A Vision on Autonomic Distributed Systems

Raimundo José de Araújo Macêdo

[1]Distributed Systems Laboratory (LaSiD) - Computer Science Department
Federal University of Bahia (UFBA) - Campus de Ondina - Salvador - BA - Brazil

macedo@ufba.br

Abstract. *The autonomic computing initiative was launched by IBM to handle the increasing management complexity of IT infrastructures and organizations. In this paper it is introduced a view on how the research on distributed systems can advance to address the autonomic perspective. We do so by discussing the impact on models, assumptions and basic building blocks and by proposing a new classification of distributed systems as a function of their capability to manage adaptation in an autonomic fashion. We also briefly discuss our achievements on designing autonomic distributed systems.*

1. Introduction

The IBM Autonomic Computing Initiative (ACI) aims at computer systems capable of self-management, making decisions autonomously from high-level policies, which are characterized by self-* properties like self-configuration, self-healing, self-optimization, self-protection, etc. [IBM 2001]. Since the initiative was launched in 2001 major research efforts have been dedicated on how to architect such systems and on mechanisms for self* properties. A commonsense in the literature is that to be autonomic, systems should be made up of a hierarchy of autonomic sub-systems from the highest level (user level) to the lowest basic components.

The idea of self-management or autonomy is much older than the IBM ACI. Indeed, even the very concept of automatic execution of computer programs in Von-Neumann architectures brings the idea of autonomous execution in the machine level. Besides, the concept of autonomy has been explored in many fields such as artificial intelligence, robotics, software engineering, network management, automation and control systems, biological computing, dependability, etc. However, the main contributions of ACI seems to be the perspective of assessing how autonomous (or autonomic) a system can be – by proposing autonomic maturity levels – and also by disseminating the urgent need for more autonomous systems in order to handle the ever increasing management complexity of modern IT systems and applications. More recently, paradigms like cloud computing, which allow higher flexibility for end users in terms of resource allocation and service compositions, have reinforced the need for autonomic systems. The fact that users do not have to plan ahead for resource provisioning, and computing resources (such as network bandwidth, CPU, etc.) are allocated on demand to meet user defined SLA (Service Level Agreement), makes these new paradigms very appealing. However, adapting to resource variability and changing user requirements at run-time is a great challenge. As a consequence, these systems need the ability to be self-manageable or autonomic, continually reconfiguring and tuning themselves to attain certain goals.

The field of distributed systems began with the first networks in the seventies, from the idea that distinct computers could cooperate and coordinate themselves to achieve certain goals, and resources could be shared between distributed processes. The research on this field can be divided into two major sub-subfields: computability models and basic algorithms at one side, and software infrastructure and tools at the other one. Computability models are about the solvability of fundamental and recurrent distributed computing problems (e.g., consensus, mutual exclusion, etc.) on given models. Software infrastructure and tools address the problems of how to develop and deploy systems from distributed processes, which involve communication standards such as remote procedure calls (RPCs), service brokers, standard protocols for object/service communication and discovery, security, and so on.

This paper tries to bridge these two fields, ACI and distributed systems, analyzing the impact of ACI on distributed systems research, with focus on computability models and basic algorithms. We begin by discussing distributed system models in section 2. In section 3 we present our view on how to develop and classify distributed systems in an autonomic perspective. Finally, in section 4 we draw a few concluding remarks.

2. Distributed System Models and Assumptions

A distributed system (DS) consists of a set of processes, which communicate and synchronize by sending and receiving messages through communication channels. Such specification is usually extended by qualifying processes and channels behavior in terms of speed and failure patterns. Research in this area is typically on the computability and complexity related to fundamental and recurrent problems encountered in distributed systems, for a variety of DS models. For instance, every pair of processes is connected by a reliable bidirectional asynchronous channel: they do not create, alter, or lose messages, and messages can spend an arbitrary unbounded time to be delivered. This model is frequently called time-free or asynchronous, and it has been proved that there is no deterministic solution to the fundamental and recurrent consensus problem in the presence of crash faults in this model [Fisher et al. 1985]. On the other hand, in synchronous systems, where message transmission and process execution delays are bounded, several problems related to distributed computing such as consensus and membership have been solved [Cristian 1991]. The bad news is that synchronous system assumptions are valid only in restricted scenarios. For instance, such assumptions do not hold on the Internet. Other researchers have investigated intermediate DS models, weaker than synchronous and stronger than asynchronous, where consensus and other fundamental problems are solvable (e.g.,[Dolev et al. 1987]).

Classical assumptions on DS models specify system behavior without considering that such behavior, in terms of model assumptions, can change over time. This is a limitation for autonomic systems where the characteristics of the underlying computing system can change dynamically. Hence, we argue that an autonomic DS should be aware of the validity of its own assumptions, following current QoS guarantees, and therefore allowing algorithms to adapt accordingly: by taking advantage of perceived synchronous behaviors, when possible, or by switching from certain synchronous solutions when the related assumptions do not hold anymore.

