Intelligent agents for high availability systems*

Fabrizio Messina[2,†], Domenico Rosaci[3,†] and Giuseppe M.L. Sarnè[4,†]

[2]University of Catania
[3]Università Mediterranea di Reggio Calabria
[4]Department of Psychology, University of Milan Bicocca, Piazza dell'Ateneo Nuovo, 1, 20126 Milan, Italy

Abstract

High availability (HA) systems are designed to guarantee availability of process and data for more then 99% of their operational time. HA are nowadays present in many contexts, both government and commercial services. In particular, the widespread of IoT smart devices has increased the need of HA systems providing broad network access. Many HA systems are built on top of the cloud, which is the standard de facto for small and medium companies to exploit reliable computational resources capable to scale up and down very quickly. Intelligent agents were developed more than 20 years ago, they have assumed a great importance in various context. Indeed, they are capable, in principle, to perceive the environment where they live and autonomously take actions to achieve goals in a wide range of contexts. Moreover, agents can improve their own performance by adopting machine learning techniques or by gaining knowledge about both environment and application domains. This position paper explores the use of intelligent agents to support core mechanisms—specifically monitoring, failure detection, and recovery—in HA systems. The discussion begins by reviewing key background concepts of HA architectures, followed by a structured characterization of their main components.

Keywords

High availability systems, High Availability cluster, Cloud computing, Software agents

1. Introduction

In the past, system availability was traditionally a requirement for mission-critical applications. Nowadays this system-wide non-functional requirement has the same importance for a wide range of commercial and government systems. Indeed, High Availability (HA) systems –designed to maintain continuous operation and accessibility, even in presence of failures– are needed in many context to guarantee essential services, as government services, as well as commercial applications, like Facebook or Linkedin, being the users connected 24 hours a day with smartphones and similar IoT smart devices[1].

As stated in [2], the flavors of system availability are *i*) continuous availability, *ii*) fault tolerance, and iii) *HA*. While the former represents an ideal state (non stop service), implying perfect components that never fail, the second one deals with the presence of faults in a real system by introducing some kind of redundancy at various levels (hardware, software, and time). Finally, High Availability systems represent a viable alternative to fault tolerant systems: in this case specific solutions to deal with Single Point of Failures are adopted.

Common solutions for HA deal with Single Point of Failures (SPOF) by means of redundant components and a sort of automation to switch from the failed component to the another one that can ensure the provisioning ot the same functionality. This kind of solution is called, in the literature, High Availability Cluster (HAC) [3]. Such solutions, widely adopted by major companies as HP and Oracle, employ a wide range of techniques to monitor the "health" of the system components (application layers and resources), to restart the application components after a failure.

For our purposes, in this paper we take into account the three main operations of i)monitoring, ii) fault detection and iii)recovery. The capability of a system to detect a fault, or even to anticipate a

26^{th} *Workshop "From Objects to Agents"*
fabrizio.messina@unict.it (F. Messina); domenico.rosaci@unirc.it (D. Rosaci); giuseppe.sarne@unimib.it (G. M.L. Sarnè); giuseppe.sarne@unimib.it (G. M.L. Sarnè)
https://www.unimib.it/giuseppe-maria-luigi-sarne (G. M.L. Sarnè)

fault, is clearly connected to the monitoring capability. Moreover a fine monitoring of the health of the systems components (i.e., applications, middleware, operating systems, device, and so on) is important to predict/detect failures. On the other hand, the system –as well as the applications– may be complex in nature, therefore the monitoring of both the system and the application may represent a challenge. Finally, the recovery/failover process should be implemented in an efficient and effective way. The first characteristic is important to not incur in expensive downtime, while the latter is crucial for obvious reason. For example, an unrealible failover mechanism may have negative effects on the availability of the system itself.

Intelligent agents[4, 5, 6, 7] have been developed for more than 20 years and have assumed a great importance in various contexts, for instance in healthcare[8]. Basically, agents are designed to perceive the environment where they are living and autonomously performing a wide range of tasks to achieve specific goals in a wide range of computational context. Furthermore, their performance can improve over time by acquiring new, specific, and updated knowledge about both environments and application domains.

Given the premises above, in this paper we discuss and analyze the possible role of intelligent agents to support high available systems. After that we describe the key concepts about HA systems and, thereafter, we focus on the general problem of monitoring fault detection and recovery, along with a broad analysis of the possible role of software agents, will be focused.

The paper is organized as follows: Section 2 discusses key concepts and terminology. Sections 3 –6 present a broad discussion about the main characteristics of the agents in the context of HA systems. In particular, as stated before, we focus on the role of software agents w.r.t. the three main operations monitoring, fault detection and recovery in HA systems. Section 7 presents some interesting related work about HA systems and the employment of software agents in HA systems. Finally, in Section 8 our conclusions are drown.

2. Key concepts and terminology

In this Section the key concepts of high availability systems, as well as basic terminology used in the following Sections, will be introduced.

A *high available (HA) computing system* can be defined as a system capable to deal with failures through the introduction of redundant components, as well as software solutions to recover/restart failed components. *Redundant components* are introduced at various levels to address the presence of SPOF. The final goal is to provide a *continuous service* or, at least, the *minimum amount of downtime*. Availability of a system can be calculated as the ratio between the amount of time a system is operational and the total time over which the system was observed, for example a year.

As stated in the introductory section of this work, High availability is nowadays a key element of production systems. Indeed, Cloud providers offers their customers to sign an SLA (Service Level Agreement) with an explicit statement about the minimum level of availability to guarantee to the customer, for example 99.5% [9].

A typical example of highly available system is represented by a HAC (High Availability Cluster): a server hosts the application, while one or more additional servers are introduced as *failover servers*, i.e. servers capable to host the main application once the main server is down due to a failure of one or more of its components. A failover solution must be designed to act *transparently* for the client and the process should be totally *automatic*, in the sense that no manual operations will be involved in the overall process. These characteristics must hold, in principle, for every solution for high availability.

