Received: 6 November 2016 | Revised: 19 August 2018 | Accepted: 2 October 2018
DOI: 10.1002/cpe.5061

WILEY

RESEARCH ARTICLE

RT-JADE: A preemptive real-time scheduling middleware for mobile agents

Tatiana Pereira Filgueiras[1] | Leonardo M. Rodrigues[2] | Luciana de Oliveira Rech[2] | Luciana Moreira Sá de Souza[2] | Hylson Vescovi Netto[2,3]

[1]Centro Universitário de Brusque - UNIFEBE, Santa Catarina, Brazil
[2]Federal University of Santa Catarina, Florianópolis, Brazil
[3]Instituto Federal Catarinense, Campus Blumenau, Santa Catarina, Brazil

Correspondence
Hylson Vescovi Netto, Federal University of Santa Catarina, Florianópolis, Brazil; or Instituto Federal Catarinense, Campus Blumenau, Santa Catarina, Brazil.
Email: hylson.vescovi@ifc.edu.br

Funding information
CAPES/Brazil; CNPq/Brazil; LEAD Clouds, Grant/Award Number: 400511/2013-4; Abys, Grant/Award Number: 401364/2014-3; ITSmartGrid, Grant/Award Number: 455303/2014-2

Summary

Mobile agents are examples of distributed systems which may dispute for the same resources on their hosts. Treating such concurrency adequately is essential, particularly in real-time applications. Due to intrinsic time restrictions, mobile agents in real-time environments are only considered successful if they fulfill their mission by respecting their deadlines. Scheduling algorithms with different policies can be applied in these scenarios. However, the efficiency of these algorithms may deviate according to the missions and deadlines of the mobile agents. Also, these algorithms can be preemptive, or calculate the order of executions without interrupting an ongoing task. In this paper, we propose a middleware extension to the JADE platform that brings real-time scheduling support with preemption to mobile agents. The proposed solution uses best effort scheduling policy in the context of soft real-time applications. We evaluate the performance of the scheduling algorithms, with and without preemption, and the impact of the selected algorithms on mission fulfillment. The results of the proposed middleware showed a great improvement on mission accomplishment when compared to the FIFO algorithm provided by the JADE platform.

KEYWORDS
JADE, mobile agents, real time, scheduling

1 | INTRODUCTION

Mobile Agents (MAs) technology has the potential of being applied in several fields in the industry as a response to scalability and latency issues of centralized systems.[1-4] With the concept of independent and self-contained software, MAs have the ability to move across hosts, transporting themselves from one system to another.[1] These agents are able to perform tasks, and this is not restricted to the system in which they began to run. MAs can also sense their execution environment, reacting quickly to changes autonomously.[1] These unique abilities bring benefits to this technology, notably in critical real-time environments. For example, mobile agents are able to overcome network latency, a common characteristic of large-scale centralized systems.[5] By executing the code locally, and quickly reacting to changes in the environment, MAs are capable of achieving missions with better response times.

Research has demonstrated the benefits of using MAs in time-constrained environments.[6-9] In medical care, due to the critical nature of the communication systems, MAs were selected as a reliable solution for clinical data mobile messaging.[9] On e-commerce, this approach can enable parallel computation by running MAs on suppliers concurrently.[8] The manufacturing industry applied MAs on production control and optimization, shifting the centralized paradigm to an autonomous distributed system.[10] MAs have also been applied to real-time distributed control systems as one step toward the automatic reconfiguration of industrial control systems.[11]

As the adoption of MAs is tightly coupled to costs of implementing solutions with this technology, researchers focused on providing middleware platforms to support the development of MAs.[12,13] Grasshopper,[14] Aglets,[15] Mobility-RPC,[16] and JADE[17] were developed for Local and Wide Area Networks (LAN and WAN, respectively). Other platforms were also proposed for Wireless Sensor Networks (WSNs),[18,19] Internet of

Things,[20] and embedded systems[12] due to great interest in mobile devices. The latter two are designed to provide mobility and communication support to MAs in wireless networks, taking into account the technical limitations of these devices (eg, battery, memory, processing).

These platforms improved the support for developing MAs. Nonetheless, time-critical applications still present challenges regarding concurrent access in remote hosts. For instance, the Mobility-RPC implementation does not have the necessary resources for preemption, such as suspending an agent, which would allow the relay of mobile agents on the use of resources. When MAs need to access the same resource, it becomes necessary to have support for scheduling based on priorities and deadlines. Currently, little attention has been given to real-time scheduling on MA platforms. Researchers proposed a real-time architecture for JADE on the supply chain domain.[6] Although this is a step toward enabling real-time support for MAs, this work has not presented mechanisms to handle concurrent real-time requests. Another work[21] proposed to model and analyze real-time systems with time constraints using the JADE platform. Although this proves to be efficient for determining a schedule for task execution, this approach does not provide a middleware solution for runtime but rather focuses on modeling and analysis simulation.

This paper presents the RT-JADE middleware extension, where a scheduling module is added to the original JADE platform to enable soft real-time applications based on best-effort policy. This implementation allows the selection of multiple scheduling algorithms to define the order of execution of MAs arriving on hosts. The following algorithms without preemption were evaluated in previous work[22]: FIFO (First In First Out), LIFO (Last In First Out), EDF (Earliest Deadline First), Priority-based Scheduling (PRIO), Deadline Monotonic (DM), and SJF (Shortest Job First). In this work, we improved such implementation by adding the following preemption algorithms to the scheduling support: Preemptive Earliest Deadline First (PEDF), Preemptive Priority-based Scheduling (PPRIO), Preemptive Deadline Monotonic (PDM), and Shortest Remaining Time First (SRTF). Additionally, we simulate different scenarios and evaluate the results based on their suitability for specific applications. The results showed a considerable improvement on mission accomplishment by adding our soft real-time support.

The next sections of this paper are divided as follows. Section 2 presents the related work. Section 3 includes some essential concepts, such as real-time scheduling and the JADE platform. Section 4 presents the RT-JADE middleware, including its architecture and mode of operation. Section 5 addresses the evaluation of the proposed approach by comparing different scheduling algorithms. Finally, Section 6 concludes the paper and presents suggestions for future work.

2 | RELATED WORK

In the field of intelligent distributed computing in computer networks, an issue that attracts much interest is the use of MAs and Multi-agent Systems. Within this context, a centralized algorithm using agents to assist two MA levels, Broker Level MA (BMA), and Supplier Level MA was proposed by Sahingoz and Erdogan.[8] The aim of this proposal was the reduction of time to accomplish missions using MA cloning. In this approach, one clone is sent to each Supplier node to meet the same missions. When the mission is over, each clone sends a reply to the BMA, which in turn chooses the best of all replies and forwards it to the Broker. When the transaction completes, a log is sent to the BMA, and then forwarded to the Broker.

Agilla[18] is a middleware solution to facilitate inter-agent communication in sensor networks using nodes' tuple spaces, which are shared by local agents and remotely accessible. This was the first work to bring the MAs programming model into WSNs. This approach employed MICA2 motes (with TinyOS operating system), which are devices with limited processing and memory capabilities. The language used in this work is higher level than assembly, but it is still in a level so much lower than Java. A more recent paper about Agilla[23] describes two implementations. The first one monitors containers to detect intrusions or movements. The second tracks the navigation of a robot in a dynamic environment. The aim was to find a safe route according to temperatures perceived on the path. Similarly, the Mobile Agent Platform for Sun Spots (MAPS)[19] is a Java-based framework that allows agent-oriented programming in WSNs. Each component of the architecture provides a minimal set of services (eg, agent creation, timing handling, message transmission) for MAs, which are modeled as multi-plan state machines. The interaction between the components of the architecture is performed through events. Although Agilla and MAPS provide support to MAs, their functionality does not envision real-time applications.

