Large-Scale Peer-to-Peer Autonomic Monitoring

João Leitão Liliana Rosa Luís Rodrigues
INESC-ID/IST
{jleitao,lrosa,ler}@gsd.inesc-id.pt

Abstract

The increasing scale and complexity of distributed system motivates the need for autonomous management. One of the key aspects in the management of distributed systems is the issue of component monitoring. Component monitoring is particularly challenging in large-scale dynamic systems, given the need to ensure that each component is monitored by at least one non-faulty component, despite joins, leaves, and failures, both at node and at network level. This paper proposes that components self-organize in an unstructured overlay network of constant degree in order to ensure that each component is always monitored by a threshold of other components.

I. Introduction

Computing environments are increasingly complex, featuring a larger and larger number of heterogeneous software and hardware components. One of the key aspects of the maintenance of such systems is the issue of component monitoring. Clearly, in large-scale system, it is impossible to perform such monitoring in a centralized manner, due to the huge load (in terms of processing and network traffic) imposed on the central monitoring component. However, the task of predefining a distributed monitoring scheme can be hopeless in a highly dynamic environment, given the constant changes in the number of components and in the network topology. The increasing complexity of this task can also overwhelm a human administrator, making the system prone to human error. This motivates the need for autonomous schemes for component monitoring, that are able to perform monitoring tasks from installation to maintenance, self-adapting to changes in the monitored system, following some high-level guidance. For instance, in a large data-center (with thousands of components) one may wish to monitor components to detect failures, in order to trigger human intervention in a timely manner. The definition of the monitoring relations across the several components in such a scenario can be overwhelming, namely when several components are added and removed from the system (*e.g.* for maintenance or upgrading purposes).

Autonomic computing deals with the inherent growing complexity of systems and components, addressing the problems introduced above. The idea behind autonomic computing is to automate the low-level and vital functions in a system, to alleviate experts from dealing with the burden of such complexity and heterogeneity. Therefore, the essence of autonomic computing is self-management, where the system is able to self-tune according to on-going changes, such as components, workloads, operational conditions, and user needs, despite the occurrence of failures. Self-management is achieved through self-configuration, self-optimization, self-healing, and self-protection [1]. The first refers to autonomous configuration of components, protocols, etc, according to specified target goals. The second refers to automated performance-tuning or operation improvement. The third refers to handling failures in the system. The fourth and last, focuses on the protection required to deal with malicious attacks and cascading failures. To achieve these goals, an autonomic system will need to continuously monitor itself.

Autonomic monitoring is vital for self-management, since it allows the system to assess its own current use and state. However, monitoring large-scale distributed systems that are faced with failures and changes due to reconfigurations is a hard task. We have already noted that large-scale prevents the use of centralized approaches. Also, large-scale prevents decentralized approaches where each component monitors every

This work was partially funded by FCT project REDICO – Dynamic Reconfiguration of Communication Protocols – (PTDC/EIA/71752/2006) through POSI and FEDER. Parts of this technical report were accepted for publication in the 2008 edition of the Distributed Autonomous Network Management Systems Workshop (DANMS'08).

other component. Furthermore, frequent changes in the system, due to runtime reconfiguration and updates, result in a dynamic distributed system, which prevent static or off-line monitoring assignments. Also, the monitoring overhead has to be acceptable for the system. Therefore, to achieve monitoring in a large-scale distributed system, it must be performed in a distributed manner, where each component monitors a small number of other components, while being resilient to frequent failures and network dynamism.

To address all these requirements, we propose a monitoring approach based on self-healing unstructured peer-to-peer overlay networks. The idea is that components can self-organize in an overlay with a given predefined neighbor degree, such that each component becomes responsible for monitoring its immediate neighbors in the overlay. The peer-to-peer overlay has the necessary flexibility to handle changes in network topology and dynamic settings, in an autonomous manner.

