STUDIA INFORMATICA
Nr 1-2 (22) Systems and information technology 2018

Andrzej BARCZAK[1],
Michał BARCZAK[2]

[1] Siedlce University of Natural Sciences and Humanities
Institute of Computer Science,
ul. 3 Maja 54, 08-110 Siedlce, Poland

[2] Mettler Toledo
ul. Poleczki 21, 02-822 Warsaw, Poland

Spreading information in distributed cloud systems using Gossip algorithm

Abstract. In a following article problem of a information sharing in distributed systemis described, as wellways of solving that problem with emphasize on Gossip protocol are presented. Furthermore the application has been creating allowing to test the Gossip protocol in a lab environment. Gossip protocol is highly parameterized and can be working in several modes. The main goal of the article is to examine the work of the Gossip algorithm, depending on the chosen mode and values of parameters, and analyses of a results.

Keywords: distributed systems, cloud systems, gossip algorithm

1. Distributed system in cloud solutions

Nowadays more and more IT systems are distributed systems or system using cloud solutions. They contains of few up to few thousands of connected computers called nodes. Such a kind of systems allows to significantly increase computing power and capitalize the resources of each of the nodes in more efficient way. Distributed systems are characterized by several features. One of most important of them is transparency. This feature guarantee that the user of a system is not aware of such a system's parameters like geographical location of each of the nodes or it's size. The breakdown of nodes should as well be not perfectible by the user. Distributed system should guarantee as well the transparency of the methods of access to it. In that case end user has the impression that system consists of a single node and not many thousands of them. Distributed system should be open for cooperation with other systems. It is pretty important that functionality of the distributed systems must be described using very precise interfaces.Another crucial feature of distributed systems is its' scalability taking into consideration both amount of nodes and their geographical location. Other point is the concurrency of the systems that allows to perform many tasks at one time. Easy way to configure and reconfigure is as well important feature of distribution system. One of the most important feature of distributed systems is their ability to share the same resources by all of the users.Ensuring all this features mentioned above is not an easyor trivial topic.

Main feature guarantees the transparency is the ability to share and distribute messages inside of a system. One of a most common solutions allowing to distribute information in a smooth way is Gossip algorithm (know as well as Gossip protocol).

2. The problem of information distribution in a cloud systems

While working with huge distributed cloud systems containing of hundred thousands of nodes, we need to deal with a problem of distributing information in a efficient and quick way. It is crucial that data, that was recived by one of the nodes, will be send fast and without too much usage of network resources. There are several algorithms that help us to deal with that problem.

2.1. One-to-One Algorithm

The simplest, however also less efficient, algorithm o distributing knowledge in a cloud is algorithm one-to-one. Mentioned algorithm is just sending information node by node. In a first iteration node receiving new piece of information sends it to his neighbour. In the flowing iterations information is distributed node by node. In case of the one-to-one algorithm the path of information flow is created. It leads to the decreasing of algorithms reliability. In case of a crash of network connection between two of the following nodes information is not send farther. Due to the fact that one node sends information just to one neighbour, full procedure is very time consuming and not efficient. Function of information propagation in time according to number of nodes is lineal and can be described as $t_p=t_i*n$ where t_p is time of propagation, t_i is time of one iteration and n is number of nodes. Assuming that the iteration time is 1 second and number of nodes is 1000 propagation time is 1000 seconds what is about

16 minutes. I would like to admit that the one-to-one algorithm is not scalable, that means that the time results will be even worse when the number of nodes increase.

2.2. All-to-AllAlgorithm

Other algorithm of spreading the information in distributed cloud systems is algorithm all-to-all.In this algorithm each of the nodes sends the newly received information as soon as he get it. Such a solution allows to increase the reliability of the distributing of information, cause compare to one-to-one algorithm, crash of connection between two of the nodes does not have such a negative effects. In case of problems with the connection between two of the nodes, information can be sand frough deferent path from other node. The problem with propagation time of the information known from algorithm one-to-one has been solved as well. Full system receive the informationalmost immediately after it appears. Unfortunately all-to-all algorithm is not free from flaws and limits. The major flaw is the amount of information sent in the system which is N^2 in each iteration where N is number of nodes. The limit of the nodes in the system derives directly from this flaw. In case of the systems containing tens of thousands of nodes, in each iteration even thousands of millions of information can be send. For both computer network and a single node it is impossible to handle such a network traffic.