In this sense automation represents a key aspect of HA systems: a high level of automation in HA systems is needed not only to replace failed component (software or hardware), but also to analyze any aspect of the life-cycle of the system components. The process of retrieving and analyzing data allows a program or, event better, an intelligent agent to gain an in depth knowledge of a system component and, as a consequence, to better manage a failover process. In principle, an agent which is able to perform data analysis about the behavior of system components, can also detect malfunctions in order

to anticipate possible failures.

In the following Section we discuss the possible role of software agents in HA systems, w.r.t. the three operations related to monitoring, failure detection and recovery. In particular, the main features and characterization of software agents in HA systems will be analyzed.

3. Monitoring, failure detection and recovery in High Availability Systems

Nowadays, HA systems are supported, in their architecture, by the cloud computing paradigm [10, 11, 12, 13, 14, 15, 16, 17, 18] which is, in turn, based on the virtualization technology [19, 20]. Indeed, cloud computing provides a broad network access, server consolidation, fast migration of virtual machines and many other features that support the provider in offering HA computing resources to their clients. This is mainly due to the flexibility offered by the virtualization and the related middleware built on it [21, 22, 23]. At the same time, applications retain their complexity[24, 25], and single point of failure still exist in every production system. Therefore the basic mechanisms to provide HA are still needed, although they are applied in different context (for instance, hypervisors and virtual machine) than 20 years ago. For example, a failover server can be a virtual machine instead of a physical machine.

In this paper the focus is on *i*) monitoring, *ii*) failure detection and *iii*) recovery, that can be considered the core mechanisms in the context of HA systems. We denote the three operations together as the MFR (Monitoring, Failure detection, Recovery) loop (see figure 1). In the following of this Section we report a few specific considerations about the main tasks of software agents to support such operations.

For the convenience of the reader, in the remaining of this work, software agents will be denoted on the basis of their responsibilities, as follows:

- `M-agent`: a software agent capable of performing one of more specialized monitoring tasks;
- `F-agent`: a software agent capable of analyzing monitoring data sent by an M-agent in order to detect a failure or a malfunction (useful to predict a failure);
- `R-agent`: a software agent capable of performing a number of specialized tasks to recover a failed computing resource;

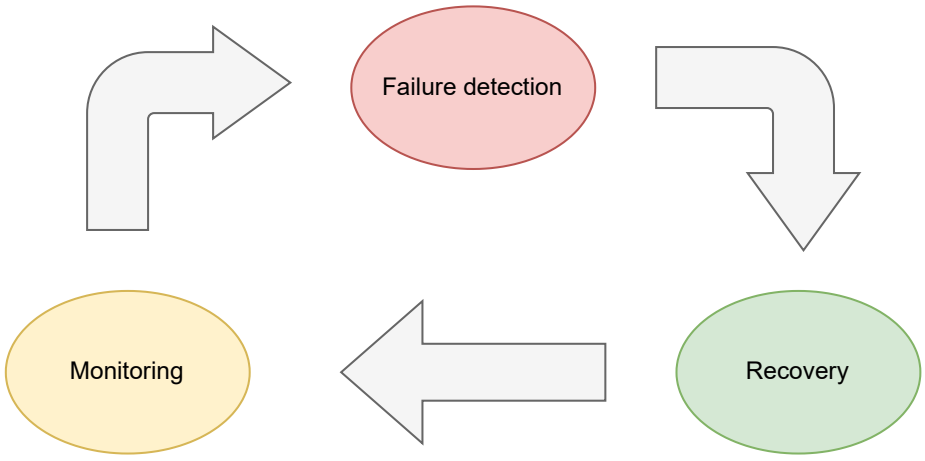

Figure 1: The loop monitoring-failure detection-recovery

4. Monitoring

Monitoring represents a crucial service in HA systems. An M-agent having the responsibility of monitoring a computing resource (physical or software) must observe continuously the entity under observation.

An M-agent can perform its task into two different way, which are labeled, for convenience, as (**O**) and (**I**):

- **(O)** the monitoring task is aimed at observing the computing resource from the *outside*, in order to detect relevant events to send to the failure detection agent: for example, sending a heartbeat message to a node;
- **(I)** a more intrusive approach, where the agent operate together with the computing resource, *inside* the same operating environment. In principle, this approach allows the agent to collect fine-grained data about the behavior of the computing resource.

Generally, the former case (O) says us that the M-agent operates by sending sort of messages to the computing resource at regular intervals from the outside, then it waits for a response that would indicate that the resource is still alive. This characterization is illustrated in Figure 2. For instance, the agent may send a simple HTTP message to a web server to get the status. In other words, the agent behavior follows a *polling-based* approach, which is commonly denoted as *reactive behaviour*. Another typical example is represented by the heartbeat message, which is periodically sent by the passive node(s) of a High Availability Cluster (HAC) to the active one by the agent of the cluster management software [3], for instance Pacemaker, which is a component of Clusterlabs [26].

With reference to Figure 2, an M-agent sends data to the F-agent (discussed in the next Section) which, in turn, analyze these data to determine whether the resource is alive or there is a malfunction, even a failure. Please note that the two agents may be the same agent. Figure 2 also represents that the computing resource may be represented by the application itself (i.e., a web server), or a virtual machine. In this last case, the agent may interface with the hypervisor or to another agent which is

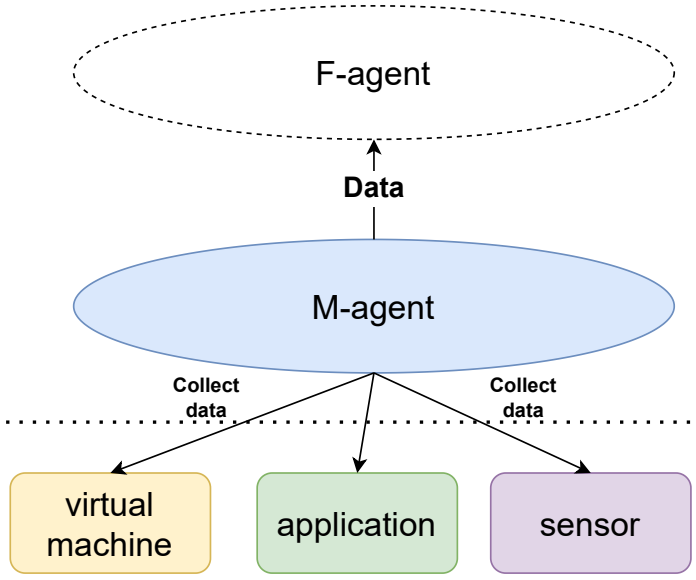

Figure 2: An M-agent performing monitoring tasks from the outside the resource environment.

running inside the virtual machine. The computing resource can be represented by a sensor, and the M-agent may send simple messages to the sensor to understand if it is still alive.