A new method for the dynamic determination of the itinerary of imprecise MAs with time constraints was proposed by Rech et al.[24] This model proposed heuristics for dynamic adaptation of the itinerary by searching the best benefit regarding the MA's mission deadline. One limitation of this approach is that it does not preempt tasks during execution. In addition to managing MAs with firm deadlines and dynamic itineraries, our work includes the handling over concurrency of shared resources, as well as preemption.

Arunachalan and Light[9] proposed a real-time middleware solution with a reliable mobile message protocol for data transfer. The objective of the work was to provide the ability to send patient information through messages (forwarded by MAs) to the hospital so that patients could be treated as efficiently as possible. Such information was handled by health-care professionals using mobile devices, such as Personal Digital Assistants (PDAs). This paper uses a specific event-based clinical data messaging specification. However, it does not describe mechanisms for handling event deadlines or concurrency.

The Mobile Real-time Supply Chain Coordination Project (MRSCC)[6] aimed to produce a real-time architecture integrated with JADE in order to enable the creation of products with MAs for members of the supply chain. In this approach, each time a user logs on, a dedicated agent is designed to meet the request. MAs migrate to different hosts, searching for the best prices of an item, and can facilitate the transaction. However, the authors do not clearly present the use of a treatment mechanism for concurrent real-time requests.

TABLE 1 Comparison of related work

Ref.	JADE[1]	Deadline[2]	Preemption[3]	Conc.[4]	Algorithms[5]	Middleware[6]
8	no	no	no	no	no	yes
18	no	no	no	no	no	yes
19	no	no	no	yes	no	yes
24	no	yes	no	no	yes	no
9	yes	no	no	no	no	yes
6	yes	no	no	no	no	no
25	yes	no	no	no	no	no
26	yes	yes	no	yes	no	no
27	yes	no	no	yes	no	no
21	yes	yes	yes	yes	no	no

Does the work: (1) use the **JADE** framework? (2) considers **deadline**? (3) allow task **preemption**? (4) control **concurrency** of shared resources? (5) use different **algorithms** for scheduling MAs? (6) use a **middleware** approach?

Kuo and Lu[25] described the use of the JADE platform to manage MAs in a microgrid setting. In this case, the JADE platform uses MAs to gather real-time information about power and load from the energy sources. This information is further used to decide if power sources should become active in the grid. However, the paper does not approach the problem of real-time scheduling for MAs. Besides, the evaluation was done using simulation only.

JADE has been modified to use real-time threads and scheduling, as an effort to evaluate the impact of the JVMs and the difference of using real-time threads versus standard threads. [26] This evaluation focused on comparing real and non-real-time scenarios. Additionally, the authors mentioned the difficulties of scheduling and preemption but did not describe how they have tackled these problems. In particular, it is unclear if they support preemption, and which scheduling algorithm was used for the experiments. Also, the focus of this work was on prioritizing the execution of threads that were already on the host, to ensure that they met their deadlines. In contrast, our work seeks to schedule the queue of incoming MAs in the host.

Cemin et al[27] developed a concept to allow dynamic reconfiguration of MAs using the JADE platform. The focus of the work was to enable the coexistence of hardware and software agents on a network, allowing them to migrate among nodes or move from software to hardware (or vice versa) using high-level services provided by the proposed approach. In this case, specific processors could be used for achieving time requirements. However, the paper only evaluated the proposed solution based on the performance of the MAs in two different execution environments: hardware versus software.

A solution proposed by Cicirelli and Nigro[21] described mechanisms for modeling and analysis simulation in real-time systems with time constraints, using the JADE platform. The evaluation comprised a case study. A recent work from the same authors brought an example of a call center and a predefined set of tasks to be treated by the system.[28] Although this work is closely related to the approach we propose here, their work does not provide a middleware solution for runtime scheduling but instead focuses on modeling and analysis simulation. Additionally, our approach presents a model for generic agents which was evaluated with ten scheduling algorithms.

Table 1 presents a comparison of the related work. We selected the criteria for the evaluation based on the functionalities proposed in the researched papers. As demonstrated by this comparison, no scientific work has so far provided all the features considered. As such, we propose to architect a solution to fulfill this gap. With a middleware capable of managing deadlines considering multiple scheduling algorithms and a more effective deadline accomplishment through preemption, our approach seeks to enhance the current state of the art.

3 | BACKGROUND

This section presents the fundamentals of this work. First, we discuss the main aspects related to the concepts of concurrency and real-time scheduling. Then, we present the JADE platform, which uses the previously mentioned concepts to allow the implementation of multi-agent systems.

3.1 | Concurrency and real-time scheduling

Concurrency can be defined as two or more processes requesting the same resource at the same time.[29] This is an essential element of distributed programming. There are two types of interaction between concurrent processes,[30] namely competition and cooperation. Competition occurs when processes are independent but must share the common resources of a computer. Cooperation occurs when (i) processes are interdependent parts of the same application or (ii) processes are explicitly interacting with each other.

Scheduling techniques are used to solve concurrency concerns between processes. The scheduler is responsible for defining the execution order among candidate processes. The primary objectives of scheduling algorithms are Fairness, Policy Enforcement, and Balance.[31] The scheduling policy can be non-preemptive or preemptive. Non-preemptive scheduling queues the tasks according to their priorities but does not interrupt a task already in execution. Two examples of non-preemptive algorithms are FIFO and SJF (both are used in this work). With preemptive scheduling, when a task with higher priority arrives, the one currently using the resource is interrupted.[32] The scheduler then releases the higher priority task for execution. This procedure eliminates the risk of a low priority task retaining control over the resource. An example of preemptive algorithms is the SRTF (also used in this work).

In dynamic load scenarios, the arrival time of tasks is not previously known. The best-effort policy can be applied in such cases. This strategy is based on probabilistic studies (or previous history) of the expected load and acceptable failure scenarios. This approach must handle runtime situations where resources are insufficient to the task (overload situations). Best-effort does not provide any guarantees that all agents will meet their deadlines at the expected time. However, the system will endeavor to achieve this. Still, mechanisms such as the deadline sacrifice of some tasks are employed in overload situations. Hence, a prior definition of what tasks can be aborted is required in the system.

3.2 | Java Agent DEvelopment framework (JADE)

JADE[17] is a structured open-source platform to make the implementation of multi-agent systems faster, in accordance with the Foundation for Intelligent Physical Agents (FIPA) specifications. JADE can be considered as a middleware that implements a development framework and an agent platform. Therefore, it provides support for agents written in Java.

The JADE platform architecture is based on the coexistence of several JVMs being distributed over independent machines with different operating systems. Each JADE platform has one or more containers. All platforms have a main-container that provides an Agent Management System (AMS) for platform addressing, delivery and send messages, MA creation/destruction, and receipt. The architecture also has a Directory Facilitator to a yellow pages service and a Remote Method Invocation (RMI) registry to retrieve and record object references (ie, agents) through their names. Platform communication is performed by using RMI, as depicted in Figure 1.

A JADE agent is an autonomous and independent process with an ID which requires communication with other agents through collaboration or competition to complete its goals.[17] Each JADE agent is a separate execution thread performing multiple tasks in response to different external events. It has simultaneous conversations, and a private queue with a finite size created and stored by the JADE communication subsystem, which is designed to achieve the lowest cost in the exchange of messages. The handling of messages exchanged between external applications and the MAs is performed by the FIFO scheduling policy.

