2}, ..., \texttt{Repl}_{j,r_j}\}$. The $\texttt{Owner}_j$ is the central element that performs all write requests. It has multiple replicas which replicate its data. We assume that these replicas are completely identical to the owner in contrast to erasure codes [1]. This means that the storage consumptions for all replicas are the same, which is given by the value $\texttt{Consum}_j$. Whenever an $\texttt{Owner}_j$ performs a write request, it delivers an update with a new version of its data to the replicas. These replicas are used to perform read requests and to overcome node failures. Their amount is fixed so far as a system parameter r_j within the interval $[R_{min}, R_{max}]$ to avoid that not too many but also not too few replicas are created. Should an owner of a data service fail, the system will elect one of the replicas with the latest version to takeover the role of owner. We also assume that the number of requests may affect the importance level of services. Services having a large amount of requests are considered to be important for the functionality of the entire system. From this point of view, important services are supposed to be rational in the sense that they want to place their replicas as well as possible in the system, by choosing only high trustworthy nodes.

3 PROBLEM STATEMENT AND BASELINE

A crucial point in our system is the trustworthiness of the service storage, especially for important services. Their data should be hosted on trustworthy nodes having a high degree of trustworthiness despite malicious behavior of nodes in the network. Therefore, we formulate the trustworthy replication management problem as follows. Given a set of services with different importance levels and a network topology with a finite number of nodes representing possible replica locations, we are interested to determine a trustworthy placement for replicas such that the trustworthiness of important data is improved by hosting them on trustworthy nodes. On the other hand, the storage of nodes should be optimized in terms of resource consumptions as well. To do so, four fundamental decisions have to be considered: (1) which services are considered as more important than others in the system, (2) when should the categorization of services be determined, (3) how to fix the right amount of replicas, and (4) where to place these replicas on nodes. Depending on these decisions, different replication strategies can be applied.

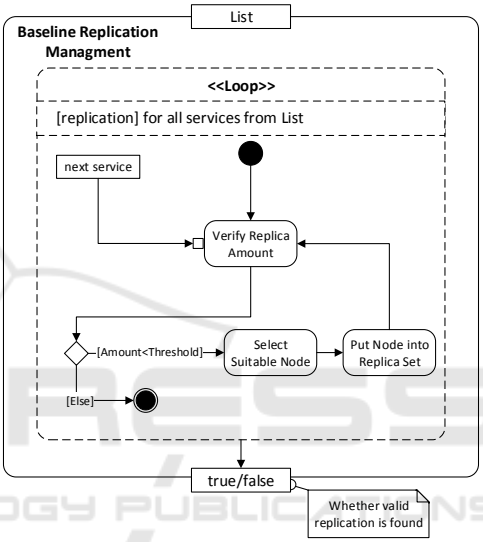


Figure 1: Elementary representation of the baseline replication management.

Figure 1 shows the baseline replication strategy considered in this work. It is based on thresholds for determining the importance level of services. These thresholds are fixed at design time and do not change during run time. Then, a scheduler is used for creating and assigning replicas. For every service, it selects a node with a uniform probability without considering its trust value or up probability and put it into a replica set. If for example a node is already selected or its storage space is full, it picks another new node, until the given number of replicas reaches the threshold R_{max} for important services or the threshold R_{min} for unimportant services. This strategy has the ability to improve the availability of service data in the system. However, it is only suited for classical systems where the benevolence of nodes is assumed (Schillo et al., 1994), because all nodes are treated identically even though some of them are less trustworthy than others. In open distributed systems, it would not provide good results due to the fact that unknown nodes might

[1] An erasure code provides redundancy without saving the identical complete copy of an object element. It divides the element into m fragments and recodes them into n fragments, where $n > m$. Details can be found in the paper referenced here (Weatherspoon and Kubiatowicz, 2002).

arbitrarily enter and leave the network at any time. Therefore, we propose to incorporate trust mechanisms into the management of replicas.