The rest of this paper is organized as follows: Section II describes the monitoring needs identified for large-scale distributed systems, and briefly overviews unstructured overlay networks and HyParView which are the basis for our proposal. Section III describes our architecture and depicts how it addresses the identified requirements. It also provides experimental results that illustrates some of the relevant properties of HyParView. Section IV overviews the related work and finally, Section V concludes the paper and presents some future directions of our work.

II. Overlay Networks for Monitoring

In this section we present the monitoring needs and relevant aspects of unstructured overlay networks, briefly describing HyParView, which builds and maintains such an overlay.

A. *Monitoring Needs*

Monitoring aims at maintaining information about the current system state, keeping track of changes in the target monitored properties. Knowing the system state is fundamental to perform the operations necessary to achieve the system goals with the desired performance. This section addresses the requirements and main challenges that autonomic monitoring faces in large-scale distributed systems.

In this work we propose a robust and autonomous monitoring architecture that works for large-scale, distributed, and dynamic networks, independently of data type and location. For that purpose there are several requirements that need to be satisfied.

The first requirement is that, for each component c in the system, there exists at any time at least one component m monitoring it. Given that failures may occur at any moment, for fault-tolerance, it is in fact desirable that each component is monitored by at least a threshold of other components. This threshold should be preserved despite the join and leave of new components in the system.

The second requirement is that the monitoring load is distributed by all the components in the system. This avoids that one, or only a small number, of components become responsible by monitoring a large portion of the system components as this could easily introduce processing or bandwidth bottlenecks in the monitoring architecture. As before, load-balancing should be preserved in face of components leaving, joining, or failing.

The third requirement is that the monitoring architecture must have the means to disseminate the relevant acquired monitoring data through the network. The dissemination mechanisms should also be highly scalable and resilient to faults, to ensure that the monitoring information (for instance alarms) reach the relevant components that are able to trigger corrective measures, despite the occurrence of faults.

We have already seen that to satisfy these requirements is a non-trivial task in large-scale dynamic systems. In particular, one needs to design a monitoring architecture that ensures that each component is permanently monitored by the desired threshold of components. Thus the monitoring overlay must self-adapt, such that when a new component joins, or a component leaves or fails, the remaining components re-distribute the monitoring load in a way that preserves such monitoring invariant. In the following text, we will show that there is an unstructured overlay network that have exactly the properties needed to satisfy these requirements. This overlay can be used not only to assign the monitoring tasks but also to disseminate the monitored information in a resilient manner when required.

B. Unstructured Overlay Networks

The peer-to-peer paradigm presents several advantages when compared with the classic client-server model, namely in terms of scalability, load balancing and fault-tolerance. A common strategy to organize and define communication patterns among peers in this paradigm is to rely in an overlay network. Overlay networks are logical networks supported, usually, by a membership service which maintains neighboring associations between nodes. One way to build such membership services is trough a peer samping service [2].

Maintaining full membership information has a prohibitive cost, not only due to the potential large size of system, but also due to the high cost of maintaining such information up-to-date in face of membership changes. Therefore, to ensure scalability, peer sampling services are usually implemented by building a local partial view, containing a sub-set of the full membership, at each node.

When the partial views of nodes are selected at random, the resulting overlay networks is said to be unstructured. This kind of overlay networks is commonly used for supporting efficient and reliable application level multicast [3], [4], as they present several interesting properties such as: scalability, autonomous operation, self-organization, and self-healing capabilities.

C. HyParView

The *Hy*brid *Par*tial *View* membership protocol, or simply, HyParView [3] is a fully decentralized membership protocol that builds and maintains an unstructured overlay network. The protocol was designed to support highly efficient and reliable application level broadcast protocols, with special interest in scenarios where large percentages of nodes may fail simultaneously.

Unlike other membership protocols, HyParView keeps two distinct partial views which are maintained using different strategies with different purposes[1].