JADE also supports the mobility of agents in a platform that can be distributed, where hosts may have distinct operating systems. A MA is transported as a Java Archive (JAR) that contains the serialized state of the agent, among other data. Its configuration can be changed during execution, moving agents between hosts when necessary. The platform includes parallelism, finite-state machine, atomic behavior, and sequencing. Concurrency is only supported between different behaviors of an agent, in a non-preemptive manner. All settings can be controlled via a remote Graphical User Interface (GUI).

4 | RT-JADE: MIDDLEWARE FOR REAL-TIME MOBILE AGENTS

The RT-JADE middleware extends the JADE platform for MAs. We address the problem through the addition of a layer for real-time scheduling support. The scheduling uses best-effort policy independent of the selected algorithm. Note that no change in the code of the original JADE was

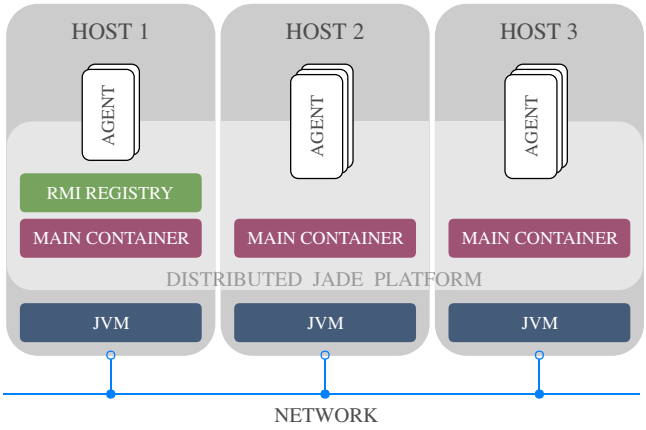

FIGURE 1 Distributed architecture of a JADE agent platform

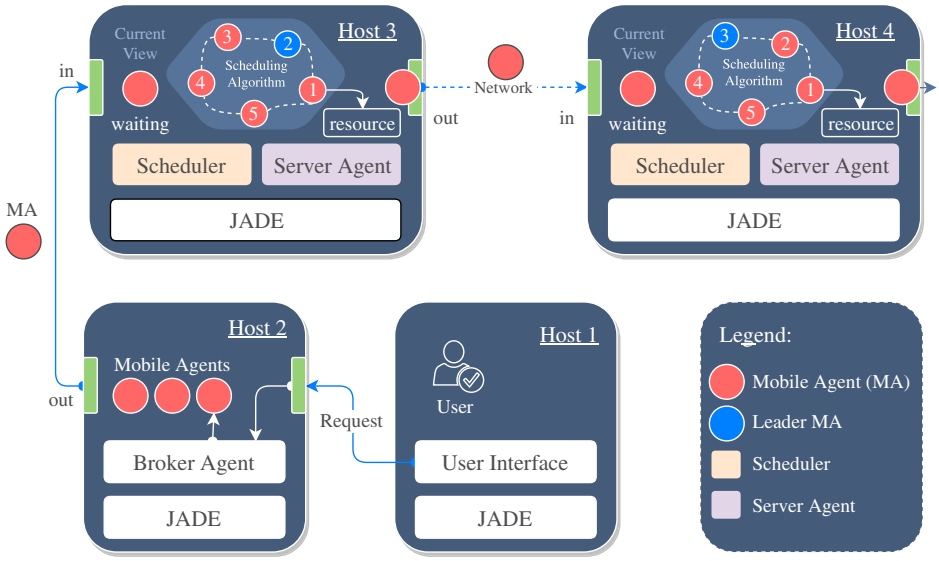

FIGURE 2 RT-JADE architecture

performed. Instead, we propose a transparent integration through the use of stationary agents and a protocol based on agent communication to define the scheduling. The source code for RT-JADE can be found at www.github.com/yeranarraila/rtjade.

4.1 | RT-JADE architecture

The RT-JADE architecture consists of stationary agents (eg, User Interface, Broker Agent, and Server Agent) and mobile agents, as depicted in Figure 2. The system also has the Schedulers, which are created by MAs. The whole system has up to *n* Server Agents, *n* Schedulers, and only one Broker Agent. The agents and the Scheduler run on the JADE platform. The function of the User Interface is to send messages to the Broker Agent and receive its replies. These messages use the ACLMessage format according to the FIPA standard.

Users can create request tuples to execute the required actions. Each request tuple contains multiple requests to a single agent, including the information regarding the required resources. The function of the Broker Agent is to instantiate one MA for each request tuple received (cf. Figure 2 - Host 1 and Host 2). According to the scheduling algorithm adopted, the Broker defines the priority elements (deadline and credential) of the MAs. Before sending them to the missions, the Broker randomly chooses a leader for that group. Additionally, the Broker Agent waits for the return of the MA to report the results of the mission to the application user through the User Interface.

A Server Agent is an interface for resource management in hosts. Its function is to receive MAs, communicate with them to provide estimated waiting times for using the resource, choose a leader for the view from the MAs contained in it, and provide access to the exclusive resource. Its internal structure (cf. Figure 3) consists of a Communication Layer, a Resource Access Layer, a Time Estimation Service, and the configuration of the scheduling algorithm that will be applied in the host. These layers are briefly described below.

The Communication Layer contains remote and local IDs (provided by JADE), which allows agents to communicate, regardless of the implemented platform or programming language. This layer has a communication interface with FIPA specifications. Finally, it has a list of linked hosts indicating the machines to which a MA can migrate. The Resource Access Layer provides the functionality required to coordinate the usage of the resources in the host. It contains a list of resources, an agent queue, and a Membership Service for view treatment.

The function of MAs is to migrate to hosts, return to the Broker, and inform the results obtained on their journey. A mission is completed only if all these steps were accomplished within the deadline. The internal structure of a MA (cf. Figure 3) consists of a Communication Layer and a Mission Management Layer. This latter layer is described below.

The Mission Management Layer contains the information required to accomplish the mission and the information necessary to provide a report once the mission is accomplished. In this layer, there is data provided by the Broker Agent, such as the mission to be performed, the requesting user ID, a deadline for the mission, a list of resources needed to complete the mission, a priority credential, and the Broker Agent host address. Additionally, it also has data provided by the Server Agent, such as the waiting time in the queue (estimated) for resource use, and the total estimated computation time required by the MA to use all the resources necessary to complete the mission in the current host. Finally, it also maintains a list of visited hosts, which is used for the final report.

The Scheduler (cf. Figure 2 - Host 3) sorts the MAs inside the view according to the configured scheduling policy. Then, it provides the ID of the MA (with the highest priority to use the resource) to the Server Agent. The Scheduler uses JADE FIPA ACL communication and has the same remote and local ID of the MA Leader that instantiated it (cf. Figure 3). The scheduler receives a current view from the MA and inserts it in the Agent Queue. Each scheduler has a distinct priority element (eg, DM = deadline, PRIO = credential). It sorts the agents according to this element and keeps the ordered view to report it to the Server Agent later.