4 THE TRUST-BASED REPLICATION APPROACH

4.1 Models and Metrics

Our aim is to provide a continuous good trustworthiness for data especially for important services and to reduce the performance overhead produced by creating too many replicas in the system as well. As a consequence, our approach should consider the following factors: (1) We want a mechanism for specifying the required trust of services based on their number of requests. (2) We need a mechanism to regularly monitor the trust behavior of nodes. (3) And we want a model to manage the amount of replicas for better trustworthiness of important services. It is worth mentioning that all solutions provided should always be round-based over the time to ensure that the system is continuously optimized at runtime.

4.1.1 Determine the Required Trust of Services

As mentioned, the number of requests plays a crucial role in the categorization of services. Services with a large number of requests are considered to be more important than others in the system. They should have a higher degree of required trust in order to be hosted on trustworthy nodes. The baseline approach does not take such a decision into account at runtime. It categorizes the importance level of services at the beginning which is then not changed during execution. Our aim is to give the system more responsibility by moving this decision from design-time to runtime. The algorithm for this strategy consists of two phases which are bounds calculation and required trust calculation phase. In the first phase, we compute the cumulative number of requests for each service $s_j \in \mathcal{S}$ at every round k as follows:

$$Request^{k}_{(s_j)} = Request^{k}_{(\texttt{Owner}_j)} + \sum_{i=1}^{r_j} Request^{k}_{(\texttt{Repl}_{j,i})} \tag{1}$$

Where $Request^{k}_{(\texttt{Owner}_j)}$ stands for the number of read and write requests performed by the owner until round k is reached, and $Request^{k}_{(\texttt{Repl}_{j,i})}$ represents the total number of read requests that were performed by every replica i of s_j until k is reached.

$$\texttt{MinRequest}^{\texttt{k}} = \{Request^{k}_{(s_j)} | \forall s_i, s_j \in \mathcal{S} : Request^{k}_{(s_i)} \geq Request^{k}_{(s_j)}\} \tag{2}$$

$$\texttt{MaxRequest}^{\texttt{k}} = \{Request^{k}_{(s_j)} | \forall s_i, s_j \in \mathcal{S} : Request^{k}_{(s_i)} \leq Request^{k}_{(s_j)}\} \tag{3}$$

Then, we determine the minimum and maximum values over these requests by Equations 2 and 3. By this means, the request number of every service is always bounded between $\texttt{MinRequest}^{\texttt{k}}$ and $\texttt{MaxRequest}^{\texttt{k}}$. Values near to the $\texttt{MinRequest}^{\texttt{k}}$ reflect unimportant services, whereas values close to $\texttt{MaxRequest}^{\texttt{k}}$ stand for important services. In the second phase, we aim to compute the required trust of each service. Assume that $\texttt{MinTrust}$ and $\texttt{MaxTrust}$ are the desired thresholds for minimum and maximum trust set in the system, where $0 \leq \texttt{MinTrust} \leq \texttt{MaxTrust} \leq 1$. Then, a value of $Request^{k}_{(s_j)}$ can be mapped to a required trust value $ReqTrust^{k}_{(s_j)}$ in the new specified range $[\texttt{MinTrust}, \texttt{MaxTrust}]$ using min-max normalization as illustrated in Equation 4.

We give a high degree of trust for important services by shifting the minimum and maximum number of requests to $\texttt{MinTrust}$ and $\texttt{MaxTrust}$, respectively. Figure 2 shows – as example – the required trust scores that can take services s_i and s_j after min-max normalization. The original distribution of requests is retained for both services and is then transformed in required trust values in the new specified range of $[\texttt{MinTrust}, \texttt{MaxTrust}]$.