We observe that an M-agent polling the computing resource to get data from the outside of the operating environment, must necessary rely on some interface published by the computing resource or by the operating environment. By this approach the agent will not be able, in principle, to inspect the computing resource from inside (i.e., get fine-grained accurate data about the behavior of the resource); on the other side the advantage is that the agent itself will not be affected by any fault of the environment hosting the computing resource.

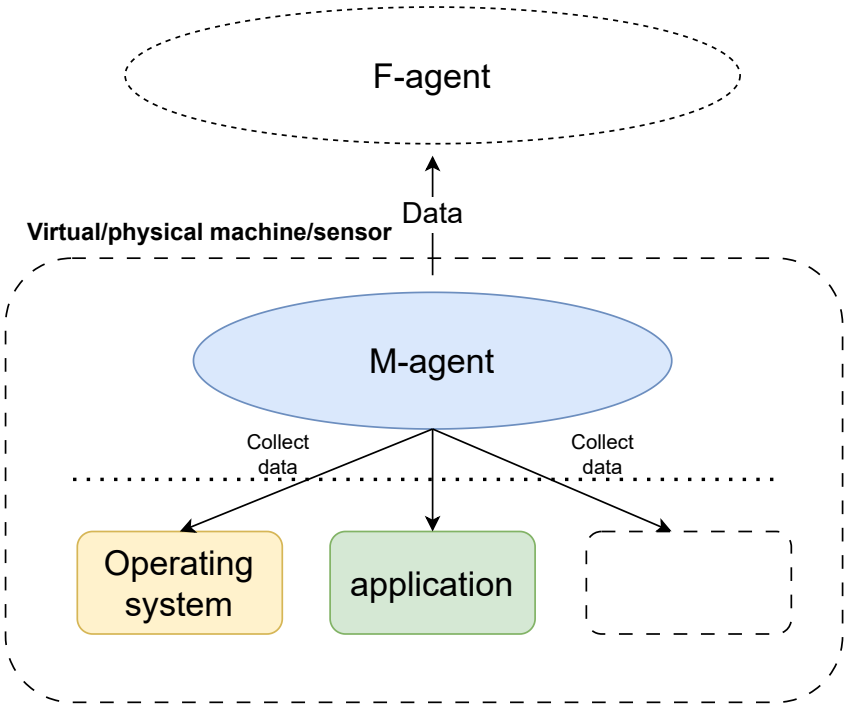

Figure 3: An M-agent performing monitoring tasks inside the resource environment.

The second characterization (I) represents the scenario in which the M-agent is running inside the execution environment of the computing resource (figure 3). For example, an M-agent running into the virtual machine can be able to inspect and analyze the log file produced by a specific application [27]. The agent may also monitor the whole operating environment (e.g. the resource usage, CPU and/or memory) to characterize the behavior of the computing resource (for instance the application), in terms of resource consumption. A suitable M-agent may, in principle, reside within a sensor, in order to characterize its behavior and send relevant data to the F-agent. This behavior can be classified as a push-based approach, because the agent operates together with the computing resource and push data to the outside to the F-agent.

In this characterization we state that the M-agent acts “proactively”. A proactive approach can be implemented also from the outside of the computing resource (O), although it is less common.

An obvious observation is that an M-agent, operating inside the operating environment of the computing resource, will be affected by any failure of the operating environment. To address this issue, an interesting approach found in literature in the context of cloud computing, proposes the combination of the poll-based and push-based approach [28]: a monitoring agent runs inside the virtual machine to notify events to another monitoring agent, which is hosted outside.

Table 1 summarizes the main features of an M-agent with respect to its deployment and behaviour on the basis of the information presented in this Section.

	(O)	**(I)**
Behavior	Observations from outside the operating environment of the computing resource	Observations from inside the operating environment of the computing resource
Reactive approach	polling-based	polling-based
Proactive approach	push-based with limitations on the quantity and quality of monitoring data	push-based
Limitation	limited data received from the interface of the operating environment and/or the computing resource	a failure of the computing environment will affect the operations of the M-agent. It should be coupled with an external (O) M-agent
Benefit	a failure of the environment will not affect, in principle, the operations of the M-agent	get a wide range of data from the operating environment of the computing resource

Table 1
Main characterization of an M-agent.

5. Failure detection

We model an F-agent as a software agent holding the responsibility to identify a failure of the computing resource from the collected data. An F-agent will send relevant information to the agent that has the responsibility to deal with the failure (i.e., the R-agent).

With reference to Figure 4, an F-agent assumes that the computing resource is in failure once it receives, from the M-agent, data indicating a certain anomaly. For instance, the M-agent did not receive for a certain amount of time any response (polling-based approach) from the computing resource. In this case, data received from the M-agent is the indication of such anomaly. This situation is depicted in the right part of Figure 4. The F-agent, in turn, notifies the R-agent that will start proper actions.

