Autonomous and scalable failure detection in distributed systems

Benjamin Satzger*, Andreas Pietzowski and Theo Ungerer

Department of Computer Science,
University of Augsburg,
D-86135 Augsburg, Germany
E-mail: satzger@informatik.uni-augsburg.de
E-mail: pietzowski@informatik.uni-augsburg.de
E-mail: ungerer@informatik.uni-augsburg.de
*Corresponding author

Abstract: The growing complexity of distributed systems makes it more and more difficult to manage them. Therefore, it is necessary that such systems will be able to adapt autonomously to their environment. They should be characterised by so-called self-x properties such as self-configuration or self-healing. The autonomous detection of failures in distributed environments is a crucial part for developing self-healing systems. In this paper, we introduce algorithms to form monitoring relations and propose to utilise these for a scalable autonomous failure detection. The evaluation of the developed algorithms indicates that they are suitable for complex, large scale and distributed systems.

Keywords: autonomous; scalable; adaptive; failure detection; distributed systems; grouping; monitoring; self-healing.

Reference to this paper should be made as follows: Satzger, B., Pietzowski, A. and Ungerer, T. (2011) 'Autonomous and scalable failure detection in distributed systems', *Int. J. Autonomous and Adaptive Communications Systems*, Vol. 4, No. 1, pp.61–77.

Biographical notes: Benjamin Satzger is a Researcher in the Department of Computer Science, University of Augsburg. Currently, he is working on techniques for self-organising and self-healing systems.

Andreas Pietzowski was a Researcher at the University of Augsburg and is now working in a software company.

Theo Ungerer is Chair of Systems and Networking at the University of Augsburg. Since 2003, he is also a Scientific Director of the Computing Center of the University of Augsburg. His current research interests are in the areas of embedded processor architectures, embedded real-time systems, ubiquitous systems and organic computing.

1 Introduction

The complexity of computer systems is steadily rising and especially distributed systems interconnect growing numbers of more and more complex heterogeneous devices. IBM has identified this trend as one of the major obstacles for the progress in the IT industry (Horn, 2001). The initiatives organic computing (OC) (Schmeck, 2005) and autonomic computing (AC) (Horn, 2001; Kephart, 2005) postulate so-called self-x properties for these systems to keep them manageable. To achieve these goals both the OC (Richter et al., 2006) and the AC community (Kephart, 2005) regard monitoring information as a basis for organic or autonomic systems.

In this paper, we propose algorithms to establish monitoring relations within a distributed system that enable them to monitor themselves in a scalable way. Amongst others, such monitoring relations can be used for a scalable failure detection. This paper is organised in seven sections. Section 2 introduces the concepts of failure detectors and it is discussed that such algorithms are usually reduced to one node monitoring another. Therefore, a further component is needed that determines who will monitor whom. Related work is highlighted in Section 3. In Section 4, we introduce the generic, novel concept of a monitoring network and based on that we state the problem of establishing monitoring relations. Then, Section 5 describes three algorithms we have developed to solve this problem. Section 6 presents the results of the evaluations we have conducted to compare the algorithms and finally, Section 7 concludes this paper.

2 Failure detectors

Failure detectors generally provide information on failures of components of distributed systems. This represents a crucial but non-trivial problem. Several impossibility studies (Chandra et al., 1996; Fischer et al., 1985; Lynch, 1989) show that perfect failure detectors cannot exist in asynchronous distributed systems. The major reason is the impossibility to distinct with certainty whether a process has failed or the communication network is just slow. Adaptive failure detectors (Chen et al., 2000; Fetzer et al., 2001; Hayashibara et al., 2004) are able to adjust changing network conditions. The behaviour of a network can be significantly different during high traffic times as during low traffic times regarding the probability of message loss, the expected delay for message arrivals and the variance of this delay.

Typically, distributed systems consisting of a finite set of processes or nodes are considered with a local failure detector attached to each process (see for example Chandra and Toueg (1996). Failure detectors identify processes they are suspecting to have crashed. For a better insight, in the following one specific failure detection algorithm (Satzger et al., 2007a,b) is presented.

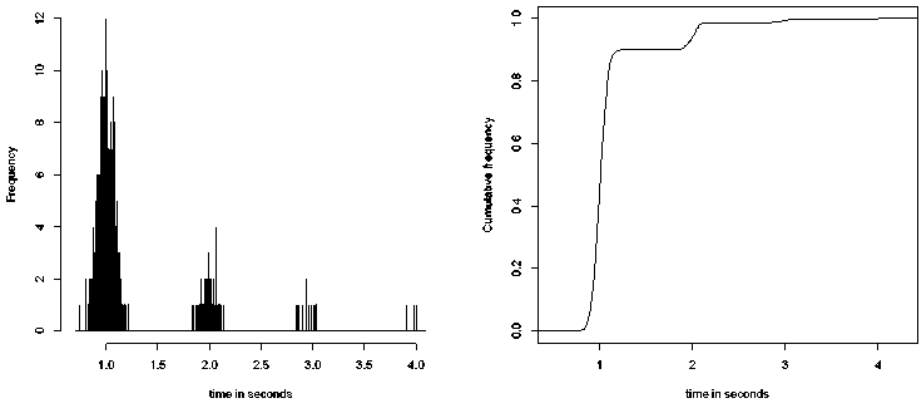
As system model two processes p and q are considered, which are connected by a communication channel. They can only communicate by sending and receiving messages. Suppose that process p is monitoring q and q sends heartbeat messages to p for every $\Delta_i = 1$ sec. Process p manages a sampling window S with information about the inter-arrival times of the last 1,000 heartbeats it has received. At a certain point during runtime S could have the form: [1.083s, 0.968s, 1.062s, 0.861s, …].