4.1.2 Assess the Trust Behavior of Nodes

A trust-based replication needs a component which generates trust values based on direct experiences to detect the presence of untrustworthy nodes in the system. This trust generation can be measured by regarding different facets of trust (Msadek and Ungerer, 2016), such as reliability, availability, functional correctness, and usability. The facet of interest in this work is availability, since we are interested to check the availability of nodes based on their uptime in the last interaction steps. Also we assume that the trust relation *Trust* between nodes is always irreflexive on $\mathcal{N}$, i.e., $\forall n_i \in \mathcal{N} :\Leftrightarrow \neg(n_i \; Trust \; n_i)$ meaning that we do not allow the possibility for nodes to assess their own trust values. Otherwise, nodes would trust themselves fully and the system will be prone to exploitation from malevolent nodes. Equation 5 shows the metric we set to calculate direct trust using the facet availability.

$$ReqTrust^{k}_{(s_j)} = \frac{Request^{k}_{(s_j)} - \texttt{MinRequest}^{\texttt{k}}}{\texttt{MaxRequest}^{\texttt{k}} - \texttt{MinRequest}^{\texttt{k}}} \cdot (\texttt{MaxTrust} - \texttt{MinTrust}) + \texttt{MinTrust} \quad (4)$$

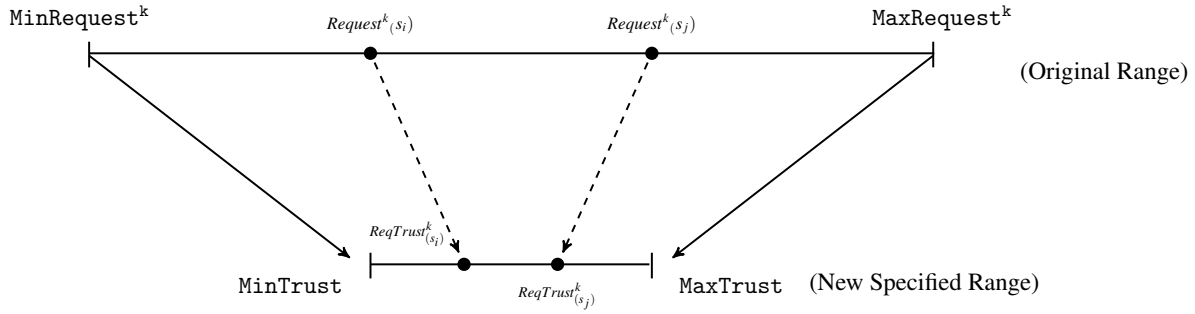


Figure 2: Mapping of service requests from the range of $[\texttt{MinRequest}^{\texttt{k}}, \texttt{MinRequest}^{\texttt{k}}]$ to required trust values in a new specified range of $[\texttt{MinTrust}, \texttt{MaxTrust}]$ using min-max normalization.

$$Trust^{k}_{(n_i,n_j)} = \begin{cases} (1-\alpha) \cdot Trust^{k-1}_{(n_i,n_j)} + \alpha \cdot \Theta^{k}_{n_j} & \text{if } k \geq 1 \\ Trust_{init} & k = 0 \end{cases} \quad (5)$$

$$\Theta^{k}_{n_j} = \frac{\text{Uptime of } n_j \text{ in period } [k-1,k]}{\text{Total time of period } [k-1,k]} \quad (6)$$

$$\Theta^{k}_{n_j} \in [0,1] \quad (7)$$

$$Trust^{k}_{(n_i,n_j)} \in [0,1] \quad (8)$$

$$k \in \mathbb{N} \quad (9)$$

In every round k, a node n_i calculates a trust value $Trust^{(k)}_{(n_i,n_j)}$ about its direct neighbor n_j based on the new observation $\Theta^{(k)}_{n_j}$ and the previous trust value $Trust^{(k-1)}_{(n_i,n_j)}$. This trust value $Trust^{(k)}_{(n_i,n_j)}$ is always within $[0,1]$ and reflects the subjective trust of node n_i in node n_j based on its experiences. A trust value of $Trust^{(k)}_{(n_i,c_i)} = 0$ means n_i does not trust n_j at all while a value of 1 stands for full trust. The factor $\alpha \in [0,1]$ decides how strong the recent observations are weighted compared to the previous ones. The larger the value α, the more the result is computed by the recent observations. Initially, the trust value of n_j is set to $Trust_{init}$ and in every round k an update occurs for $Trust^{(k)}_{(n_i,n_j)}$. With increasing number of mutual interactions over k, we expect to correctly estimate the behavior of nodes in the system. This is very important to prevent the hazardous placements of replicas on nodes.