On the other hand, when the M-agent is running inside the operating environment of the computing resource (see the left part of Figure 4), the `F-agent` will receive from the `M-agent` detailed monitoring information about the computing resource. In this case, the F-agent must hold suitable analysis capabilities to extract relevant information to detect a failure or a malfunction. In particular, the ability of an F-agent to detect malfunction, may help to predict possible failures. That represents a powerful but not common approach: indeed, in this case the F-agent must hold a certain confidence or expertise in the application domain, in order to extract relevant data from the data received by the `M-agent`.

To this regards, it is worth mentioning the work in [29], where the authors propose a middleware to ensure high availability in the cloud. They employed the Effective Descriptive Set Theory to determine a model of fault detection for real life applications running on the cloud; then, they present the design of a deterministic algorithm to achieve automatic allocation of backup nodes in place of the nodes affected by the faults. Indeed, their approach is proactive in nature for the monitoring and notification of faults. Another interesting study following a proactive approach is that in [30] where the authors deal with the issue of the HA requirements in the cloud by implementing the dynamic incorporation of HA features into the deployed applications. This approach may be considered similar to that in [27], where the authors propose to employ agents to monitor the VMs to collect data about the application behavior and the resources used by the application themselves.

Last but not least, we illustrate in the left of Figure 4, a couple of M-agents. The former operates outside of the computing environment and the latter operates inside the computing environment. This is the correct approach because, as resumed in Table 1, an M-agent operating inside the same operating environment of the computing resource, can be subject to the failure of the operating environment itself. In this manner, the other M-agent will be able to notify to the F-agent the failure of the computing environment.

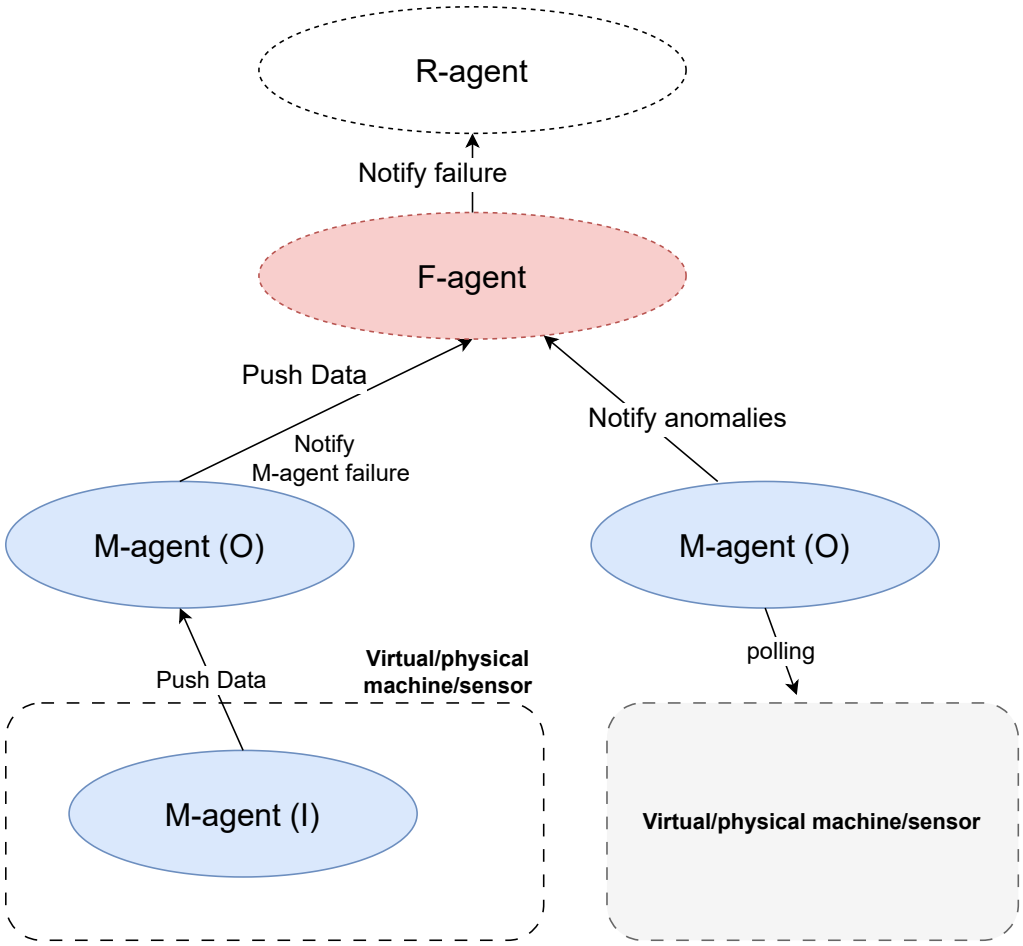

Figure 4: The interactions between an F-agent and the M-agent.

6. Recovery

An `R-agent` must be capable of performing recovery –if a failure is detected– of a resource (i.e., an application, a virtual machine, a sensor, etc.). Similarly to M and F agents, this agent must hold a relevant knowledge about the application/environment to execute suitable procedures aimed at recovering the service with the minimum impact in terms of total measured availability.

An R-agent can operates outside or inside the computing environment (Figure 5). In the first case, if the R-agent has to restart the computing environment, it can achieve this task by means of suitable interfaces. For instance, in a virtual environment, a virtual machine can be resumed or restarted in a different physical machine; similarly, in a HAC, the application (e.g. a DB or a web server) will be restarted in a failover server.

An R-agent running inside the computing environment will be capable, in principle, to recover the computing resource running in the computing environment (e.g. a scheduler, a daemon, a web server, an operating system, etc) from the inside.