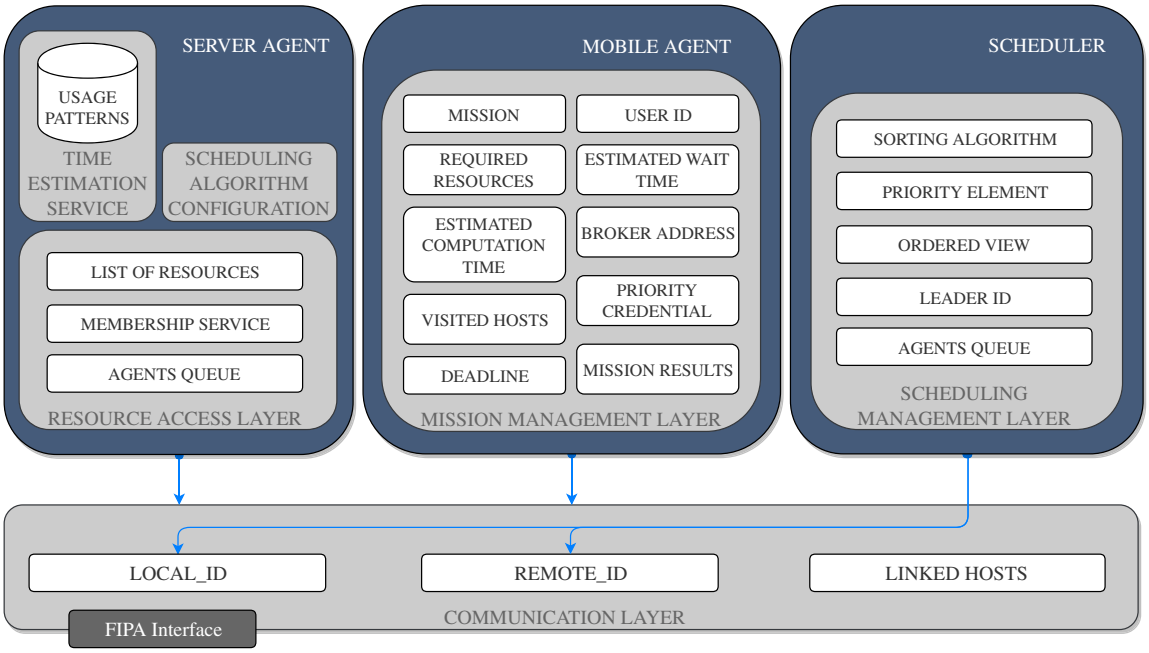

FIGURE 3 Server agent structure

4.2 | Execution model of real-time mobile agents

The model adopted considers a set of computers connected in a network, where hosts run the RT-JADE middleware. MAs can migrate to these hosts, perform their missions and, finally, leave to other hosts (cf. Figure 2). Note that the JADE platform has support for network failures by checking it before launching the MAs. Therefore, it is assumed that network failures are handled automatically by the JADE framework.

A mission is defined as a sequence of hosts (itinerary) that an agent must visit in order to consume resources. The mission is accomplished once the itinerary is fulfilled and resources are consumed in each host. In this case, each host can receive a maximum amount of MAs, limited only by its memory and processing capacity. Therefore, for real-time applications, it is necessary to organize agents following a scheduling algorithm, defining the order that agents must follow to have exclusive usage of the resource.

The interaction between client and server is accomplished through the use of MAs, although this is transparent for the user. For simplicity, we assume that each host has only one resource, which is exclusively used by the currently authorized MA. This means that the resource may only be used by another MA when the current MA releases that resource (either through mission accomplishment or scheduler preemption). Due to the IN/OUT agent dynamics in a host, the solution of agent scheduling is not trivial.

In this proposal, we have adopted the use of *views*, which indicate the MA set in a host at any given moment. The MAs at $view_i$ (current view) are ordered for execution according to the configured scheduling algorithm (cf. Figure 2, host 3), which runs only one time for a view. A view change (to $view_{i+1}$) can happen when a new MA arrives at the host (with preemption activated), or every time a MA in $view_i$ leaves the host and at least one MA is waiting to be scheduled (cf. Figure 2 - host 3). In these cases, a new view ($view_{i+1}$) is established, and the scheduling algorithm runs to set the new order in this view.

Depending on the scheduling algorithm adopted, the previous view order can be maintained, only inserting the waiting agents at the end of the queue. In preemptive cases, the new order is only calculated (and the view changed) if the incoming agent has a higher priority than the MA currently executing. Nevertheless, there are situations where the order is changed causing delays on the deadlines of re-ordered agents. In such case, the scheduled MA can give up waiting for the resource and, as a consequence, exit the queue and leave the host (migrating to another host with less overhead or going back to the Broker).

4.3 | Real-time scheduling algorithm for mobile agents

As explained in Section 4.2, scheduling is only performed for agents in the current view ($view_i$). This procedure can occur both in normal or crash runs. Figure 4 depicts the communication between the proposed components to define the schedule during a normal run, ie, an execution without crashes.

After receiving requests from users and creates a MA for each request, the Broker randomly chooses a leader for that group. The leading MA identifies itself on the ServerAgent upon arriving at the Host. The ServerAgent waits a time N for the response from the leader. If there is a timeout, the Server Agent randomly chooses one MA in the view to become the view leader (phase I). The elected MA is responsible for launching the Scheduler (phase II), which will define the order of execution of the MAs in the current view. The Scheduler requests to each MA

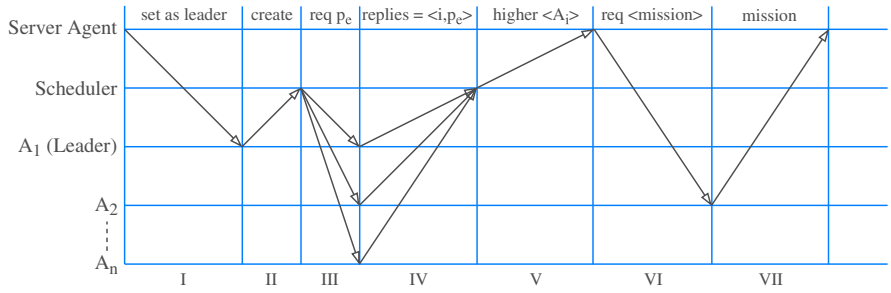

FIGURE 4 Scheduling on normal run

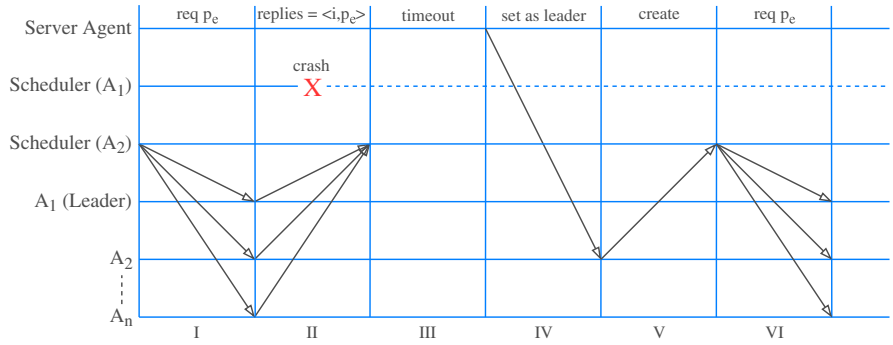

FIGURE 5 Scheduling after a crash

in the current view ($A1, A2, \ldots, A_n$) the priority element (p_e) for the scheduling algorithm (phase III), as set by the Broker. After receiving the data (phase IV), it chooses the highest priority based on the scheduling algorithm and informs the MA ID to the Server Agent (phase V). Hence, the MA with the highest priority receives authorization (phase VI) to use the exclusive resource (in this example, A2 has the highest priority). Once the agent finishes the resource usage, it informs the Server Agent and leaves the host (phase VII).*

Unwanted situations may also occur in the system during a scheduling procedure. For example, Figure 5 illustrates the behavior of the system during a crash. In this scenario, if a reply has not arrived before the timeout, the Server Agent randomly chooses a new leader to start a new view and re-initiates the process. This procedure avoids crashes in the whole system.