4.1.3 Perform the Placement of Replicas

A crucial point in the baseline algorithm is the trustworthy placement of service data. To guarantee a specific level of trustworthiness the nodes hosting the replicas have to be chosen correctly. An optimal selection of these nodes is not considered in the baseline version. Therefore, we are interested in our approach to improve the replica placement by considering the trust values of each replica. Let us assume the required trust of a service s_j is known and that $\texttt{Owner}_j$ has multiple replicas $\{\texttt{Repl}_{j,1}, \texttt{Repl}_{j,2}, ..., \texttt{Repl}_{j,r_j}\}$ which replicate its $\texttt{Data}_j$. These replicas are distributed on different nodes and the trust behaviors of nodes are independent of each other. The probability of $\texttt{Data}_j$ to be in a trustworthy state at round k is represented by $Trust^{k}_{(\texttt{Data}_j)}$ and its untrustworthiness is denoted by $\overline{Trust^{k}_{(\texttt{Data}_j)}} = 1 - Trust^{k}_{(\texttt{Data}_j)}$. It is obvious that $\texttt{Data}_j$ is in an untrustworthy state if and only if all replicas as well as the owner are not trustworthy. So the untrustworthiness of $\texttt{Data}_j$ can be determined by Equation 10.

$$\overline{Trust^{k}_{(\texttt{Data}_j)}} = \overline{Trust^{k}_{(\texttt{Owner}_j)}} \cdot \prod_{i=1}^{r_j} \left(\overline{Trust^{k}_{(\texttt{Repl}_{j,i})}} \right) \quad (10)$$

This means that the probability of at least one replica to be in a trustworthy state can be written as follows:

$$Trust^{k}_{(\texttt{Data}_j)} = 1 - \left(\overline{Trust^{k}_{(\texttt{Owner}_j)}} \cdot \prod_{i=1}^{r_j} \left(\overline{Trust^{k}_{(\texttt{Repl}_{j,i})}} \right) \right) \quad (11)$$

To ensure that this probability is always greater than or equal to $ReqTrust^{k}_{(s_j)}$, we make use of Equation 12 as a condition.

$$1 - \left(\overline{Trust^{k}_{(\texttt{Owner}_j)}} \cdot \prod_{i=1}^{r_j} \left(\overline{Trust^{k}_{(\texttt{Repl}_{j,i})}} \right) \right) \geq ReqTrust^{k}_{(s_j)} \quad (12)$$

By this means, solving the placement problem consists of minimizing the amount of replicas r_j needed such that the constraint of Equation 12 is met. This is very important to prevent the hazardous placements of replicas on nodes and to reduce the replication overhead during runtime.

4.2 The Replica Management Approach

The interaction between the activities of the trust-based replication approach is illustrated in Figure 3.

In the first step, we make use of the trust metric of Equation 5 — by regarding availability as a facet of trust — to assess the behavior of nodes. Our metric takes the advantage to converge to the true hidden trust values of nodes with increasing number of mutual interactions over k. This is very important to detect trust anomalies in node behavior and to allow owners to decide where to place their replicas trustworthily in the system. Then an update is initiated at every round k to refresh the trust values of nodes as well as the recognition of important services at runtime. Services with a large amount of requests are considered to be important for the functionality of the entire system. As a consequence, we give them a high degree of required trust using Equation 4. Then, replicas are managed for every service so that the condition of Equation 12 will hold. This replication management is accomplished by removing, replacing or adding the minimum number of replicas for every service. The smaller the amount of replicas is, the lower the replication cost and the better the performance of the system will be. Using trust, our replication management has the following benefits compared to conventional replication systems: On the one hand it reduces the overall number of replicas produced in the system. This optimization cost is continuously performed over the system lifetime. On the other hand, it improves the trustworthy placement of replicas on nodes so that the more important replicas will be always placed only on highly trustworthy nodes.