Furthermore, p stores the time of the last received heartbeat called *freshness point f*. Based on the sampled inter-arrival times and f, the algorithm estimates the probability that q has failed. On the left hand, Figure 1 shows the values of S as a histogram. The shape of the histogram depends mainly on Δ_i and the communication channel connecting q and p. In this particular example, Δ_i is one second and the communication channel has a message loss rate of 10% and a certain fluctuating message sending delay. The peak at two seconds arises from one lost heartbeat message, the peak at three seconds arises from two consecutive lost heartbeat messages, and so forth.

This histogram can be seen as an approximation of the *probability density function* of the distribution of the inter-arrival times. Based on this histogram, the cumulative frequencies of the values in S are easily computable. The cumulative frequencies in turn can be seen as an approximation of the corresponding *cumulative distribution function*. A cumulative distribution function (CDF) completely describes the probability distribution of a real-valued random variable, in our case the inter-arrival times of the heartbeat messages. The CDF $F(t_\Delta) = P(X \leq t_\Delta)$ represents the probability that an inter-arrival time takes on a value less than or equal to t_Δ. The values of the CDF are also a reasonable indicator for the crash of q. Assume p is waiting since time t for the next heartbeat from q. $F(t_\Delta) = x$ means 'p is waiting for the next heartbeat message since t_Δ seconds and the probability that no further heartbeat message arrives is x'. The longer p is waiting for the next heartbeat the higher the values of F will be.

On the right side, Figure 1 shows the cumulative frequencies of the values in S that is used as an estimation of the real CDF of the inter-arrival times and to compute a suspicion value for the failure of q. Like many other algorithms, the discussed failure detector only deals with one process/node monitoring another. To allow the nodes of a distributed system to monitor themselves, a strategy is needed to determine who is monitoring whom. The simplest way would be to let any two nodes monitor each other. Obviously, within complex distributed systems, this is not practicable due to scalability reasons. In this work, strategies are proposed to install monitoring relations among the components of a distributed system in a way to maintain scalability. In the following section, a survey of approaches for scalable failure detection is given.

Figure 1 A failure detection algorithm



3 Related work

To supply adequate support for large scale systems, *hierarchical* failure detectors define some hierarchical organisation. Bertier et al. (2003) introduce a hierarchy with two levels: a local and a global one, based on the underlying network topology. The local groups are LANs, bound together by a global group. Every local group elects one leader that is member in the global group. Within each group any member monitors all other members. Different from Bertier et al. (2003), in this work the existence of some classifying concept like a LAN is not required. Monitoring relations can also be built within a network of equal nodes.

Gossiping is a method of information dissemination within a distributed system by information exchange with randomly chosen communication partners. Baker and Shostak (1972) discussed a gossiping system with ladies and telephones. They investigated the problem of *n* ladies, each of them knows some item of gossip not known to the others. They use telephones to communicate, whereas the ladies tell everything they know at that time whenever one lady calls another. The problem statement was 'How many calls are required before each lady knows everything?'. Demers et al. (1987) pioneered gossiping in computer science as a way to update and ensure consistent replicas for distributed databases.

Van Renesse et al. (1998) have been the first using gossiping for failure detection to cope with the problem of scalability. In their basic algorithm, each process maintains a list with a heartbeat counter for each known process. At certain intervals every process increments its own counter and selects a random process to send its list to. Upon receipt of a gossip message the received list is merged with its own list. Each process also maintains the last time the heartbeat counter has increased for any node. If this counter is increased for a certain time then the process is considered to have failed. In addition to this basic gossiping, the authors specify a multi-level gossiping algorithm that does not choose the communication partners completely random but dependent on the underlying network. Basically, they try to concentrate the traffic within subnets and to decrease it across them. Thus, the scalability can be further improved. A disadvantage is that the size of gossip messages grows with the size of processes that causes a relatively high network traffic. Furthermore, the timeout to prevent false detections has to be rather high and since every process checks failures of processes by its own, false detections cause inconsistent information.

The SWIM protocol, based on the work of Gupta et al. (2001) and described in a paper of Das et al. (2002), faces the mentioned drawbacks as it uses a separate failure detector and failure dissemination component. The failure detector component detects failures while the dissemination component distributes information about processes that have recently either left, or joined, or failed. Each process periodically sends a ping message to some randomly chosen process and waits for it to request. In this way, failures can be detected and are then disseminated by a separate gossip protocol. The separation of failure detection and further components as proposed in Das et al. (2002) is taken up in this work. While in the previous section a failure detection component is introduced, here it is dealt with the dissemination component.

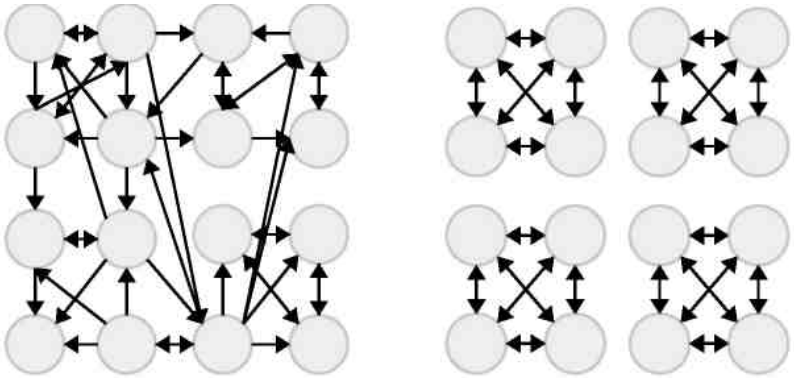
Horita et al. (2005) present a scalable failure detector that creates dispersed monitoring relations among participating processes. Each process is intended to be monitored by a small number of other processes. In almost the same manner as in systems mentioned above, a separate failure detection and information propagation are used. Their protocol tries to maintain each process being monitored by k other processes. As a typical number for k they declare 4 or 5. When a process crashes, one of the monitoring processes will detect the failure and propagate this information across the whole system. In addition to the description of their failure detector, Horita et al. compare the overheads of different failure detection organisations in their paper. The grouping mechanism of Horita et al. (2005) is based on a random construction of monitoring relations. Each node selects a certain amount of randomly chosen nodes, which then serve as its surveillants. Hence, it is not taken into account how well a node is suited to monitor another. One motivation for this work is to take such an optimality criterion into account.