2.3. Gossip algorithm

Xerox company at the end of 1980's struggled with a problem of replicating data on few thousands instances of distributed data base. Crucial aspects for Xerox were both time of spreading big amount of information and the reliability of the replication process. One of the propositions was Gossip algorithm called as well Epidemic algorithm. The rule of operation of this algorithm is very similar to spreading either gossip or the diesis in some population [6].Nodes of the system having information sends it to chosen neighbour in a iterative way. The way of selecting the neighbour might be random, but it does not need to be such. Each of the nodes might calculate dynamic parameters of connection (like transmission time or quality parameters of transport medium). Based on that parameters for each of nodes factor will be assigned. Nodes with higher factor will be prioritised while selecting neighbour for information sending.

For nodes in a system, based on possessing of information, one of three following statuses (states) can be assigned [8]:

1. Infected by the information (I)
2. Susceptible (S). This status is assigned to the nodes not having information.
3. Removed (R). Node can be in state R when the information it has is old and is not distributed any more.

There exists two main models of changing of statuses - Model SI and Model SIR.

In Model SI there is possibility only for changing state from S to state I,while in SIR Model change from I to R is also allowed. The Gossip algorithm can work in three different modes: push, pull and hybrid push-pull. In case of SI push implementation, nodes being in S state only are passively listening and waiting for new information. In the pull version nodes, with status S assigned, as well sends requests for a new piece of information. The hybrid model is combination of Push and Pull. Pseudo code of the Gossip algorithm in SI model is presented in listing 1.

```
Public void Gossip()
{
while (true)
{
Thread.Sleep(timePeriod);
NodeselectedNode = GetRandomNode();
if (this.push)
{
if (this.state == NodeState.Infected)
{
SendUpdate(selectedNode, info);
}

}
if (this.pull)
{
SendUpdateRequest(selectedNode);
}
}
}
Private void OnGetUpdate(UpdateArgsargs)
{
args.reciver.informations.Add(args.info);
args.reciver.state = NodeState.Infected;
}

Private void OnGetUpdateRequest(UpdateArgsargs)
{
if (this.state == NodeState.Infected)
{
SendUpdate(args.sender,args.info);
}
}
```

Listing 1. Gossip Algorithm in SI model. Source: own work

It is wealth mentioning that in SI model for both push and hybrid implementation data spreading will not be stopped even when all the nodes have received information. In case of pull implementation this problem not occurs as long as all the nodes has information about amount of data that need to be redistributed. If all of the nodes received required number of information, they will stop sanding update requests. Supposing that nodes do not have knowledge about amount of information in a system, pull requests will be sent as well, even when system is in consistent state. Constant sending requests and information may lead to overloading of a network. One of the solutions of described problem is SIR model. In SIR model for each piece of information the ageing factor is added. After reaching this value node is set into state R and stops distributing information. Specifying ageing factor is not an easy task. Many aspects, like for example number of nodes and delays in computer network, needs to be taken into consideration. Underestimation of the factor may lead to not sending information to some of the nodes. Overestimation however cause sending too big number of information and overloading the network. There are several methods of estimating the ageing factor. Most of them requires sending the return message after receiving known information. In first implementation node can be set into status R after sending defined number of the obsolete information. Next one allows to stop sending information , with probability P, after sending every duplicated message. In last implementation one of the nodes having obsolete data is chosen to be set in status R [7]. Pseudo code of Gossip protocol in a SIR model is presented in Listing 2. The differences in implementation between SI and SIR model were highlighted in yellow.

```
Public void Gossip()
{
while (true)
{
Thread.Sleep(timePeriod);
Node selectedNode = GetRandomNode();
if (this.push)
{
if (this.state == NodeState.Infected)
{
SendUpdate(selectedNode, info);
}

}
if (this.pull)
{
SendUpdateRequest(selectedNode);
}
}
}

Private void OnGetUpdate(UpdateArgsargs)
{
args.reciver.informations.Add(args.info);
args.reciver.state = NodeState.Infected;
SendResponse(args.sender, informationStatus);
}

Private void OnGetResponse()
{

if    (RemovalImplementation.Number    ==    remImplementation&&resendedMessages    ==
obsolescence)
{
this.state = NodeState.Removed;
}
elseif (remImplementation == RemovalImplementation.Probability)
{
if (GetProbablity())
{
this.state =NodeState.Removed;
}

else
{
resendedMessages++;
}
}
}

privateboolGetProbablity()
{
var rand = new Random();
var p = rand.Next(1, obsolescence);
return p < obsolescence;
}

Private void OnGetUpdateRequest(UpdateArgsargs)
{
if (this.state == NodeState.Infected)
{
SendUpdate(args.sender, args.info);
}
}
```