Figure 5 illustrates these two typical scenarios: the left part represents an R-agent running inside the computing environment, trying to restart the computing resource. If such tentative does not succeed, the R-agent running outside the computing environment will start an alternative environment along

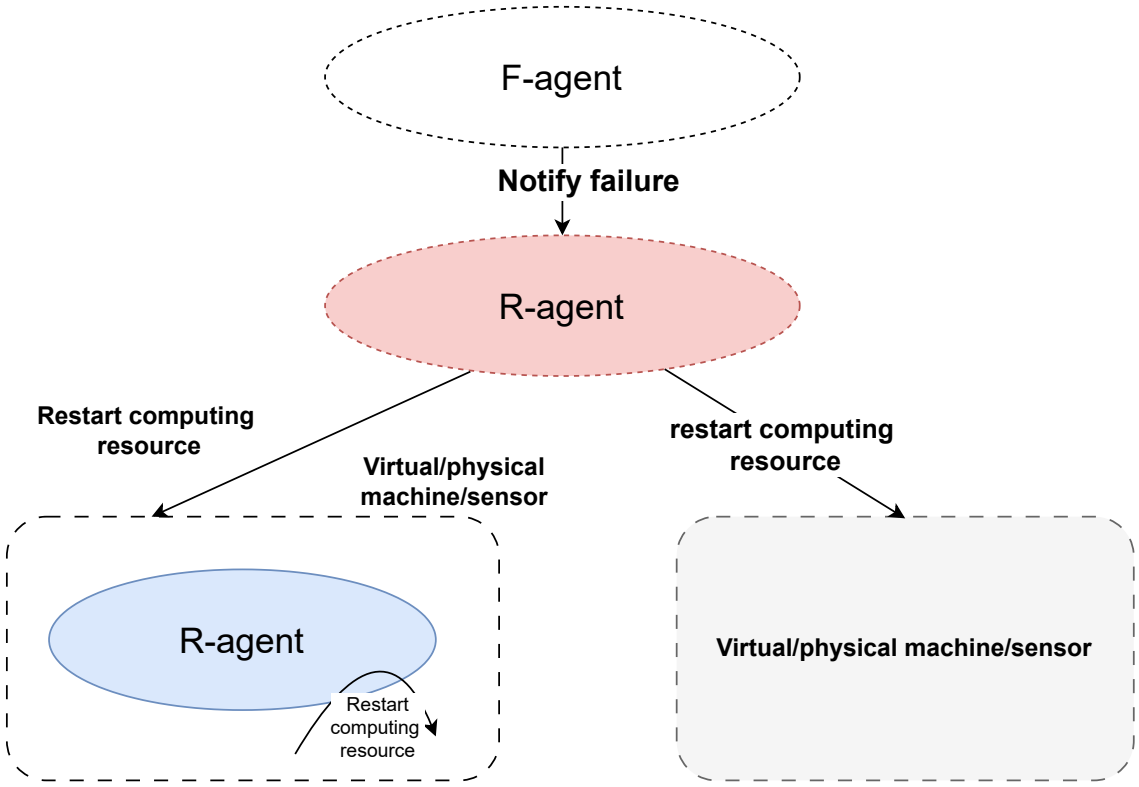

Figure 5: The interactions between an R-agent and the computing resource.

with the computing resource. It is the typical case of a service or an application running in a HAC: if the application is affected by a failure, the resource agent tries to restart the application itself, thereafter the application is restarted in a a failover server.

For instance, Pacemaker (which is the main component of the Clusterlabs project) will manage a failure trying to *i*) restart the computing resource/service (e.g. a web server) and, thereafter, if the tentative has not given success, *ii*) trying to start the service into the failover node [3]. In other words, recovery actions may be classified into two macro-categories: *i*) actions aimed at restarting the failed resource (e.g. a virtual machine, a web server, etc.), and *ii*) actions aimed at recovering the service by employing a redundant resource.

On the basis of the considerations reported above, we outline the main characteristics of a R-agent:

knowledge represents the basic foundation for its own capabilities: the R-agent must hold an in depth knowledge of the computing resource, in order to understand how to deal with the failure;

capabilities are strictly related to the knowledge of the agent itself; an R-agent can start operating holding basic knowledge, therefore basic capabilities to recover the computing resource;

reliability and availability: ***i)*** an R-agent should be 100% available because any unavailability may have negative effects on the overall availability of the system itself; ***ii)*** it should be reliable because any malfunction in the MFR loop will have a negative impact on the total system availability.

For example an R-agent for a web server must be aware of the computing environment (e.g., OS version, web server version and so on) as well as the key tools needed to manage the corresponding process. Similarly, an R-agent that needs to resume a virtual machine from a snapshot must be capable

of exploiting specific tools and/or services in the virtualization platform or cloud platform. In addition, during the lifetime of the computing resource, every time the R-agent deals with a failure, in principle it gains knowledge. For example, an R-agent may gain knowledge about the time needed to restart the computing resource in the same computing environment, or the time needed to restart the computing resource/service in a failover computing environment. In principle, this knowledge can help the R-agent to perform better in the future.

Moreover, an R-agent can be further categorized on the basis of its own capabilities:

specialized the R-agent acts as an `orchestrator` of services and tools that must be coordinated in order to perform efficient and effective recovery of the computing resource;

unspecialized the R-agent relies on very simple mechanism to recovery the computing resource.

For instance, if the R-agent does not rely into specific and/or sophisticated services as, for example checkpoints, to provide a restoration of the state of the application or part of it, it can be categorized as unspecialized. An example of an unspecialized R-agent my be represented by the *resource agent* of Pacemaker (component of Clusterlabs) that contains the code to re-start a web server in a failover node.

Conversely, a specialized R-agent is capable of relying on specialized service to restore the computing resources with the minimum losses for the application itself. For example, an R-agent could collect snapshot of a virtual machine at regular intervals, in order to restore the state of the application at a certain time.

The main characteristics of an R-agent are summarized in Table 2.