A small symmetric *active view* with a size of $\log(n) + 1$ where n is the total number of nodes in the system, which is mainly used to support communication among peers. This view is maintained using a reactive strategy which means that it is only updated in response to some external event that affects the overlay (*e.g.* a node joining or leaving). TCP connections are maintained to each neighbor in these partial views. TCP is used as an unreliable failure detector [3], which facilitates the implementation of the reactive maintenance strategy.

Each node also maintains a larger *passive view* usually k times larger than the active view, whereas k is a constant related with the fault tolerance level of the protocol. The passive view is maintained by a cyclic strategy therefore, periodically, each node performs a shuffle operation with one random node in the overlay that results in the update of both nodes passive views. This partial view is used for fault tolerance and serves as a backup list of nodes that is used to attempt to fill the active view when some of the nodes in it are suspected as being failed.

Although originally HyParView was designed to support reliable broadcast in combination with an efficient flooding technique, a companion protocol named *Plumtree* [5] allows to efficiently build and maintain a highly resilient spanning tree embedded in the overlay maintained by HyParView's active view, which allows to lower the overhead of the dissemination protocol without impairing its resilience.

[1]In fact, the protocol is said to be Hybrid because it combines these different strategies.

III. Architecture

We propose a monitoring architecture based on the unstructured overlay defined by the active view of the HyParView protocol. In our approach, each component of the system is mapped to a node, or peer, in the HyParView overlay. The monitoring relations are mapped on top of the neighboring relations defined by the overlay, therefore if node n is neighbor of node p in the overlay, component n monitors component p. The reader should notice that because active views in the HyParView protocol are symmetric this also implies that component p is neighbor of component n and therefore p also monitors n.

In order to join the monitoring overlay, components simply have to execute the Join procedure of the HyParView protocol. To that end, a new component has to know another existing (correct) component in the system and contact it. The Join procedure ensures, through random walks in the overlay maintained by HyParView, that to the new component will be attributed a *threshold* of neighbors and, due to the symmetric nature of neighboring relations, be known and monitored by that same amount of other random components.

The reader should notice that the reactive nature of neighboring relations in the active view of HyParView, which promotes the stability of these relationships, allows to have monitoring operations that are able to observe the evolution of relevant properties in components, enabling more complex and history based measures to be taken from the system. Such approach is not possible using other well known gossip-based membership protocols such as Cyclon [4].

An explicit Leave mechanism had to be added to HyParView. Originally the protocol did not require this, as a leaving node could be simply handled as a failed node. However, for monitoring purposes, this could generate incorrect alarms, that a component had failed.

A. Ensuring monitoring requirements

In Section II-A we have identified three requirements for monitoring dynamic large-scale distributed systems. In this section we discuss how our architecture addresses those requirements.

It has been shown that the overlay network maintained by HyParView is connected, meaning that for each component, there is at least one path that connects that component to all other components in the system. Therefore every component in the system is guaranteed to have, at any given time, at least one neighbor. Moreover, HyParView is self-healing and is able to recover from large failures, as high as 80% of simultaneous node failure. Thus our architecture satisfies the first requirement: for every component c, there exists at any given time at least one component m that monitors it. Also, the reactive nature of the active view combined with the additional passive view allows HyParView to recover these properties after large failures faster than other gossip-based membership protocols.

The second requirement is to distribute evenly the monitoring load among all components in the system. This is also ensured by HyParView's properties. Active views have a limited size t; Moreover, HyParView is able to ensure that almost every node (in our experiments 97.79%) has a full active view. Therefore, (almost) every component of our system will monitor t other components, and because the active view is symmetric, these component will also be monitored by t other components, ensuring a monitoring threshold of t, as discussed earlier in the paper.

The reader should notice that, although the remaining 2.21% nodes do not have a full active view, they do not contain an empty view, and therefore it is ensured that they monitor, and are monitored, by a lower *threshold* of t'. As we show in Section III-B, t' is never below 2.