Listing 2. Gossip Algorithm in SIR model. Source: own work

Other crucial problem, existing in Gossip algorithm, is gathering knowledge about system topology by each of a nodes. In case of huge systems it requires establishing very big list of connections on every node, what can have negative influence on scalability of the Gossip algorithm. Finding out new joined nodes is as well problematic issue. It requires from nodes sharing information about the topology. Such a procedure generates additional network traffic. Another solution states creating one central managing node that will have full knowledge about system structure. It is as well not perfect resolution cause the reliability of a system is becoming lower. In most efficient implementation every node has just partial information about a system. Assuming that combination of information from all the nodes allows to send information between every node it is best solution. More than that information about system topology can be as well sent with Gossip messages.

It is wealth mentioning that Gossip algorithm allows to spread more than one information in a system in the same time. Taking it into consideration possiblility that not all nodes will have the same information, opposite to all-to-all algorithm full knowledge and coherence will be achieved after few iterations. Gossip algorithm is an algorithm of final coherence, that not guarantee strong coherence of a system. That is why it cannot be used in systems requiring very strict coherence like transactional bank systems or internet stores.

Some of the implementation of Gossip algorithm allows to send information to more the one neighbor in the same iteration. It for sure reduce number of iterations and speeds up the process of knowledge sharing, however we need to remember that choosing to big number of neighbors to sent information will lead to creation almost all-to-all algorithm and overloading network.

2.4. The description of a application and lab environment.

Algorithm Gossip has been implement using .NET Remoting technology in C# programming language. Application allows to spread more than one information at one time and for changing the Gossip model (SI and SIR). As well it is possible to specify number of neighbors for which information will be sent in one iteration and the ageing factor. In case of SIR model user is able to select method of choosing nodes that should be assigned to R status. Algorithm Mode can be parameterised as well. Application shows number of sent network packages corresponding with Gossip protocol. As well the state of each node (number of information it has) can be shown. The lab environment contains of ten connected servers (nodes), on which the test application is running. The nodes were connected in two deferent configurations. In first one the notes are connected into a complete graph while in second one into binary tree.

2.5. Analysis of Gossip Algorithm

The way Gossip algorithm in SI model is working, has been examined for five, six, eight and ten nodes. For five nodes, servers has been connected in a topology of a binary tree presented on Figure 1.

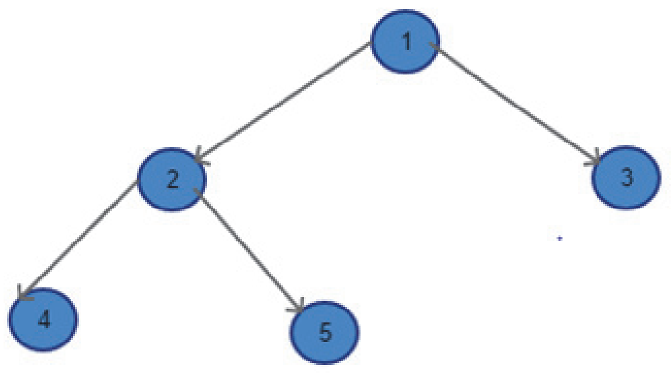

Figure 1. Network topology for five nodes. Source: own work

Examination has been done for three modes (pull, push and hybrid). One or two pieces of information had been sent. The information was sent either from node one, five or both of them. In case of push method and sending the information from first node system reached the coherency in iteration number five. For both pull and hybrid modes it took place in second iteration. When the spreading node was the one with number five the results were five iteration for push method, four for pull and three for hybrid. Assuming that the information was sent from both nodes one and five the performance was the same for hybrid and pull. In both cases the information was spread in three iterations. In the push mode, system needed one iteration more to send information to all nodes.