Not only crashes can cause the system to change the leader. After the MA Leader accomplishes its mission and leaves the host, a new leader is chosen for the next view. If the current view and the previous view are the same, ie, there was no incoming MA or preemption in the host, a new scheduling is not necessary. Instead, the Scheduler communicates the ID of the next MA for the exclusive use of the resource to the Server Agent.

Figure 6 presents the scheduling algorithms (preemptive and non-preemptive). Notations are presented in Table 2. The actions of the Server Agent algorithm are described in lines 3-27. When the host receives a MA, the Server Agent must verify if there are elements in the queue (line 4). When the queue is empty, it means that there is no leader and that the agent can be immediately activated (line 7). If the queue already has elements, it becomes necessary to verify if preemption is enabled (line 9). With preemption (line 10), the algorithm will always verify the priority of execution as soon as the agent arrives in the host (line 36). Without preemption (line 12), the agent will be added to a waiting queue until the process to generate a new view is triggered.

If a MA leaves the host before the exclusive resource is used (line 15), this MA is simply dequeued. In this case, it is not necessary to recalculate the priorities, as they will continue to be ordered inside the queue. When the MA leaves the queue, after the resource use (line 18), a new view is generated. This new view will add all the elements in the waiting queue to the execution queue (line 24), and the order of priorities will be recalculated (line 25). Once the order is defined, the agent with the highest priority is activated (line 27).

The actions of the MA consist of a verification that if it is the leader. If a MA is the leader, it creates an instance of the Scheduler (line 28). Finally, the Scheduler actions (lines 29-42) consist of three main tasks (subroutines): calculate the order of priorities, send a MA to the exclusive use of a resource, and attempt preemption. These actions are described below.

To calculate the order of priorities, first the Scheduler requests to each MA in the current view the priority elements p_e (lines 29-32). A p_e can be a deadline, a credential, or both. The Scheduler waits for the replies, storing them into a queue (line 31). In case of a timeout, no p_e value will be defined for the agent. As a consequence, the priority of this element will be reduced. Once the priority elements are defined, the Scheduler is started. The function *schedule()* sorts the received values (line 33). For FIFO policy, the leader only requests to the Server Agent to activate the next MA in the queue.

*The MA checks from time to time whether its host wait time is greater than or equal to 65% of its deadline. If it crosses this threshold, the MA immediately returns to the Broker. This 65% threshold value was arbitrarily defined after performing several tests, where it was found that this is the tightest limit for a mobile agent to reach the host within its deadline.

```
Shared variables:
      1.  queue = ∅          {queue of agents in the current view}
      2.  waiting_queue = ∅  {queue of agents not yet in the current view}

{Server Agent}
Main Task
      3.  On arrival of Ai
      4.    IF (queue is empty) THEN
      5.          enqueue(queue, Ai)
      6.          chooseLeader(queue)
      7.          activateAgent()
      8.    ELSE
      9.          IF (preemption is active && resource is allocated) THEN
      10.               attemptPreemption(Ai)
      11.         ELSE
      12.               enqueue(waiting_queue, Ai)
      13.         ENDIF
      14.   ENDIF
      15. On Ai leave before resource use
      16.   dequeue(queue, Ai)
      17.   dequeue(waiting_queue, Ai)
      18. On Ai leave after resource use
      19.   dequeue(queue, Ai)
      20.   IF (Ai is Leader) THEN
      21.         chooseLeader(current_view)
      22.   ENDIF
      23.   IF (waiting_queue has elements) THEN
      24.         enqueue(queue, waiting_queue)
      25.         runScheduler()
      26.   ENDIF
      27.   activateAgent()

{Mobile Agent}
subroutine setLeader(Leader)
      28. launchScheduler()

{Scheduler}
subroutine runScheduler()
      29. FOR each Aj in queue DO
      30.   pe = requestPE(Aj, t1)
      31.   enqueue(prioQueue, pe)
      32. ENDFOR
      33. schedule(queue, prioQueue)

subroutine activateAgent()
      34. send higher priority Aj in queue to exclusive resource

subroutine attemptPreemption(Ai)
      35. pe = requestPE(Ai, t1)
      36. IF Agent in execution(Aj) has lower priority than Ai THEN
      37.   suspend(Aj)
      38.   enqueue(queue(0), Aj)
      39.   send Ai to exclusive resource
      40. ELSE
      41.   enqueue(waiting_queue, Ai)
      42. ENDIF
```

FIGURE 6 Scheduling algorithm

The preemptive behavior is defined in lines 35-42. If this functionality is activated, the p_e of the arriving agent must be compared only to the p_e of the agent currently using the resource (line 36). If the priority of the arriving agent is higher, then the agent occupying the resource will be suspended and added back to the execution queue. The arriving agent will have access to the resource (line 39). If the priority of the arriving agent is lower, the agent is simply added to the waiting queue (line 41). The *activate()* and *suspend()* functions (lines 27 and 37) are provided by JADE.

TABLE 2 Notations

Symbol	Description
A_k	Mobile Agent *k*
enqueue(q,e)	Inserts element (e) in queue (q)
dequeue(q,e)	Removes element (e) from queue (q)
p_e	Priority Element

5 | EVALUATION

This evaluation compares the impact on mission accomplishment when applying different scheduling algorithms. Thus, ten scheduling algorithms were considered[33]: FIFO (First In, First Out) – already supported by JADE, LIFO (Last in, First Out), EDF (Earliest Deadline First), PEDF (Preemptive EDF), DM (Deadline Monotonic), PDM (Preemptive Deadline Monotonic), PRIO (Priority-Based Scheduling), PPRIO (Preemptive Priority-Based Scheduling), SJF (Shortest Job First), and SRTF (Shortest Remaining Time First).

In FIFO, tasks are delivered according to their arrival order, without warranties about deadline accomplishment. LIFO is the opposite of FIFO, considering the arrival order. The first task which arrives is the last that will be executed. As FIFO, there are not compromise with deadlines in LIFO. In EDF, dynamic priorities are assigned to the tasks, according to their deadlines. Tasks with deadlines closer have higher priority. PEDF allows a task, even without having finished its execution, to be preempted by a task with shorter deadline. The disadvantage, in this case, is the possibility of numerous switchings between tasks. The DM scheduling uses static priorities. Tasks with fewer absolute deadlines have higher priority. PDM is a preemptive version of scheduling with static priorities. In PRIO, tasks receive priority statically (when they are created) or dynamically (according to statistics of the last execution). In this case, some tasks can never be executed since other tasks with higher priorities can always arrive. PPRIO is similar, however, it allows a lower priority task to be preempted by a higher priority task. The preempted task is resumed as soon as the highest priority task ends its execution. In SJF, shorter tasks (with lower duration) receive higher static priority. The SRTF is a preemptive variant of the SJF scheduler. Initially, the shortest jobs are executed first. When a new job arrives, its execution time is evaluated (including the execution time of the job that is executing). If this estimated time is lower than the execution time of the current job executing, the new job gain the right of execution and the current job is preempted.