	R-Agent for restart	**R-Agent for failover**
Behavior	operates inside the computing environment (virtual machine, physical machine, sensor, and so on) of the computing resource	can operates from the outside. It can performs failover through by using a redundant resource (virtual machine, passive/active server, sensor in standby, and so on)
Limitation	the R-agent operates inside the computing environment, as a consequence it will be affected by a failure of the environment itself	the R-agent operates outside of the computing environment of the resource, a consequence it cannot control the computing resource.
Benefit	The R-agent is able, in principle, of taking a sequence of actions to recover the computing resource from inside its computing environment	the R-agent is not affected from any failure of the computing environment
Knowledge and capabilities	knowledge necessary in the application domain, in order to perform accurate actions to restart the computing resource	specific in the domain of the computing environment, for example hot to restart a virtual machine in a cloud environment
Reliability and availability	limited to the fact that the agent is alive in the same computing environment of the computing resource	an R-agent for failover can have a 100% reliability and availability, as it runs outside of the computing environment of the computing resource
Specialized vs unspecialized	a specialized agent is able to orchestrate tools and services to recover the application in the same environment, if possible; an unspecialized agent relies on very simple mechanisms to recover a sevice/application	a specialized agent is able to orchestrate tools and services to restart the application in a suitable alternative environment, if possible; an unspecialized agent relies on very simple mechanisms to recover a device/application in a suitable/alternative environment

Table 2
Main characteristics of an R-agent.

7. Related work

Software agents can be employed in a wide range of computational contexts [31]. We studied the use of software agents along with suitable models in various contexts as social networks [32, 33], trust [25, 34], IoT[35, 36, 37, 38, 39, 40, 41, 42], and cloud computing [18, 27].

Since in this work we focus in the contribution of software agents to HA systems, in this Section we discuss a few relevant work in the literature about existing solutions for HA systems in general, as well specific works about software agents supporting HA systems.

7.1. Challenges and solutions for HA systems

HA requirements can be addressed through several different solutions at many levels (e.g., applications, DB, operating system, physical/virtual machine, etc) [3]. Such solutions include the employment of redundant resources and sort of automation to orchestrate the redundant resources. A common solution for a wide range of applications is represented by the adoption of HAC, based on various different configurations of redundant resources, as for instance active-active or active-passive, N+1, N+N and so on [3]. As observed and remarked in the previous sections of this work, the overall management of the MFR can benefit from a high level of automation and coordination.

In order to support the statement above, we observe that, although many of the current HA solutions provide application monitoring, application-specific errors are often not captured. Indeed, application-related errors are often difficult to monitor (for instance, an application temporarily unresponsive) but, at the same time, they represent an opportunity to improve the real value of Mean time between failure (MTBF). To address this problem there is any specific literature, as the solution will depend on the specific application and, as a consequence, on the availability of additional modules and procedures. In this field, specialized software agents can provide an excellent support to the HA requirement of specific applications.

Moreover, the environment aspect is nowadays crucial for HA system (as also remarked in the Sections 4–6), and it may represent a challenge. Indeed, since in virtual environments hosts and guest are separated, the coordinated monitoring of both environment can be complicated: the guest is not generally aware of the host and, at the same time, the host may not be aware of the guest problems. The problem has been explored in [43]. For this reason we have taken into account the environment, and state that an agent can be executed inside and/or outside of the application environment. Moreover, with respect to this specific aspect, the adoption of intelligent agents and, overall, multi-agent systems [44] can address this concern.

A similar limitation is represented by the public cloud environment [45]. Indeed, the provisioning of IaaS (infrastructure as a Service) computational resources in the public cloud offers a broad range of advantages but, on the other hand, limited control of resources. At the same time, high availability for Enterprise applications requires accessing some elements of the hosts. This problem has been discussed in [9], on which the authors focus on the fact that current availability solutions do not rely on the specific application.

7.2. Software agents for high available systems

As previously illustrated in Section 7.1, common solutions for HA systems are realized to implement HA clusters, and resource agents are, in fact, simple functions coordinated by an HA manager.

For example, common cluster management software solutions, like that of IBM [46] or Clusterlabs [3, 47] there exists a distinction between HA manager, which holds the control of the cluster management services, and those services called HA service agents (IBM) or resource agents (Clusterlabs). Those agents have the responsibility of service monitoring and to perform very simple operations like starting and arresting the service, and simple monitoring of the service daemon. It is clear that, in production systems, software agents are very simple, they still lack of a more sophisticated implementation of agents for HA systems.

In [48] the authors deal with the problem of supporting mission critical Web Services. They propose to improve the architecture of web services by providing more autonomic behavior: configuring themselves, diagnosing faults, and self-managing. They proposes extensions to the Web Services architecture to support mission-critical applications, which must hold properties of high availability. They provide additional features in order to enrich Web Services with strong reliability aspects, as replication, data streaming, sophisticated user-programmable failure detection, reliable messaging, and events. The interesting aspect of the work is that the authors provide a support for consistent, coordinated behavior even when a system includes large numbers of lightweight components. In this sense, the authors extended the Web Service platform with a distributed intelligence supporting availability of high number of distributed objects.

The work in [49] represents an interesting review of the software agent technology by taking into account the use of software agents in a specific mission-critical environment (oil and gas industry). They claim that requirements as environmental safety, high availability and reliability cannot be provided with traditional software approaches. Software agents are capable, in principle, to sense the environment and make decisions without any human intervention. The authors provide interesting thought about how software agents can be used to improve decisions within specific industrial processes. They proved that, in the specific area of mission-critical business, three core processes can be supported by less than 10 software agents in its initial outline.

The authors of a recent work [50] proposes an approach to replicate in the cloud a number of software agents associated to the control of manufacturing resources. They proved that replicating agents in Cloud Manufacturing Control architectures results in a HA decentralized control system. They state that the methodology represents an extension of the generic agentification process, i.e. associate a software agent to a physical entity in order to simplify the access to the resource's operations. Moreover, software agent can be easily integrated in standard multi-agent system (MAS) framework. Indeed, the authors used JADE [51] to validate their approach.

Since high availability is nowadays provided by cloud computing systems, a few works have discussed the relation between cloud computing and software agents, for example [52]. The paper examines the main characteristics of a Cloud computing systems, as for example elasticity, which allows users to adapt the computing platform to variable needs. The paper focuses on the implementation of high performance complex systems and intelligent applications by using of cloud systems and software agents. The author emphasizes that, from the convergence of Cloud computing and software agents, a few improvements can be provided into the platform itself, i.e. intelligent and flexible cloud services, autonomous and pro-active services, autonomic clouds. As HA is one of the master requirement provided by cloud computing services, this analysis provided evidence that the main features of software agents provide an important support to HA in such systems.