For six nodes binary tree topology presented on Figure 2 was used.

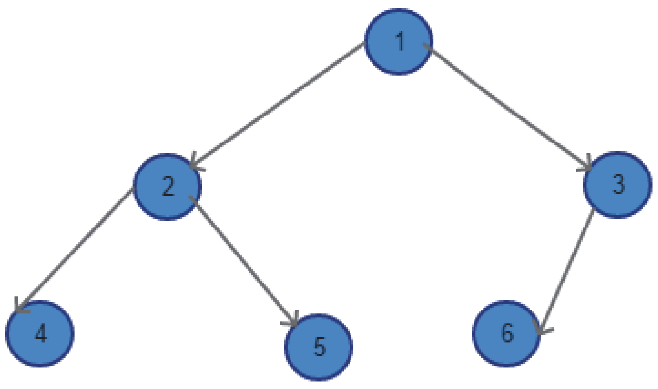

Figure 2. Network topology for six nodes. Source: own work

The same as in case of five nodes one or two pieces of information was sent. They were propagated from nodes number one, six or both.When the information was sent from first node, the most efficient occurred to be algorithm Gossip in a hybrid mode. The system was coherent in third iteration. Using push and pull modes, nodes needed five iteration to spread the information. In the push mode information was distributed in sixth iteration, while in pull mode not until eighth.We can observe similar results, when the data packages were sent from both node one and six at ones. Using hybrid mode algorithm was able to spread information in three iterations. For push and pull six iterations were needed to achieve such a result. For six nodes, as well examination of spreading information in system of complete graph topology was done. One or two pieces of information was sent. In both cases most efficient occurred to be hybrid mode. For both one and two pieces of information system get coherent in second iteration. In pull mode it took place in third iteration forone information and forth for two pieces of information. In both cases Push mode needed four iteration to spread information over the system.

Analogous tests were made for a system containing eight nodes. Binary tree topology of connections between servers is presented on Figure 3.

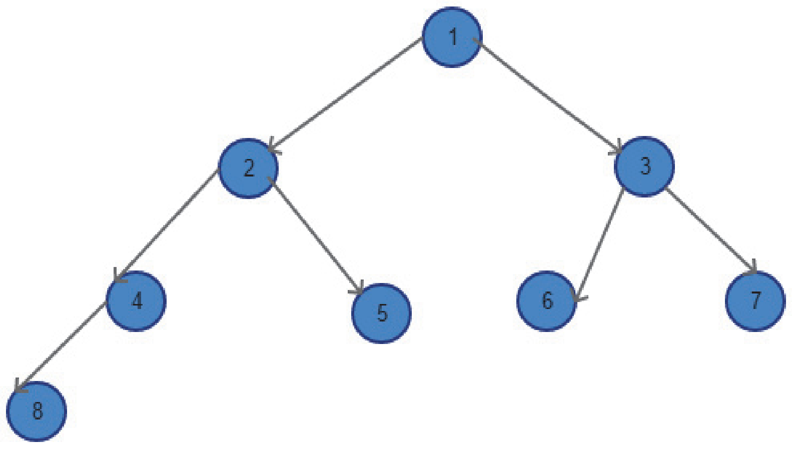

Figure 3. Network topology for eight nodes. Source: own work

For hybrid and pull modes, when infected node was that with number one, information was distributed in second iteration. Algorithm Gossip in push mode needed six iteration to achieve it. For information distributed from node number eight, in hybrid mode, system get coherent in sixth iteration. Using pull method, algorithm was able to spread information in seven iterations, while using push in twelfth. Distributing two data packages (one from node seven and one from node eight) lead to worse results. Gossip protocol in a push mode needed up to twenty seven iteration to spread information. Pull method was able to redistribute information in eleven iteration and hybrid in nine. For hybrid mode sending three pieces of information was tested. They were distributed from nodes number eight, seven and six. Coherence was achieved in eighth iteration. Next measurement for nodes connected into complete graph were done. Spreading one information took four iterations for push, six for pull and three for hybrid. When number of data packages was extended to two, the most efficient occurred to be hybrid and push modes. In both cases system was coherent in forth iteration. Pull mode needed one iteration more to achieve that. For hybrid mode the number of selected neighbors, for information distribution in one iteration, was extended to two. It effected in spreading one information in two iterations.