In the following, monitoring networks and algorithms to form monitoring relations are introduced and evaluated. This approach is independent from the used monitoring component but can be used together with a failure detector as introduced above. The separation of the monitoring itself and the formation of monitoring relations allows to create generic services. As clarified above, the separation of information propagation and monitoring has been identified as an important characteristic by many researchers. In the area of scalable failure detectors, the consideration of the suitability of monitoring relations has been neglected so far. For instance, the work of Horita et al. (2005) proposes to choose surveillants *randomly*. To take suitability information into account can improve the performance and reduce the overhead of monitoring components like failure detectors.

4 Monitoring networks

A *monitoring network Net*, a network of monitoring relations, is represented as a triple (N, M, s), where N is the set of nodes/processes of a network, $M \subseteq N \times N$ is the monitoring relation and s is a function from N x N to a real value within [0, 1]. For each tuple $(u, v) \in N \times N$, $s(u, v)$ is the suitability of node u to monitor node v. This suitability can depend on different aspects such as the latency of a connection, the reliability of a node, its load and so on. If a node u is not able to monitor another node v at all, $s(u, v)$ should output 0. The monitoring relation defines which monitoring relations are established, that is $(u, v) \in M$ means node u is currently monitoring node v. $(u, v) \in M$ is also denoted with $u \to v$. The relation M is irreflexive, that is it is not allowed that a node is monitoring itself. The term $\overset{*}{\to} v$ is defined as all nodes monitoring v, that is $\overset{*}{\to} v := \{u \in N \mid u \to v\}$. Similar, $u \overset{*}{\to}$ outputs all nodes u is monitoring, that is $u \overset{*}{\to} := \{v \in N \mid u \to v\}$.

Figure 2 Types of monitoring relations



The task of a grouping algorithm is basically, given a monitoring network Net = (N, M, s) and a positive integer $m < |N|$, to establish monitoring relations such that every node of the network is monitored by at least m nodes. In this work, two flavours of this problem are distinguished, namely, *individual monitoring relations* also called *dispersed monitoring relations* and *closed monitoring groups*. In the former, monitoring relations can be set for each node individually while in the latter nodes form groups with mutual monitoring relations. The number m of surveillants for each node can be defined by the user. Typically, a higher number of surveillants not only provide a higher reliability but also cause a higher overhead. In Figure 2 (left), an instance of individual monitoring relations of a monitoring network is illustrated with $m = 3$. Thereby, the illustration of the suitability information has been omitted. Figure 2 (right) shows a corresponding partition of a network into monitoring groups. In the following, problem definitions of establishing individual monitoring relations and monitoring groups are given.

4.1 Individual monitoring relations

A positive integer m, where $m < |N|$, establishes monitoring relations M such that $\forall n \in N$ holds $|\overset{*}{\rightarrow} n| = m$. This means each node is monitored by m other nodes. Furthermore, the algorithm should maximise the suitability of the grouping to establish adequate monitoring relations. Therefore, the term

$$\sum_{\upsilon \in N} \sum_{u \in \overset{*}{\rightarrow} \upsilon} s(u, \upsilon)$$

should be maximised by the grouping algorithm. The optimisation of the suitability is a quality criterion for grouping algorithms, but it is not postulated that the algorithms output an optimal solution as it is more important to find solutions in all cases as fast as possible. The term *monitoring group* or simply *group* in the context of individual monitoring relations can be understood as all nodes monitoring one particular node, whereas the latter is the leader of the group. Thus, in a network of n nodes there are also n groups: each node $\upsilon \in N$ is the leader of the group $\{\upsilon\} \cup \overset{*}{\rightarrow} \upsilon$.

4.2 Closed monitoring groups

Different from the dispersed individual monitoring relations, a closed monitoring group is a group of nodes in which all members monitor each other. In addition to the individual monitoring relations, constraints regarding the monitoring relations M are holding: M must be symmetric and transitive in order to produce closed monitoring groups. In another point, the problem of finding monitoring groups is relaxed, compared to individual monitoring relations, as it is not always possible to find groups of the size $m + 1$ resulting in m surveillants per node in the group. If for instance a network has three nodes and monitoring groups of size 2 need to be established, this leads to an unsolvable problem. For such cases, also closed monitoring groups of bigger sizes are allowed. In detail, the problem $\forall\ n \in N$ holds $|\overset{*}{\rightarrow} n| = m$ is relaxed to $\forall\ n \in N$ holds $|\overset{*}{\rightarrow} n| \geq m$. A very simple solution to this problem is to combine the whole network into one group. This is a valid solution as just $|\overset{*}{\rightarrow} n| \geq m$ is postulated. However, the number of surveillants per node should be as close as possible to m. This represents a soft constraint similar to the maximisation of the suitability criterion.

Two nodes are in the same closed monitoring group if they are monitoring each other. An additional requirement for such monitoring groups is that each group has one node, which is declared as leader. Such a role is needed by many possible applications based upon grouped nodes, for example to have one coordinator or contact for each group. An instance where one leader per group is necessary is the formation of hierarchical groups. Whether individual monitoring relations or monitoring groups are more adequate depends on the environment and the monitoring task.