The final requirement identified was to support an efficient, scalable, and highly fault-tolerant dissemination strategy, that could propagate the relevant monitoring information to the components with the responsibility, or ability, to initiate corrective measures. In our architecture, monitoring is a shared task among all components therefore, information must be broadcasted to all components in the system. To this end, we propose the use of the broadcast primitive introduced in [3] to which HyParView was originally designed. This is a scalable service, as it shares the load of disseminating information evenly across all nodes in the system. Moreover, it is efficient and ensures that all participants receive all broadcast messages as long as the overlay remains connected.

Each component has an embedded monitor for acquiring the monitored data. This monitor keeps the acquired data and combines it with other received data. Each monitor requests from its neighbors data regarding the monitored properties. According to its configuration the monitor can either disseminate the data or store it for posterior analysis. This can be done periodically, at given time intervals, or whenever some particular change happens. In the latter scenario, an alarm can be generated and disseminated to all other component monitors. In terms of monitoring, these alarms are useful since they can notify a relevant change that triggers some self-reconfiguration in components.

Although most components act as data sources, some components may not be sources themselves but still require access to the monitoring data from other components. One example are autonomic managers, described in [1], where one or multiple managers depend on monitoring information to perform reconfiguration actions. To support this, a monitor can operate in two distinct modes: *active*, where it acquires and disseminates monitoring information, besides receiving information from other components; and *passive*, where the monitor only receives and collaborates in the dissemination of information from other components.

B. Evaluation

In this section we illustrate the relevant properties of HyParView that make it the appropriate infrastructure for our monitoring architecture. We evaluate these properties through simulation in the *PeerSim* [6] simulator, using its cycle based engine. In this context, a simulation cycle is a virtual time unit that starts with the transmission of a broadcast message by a random node and ends when no message is in transit in the network (*e.g.* all produced messages in that cycle were either delivered or lost).

We executed several experiments in a system composed of 10.000 components. HyParView was configured with an active view size of 5 and a passive view of 30. All other parameter associated with the internal operation of the protocol were set to the same values described in [3].

Figure 1 depicts the degree distribution (*e.g.* the number of neighbors) of components in a HyParView overlay. The overlay was generated by having every node join the overlay using a single contact node.

Notice that the large majority of components in the system have a full active view of 5 neighbors. There is no component in the system with 1 or fewer neighbors. This ensures that: *i)* the monitoring load is evenly distributed through all components, and *ii*) that all components in the system are monitored by at least 2 other components, being the large majority monitored by the target *threshold* of 5 other components.

Figure 2 depicts the number of simulation cycles required by HyParView to recover the overlay connectivity in face of different simultaneous component failures that range from 10% to 90% of the original components in the system. We compare the performance of HyParView with that of Cyclon [4] a well known gossip-based membership protocol which maintains an unstructured overlay network. Cyclon was configured in our experimental setup with a partial view size of 35 (the sum of both active and passive views of HyParView). This comparison serves to show that HyParView is a better suited than other existing protocols (such as Cyclon) to support a large-scale monitoring architecture.

Notice that HyParView connectivity is regained in a single simulation cycle in scenarios where, at most, 60% of the components of the system fail simultaneously, whereas Cyclon takes increasingly larger times to recover its connectivity, raging from 42 to 80 simulation cycles. In a catastrophic scenario where 90% of the components in the system fail simultaneously, HyParView only requires 4 simulation cycles to recover, whereas Cyclon takes 159 simulation cycles.