Similar experiments were made for system containing ten nodes.The topology of network connections between servers is presented on Figure 4.

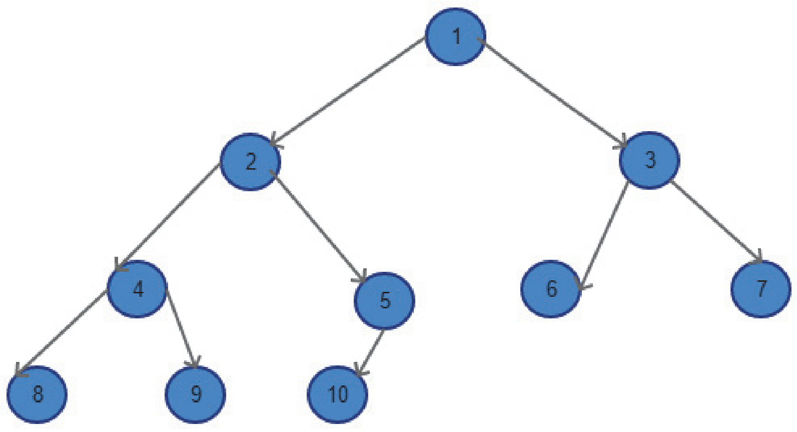

Figure 4. Network topology for ten nodes. Source: own work

When information was sent from node number one in push mode, the system achieved coherency in the fourteenth iteration. In the pull mode, this took place in the ninth iteration, while in the hybrid mode in the fifth. Considering the first node infected was node number ten, the Gossip algorithm, in both push and pull mode, spread the data in eleven iterations. In hybrid mode, each server received information after six iterations. For two pieces of information sent from nodes nine and ten in push mode, the broadcast occurred in seven passes. For the pull mode this was done in the fifteenth iteration and for the hybrid mode in the fifth. After connecting all ten nodes in a full graph topology, the results are much better. For one piece of information sent in the push mode, the system was coherent after the fourth iteration. In the pull mode it took three more iterations, while in the hybrid mode the message was sent within two iterations. In the case of sending two messages in the push mode, the information was disseminated during six iterations, in the pull mode during seven, while in the hybrid mode during three.

In the SIR model, for five nodes connected in topology of full graph, hybrid mode and one message, tests of various model configurations were made. In the SI model, every node within forty-five iterations sent ninety-two packages related to the exchange of information (requests to send a message and the messages themselves). With a larger number of nodes, it can be a significant load for the network. In order to minimize the number of requests and information sent, an SIR model was introduced, allowing to stop broadcasting information under certain conditions.The behavior of the Gossip algorithm in three variants of the SIR mode was investigated. The first was to stop sending messages with a set probability P / the probability of changing the status of information from the state I to R / at the moment when the neighboring node sends information that it already received particular message before. The tests began with a P value of 0.05. For this configuration, nodes sent from seventy-seven to one hundred and five packages during forty-five iterations. Considering that this value also contains the amount of feedback messages and that number of network packages is smaller than in the SI model, the effect is satisfying. Then the value of P was increased to 0.1. The result of the increase was visible and resulted in a reduction in the number of packages from individual nodes to a number between sixty and seventy. After another increase in the value of P to 0.2, the number of packages sent from one node was reduced to a value between fifty and fifty seven. A further increase in the value of P did not bring such large changes. For P equal to 0.3, nodes send out fifty to fifty five packages, while for a P of 0.5, the result is between forty five and fifty packages. For the higher probability values, it was already possible to observe the lack of final system coherency. The information was set in R status on all nodes before it was sent out over the entire system.