5 Grouping algorithms

Three grouping algorithms are introduced in this section, one to establish individual monitoring relations, two to form closed monitoring groups. The algorithms are tailored to solve these problems in a distributed manner. Furthermore, it is not assumed that all nodes have information about all other nodes what would simplify the problem significantly. The nodes of a self-monitoring network Net = (N, M, s) do not know about the suitability s, that is how suitable other nodes are to monitor it, until they receive a message from a node with information about that. The suitability also might change over time. In the following, the usage and relevance of suitability metrics for monitoring relations are discussed. Then, three algorithms are presented that provide the desired grouping capabilities. To be able to establish suitable monitoring relations, the nodes of a network need information about each other. Such information might be the quality of the network connection of two nodes, the reliability of a node, and so on. Each node is holding relevant information about a number of other nodes to allow the computation of suitability information.

The establishment of monitoring relations within a network Net = (N, M, s) can be based on different aspects. Therefore, the suitability function s has to be defined accordingly. Note that the suitability information typically is not computable before nodes receive information from other nodes. If it is, for instance, desired that nodes should be monitored by nodes with a similar hardware equipment and a fast network

connection, the suitability function could be set to $s(u,\upsilon) = [h(u,\upsilon) + n(u,\upsilon)]/2$, where $h(u, \upsilon)$ returns a value within [0, 1], indicating the similarity of the hardware equipment of u and υ, and $n(u, \upsilon)$ returns a value within [0, 1], indicating the performance of the network connection. Such a scenario would make sense if a fast network connection improves the monitoring quality and in the case of an outage of a node, another node with similar hardware equipment is likely to have the ability to inherit the tasks of the failed node. Thus, the setting of the suitability function influences the establishment of monitoring relations. The definition of a suitability function should reflect the requirements of a monitoring system. All relevant factors should be included and weighted according to its importance.

Now, three algorithms to establish monitoring relations in an autonomous distributed way are presented: INDIVIDUAL, which constructs individual monitoring relations, MERGE and SPECIES, which install monitoring groups. The idea of INDIVIDUAL is very simple: each node tries to identify the m most suitable nodes and asks them to monitor it. In the initial state of the algorithm MERGE, each node forms a group consisting of one node which it is leader of. Groups merge successively until they reach a size greater than m. SPECIES distinguishes between the two species *leader* and *non-leader*. The specificity of a node is random-driven. Non-leaders try to join a group whereas each group is controlled by one leader. In the case of an inadequate ratio of leaders to non-leaders, nodes can change its specificity. This paper presents the basic functionality of the algorithms. For a formal listing, it is referred to (Satzger and Ungerer, 2008) where you can find the pseudocode of all three algorithms.

5.1 Individual

Individual monitoring relations denote monitoring responsibilities set individually for each node. Using the suitability function, nodes can identify suitable surveillants. The most suitable ones are asked to monitor it. Therefore, nodes send monitoring requests to other nodes and wait for their acknowledgement. This process is repeated until the node has established m acknowledged monitoring relations.

A further requirement for individual grouping algorithms, which is omitted here could be that each node u monitoring a node υ needs to know all other nodes also monitoring υ, that is if $u \rightarrow \upsilon$ then u needs to know the set $\overset{*}{\rightarrow}\upsilon$. This might be necessary as in the case of a failure of υ, all monitoring nodes could for example, have to hold some kind of vote to gather a consistent view and to plan repairing actions, respectively. This feature of closed monitoring groups could easily be integrated into INDIVIDUAL. This has not been done in order to investigate the more general algorithm as stated here.

5.2 Merge

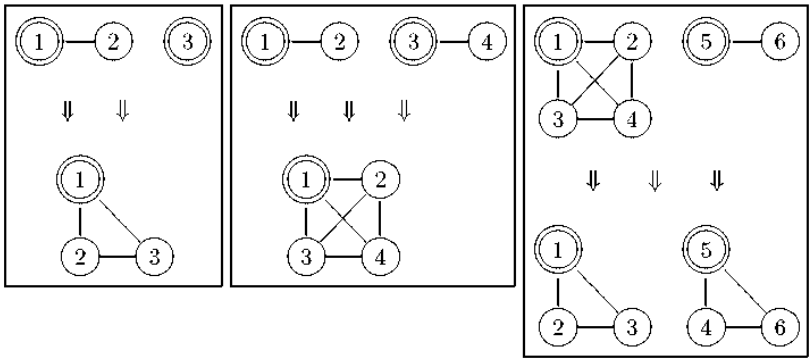
In this section, the MERGE algorithm is discussed that establishes closed monitoring groups. Within these groups all nodes monitor each other. Every group has a group leader. Typically, the initial situation is a monitoring network Net = (N, S, Ø) without monitoring relations and a number m that determines the desired number of surveillants. During the grouping of the nodes into monitoring groups, existing groups smaller than $m + 1$ merge with other groups until the resulting group has enough members. Due to this mechanism the maximal size can be limited by $2\cdot(m + 1)–1$. If there exists, for

example, a group of size $2\cdot(m + 1)$, it can be split into two groups of valid size $m + 1$. The group leaders that belong to a monitoring group smaller than $m + 1$, ask suitable other group leaders to merge their groups. If this request is accepted the groups merge and the requesting group leader must give off its leadership. The requested group leader is the leader of the newly formed group. After such a merging process, the group leader informs all members about the new group. Nodes that have lost the leadership adopt a completely passive role in the further grouping process and are not allowed to accept merging request from other leaders anymore.