IV. Related Work

The Ganglia system [7] aims at scalable and distributed monitoring in high performance computing systems. It focuses on clusters, grids, and large scale systems with high availability. Ganglia architecture

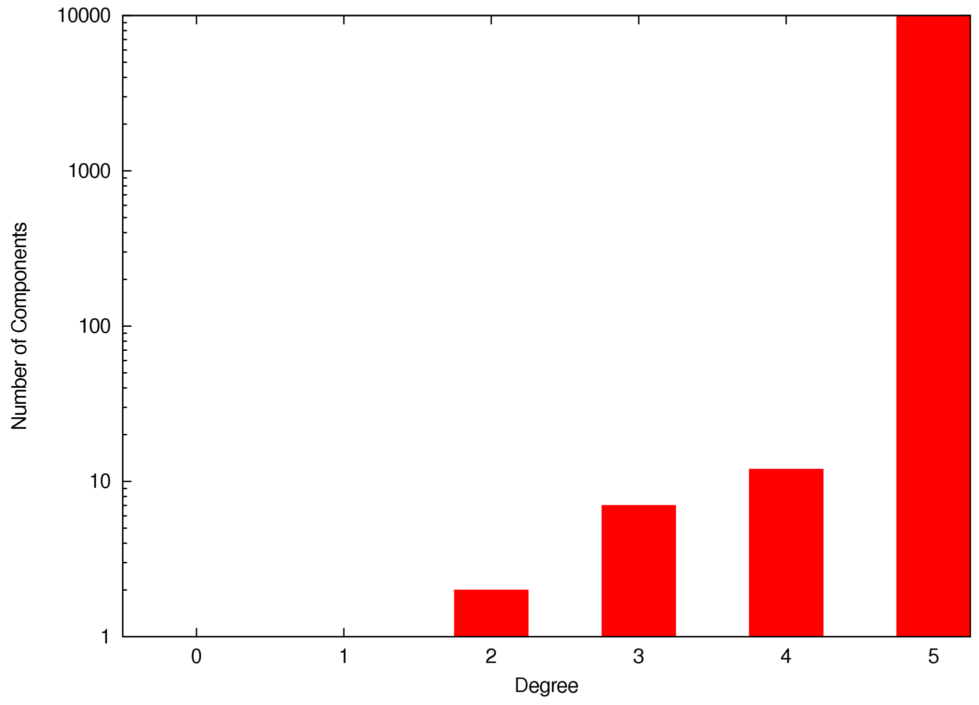

Fig. 1. Degree Distribution

relies in a multicast-based protocol to monitor system-wide state and in a tree of point-to-point connections to aggregate clusters state information. Unfortunately, the authors main target are stable environments, whereas our work focus on, eventually highly, dynamic large scale systems. Moreover, their solution requires IP multicast which might not be available in large scale systems.

Another system is Astrolabe [8], that features information management, offering support to detect changes and failures, as well as, support to perform the necessary reconfigurations as reaction. It monitors the state of a collection of distributed resources, providing summaries of this information to users. Astrolabe architecture relies in an agent, running in each node, that executes a gossip protocol. The membership update also depends on a failure detection mechanism. Moreover, it requires the maintenance of an explicit tree of nodes and also the selection of representatives (leaders) within portions of the tree (subtrees).

Finally, in [9] the authors propose a scheme for building a distributed failure detector based on gossiping. Although the authors propose a peer-to-peer model, their architecture is only based in the random exchange of messages between the several nodes, whereas our work mainly relies in the properties of an overlay topology. Moreover, the authors do not explain how they maintain membership relation between nodes, and they only assume a small percentage of node failures. Their work can however, be seen as complementary to our own, in the sense that we could rely in a similar technique to provide failure detection in our architecture.

V. Conclusions and Future Work

In this paper we proposed a monitoring approach based on self-healing unstructured peer-to-peer overlay networks, more specifically by relaying in the special properties provided by the HyParView protocol. We also identified a set of monitoring requirements for large scale dynamic distributed systems and showed how our approach addresses these specific needs. Experimental work illustrates the properties of HyParView that make it an appropriate support for a monitoring service.