5 EVALUATION

In this section, we investigate the effectiveness of the trust-based replication approach. For the purpose of evaluating and testing, an evaluator based on our TEM (Anders et al., 2013) middleware has been implemented which is able to simulate our approach over a period of 2000 rounds. The evaluation network consists of 1000 nodes where all nodes are able to communicate with each other using message passing. Each node has a random storage capacity and is judged by an individual trust value without any central knowledge. The trust values of nodes are generated in two steps. Firstly, according to (Bernard and Le Fessant, 2009; Rzadca et al., 2010), we created different behaviors of nodes with different proportions in the system:

- **Durable Trustworthy:** with a mean value of 0.95 and proportion of 10%
- **Stable Trustworthy:** with a mean value of 0.87 and proportion of 25%
- **Unstable Trustworthy:** with a mean value of 0.75 and proportion of 30%
- **Erratic:** with a mean value of 0.33 and proportion of 35%

Then, we added a Gaussian noise of $\sigma = 0.1$ to each trust and capped the resulting value into $[0,1]$. The evaluation has been conducted using 100 services (i.e., 50% of them are important and 50% unimportant). The service assignment has been performed randomly to ensure that the proposed approach is evaluated under a great variety of start conditions. After the assignment, replicas are created in the system using the following parameters:

- The minimum replication factor R_{min} is set to 5.
- The maximum replication factor R_{max} is set to 20.

Our goal is to recognize the importance level of services solely based on the number of requests and to improve the trustworthy placement of replicas at runtime. Furthermore, replication cost should be reduced in contrast to the baseline approach in order to get a better performance in the overall system. Each evaluation scenario has been replayed 500 times and the results have been averaged.

5.1 Trust Examination

In the following, it is demonstrated how the placement of replicas can be improved — using the proposed algorithm — in response to trust changes in the environment. Therefore, the importance level of services is changed during runtime. We varied the rate of requests for every service and compared the deviation of actual trust from the required trust. Figures 4(a) and 4(b) show the results of this experiment without and with trust, respectively.

The results attest a good performance for the trust-based replication compared to the baseline approach.