Let us consider an example where m is 2, that is groups of minimum size 3 are formed. In Figure 3 (left), two groups are examined, one consisting of Nodes 1 and 2 whereas Node 1 is leader and the other group consisting only of Node 3. Node 3 is requesting the group of Node 1 to merge. After the merge process, a new group is formed with exactly 3 members and Node 1 is leader of that group. Merge requests are never denied by leaders. Thus, as you can see in Figure 3 (middle), it is possible that groups that emerge have more than the desired $m + 1$ members.

If groups become greater or equal to $2\cdot(m + 1)$, as illustrated in Figure 3 (right), a splitting is performed resulting in two monitoring groups that both have at least m+1 members, what is enough to stop active merging activities. Thus, the resulting group sizes of the MERGE algorithm are always between $m + 1$ and $2\cdot(m + 1) - 1$.

Figure 3 Functionality of the MERGE algorithm



5.3 Species

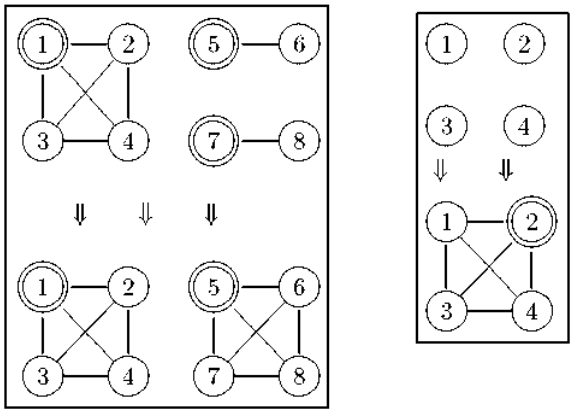
Like the MERGE algorithm, SPECIES also installs closed monitoring groups. It is based on the existence of two species: *leader* and *non-leader*. Leaders are group manager and each group contains exactly one leader. Non-leaders contact the most suitable leader trying to join its group. The specificity of a node is random-driven and dependent on the value of m. Consider a network consisting of n nodes. The optimal number of leaders is $(n/m+1)$ as the following example illustrates: Within a small network of 12 nodes, closed monitoring groups need to be installed with $m = 2$, that is two surveillants per node or groups of size three. The optimal case for that are four groups of size three. Thus, $(n/m+1) = (12/3) = 4$ leaders are needed and so non-leaders can join. Therefore, the SPECIES algorithm selects every node as leader with probability $(1/m+1)$ and non-leaders otherwise. As it is worse to have too many leaders than too few, the

probability of a node to become a leader can be adjusted to, for example $(0.8/m+1)$. However, the random assignment of species to nodes does not guarantee a valid distribution into leaders and non-leaders. Thus, if a leader recognises that there are too many of them, they can toggle their species and transform into a non-leader. Vice versa, if nodes cannot find leaders to join they transform into a leader with a certain probability.

The network shown in Figure 4 (left) contains too many leaders. In this case, *m* is 3 that means groups of sizes of at least 4 need to be formed. However, this is not possible in this example. If no non-leader joins the groups smaller than 4, their leaders try to contact other leaders in order to find groups with enough members to poach some non-leaders. If this also fails, leaders then transform to non-leaders with a certain probability. This happens with Node 7 in this example. After that transformation a valid grouping is possible.

Figure 4 (right) shows the contrary situation as above; too few leaders are available, in this case even none. If non-leaders are unable to find any leader, they become leader with a certain probability. There are two cases how non-leaders join a group. If they do not belong to a group yet, they themselves care to find a group and join it. Leaders controlling an undersized group try to find oversized groups and ask their leaders to handover non-needed members.

Figure 4 Functionality of the SPECIES algorithm



6 Evaluation

An evaluation for the above introduced algorithms is provided in this section. For the purpose of evaluating and testing, a toolkit has been implemented, which is able to simulate distributed algorithms based on message passing. It is written in Java and allows the construction of networks consisting basically of nodes, channels that connect two nodes, and algorithms running on nodes. As the simulation runs on one single computer, a random strategy selects the next node whose algorithm is executed partially. Thus, the asynchronous behaviour of distributed systems is covered. It is assumed that the communication channels do not drop messages and deliver them in the correct order.

The nodes of the monitoring network Net = (*N*, *M*, s) used for the evaluation are theoretically arranged as a grid. The distance of two nodes *u*, *v* within the grid determines

their mutual monitoring ability. Thus, the suitability has been set to the reciprocal value of the Euclidean distance of the nodes within the grid. The nodes have sufficient information only about a certain number of nodes to compute a suitability value. This models the concept that in many networks nodes do not know everything but have a limited view.

The introduced grouping algorithms are evaluated within different scenarios. The evaluation focuses on the scalability of the establishment of monitoring relations, the optimality of the relations regarding the suitability metric, and the failure tolerance of a system if failure detectors are used together with the grouping approach. The evaluations have been conducted using different sets of parameters like the values for the desired number of surveillants m and the amount of information about other nodes. Each evaluation scenario has been replayed 1,000 times whereas the results have been averaged.

Recall, a monitoring network Net is represented as (N, M, s), where N is the set of nodes of a network, $M \subseteq N \times N$ is the monitoring relation, and s is a function from $N \times N$ to a real value within [0, 1]. The task of a grouping algorithm is, given a positive integer $m < |N|$, to establish monitoring relations such that every node of the network is monitored by at least m nodes.