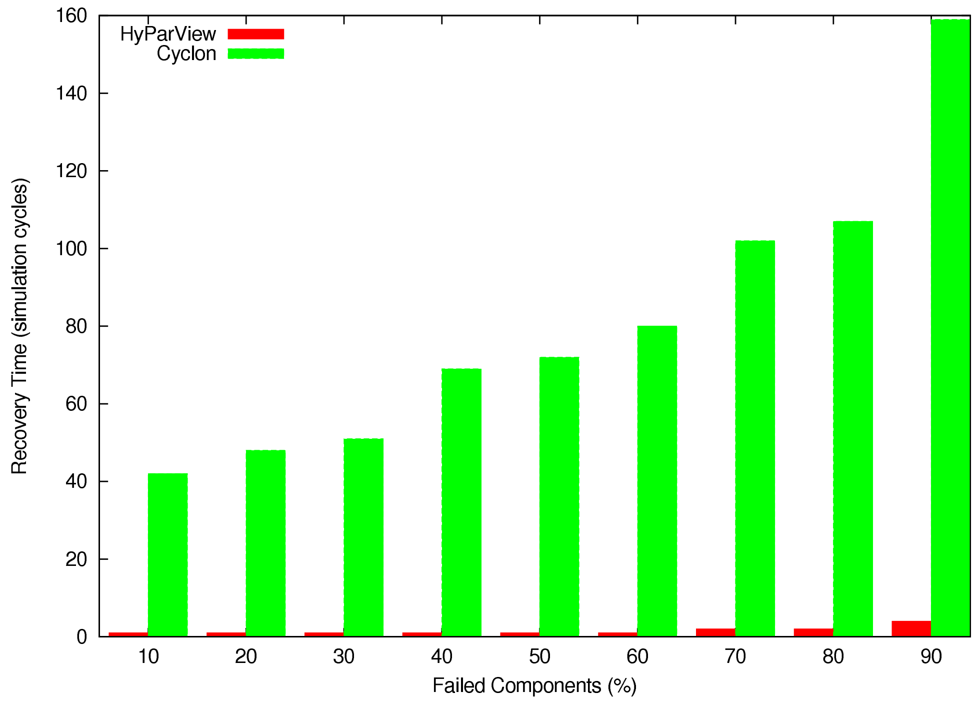

Fig. 2. Recovery Time

As future work we would like to extend our approach to handle more complex communication patterns by the monitoring service. For instance, we would like to experiment with structured overlay networks (DHT's) as a building block to support queries over the monitoring system. Moreover we would like to use these DHT's in combination with efficient broadcast protocols such as Plumtree [5] in order to support a publish-subscribe interface for the information dissemination component.

In terms of self-management, we would like to extend the proposed architecture with mechanisms for self-configuration of system components, based on the acquired monitoring information. These mechanisms would include a configuration policy, describing how the components should self-adapt, and managers to coordinate the configuration among different nodes.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "The peer sampling service: experimental evaluation of unstructured gossip-based implementations," in *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 79–98.

[3] J. Leitão, J. Pereira, and L. Rodrigues, "HyParView: A membership protocol for reliable gossip-based broadcast," in *DSN '07: Proc. of the 37th Annual IEEE/IFIP Intl. Conf. on Dependable Systems and Networks*. Edinburgh, UK: IEEE Computer Society, 2007, pp. 419–429.

[4] S. Voulgaris, D. Gavidia, and M. Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, June 2005. [Online]. Available: http://dx.doi.org/10.1007/s10922-005-4441-x

[5] J. Leitão, J. Pereira, and L. Rodrigues, "Epidemic broadcast trees," in *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS'2007)*, Beijing, China, Oct. 2007, pp. 301 – 310.

[6] "Peersim p2p simulator," http://peersim.sourceforge.net/.

[7] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[8] R. V. Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Transactions on Computer Systems*, vol. 21, no. 2, pp. 164–206, 2003.

[9] R. van Renesse, Y. Minsky, and M. Hayden, "A gossip-based failure detection service," in *Middleware'98, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, England, September 1998, pp. 55–70.