We selected a minimum mission scenario to perform the tests, which is capable of using all the functionality of RT-JADE. We consider that if the alternative scheduling algorithms perform better than the default provided by JADE in this scenario, likely similar results can be obtained on more complex applications.

The scenario we considered is inspired in a management information system, where there are at least two access levels. The first one gives operational information (OI), available for all business departments. The second one offers strategic information (SI) which is accessible only by managers. There is a dependency relation between these resources: OI is collected for a later processing and generation of SI. To test this scenario, we defined two server agents, each containing only one exclusive resource (cf. Figure 7). The first one (SA-OI) provides information originated from a database search resource (R1). The second (SA-SI) has a mathematical resource (R2).

The mission consists of departing from the Broker Agent component, passing through SA-OI, and, according to the access level, go to SA-SI for further data collection or return to the Broker Agent. For these tests, the access levels of the agents were randomly selected, resulting in different mission itineraries.

In this particular case, many tests were performed to estimate the minimum deadlines for MAs to accomplish their missions. First, a MA was released in the architecture with 50 ms deadline in 100 iterations, and it was realized that this time was not enough for the completion of its mission. After that, the deadline was increased to 100 ms and so on, until it reached the maximum value of 300 ms. This value of 300 ms allowed the MA to return to the Broker with either a totally or partly completed mission, being the smallest deadline used for the load of 5 mobile agents. Using the same deadline for the 20 MA load, it was noticed that there were no MA that fulfilled the mission, neither totally or partially. New tests were done with a load of 20 mobile agents, increasing gradually, every 50 ms until arriving at the value of 700 ms. The other deadline values, for both 5 and 20 MAs, were chosen in the same way, with the same tests. However, always observing the range where the increase of MAs with totally completed missions was significant. Therefore, the deadline values in the worst and best cases found and assumed during the tests were, respectively: for 5 mobile agents (Worst case: 300 ms, Best case = 1100 ms); for 20 mobile agents (Worst case = 700 ms, Best case = 2500 ms).

The deadline was provided to the agent at the moment that it is departing from the Broker Agent. To evaluate this proposal, we measured the Mission Accomplishment achieved by the agents in different environment setups.

The architecture components were implemented in JAVA (JDK 1.6.0_19), using the JADE framework (v.4.0.1). The evaluation environment consisted of three machines connected to a LAN: (i) Intel Core2Duo 2.4GHz, 1GB RAM; (ii) Intel Core2Duo 1.6GHz, 1.5GB RAM; and (iii) Intel Core Quad 3.0GHZ, 4GB RAM. All machines ran on Windows operating system.

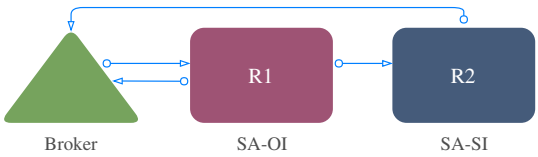

FIGURE 7 Scenario for evaluation of RT-JADE

5.1 | Mission accomplishment

The system was evaluated with two workloads: 5 agents (light) and 20 agents (heavy). Agents had to repeat their missions as soon as they returned to the Broker Agent. Each mission was executed 100 times, resulting in 500 missions on the light workload and 2000 missions on the heavy workload. Table 3 summarizes the parameters used in these assessments.

For the first workload, we selected five different deadlines which provide an overview of the system behavior: 300 ms, 500 ms, 700 ms, 900 ms, and 1100 ms. The workloads were executed for each scheduling policy and deadline. As result, we could verify the number of agents that fulfilled their missions, and the ones that did not meet their deadlines.

The results obtained from our experiments are depicted in Figure 8. For the first workload, agents running under LIFO, EDF, and DM policies fulfilled more missions than those scheduled by FIFO policy. Particularly, EDF and LIFO had a better overall performance than FIFO on this workload. FIFO did not allow all agents to complete their missions even for the largest deadline evaluated.

Table 4 presents the improvements obtained with the adoption of scheduling algorithms in the JADE platform when compared to FIFO policy. The majority of the algorithms evaluated presented higher mission accomplishment when compared to the original policy. In particular, for the shortest deadline, the LIFO algorithm showed results 2466% higher than FIFO for the same setup.

The expressive positive results obtained by LIFO were further analyzed. The improvements achieved with this algorithm can be visualized in Figure 9. This is explained by the complexity of the sorting algorithm applied. This sorting algorithm is based on the Collections.reverse() Java method, which has linear cost O(n), whereas the other algorithms use the FastQuickSort, which has an average and best-case performance of O(n log n) and $O(n^2)$ for worst-case.

Agents arrive at the hosts without a specific order. FIFO and LIFO do not consider priorities. The best-case scenarios for these methods can occur when agents with few resources could stay ahead of agents with more resources. In worst-case scenarios, agents with more resources stay ahead of agents with few resources. This case will result in a less amount of missions accomplished.

The SJF and SRTF algorithms performed worse than FIFO for some of the deadlines. The main reason for this behavior lies in the time estimation for the tasks. The missions were very simple and, in most cases, had the same average estimated time for execution. As such, the algorithm consumed more resources than FIFO to re-calculate the order of execution, but in most cases, the order was equivalent to FIFO. Although the performance of these algorithms in this testbed did not demonstrate advantages, it still can prove to be efficient in systems with higher complexity and different estimated times for tasks.

TABLE 3 Summary of tested workloads

Workload	1	2
Number of Agents	5	20
Number of Missions	500	2000
Tested Deadlines (in milliseconds)	300, 500, 700, 900, and 1000.	700, 1500, 2000, and 2500

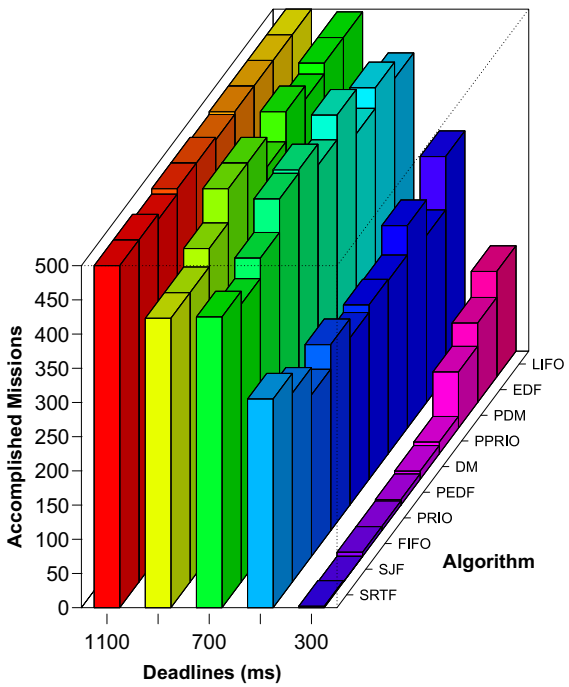

FIGURE 8 Overview of mission accomplishment of evaluated algorithms

TABLE 4 Mission accomplishment: comparison of algorithms

Deadline	Scheduling Algorithm								
	LIFO	EDF	PDM	PPRIO	DM	PEDF	PRIO	SJF	SRTF
300	2467	1833	1267	183	100	33	0	-100	-67
500	36	3	25	7	8	3	15	17	29
700	0	6	-1	14	5	12	11	-7	-3
900	10	10	7	11	1	11	11	-6	-6
1100	2	2	2	2	1	2	2	2	2

Algorithms performed better than FIFO (in %).