Another implementation of the Gossip algorithm in the SIR model assumes switching to the R state after sending n redundant messages. Assuming n equal three, each node sent fifty to fifty three packages. The results for n equal to five are identical. At n of six, this number increased to fifty four - fifty-seven packages. Assuming n equal to eight, we can observe another increase in the number of packages to a value between fifty five and sixty.

The next implementation allowed sending by a predetermined management node to a randomly selected neighbor containg a message, request to change the information's state to R. In this implementation, each node sent within seventy messages. A summary of the test results for the number of packets sent is presented in Table 1.

Table1. Number of sent packages in Gossip algorithm. Source: own work

	Minimal number of packages	**Maximal number of packages**
SI		
	92	92
SIR with Probablity P		
0.05	77	105
0.1	60	70
0.2	50	57
0.3	50	55
0.5	45	50
After *n* redundant mesages		
3	50	53
5	50	53
6	54	57
8	55	60
Random	68	73

The researches were made on a relatively small number of nodes, so the question arises about the efficiency of the algorithm for much larger systems. In the case of a full graph topology, it can be assumed that the probability *p* of choosing each of the n neighbors is equal to 1 / n. The Gossip algorithm of the SI model and in the push mode can be described using a mathematical model known from epidemiology, showing the number of infected people (in our case nodes) depending on the time. The formula for the number of infected nodes is shown below [1]:

$$U(t)=\frac{n+1}{1+n*e^{-(n+1)*p*t}}(1)$$

where: n is the number of nodes, and t is the number of iterations. Assuming that the number of nodes is one hundred thousand, we need only twenty-four iterations to achieve coherency. For a million nodes, the number of iterations only increases to twenty eight. For the pull and hybrid models the results should be even better. Figure 5 shows the percentage share of infected nodes in iterations for ten, one hundred, one thousand, ten thousand one hundred thousandand and million nodes.

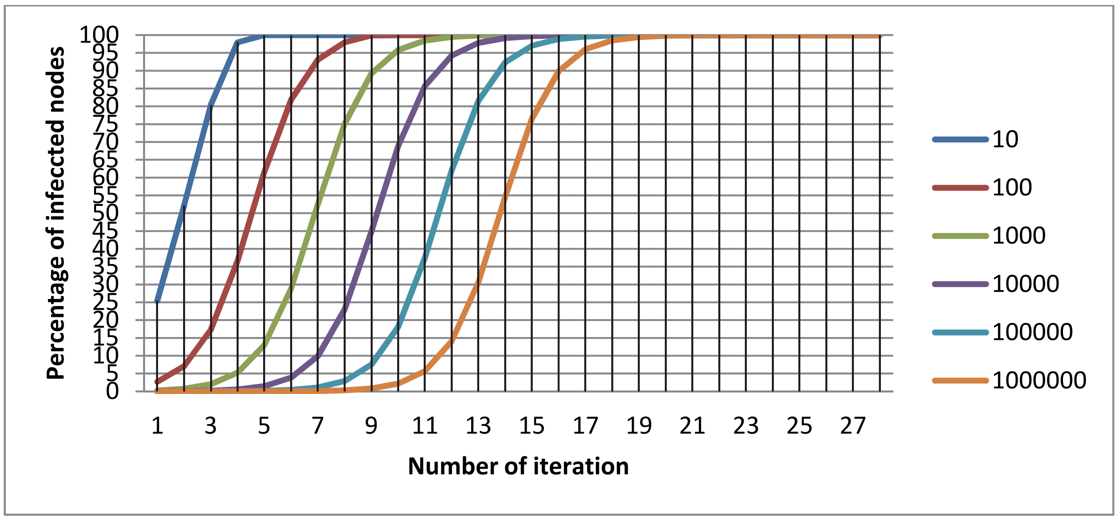

Figure 5. Percentage of infected nodes In each of the iterations. Source: [1]

The chart shows that after the seventeenth iteration system coherence for the number of nodes less than a million is more than ninety percent, and from the nineteenth iteration more than ninety-nine percent.