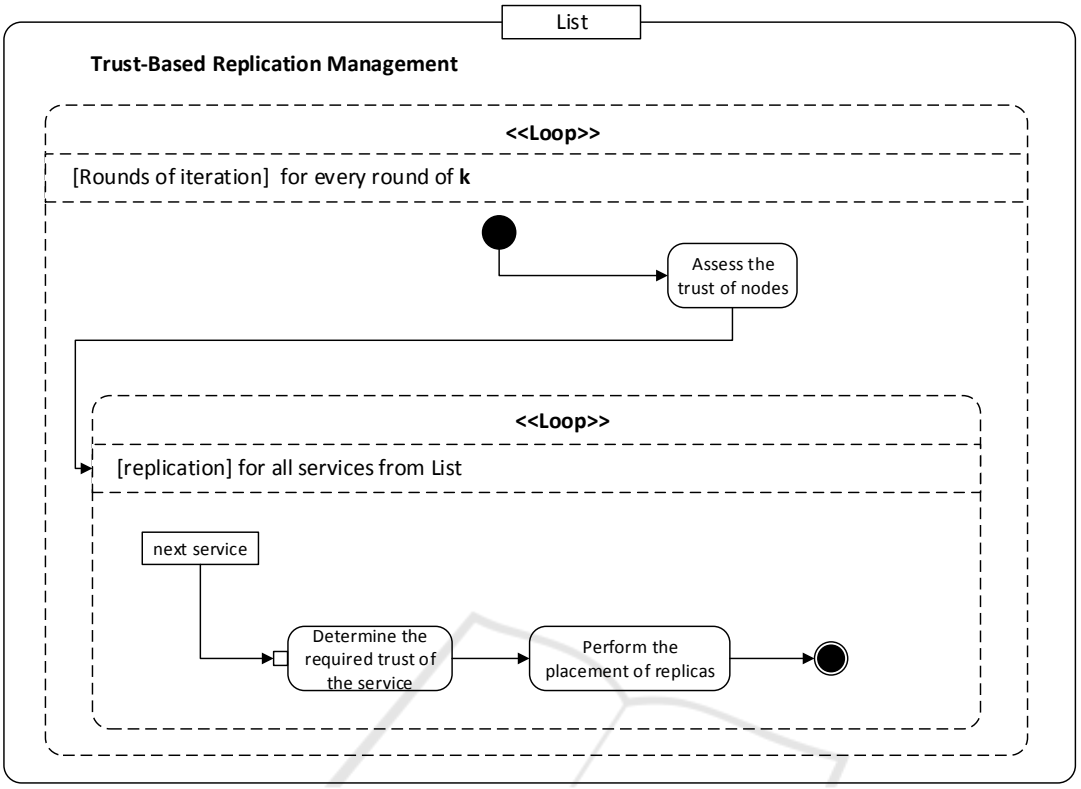


Figure 3: Elementary representation of the trust-based replication management.

Most of services show either an equal or a better actual trust than the required trust. However, there exist a small number of services which have a less trustworthy state than the required value. This is explained by the fact that nodes in our system have a limited capacity available for storage. This makes the replica process difficult for some services to find still unloaded trustworthy nodes on which to place their replicas.

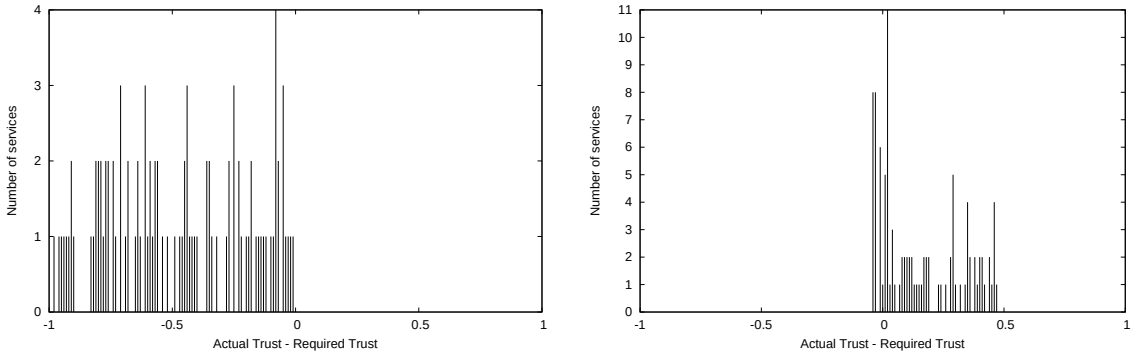
5.2 Overhead Examination

To establish replication, replicas need to be created in the system. In the following, this overhead is investigated for the baseline and the trust-based replication approach. For this experiment, the same settings as above have been used and Figure 5 shows its result.

The values on the x-axis stand for the replication approach used and the total number of replicas is depicted on the y-axis. To perform detailed measurements, we separated the important services from unimportant ones and calculated also the replica overhead created for each category. From the results, we can observe that the overhead is reduced by about 20 % for important services. But for unimportant services, the overhead is maintained nearly at a constant level. This is because the condition set in the system does not tolerate to produce less replicas than R_{min} for unimportant services. Overall, we can say that the cost of replication is decreased using our approach by about 14% for all services. This means that the consideration of trust does not prevent our algorithm to save overhead by placing the replicas.