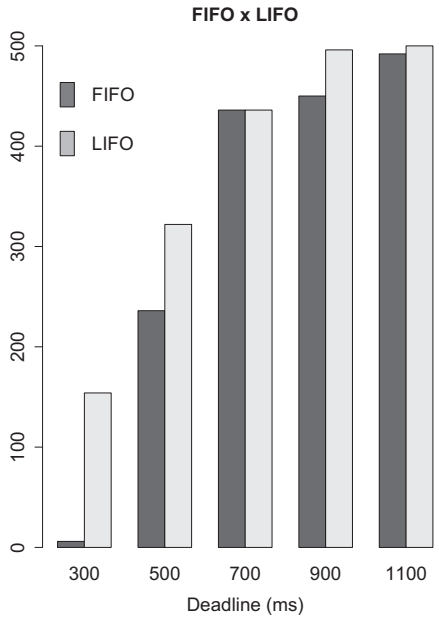

FIGURE 9 Comparison of LIFO and FIFO for 5 agents

For the second workload, we selected a different set of deadlines for evaluating the system: 700 ms, 1500 ms, 2000 ms, and 2500 ms. It was necessary to shift the window of deadlines to accommodate the additional overhead caused by the more complex tasks executed by the agents on SA-OI/SA-SI scenario. Table 3 summarizes the parameters used in this assessment. Figure 10 presents the results obtained for this workload.

This workload also confirms the hypothesis that the FIFO policy limits the potential capacity for executing agents in JADE. In most cases, FIFO had a worse performance in terms of mission accomplishment. Table 5 presents a comparison of algorithms with 20 agents concurring.

As FIFO did not allow agents to fulfill their mission for a deadline equals to 700 ms, the first line of Table 5 indicates an infinite increase. For a better comparison, we included an extra-line in this table (the 700* line) comparing all algorithms to the algorithm that presented the lowest performance (ie, PRIO). Similar to the results obtained in the light workload, EDF and PDM policies also performed better than FIFO policy. The PRIO scheduling policy presented satisfactory results, always fulfilling more missions than FIFO. For this simulation, the FIFO policy has the worst results in three of the four ranges evaluated.

In our experiments, a system with critical tasks (eg, lower deadlines) could be benefited from the usage of other methods other than FIFO, which is the default method of JADE. Systematically, the methods EDF, PDM, PPRIO, and DM presented great mission accomplishment results when compared with FIFO (Tables 4 and 5). The rational indicates that these methods keep presenting better results better on systems with more agents in the same way as in our experiments (ie, 5 and 20 agent loads).

5.2 | Impacts of preemption on mission accomplishment

In this section, we evaluate the impact of preemption on the proposed architecture. The principle of the preemptive approach is to halt the execution of a MA to give access to the exclusive resource to another higher priority agent. To compare the impacts of preemption (Table 6), we calculated the delta (Δ) for mission accomplishment between preemptive and non-preemptive algorithms.

The results demonstrate that preemption has an impact on the overall system performance, in particular under restrictive deadlines (300 ms and 1500 ms). This is explained by the frequent halt on execution that occurs under these conditions and the management costs associated with them. The halts ensure that higher priority tasks gain access to the resources. However, if halts occur frequently and the system does not have enough time to resume halted tasks, the result is a high number of halted missions that do not fulfill their deadlines.

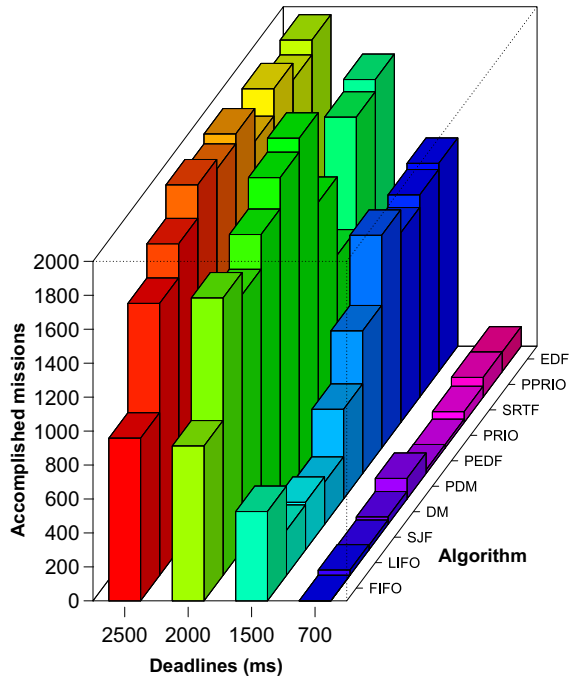

FIGURE 10 Mission accomplishments with 20 agents, different algorithms and deadlines

TABLE 5 Mission accomplishment: comparison of algorithms with 20 agents concurring

Deadline	Scheduling Algorithm								
	LIFO	EDF	PDM	PPRIO	DM	PEDF	PRIO	SJF	SRTF
700	∞	∞	∞	∞	∞	∞	∞	∞	∞
700*	55	475	505	480	120	5	0	220	30
1500	-50	133	0	126	-53	60	138	112	-47
2000	79	89	108	81	87	117	59	8	65
2500	67	104	103	95	109	109	88	105	88

Algorithms performed better than FIFO (in %)

The adoption of preemption should be considered for scenarios where certain tasks must be able to execute immediately. This is a frequent requirement for applications with safety considerations (eg, an assembly line must stop immediately if there is a risk of an accident).

It is also worth noting that, under certain conditions, the use of preemption results in improvements on the mission accomplishment metric, eg, all algorithms with 2000 ms as a deadline. Therefore, for a system design, it is important to evaluate the conditions under which the system will execute (deadlines, load, prioritization requirements) to select the appropriate scheduling policy.

TABLE 6 Mission accomplishment for preemptive and non-preemptive scheduling algorithms

Agents	Deadline (ms)	Δ(PEDF-EDF)	Δ(PDM-DM)	Δ(PPRIO-PRIO)	Δ(SRTF-SJF)
5	300	-108	70	11	2
	500	0	41	-19	29
	700	30	-29	10	19
	900	4	24	0	0
	1100	0	4	0	0
20	700	-94	77	96	-38
	1500	-387	280	-62	-837
	2000	254	185	202	516
	2500	47	-58	61	-163

6 | CONCLUSION

This paper presented the RT-JADE, a middleware with real-time scheduling support for JADE, which uses the concept of views to manage the scheduling of MAs in this platform. The proposed solution was evaluated with ten different scheduling algorithms. The adoption of other scheduling algorithms proved to increase the mission accomplishment metric in multiple cases. In particular, the use of LIFO, EDF and PDM algorithms resulted in great gains in the mission accomplishment performance under a restrictive deadline (eg, 300 ms), when compared to the original scheduling policy (ie, FIFO) provided by JADE.

In addition, the use of preemption in the scheduling of MAs brought new insights to the system behavior. We conclude that preemption is a powerful tool to be adopted under certain scenarios. Nonetheless, this approach must be used with prudence. By using preemption, high-priority tasks can halt the execution of tasks to gain access to the resource. This approach has good results under low-frequency halts. Under high-frequency halts, the system spends considerable resources on managing preemption, which in turn affects the mission accomplishment metric.

RT-JADE proved to be effective in allowing the use of time constraints on MAs. From the tests performed in this work, it can be concluded that there is the possibility of increasing both the number of nodes to be traversed in the network as well as the number of MAs to travel through the system as a whole. RT-JADE was created with the intention of allowing the distributed system developer to configure both the number of nodes present in the system and the number of MA per request, being their deadlines adjusted accordingly to their mission requirements. This study opened prospects for further research in the area of real-time scheduling for MAs. The calculations of the waiting time in the queue can be improved to consider an average based on queuing theory. With the analysis of the scheduling algorithms presented in this paper, it becomes possible to propose an adaptive scheduling approach, considering the deadlines and the number of concurrent agents in the host.