3. Summary

Based on the results of the tests carried out and the calculations made, it can be argued that the Gossip algorithm is perfectly suited for the spreading of information in distributed systems. However, it should be taken into account that the effectiveness of the algorithm depends on many factors. The topology of connections between nodes has a fundamental influence on the operation of the gossip algorithm. In hierarchical structures, which is a tree topology surveyed, information needs more iteration to reach all the nodes than in topology of complete graph. In the tree structure, the fact from which of the nodes information will be sent is not without significance. The deeper the infected server is in the network structure, the more iterations are needed to bring about system integrity. The choice of algorithm's operating modes also affect onthe efficiency of its work. The fastest mode is hybrid mode. Slightly worse results can be observed for the pull mode. The weakest effects gives the use of the push mode. The hybrid mode, unfortunately, is not free of drawbacks. The main one consists in sending a large number of network packages. Each of the nodes operating in this mode, during each iteration, sends out one to two packages. The first package is responsible for sending a request message. The second is sent if the table of the information is not empty. The amount of packets sent can be significantly reduced using the SIR algorithm of the Gossip algorithm. It allows not to broadcast information that has already been sent to many nodes. There are several implementations that allow to set R status for the message, causing the message not to be distributed. The most effective implementation was the transition to the status of R with a constant probability P, in the case of obtaining message about redundant information. For the probability P from 0.2-0.5, the number of packages sent has been halved. Implementation allowing to determine the status R after sending redundant message n times gives slightly worse results. Random selection of nodes to set the R status for information is the least effective and seems to be the most risky of all described approaches. In specific situations incorrect selection of the node may cause that the system will not achieve the final coherence. This happens when the node being drawn is the only one that has a connection to a part of the system and will not be able to send information there. I think that the interesting aspect of the Gossip algorithm is its scalability. In the hybrid mode and the full graph topology, I did not notice a significant difference in the number of needed iterations while sending one message for six nodes and sending two messages to ten nodes.

During the study of the Gossip protocol in the push mode, in particular in the hierarchical network topology, one could notice a large dependence of the protocol's effectiveness on the selection of neighboring nodes when sending a message. In the case of two neighbors, the probability of drawing each of them is equal to 0.5. It may happen that in the course of several iterations, the same neighbor will be drawn. This will result in the information not reaching the other of the adjacent nodes and extending the data transfer process. The solution to this problem is to create a local dictionary to determine for which nodes the information was sent. This would eliminate from the drawing the nodes that received the message from the draw in subsequent iterations. The described problem does not occur when we have many adjacent nodes, because the probability of drawing each of them becomes smaller as the number of connected nodes increases. It is obvious that during the implementation of the Gossip algorithm, special attention should be paid to the problems of mutual distributed exclusion. Exclusion can occur when multiple nodes at the same time try to send a message to the same node. The solution may be to use of an algorithm that solves this problem, such as the Ricard Agrawal’s algorithm based on requests. It should be mentioned that the Gossip algorithm is highly scalable, as shown by mathematical methods. This favors the popularity of solutions based on the this algorithm. It is used, among others, in the Apache Cassandra software, which is a free tool for managing no-SQL distributed databases. Another system using the epidemiological algorithm is SERF [10], used to manage server farms. The Gossip protocol has also been applied to Amazon Web Services [4].

References

1. Sławomir Zborowski, „Algorytmy w Chmurach. Część 1: Gossip” Programista, numer 1/2016 (44).
2. http://alvaro-videla.com/2015/12/gossip-protocols.html
3. http://sbrc2010.inf.ufrgs.br/resources/presentations/tutorial/tutorial-montresor.pdf
4. http://status.aws.amazon.com/s3-20080720.html
5. https://www.cs.cornell.edu/courses/cs6452/2012sp/papers/gossip-podc05.pdf
6. https://www.cis.upenn.edu/~bcpierce/courses/dd/papers/demers-epidemic.pdf
7. http://www.inf.u-szeged.hu/~jelasity/dr/doktori-mu.pdf
8. https://managementfromscratch.wordpress.com/2016/04/01/introduction-to-gossip/
9. http://www.prismmodelchecker.org/casestudies/gossip.php
10. https://www.serf.io/docs/internals/gossip.html