6 RELATED WORK

As far as we know, a previous study of replication for self-healing systems based on trust in open and distributed environments has never been done before. However, some works have already been done on their design, for instance Spanner (Corbett et al., 2013) which is a system designed by Google, OriFile (Mashtizadeh et al., 2013), MinCopysets (Cidon et al., 2013), Scada (Kirsch et al., 2014), CalvinFS (Thomson and J. Abadi, 2015) etc. Our approach differs from the current state of the art in four major points. First, the benevolence assumption of nodes is not made in our work. In contrast, we use the social concept of trust to mitigate hazards that can occur from placing replicas randomly in the system. Second, our approach can adapt to changing behavior of nodes as well as to changing condition of services. Third, it possesses a regulation mechanism to save



(a) Using the baseline replication approach. (b) Using the trust-based approach.

Figure 4: The density of trust deviations.

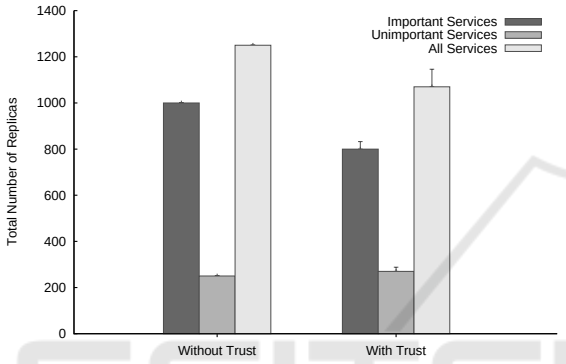


Figure 5: Measuring the impact of overhead for different categories of services using the trust-based vs. the baseline replication approach.

replica overhead at runtime, and finally it is applicable to any kind of trust-aware Recommender Systems (Kiefhaber et al., 2013). Thus, we want to compare our mechanism to systems that employ the same approach. The closest approach to us is to perform the replica placement based on a forecast regarding the availability of nodes. For example, the authors of (Li et al., 2015) analyze how to maximize the number of objects that remain available when node failures occur. Existing systems based on such availability placement include (Rzadca et al., 2010; Mills et al., 2015; Bhagwan et al., 2003; Bernard and Le Fessant, 2009). The main disadvantage of those approaches is that they do not take the priority of services into account and need a high computation power to perform the placement. This would not be suitable for ubiquitous or embedded systems. Very recently, the authors of (H. Noor et al., 2016) proposed an interesting replication technique based on trust to minimize the impact of inoperable TMS instances in the system. Based on an artificial exponential distribution to model trust, the authors estimate the trust of each node and determine the amount of replica accordingly. However, as exponential distributions are memoryless they cannot be used to predict trust as we do in this work. The second disadvantage of this approach is that it does not consider the number of requests to determine the importance level of services at runtime.

7 CONCLUSION

In this paper, a novel replication approach for self-healing systems is proposed. It is based on the notion of trust to improve the trust distribution of replicas and to minimize replication overhead in the system. The algorithm makes use of a trust metric to model the trust relationship between nodes, which is missing in most existing state of the art systems. Then, a mathematical model is formulated to determine the required trust of each service based on how many times it was requested in the last rounds. This is important to enhance the management of replicas for better trustworthiness of important services. An evaluation is provided with respect to a real-world Organic Computing middleware. Overall, the results show a better trust distribution for replicas with a significant reduction in overhead when compared to the baseline. However, there are still some studies to be done for future work. For instance, further improving the trust distribution of replicas and further decreasing the replication overhead. We also plan to investigate more the categorization of services by including other factors such checkpoint size and service centrality (Cao et al., 2016; Pantazopoulos et al., 2011).

REFERENCES

Anders, G., Siefert, F., Msadek, N., Kiefhaber, R., Kosak, O., Reif, W., and Ungerer, T. (2013). Temas a trust-enabling multi-agent system for open environments. Technical report, Universität Augsburg.

Bernard, S. and Le Fessant, F. (2009). Optimizing peer-to-peer backup using lifetime estimations. *2nd International Workshop on Data Management in Peer-to-peer systems*, page 8.

Bhagwan, R., Moore, D., Savage, S., and Voelker, G. M. (2003). Replication strategies for highly available peer-to-peer storage. *Future Directions in Distributed Computing*, Volume 2584 of the series Lecture Notes in Computer Science:153 – 158.