ACKNOWLEDGMENTS

This paper is dedicated to the memory of our dear colleague Prof. Lau Cheuk Lung, who motivated us to pursue this work. His contribution to this work was of great significance. Additionally, the authors would like to thank the financial support from CAPES/Brazil and CNPq/Brazil, offered through the following projects: LEAD Clouds (400511/2013-4), Abys (401364/2014-3), and ITSmartGrid (455303/2014-2).

ORCID

Hylson Vescovi Netto http://orcid.org/0000-0002-1929-7743

REFERENCES

1. Lange DB, Oshima M. Seven good reasons for mobile agents. *Commun ACM*. 1999;42(3):88-89. https://doi.org/10.1145/295685.298136 doi:10.1145/295685.298136
2. Gavalas D, Tsekouras GE, Anagnostopoulos C. A mobile agent platform for distributed network and systems management. *J Syst Softw*. 2009;82(2):355-371.
3. Symonds M. *Agent-Based Computing in Java* [Master's thesis]. San Jose, CA: San Jose State University. http://scholarworks.sjsu.edu/etd_projects/599; 2018
4. Taif F, Namir A, Azouazi M. Modeling, design and development of a multi-agent decision support system for the real-time control of the operating theaters. In: *Advances in Intelligent Systems and Computing*. New York, NY: Springer; 2018:3-16.
5. Prim RC. Shortest connection networks and some generalizations. *Bell Syst Technical J*. 1957;36(6):1389-1401.
6. Shemshadi A, Soroor J, Tarokh MJ. Implementing a multi-agent system for the real-time coordination of a typical supply chain based on the JADE technology. Paper presented at: IEEE International Conference on System of Systems Engineering; 2008; Singapore.
7. Baek J-W, Kim G-T, Yeom H-Y. Cost-effective planning of timed mobile agents. Paper presented at: International Conference on Information Technology: Coding and Computing; 2002; Las Vegas, NV.
8. Sahingoz OK, Erdogan N. A two-leveled mobile agent system for e-commerce with constraint-based filtering. Paper presented at: International Conference on Computational Science; 2004; Krakow, Poland.
9. Arunachalan B, Light J. Agent-based mobile middleware architecture (AMMA) for patient-care clinical data messaging using wireless networks. Paper presented at: 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications; 2008; Vancouver, Canada.
10. Wang S, He D, Long H, Goh MW. An intelligent manufacturing system: agent lives in adhesive slice. *Int J Comput Scie Netw Security*. 2006;6(5A):73.
11. Brennan RW, Fletcher M, Norrie DH. An agent-based approach to reconfiguration of real-time distributed control systems. *IEEE Trans Robot Autom*. 2002;18(4):444-451.
12. Chen B, Cheng HH, Palen J. Mobile-c: a mobile agent platform for mobile C/C++ agents. *Softw: Pract Exper*. 2006;36(15):1711-1733.
13. Wierlemann T, Kassing T, Harmer J. The on the move project: Description of mobile middleware and experimental results. In: *Advances in Wireless Communications*. New York, NY: Springer; 2002.
14. Bäumer C, Magedanz T. Grasshoppera mobile agent platform for active telecommunication networks. In: *Intelligent Agents for Telecommunication Applications*. New York, NY: Springer; 1999.
15. Lange DB, Oshima M, Karjoth G, Kosaka K. Aglets: Programming mobile agents in Java. Paper presented at: International Conference on Worldwide Computing and Its Applications; 1997; Tsukuba, Japan.

16. Gallagher N. Mobility-RPC. github.com/npgall/mobility-rpc. Accessed on July 10, 2018.
17. Bellifemine FL, Caire G, Greenwood D. *Developing multi-agent systems with JADE*. New York, NY: John Wiley & Sons; 2007.
18. Fok C-L, Roman G, Lu C. Mobile agent middleware for sensor networks: An application case study. Paper presented at: 4th International Symposium on Information Processing in Sensor Networks; 2005; Los Angeles, CA.
19. Aiello F, Fortino G, Gravina R, Guerrieri A. MAPS: a Mobile Agent Platform for Java Sun SPOTs. Paper presented at: 3rd International Workshop on Agent Technology for Sensor Networks; 2009; Budapest, Hungary.
20. Fortino G, Trunfio P. *Internet of Things Based on Smart Objects, Technology, Middleware and Applications*. New York, NY: Springer; 2014.
21. Cicirelli F, Nigro L. Modelling and analysis of parallel/distributed time-dependent systems: An approach based on jade. In: *Internet and Distributed Computing Systems*. New York, NY: Springer; 2014.
22. Filgueiras TP, Lung LC, de Oliveira Rech L. Providing real-time scheduling for mobile agents in the jade platform. Paper presented at: 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing; 2012; Shenzhen, China.
23. Fok C-L, Roman G-C, Lu C. Agilla: a mobile agent middleware for self-adaptive wireless sensor networks. *ACM Trans Autonomous Adapt Syst (TAAS)*. 2009;4(3):16.
24. Rech L, Montez C, de Oliveira RS. Itinerary determination of imprecise mobile agents with firm deadline. *Web Intelligence Agent Syst*. 2008;6(4):421-439.
25. Kuo M-T, Lu S-D. Design and implementation of real-time intelligent control and structure based on multi-agent systems in microgrids. *Energies*. 2013;6(11):6045-6059.
26. Król D, Nowakowski F. Practical performance aspects of using real-time multi-agent platform in complex systems. Paper presented at: International Conference on Systems Man and Cybernetics; 2013; Manchester, United Kingdom.
27. Cemin D, Götz M, Pereira CE. Dynamically reconfigurable hardware/software mobile agents. *Design Autom Embedded Syst*. 2014;18(1-2):39-60. https://doi.org/10.1007/s10617-013-9116-3
28. Cicirelli F, Nigro L. Control aspects in multiagent systems. In: *Intelligent Agents in Data-intensive Computing*. New York, NY: Springer; 2016.
29. Barland I, Greiner J, Vardi M. Concurrent Processes: Basic Issues, Connexions Web site. 2005. https://archive.cnx.org/contents/d5669d86-59b6-4fe6-a427-b14227e98f77@16/concurrent-processes-basic-issues
30. Shrivastava S, Banâtre J-P. Reliable resource allocation between unreliable processes. In: *Reliable Computer Systems*. New York, NY: Springer; 1985.
31. Chang J, Zhou W, Song J, Lin Z. Scheduling algorithm of load balancing based on dynamic policies. Paper presented at: 6th International Conference on Networking and Services(ICNS); 2010; Warsaw, Poland.
32. Khanna S, Zolnowsky J. Realtime scheduling in sunos 5.0. Paper presented at: USENIX Winter Conference; 1992; San Francisco, CA.
33. Ramamritham K, Stankovic JA. Scheduling algorithms and operating systems support for real-time systems. *Proc IEEE*. 1994;82(1):55-67.

How to cite this article: Filgueiras TP, Rodrigues LM, Rech LdO, de Souza LMS, Netto HV. RT-JADE: A preemptive real-time scheduling middleware for mobile agents. *Concurrency Computat Pract Exper*. 2018;e5061. https://doi.org/10.1002/cpe.5061