8. Conclusions

This paper analyzed a few important aspects related to the possible contribution given by intelligent agents to HA Systems. In particular, after introducing a few well known aspects of HA systems, we have focused on the typical loop monitoring-failure detection-recover of such systems. Then we have analyzed the possible use of software agents in the main operations involved in such a loop. In particular, we have characterized three types of agent: *i*) M-agent, *ii*) F-agent and *iii*) R-agent. As a first output of our analysis, it has emerged that a common aspect of the agents is their deployment. Any of these agents will operate inside the application environment or outside. Even better, we found that both type of deployment should be implemented in order to get the maximum support.

For an M-agent, we have focused on the available mechanisms to perform monitoring –proactive vs reactive–, while for F-agent we have focused on the interactions with R-agent and M-agent, as well as the analysis operation on the data received by M-agents to detect a failure and/or a malfunction. Finally, we have characterized R-agent with respect to the twofold possible contribution of such an agent: *i*) trying to restart the application and/or the failed resource, or *ii*) implement some failover mechanisms

–i.e., restarting the service in a redundant resource. We have focused on the fact that those operations requires an in depth knowledge about the computing environment and/or the applications, as well as certain specific capabilities.

We have also highlighted as intelligent agents, in order to support high available systems, should be executed in the application environment; in this way they can collect relevant, high quality monitoring data which, in turn, allow agents to perform sophisticated actions to guarantee the desired level of availability. In this sense, the ability of an agent to gain knowledge about the application and the environment can represent an addition about standard and/or naive solutions.

Acknowledgments

This work has been supported by the project "Piano Pia.ce.ri 2024–2026" granted by the University of Catania.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] M. De Benedetti, F. Messina, G. Pappalardo, C. Santoro, Jarvsis: a distributed scheduler for iot applications, Cluster Computing (2017) 1–16.

[2] I. Pramanick, High availability, The International Journal of High Performance Computing Applications 15 (2001) 169–174.

[3] P. Somasekaram, R. Calinescu, R. Buyya, High-availability clusters: A taxonomy, survey, and future directions, Journal of Systems and Software 187 (2022) 111208.

[4] M. Wooldridge, Intelligent agents, Multiagent systems: A modern approach to distributed artificial intelligence 1 (1999) 27–73.

[5] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, et al., Unity: A general platform for intelligent agents, arXiv preprint arXiv:1809.02627 (2018).

[6] I. Rudowsky, Intelligent agents, Communications of the Association for Information Systems 14 (2004) 14.

[7] P. De Meo, F. Messina, D. Rosaci, G. M. L. Sarné, An agent-oriented, trust-aware approach to improve the qos in dynamic grid federations, Concurrency and Computation: Practice and Experience 27 (2015) 5411–5435.

[8] S. Iqbal, W. Altaf, M. Aslam, W. Mahmood, M. U. G. Khan, Application of intelligent agents in health-care, Artificial Intelligence Review 46 (2016) 83–112.

[9] M. Nabi, M. Toeroe, F. Khendek, Availability in the cloud: State of the art, Journal of Network and Computer Applications 60 (2016) 54–67.

[10] R. Giunta, F. Messina, G. Pappalardo, E. Tramontana, Providing qos strategies and cloud-integration to web servers by means of aspects, Concurrency and Computation: Practice and Experience 27 (2015) 1498–1512.

[11] F. Messina, G. Pappalardo, C. Santoro, Integrating cloud services in behaviour programming for autonomous robots, in: Algorithms and Architectures for Parallel Processing: 13th International Conference, ICA3PP 2013, Vietri sul Mare, Italy, December 18-20, 2013, Proceedings, Part II 13, Springer International Publishing, 2013, pp. 295–302.

[12] F. Messina, G. Pappalardo, D. Rosaci, C. Santoro, G. M. L. Sarné, A trust model for competitive cloud federations, Complex, Intelligent, and Software Intensive Systems (CISIS) (2014) 469–474.

[13] L. Qian, Z. Luo, Y. Du, L. Guo, Cloud computing: An overview, in: IEEE international conference on cloud computing, Springer, 2009, pp. 626–631.