Cao, J., Arya, K., Garg, R., Matott, L. S., Panda, D. K., Subramoni, H., Vienne, J., and Cooperman, G. (2016). System-level scalable checkpoint-restart for petascale computing. *CoRR*, abs/1607.07995:1–18.

Cidon, A., Stutsman, R., Rumble, S., Katti, S., Ousterhout, J., and Rosenblum, M. (2013). Mincopysets: Derandomizing replication in cloud storage. *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation*.

Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., and Woodford, D. (2013). Spanner: Google’s globally distributed database. *ACM Trans. Comput. Syst.*, 31:8:1–8:22.

H. Noor, T., Z. Sheng, Q., Yao, L., Dustdar, S., and H.H. Ngu, A. (2016). Cloudarmor: Supporting reputation-based trust management for cloud services. *IEEE Transactions on Parallel & Distributed Systems*, 27:367 – 380.

Halsall, F. (1996). *Data Communications, Computer Networks, and Open Systems*. Addison-Wesley; Electronic Systems Engineering Series.

Kiefhaber, R., Jahr, R., Msadek, N., and Ungerer, T. (2013). Ranking of direct trust, confidence, and reputation in an abstract system with unreliable components. In *The 10th IEEE International Conference on Autonomic and Trusted Computing (ATC-2013)*.

Kirsch, J., Goose, S., Amir, Y., Wei, D., and Skare, P. (2014). Survivable scada via intrusion-tolerant replication. *IEEE Transactions on Smart Grid*, 5:60 – 70.

Li, P., Gao, D., and Reiter, M. K. (2015). Replica placement for availability in the worst case. *IEEE 35th International Conference on Distributed Computing Systems (ICDCS)*, 67:599 – 608.

Mashtizadeh, A. J., Bittau, A., Huang, Y. F., and Mazières, D. (2013). Replication, history, and grafting in the ori file system. *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ACM:151–166.

Mills, K. A., Chandrasekaran, R., and Mittal, N. (2015). Algorithms for replica placement in high-availability storage. *The Computing Research Repository (CoRR)*, abs/1503.02654.

Msadek, N. (2016). *Increasing the Robustness of Self-Organizing Systems By Means of Trust Practices*. PhD thesis, University of Augsburg.

Msadek, N. and Ungerer, T. (2016). Trust as important factor for building robust self-x systems. In *Trustworthy Open Self-Organising Systems*.

Pantazopoulos, P., Karaliopoulos, M., and Stavrakakis, I. (2011). Centrality-driven scalable service migration. *Proceedings of the 23rd International Teletraffic Congress*, 978-0-9836283-0-9:127–134.

Reif, W., Anders, G., Seebach, H., Steghöfer, J.-P., Elisabeth, A., Hähner, J., Müller-Schloer, Christian, and Ungerer, T. (2016). *Trustworthy Open Self-Organising Systems*. Birkhäuser.

Rzadca, K., Datta, A., and Buchegge, S. (2010). Replica placement in p2p storage: Complexity and game theoretic analyses. *IEEE 30th International Conference on Distributed Computing Systems (ICDCS)*, 67:599 – 609.

Schillo, M., Funk, P., Stadtwald, I., and Rovatsos, M. (1994). Using trust for detecting deceitful agents in artificial societies. *Proceedings of the Ibero-American Conference on Artificial Intelligence (IBERAMIA '94)*, 14:825 – 848.

Thomson, A. and J. Abadi, D. (2015). Calvinfs: Consistent wan replication and scalable metadata management for distributed file systems. *13th USENIX Conference on File and Storage Technologies (FAST 15)*, USENIX Association:1 –14.

Weatherspoon, H. and Kubiatowicz, J. D. (2002). Erasure coding vs. replication: A quantitative comparison. *Peer-to-Peer Systems*, 2429 of the series Lecture Notes in Computer Science:328 – 337.