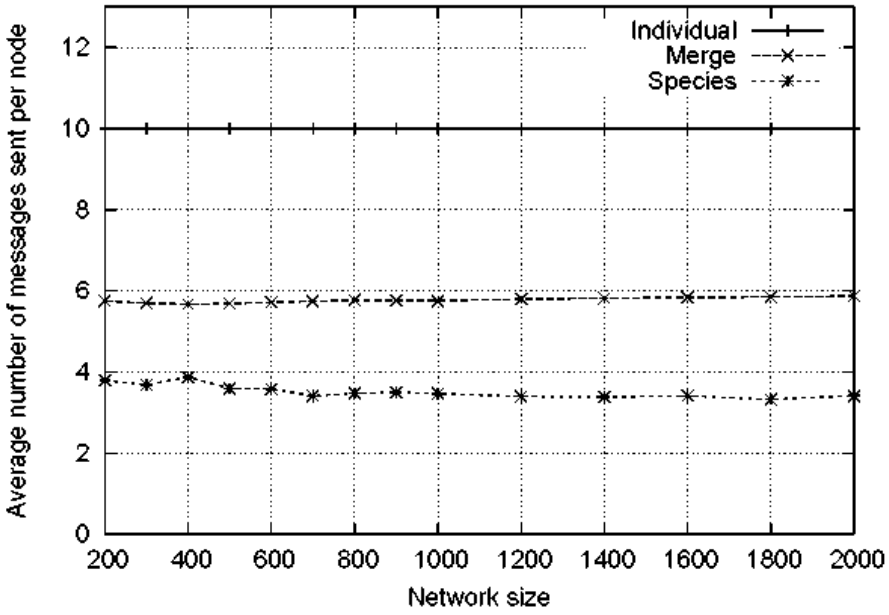
As suitability function $s(u, v)$, the reciprocal value of the Euclidean distance of the nodes u and v is used. At the beginning the monitoring relation is empty, that is $M = \phi$. This means that the network is in a state where no monitoring relations are established yet. To model the fact that nodes do not have a complete view of the whole network, the value κ describes the part of the network each node is aware of. A value of $\kappa = 10$ means that each node has information about 10 randomly chosen nodes. This information is assumed to have been gathered by past communication processes with these nodes. In the following, the results of the conducted evaluations are presented.

6.1 Scalability

Messages need to be sent to establish monitoring relations. In the following, this overhead is evaluated for the proposed grouping algorithms. All experiments have been conducted according to the description given above. First, the scalability regarding the network size is evaluated. In this experiment, the number of desired surveillants m is set to 5 while each node knows 50 other nodes, that is $\kappa = 50$. Figure 5 shows the results of this experiment, whereas the values on the x-axis stand for the network size and the average number of messages sent by each node is depicted on the y-axis.

As m is 5, each node executing the INDIVIDUAL algorithm needs 10 messages, 5 monitoring requests and 5 responses. MERGE needs less than 6 messages and SPECIES less than 4. The results indicate that all three algorithms can be classified as being independent from the network size, as the nodes basically do not send more messages within a bigger network. The algorithm SPECIES performs even better in bigger networks. The reason for this behaviour is the random-driven determination of the specificity. The aim of that process is to achieve a division into leaders and non-leaders of a defined ratio. In general, the bigger the network the better this ratio is met. Thanks to the independence of the overhead caused by the grouping algorithms from the network size, all introduced algorithms seem suitable to be applied within complex distributed systems.

Figure 5 Scalability of grouping algorithms regarding network size ($\kappa = 50$)



The next section examines the monitoring relations with respect to their suitability according to the suitability function. All following evaluations are conducted with a network size of 1,000 nodes.

6.2 Suitability

As stated in the specification, the algorithms are supposed to take the suitability of the nodes into account. This means the term

$$\sum_{v \in N} \sum_{u \in \overset{*}{\rightarrow} v} s(u, v) \tag{1}$$

should be maximised. The average suitability within the evaluation network is about 0.09. This means a random grouping produces monitoring relations of about that value. Figures 6 and 7 show the results of the experiments concerning the suitability of the algorithms for different values of κ (10 and 100). This parameter represents the size of the nodes' view on the network. The x-axis represents the number of surveillants per node, the y-axis depicts the average suitability of the formed groups based on Equation 1. In all cases, smaller groups tend to result in better suitability values. SPECIES and especially MERGE handle grouping with limited information very well. In the case of full information about the network ($\kappa = 1{,}000$), INDIVIDUAL would perform optimal.

Figure 6 Suitability of grouping algorithms ($\kappa = 10$)

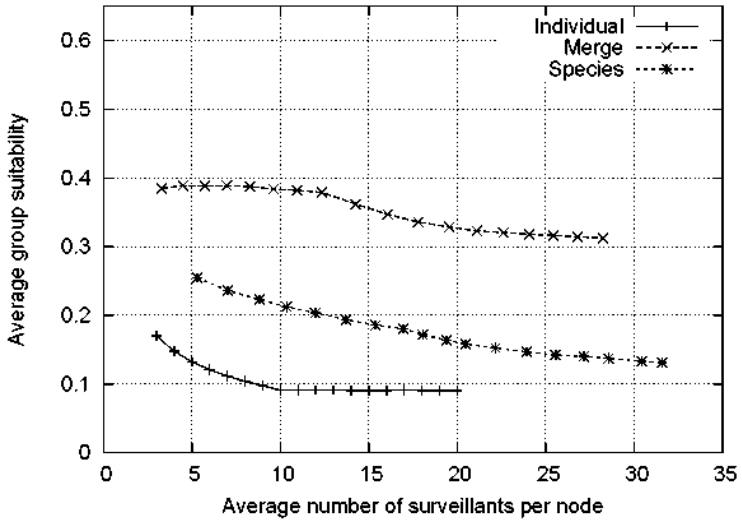
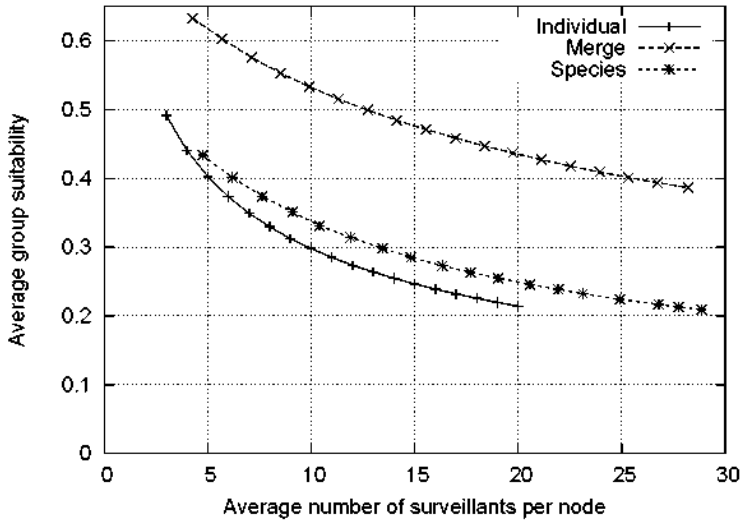