[14] N. Antonopoulos, L. Gillam, Cloud computing, volume 51, Springer, 2010.
[15] R. Buyya, J. Broberg, A. Goscinski, Cloud computing, Principles and Paradigms, Publisher (2011).
[16] M. N. Sadiku, S. M. Musa, O. D. Momoh, Cloud computing: opportunities and challenges, IEEE potentials 33 (2014) 34–36.
[17] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., Above the clouds: A berkeley view of cloud computing, Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS 28 (2009) 2009.
[18] A. Comi, L. Fotia, F. Messina, G. Pappalardo, D. Rosaci, G. M. L. Sarné, A reputation-based approach to improve qos in cloud service composition, in: 2015 IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE, 2015, pp. 108–113.
[19] F. Calzolari, S. Arezzini, A. Ciampa, E. Mazzoni, A. Domenici, G. Vaglini, High availability using virtualization, in: Journal of Physics: Conference Series, volume 219, IOP Publishing, 2010, p. 052017.
[20] M. Portnoy, Virtualization essentials, volume 19, John Wiley & Sons, 2012.
[21] T. Rosado, J. Bernardino, An overview of openstack architecture, in: Proceedings of the 18th international database engineering & applications symposium, 2014, pp. 366–367.
[22] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The eucalyptus open-source cloud-computing system, in: 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE, 2009, pp. 124–131.
[23] D. Milojičić, I. M. Llorente, R. S. Montero, Opennebula: A cloud management tool, IEEE Internet Computing 15 (2011) 11–14.
[24] F. Messina, G. Pappalardo, C. Santoro, Complexsim: a flexible simulation platform for complex systems, International Journal of Simulation and Process Modelling 6 8 (2013) 202–211.
[25] A. Comi, L. Fotia, F. Messina, D. Rosaci, G. M. L. Sarné, A partnership-based approach to improve qos on federated computing infrastructures, Information Sciences 367 (2016) 246–258.
[26] Pacemaker, Clusters from scratch, 2024. URL: https://clusterlabs.org/projects/pacemaker/doc/3.0/Clusters_from_Scratch/html/.
[27] F. Messina, G. Pappalardo, D. Rosaci, G. M. L. Sarné, An agent based architecture for vm software tracking in cloud federations, in: 2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems, IEEE, 2014, pp. 463–468.
[28] H. Chan, T. Chieu, An approach to high availability for cloud servers with snapshot mechanism, in: Proceedings of the industrial track of the 13th ACM/IFIP/USENIX international middleware conference, 2012, pp. 1–6.
[29] A. Imran, A. U. Gias, R. Rahman, A. Seal, T. Rahman, F. Ishraque, K. Sakib, Cloud-niagara: A high availability and low overhead fault tolerance middleware for the cloud, in: 16th Int'l Conf. Computer and Information Technology, IEEE, 2014, pp. 271–276.
[30] A. Kanso, Y. Lemieux, Achieving high availability at the application level in the cloud, in: 2013 IEEE Sixth International Conference on Cloud Computing, IEEE, 2013, pp. 778–785.
[31] H. Nwana, M. Wooldridge, Software agent technologies, BT Technology Journal 14 (1996).
[32] P. De Meo, F. Messina, D. Rosaci, G. M. L. Sarné, Forming time-stable homogeneous groups into online social networks, Information Sciences 414 (2017) 117–132.
[33] G. Fortino, L. Fotia, F. Messina, D. Rosaci, G. M. L. Sarné, A social edge-based iot framework using reputation-based clustering for enhancing competitiveness, IEEE Transactions on Computational Social Systems 10 (2023) 2051–2060. doi:10.1109/TCSS.2022.3208376.
[34] G. Fortino, L. Fotia, F. Messina, D. Rosaci, G. M. L. Sarné, Trusted object framework (tof): A clustering reputation-based approach using edge computing for sharing resources among iot smart objects, Computers & Electrical Engineering 96 (2021) 107568.
[35] G. Fortino, F. Messina, D. Rosaci, G. M. L. Sarné, Using trust measures to optimize neighbor selection for smart blockchain networks in iot, IEEE Internet of Things Journal 10 (2023) 21168–21175. doi:10.1109/JIOT.2023.3263582.
[36] G. Fortino, L. Fotia, F. Messina, D. Rosaci, G. M. L. Sarné, A blockchain-based group formation strategy for optimizing the social reputation capital of an iot scenario, Simulation Modelling

Practice and Theory 108 (2021) 102261.
[37] G. Fortino, L. Fotia, F. Messina, D. Rosaci, G. M. L. Sarnè, A social edge-based iot framework using reputation-based clustering for enhancing competitiveness, IEEE Transactions on Computational Social Systems 10 (2022) 2051–2060.
[38] G. Fortino, F. Messina, D. Rosaci, G. M. L. Sarnè, Using trust measures to optimize neighbor selection for smart blockchain networks in iot, IEEE Internet of Things Journal 10 (2023) 21168–21175.
[39] F. Messina, C. Santoro, F. F. Santoro, Enhancing security and trust in internet of things through meshtastic protocol utilising low-range technology, Electronics 13 (2024) 1055.
[40] F. Messina, D. Rosaci, G. M. L. Sarnè, Applying trust patterns to model complex trustworthiness in the internet of things, Electronics 13 (2024) 2107.
[41] G. Fortino, F. Messina, D. Rosaci, G. M. L. Sarne, Improving computational efficiency of the tons algorithm in selecting neighbor agents in blockchain trust-based iot environments, in: CEUR WORKSHOP PROCEEDINGS, volume 3735, CEUR-WS, 2024, pp. 84–97.
[42] F. Messina, D. Rosaci, G. M. L. Sarnè, A neural-symbolic approach to extract trust patterns in iot scenarios, Future Internet 17 (2025) 116.
[43] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, P. F. Chan, Leveraging virtualization to optimize high-availability system configurations, IBM Systems Journal 47 (2008) 591–604.
[44] W. Van der Hoek, M. Wooldridge, Multi-agent systems, Foundations of Artificial Intelligence 3 (2008) 887–928.
[45] P. Hofmann, D. Woods, Cloud computing: The limits of public clouds for business applications, IEEE Internet Computing 14 (2010) 90–93.
[46] HAagents, High availability services and service agent, 2024. URL: https://www.ibm.com/docs/en/pcmc/4.2.1?topic=availability-services-service-agent.
[47] clusterlabs, Clusterlabs, 2025. URL: https://clusterlabs.org/.
[48] K. Birman, R. Van Renesse, W. Vogels, Adding high availability and autonomic behavior to web services, in: Proceedings. 26th International Conference on Software Engineering, IEEE, 2004, pp. 17–26.
[49] E. Landre, J. Ølmheim, G. O. Wærsland, H. Rønneberg, Software agents—an emergent software technology that enables us to build more dynamic, adaptable, and robust systems, in: SPE Annual Technical Conference and Exhibition?, SPE, 2006, pp. SPE–103354.
[50] S. Răileanu, F. D. Anton, T. Borangiu, S. Anton, Design of high availability manufacturing resource agents using jade framework and cloud replication, Service Orientation in Holonic and Multi-Agent Manufacturing: Proceedings of SOHOMA 2017 (2018) 201–215.
[51] F. Bellifemine, A. Poggi, G. Rimassa, Jade–a fipa-compliant agent framework, in: Proceedings of PAAM, volume 99, London, 1999, p. 33.
[52] D. Talia, et al., Cloud computing and software agents: Towards cloud intelligent services., in: WOA, volume 11, Citeseer, 2011, pp. 2–6.