In our research we have developed general self-aware models and related algorithms that can sense the computing environment and adapt to current conditions, adopting new assumptions when possible. Self-awareness was realized by us by partitioning the set of processes into three disjoint sets (*life, uncertain and down*) that are automatically updated to reflect the current state of the computing and communication systems [Gorender et al. 2007]. It could be argued that adopting worst-case assumptions (i.e., asynchronous assumptions) would suffice, so that assumptions do not need to be changed at all. Though this argument is true in the sense of algorithm correctness, such an approach can indeed have a very negative impact on the DS effectiveness. For instance, if QoS of processes and channels improve so that parts of the system behave as synchronous sub-systems, then more robust solutions can be deployed, tolerating a larger number of component failures [Macêdo and Gorender 2009]. On the other hand, if strong assumptions do not hold in a given run, self-awareness will provide the required knowledge to adapt services to a degraded, though still operational, service. Without model self-awareness such runtime adaptation would not be possible.

3. From Adaptive to Autonomic Distributed Systems

A distributed system may evolve to adapt to new application requirements or computing environment changes (due to failures, addition or removal of computing resources, etc.). Therefore, being adaptive is a requirement in the first place, to cope with the dynamic nature of such systems. Hence, adaptive systems have been the focus of many research efforts in the past in areas such as dependability, real-time systems, software engineering, etc. For instance, a fault-tolerant system is adaptive by definition, because it will work correctly despite the failure of some of its components: i.e., they are adaptive regarding component failures. One can go even further, designing systems that can modify the fault-tolerance mechanisms at runtime to adhere to new computing conditions, or to optimize some performance aspects such as message overheard. In order to help to understand the adaptiveness nature of a DS regarding its autonomic capability, in this paper we propose to classify autonomic distributed systems according to the degree of control they can exercise upon the related adaptive mechanisms, if any, as follows.

1. Non adaptive: no adaptation is provided
2. Offline adaptive: adaptation is possible but only before execution - no runtime adaptation
3. Online adaptive: adaptation is realized at runtime from pre-defined objectives - policies cannot be controlled at runtime
4. Autonomic adaptive: adaptation policies or objectives can be modified at runtime

IBM proposed 5 maturity levels to classify autonomic systems. At level 5, called full autonomic, management functions are not only automated, but the focus is on achieving business goals and managing business processes, not just managing the IT infrastructure that supports those business processes. At level 4 there will be a fully automated control loop in which autonomic managers can sense and respond to changes in the environment. Although the view in this paper is focused solely on distributed system mechanisms (not the whole IT infrastructure and organizations) we could apply the following simplified mapping to both views. The first three maturity levels (basic, managed, predictive) would map to non-adaptive distributed systems. The adaptive level (number 4) would map to offline adaptive and online adaptive DSs. Finally, level 5 (full autonomic) maps to autonomic adaptive DSs.

In order to deploy autonomic distributed systems, basic DS building blocks should allow upper-layer applications or services to control their behavior from certain policies or objectives. The policies or objectives may specify QoS requirements and/or computing environment constraints, or both, and a key design point is to define autonomic objectives or derive them from upper-layer SLAs. We have instantiated this general approach to implement autonomic failure detectors, autonomic group communication, and byzantine replication[1], which are important DS building blocks. In the failure detector case, the upper-layer system (a human user or system) can control the desired failure detection QoS under the available resources, such as detection time and detection accuracy. In the group communication case, message blocking time and protocol overhead can be controlled, and in the byzantine replication case, batch size and timeout are regulated by the controller in order to optimize message throughput and delivery time. In the first two cases (failure detection and group communication), building blocks are adaptive autonomic, and in the latter case, just online adaptive, as its objective – optimizing throughput – is not modified at runtime. Online adaptive is a weaker form of an autonomic distributed system.

4. Final Remarks

In order to help to understand the relation between the autonomic computing initiative and DS research, we have proposed in this paper a classification of distributed systems according to its adaptiveness nature, ranging from non-adaptive to autonomic adaptive systems, and presented some developments we have realized in the Distributed Systems Laboratory (LaSiD)/UFBA in light of this new classification. A great challenge in designing autonomic distributed systems is on how to map business rules and policies into autonomic distributed system objectives or adaptiveness policies. This mapping will greatly depend on the mastering of the related DS building block dynamics and its performance tradeoffs. Modeling the behavior of the computing environment and distributed system protocol is another challenge, which we have successfully addressed by applying feedback control loop theory.

References

Cristian, F. (1991). Reaching agreement on processor-group membership in synchronous distributed systems. *Distributed Computing*, 4(4):175–187.

Dolev, D., Dwork, C., and Stockmeyer, L. (1987). On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97.

Fisher, M. J., Lynch, N., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382.

Gorender, S., Macêdo, R. J. A., and Raynal, M. (2007). An adaptive programming model for fault-tolerant distributed computing. *IEEE Trans. on Dependable and Secure Computing*, 4(1):18–31.

IBM (2001). Autonomic computing: Ibm's perspective on the state of information technology. Technical report, IBM Coorporation, USA, New York.

Macêdo, R. J. A. and Gorender, S. (2009). Perfect failure detection in the partitioned synchronous distributed system model. In *Proc. of IEEE International Conference on Availability, Reliability and Security (ARES 2009)*, pages 273–280.

[1]the related published papers are available on http://www.lasid.ufba.br/publicacoes/papers.xml