Figure 7 Suitability of grouping algorithms ($\kappa = 100$)



6.3 Failure tolerance

In this section, the gain of applying the proposed grouping techniques with respect to failure tolerance is investigated. To evaluate the failure tolerance of the monitoring relations, the following methodology is used. It is assumed that a certain percentage of randomly chosen nodes within the network fail simultaneously, that is they crash and do not recover. Using failure detectors, nodes monitor each other according to the installed

monitoring relations by a grouping algorithm. It is assumed that failure detectors eventually detect the failure of a node. An undetected failure means the failure of a node that remains undetected. In this setting, this is only possible if a node and all its surveillants fail simultaneously.

The detection of a failure is the prerequisite of a subsequent repair or self-healing, respectively. If a node has no surveillant, its failure equals to an undetected failure. If in a network any node monitors all other nodes, only the complete failure of the whole network results in undetected failures. However, for more complex systems, the latter monitoring strategy typically introduces an excessive overhead. Before the evaluation results are presented, a short view on failure tolerance motivated by probability theory is given.

Let X be the number of elements within a set, $Y \leq X$ the number of elements within this set possessing a feature F, and $x \leq X$ the number of elements that are randomly chosen from the set. The probability of k elements with feature F being in the randomly chosen set is then

$$\frac{\binom{Y}{k}\binom{X-Y}{x-k}}{\binom{X}{x}},$$

according to the hypergeometric distribution (Feller, 1970). Considering a network Net = (N, M, s) and a number of surveillants per node of $m < |N|$. If ϕ random nodes of the network fail, where $m + 1 \leq \phi \leq |N|$, the probability for the undetected failure of a certain node is

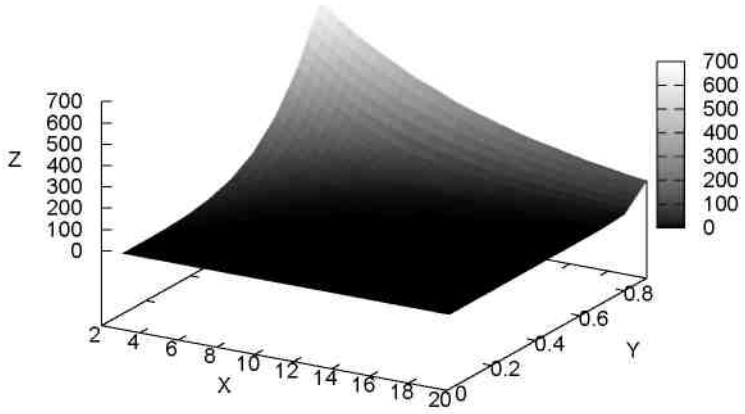
$$\frac{\binom{m+1}{m+1}\binom{|N|-m+1}{\phi-m+1}}{\binom{|N|}{\phi}} = \frac{\binom{|N|-m+1}{\phi-m+1}}{\binom{|N|}{\phi}}.$$

If ϕ is lower than $m + 1$, the probability for an undetected failure is obviously 0. If for instance $\phi = 10\%$ of the nodes of a network Net = (N, M, s) consisting of 100 nodes fail, whereas each node is monitored by $m = 3$ nodes, then the probability for a certain node $\eta \in N$ to fail undetectably is:

$$\frac{\binom{|N|-m+1}{\phi-m+1}}{\binom{|N|}{\phi}} = \frac{\binom{100-4}{10-4}}{\binom{100}{10}} \approx 5.10^{-5}.$$

The following simulations have been conducted as before with a network Net = (N, M, s) of 1,000 nodes. Monitoring relations are established with all three proposed grouping algorithms and different values for m. It is measured how many undetected failures occur if a certain percentage ϕ of random nodes fail. All three algorithms show the same basic behaviour while INDIVIDUAL performs slightly better than the other two algorithms. The algorithm INDIVIDUAL performs best because it has no variance in the number of surveillants. The parameter m exactly determines the resulting group size, that is the number of surveillants. For closed monitoring groups this number varies. Figure 8 presents the result using the example of Individual. The x-axis shows the average number of surveillants per node, the y-axis stands for the percentage of failed nodes and the z-axis the number of undetected failures.

Figure 8 Failure tolerance of grouping algorithm INDIVIDUAL



In all cases, the number of undetected node failures decrease with a higher number of surveillants and a percentage of node failures. As you can see, undetected node failures only occur with a low number of surveillants and many node failures. These results can be used as a utility to choose an adequate value for the number of surveillants, which is a balancing act between overhead and failure tolerance.

7 Conclusions

In this work, an approach for an autonomous, scalable failure detection in distributed systems has been presented. In order to develop generic and flexible failure detection services, the separation of the mutual failure detection task and the installation of monitoring relations are preferable. We have formulated a novel precise problem statement for the installation of monitoring relations, which takes suitability information into account. Three algorithms have been devised, which solve this problem and have been compared regarding their scalability, suitability, and the failure tolerance they are providing. The evaluation shows that the overhead of all proposed algorithms is independent from the network size. Therefore, they are suitable for complex, large scale and distributed systems. Each algorithm needs only a very limited number of messages per node in order to fully install monitoring relations. Existing algorithms dealing with similar problems are either too complex for self-healing systems or unable to consider the suitability of monitoring relations. The conducted evaluations and theoretical considerations regarding the failure tolerance can be used to determine the necessary number of monitoring nodes. The formation of hierarchical groups could further improve the monitoring relations and could be a starting point for future work.

References

Baker, B.S. and Shostak, R. (1972) 'Gossips and telephones', *Discrete Math*, Vol. 2, No. 3, pp.191–193.

Bertier, M., Marin, O. and Sens, P. (2003) 'Performance analysis of a hierarchical failure detector', in *Proceedings 2003 International Conference on Dependable Systems and Networks (DSN 2003)*, pp.635–644, San Francisco, CA, USA, IEEE Computer Society.

Chandra, T.D. and Toueg, S. (1996) 'Unreliable failure detectors for reliable distributed systems', *Journal of the ACM*, Vol. 43, No. 2, pp.225–267.

Chandra, T.D., Hadzilacos, V. and Toueg, S. (1996) 'The weakest failure detector for solving consensus', *Journal of the ACM*, Vol. 43, No. 4, pp.685–722.

Chen, W., Toueg, S. and Aguilera, M.K. (2000) 'On the quality of service of failure detectors', in *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2000)*, New York: IEEE Computer Society Press.

Das, A., Gupta, I. and Motivala, A. (2002) 'SWIM: SCALABLE weakly-consistent infection-style process group membership protocol', in *DSN*, pp.303–312, IEEE Computer Society.

Demers, A.J., Greene, D.H., Hauser, C., Irish, W. and Larson, J. (1987) 'Epidemic algorithms for replicated database maintenance', in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pp.1–12, Vancouver, British Columbia, Canada.

Feller, W. (1970) *An Introduction to Probability Theory and its Application*, Vol. 1, New York: John Wiley and Sons.

Fetzer, C., Raynal, M. and Tronel, F. (2001) 'An adaptive failure detection protocol', in *PRDC '01: Proceedings of the 2001 Pacific Rim International Symposium on Dependable Computing*, p.146, Washington, DC, USA. IEEE Computer Society.

Fischer, M.J., Lynch, N.A. and Paterson, M.S. (1985) 'Impossibility of distributed consensus with one faulty process', *Journal of the ACM*, Vol. 32, No. 2, pp.374–382.

Gupta, I., Chandra, T.D. and Goldszmidt, G.S. (2001) 'On scalable and efficient distributed failure detectors', in *PODC: 20th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*.

Hayashibara, N., Défago, X., Yared, R. and Katayama, T. (2004) 'The f accrual failure detector', *SRDS*, pp.66–78, IEEE Computer Society.

Horita, Y., Taura, K. and Chikayama, T. (2005) 'A scalable and efficient self-organizing failure detector for grid applications', in *SC'05: Proc. The 6th IEEE/ACM International Workshop on Grid Computing CD*, pp.202–210, Seattle, Washington, USA, IEEE/ACM.

Horn, P. (2001) *Autonomic Computing: IBM's Perspective on the State of Information Technology*. Available at: http://www.research.ibm.com/autonomic.

Kephart, J.O. (2005) 'Research challenges of autonomic computing', in *ICSE '05: Proceedings of the 27th International Conference on Software engineering*, pp.15–22.

Lynch, N. (1989) 'A hundred impossibility proofs for distributed computing', in *PODC '89: Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, pp.1–28, New York, NY: ACM Press.

Renesse, R.V., Minsky, Y. and Hayden, M. (1998) 'A gossip-style failure detection service', *Technical Report TR98-1687*, Cornell University, Computer Science.

Richter, U., Mnif, M., Branke, J., Müller-Schloer, C. and Schmeck, H. (2006) 'Towards a generic observer/controller architecture for organic computing', in C. Hochberger and R. Liskowsky (Eds), *INFORMATIK 2006 – Informatik für Menschen*, volume P-93 of *GI-Edition – Lecture Notes in Informatics*, pp.112–119, Bonn, Germany, Köllen Verlag.

Satzger, B., Pietzowski, A., Trumler, W. and Ungerer, T. (2007a) 'A new adaptive accrual failure detector for dependable distributed systems', in *SAC 2007: Proceedings of the 22nd ACM Symposium on Applied Computing*, pp.551–555, New York, NY, USA: ACM.

Satzger, B., Pietzowski, A., Trumler, W. and Ungerer, T. (2007b) 'Variations and evaluations of an adaptive accrual failure detector to enable self-healing properties in distributed systems', in *ARCS 2007: Proceedings of the 20th International Conference on Architecture of Computing Systems*, volume 4415 of *Lecture Notes in Computer Science*, pp.171–184, Springer.

Satzger, B. and Ungerer, T. (2008) 'Grouping algorithms for scalable self-monitoring distributed systems', in *Autonomics 2008: Proceedings of the 2nd ACM/ICST International Conference on Autonomic Computing and Communication*.

Schmeck, H. (2005) 'Organic computing', *Künstliche Intelligenz*, Vol. 5, No. 3, pp.68–69.