Future Generation Computer Systems 29 (2013) 2284–2294

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Self-healing of workflow activity incidents on distributed computing infrastructures

Rafael Ferreira da Silva [a,*], Tristan Glatard [a], Frédéric Desprez [b]

[a] University of Lyon, CNRS, INSERM, CREATIS, Villeurbanne, France
[b] INRIA, University of Lyon, LIP, ENS Lyon, Lyon, France

## HIGHLIGHTS

- Autonomous detection/handling of operational incidents in workflow activities.
- Executions speed up to a factor of 4 and consume up to 26% less resources.
- Self-healing loop properly detects unrecoverable errors.

## ARTICLE INFO

*Article history:*
Received 10 August 2012
Received in revised form
14 February 2013
Accepted 6 June 2013
Available online 21 June 2013

*Keywords:*
Error detection and handling
Workflow execution
Production distributed systems

## ABSTRACT

Distributed computing infrastructures are commonly used through scientific gateways, but operating these gateways requires important human intervention to handle operational incidents. This paper presents a self-healing process that quantifies incident degrees of workflow activities from metrics measuring long-tail effect, application efficiency, data transfer issues, and site-specific problems. These metrics are simple enough to be computed online and they make little assumptions on the application or resource characteristics. From their degree, incidents are classified in levels and associated to sets of healing actions that are selected based on association rules modeling correlations between incident levels. We specifically study the long-tail effect issue, and propose a new algorithm to control task replication. The healing process is parametrized on real application traces acquired in production on the European Grid Infrastructure. Experimental results obtained in the Virtual Imaging Platform show that the proposed method speeds up execution up to a factor of 4, consumes up to 26% less resource time than a control execution and properly detects unrecoverable errors.



## 1. Introduction

Distributed computing infrastructures (DCI) are becoming daily instruments of scientific research, in particular through scientific gateways [1] developed to allow scientists to transparently run their analyses on large sets of computing resources. While these platforms provide important amounts of resources in an almost seamless way, their large scale and the number of middleware systems involved lead to many errors and faults. Easy-to-use interfaces provided by these gateways exacerbate the need for properly solving operational incidents encountered on DCIs since end users expect high reliability and performance with no extra monitoring or parametrization from their side. In practice, such services are often backed by substantial support staff who monitors running experiments by performing simple yet crucial actions such as rescheduling tasks, restarting services, killing misbehaving runs or replicating data files to reliable storage facilities. Fair QoS can then be delivered, yet with important human intervention.

For instance, the long-tail effect [2] is a common frustration for users who have to wait for a long time to retrieve the last few pieces of their computations. Operators may be able to address it by rescheduling tasks that are considered late (e.g. due to execution on a slow machine, low network throughput or just loss of contact) but detection is very time consuming and still approximate.

Automating such operations is challenging for two reasons. First, the problem is online by nature because no reliable user activity prediction can be assumed, and new workloads may arrive at any time. Therefore, the considered metrics, decisions and actions have to remain simple and yield results while the application is still executing. Second, it is non-clairvoyant due to the lack of information about the applications and resources in production conditions. Computing resources are usually dynamically provisioned from heterogeneous clusters, clouds or desktop grids without any reliable estimate of their availability and characteristics. Models of application execution times are hardly available either, in particular on heterogeneous computing resources.

* Corresponding author. Tel.: +33 0 4 7243 7.
*E-mail addresses:* rafael.silva@creatis.insa-lyon.fr, rafaelsilvajp@gmail.com (R. Ferreira da Silva), glatard@creatis.insa-lyon.fr (T. Glatard), Frederic.Desprez@inria.fr (F. Desprez).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.06.012

A scientific gateway is considered here as a platform where users can process their own data with predefined applications workflows. Workflows are compositions of *activities* defined independently from the processed data and that only consist of a program description. At runtime, activities receive data and spawn *invocations* from their input parameter sets. Invocations are assumed independent from each other (bag of tasks) and executed on the DCI as single-core *tasks* which can be resubmitted in case of failures. This model fits several existing gateways such as e-bioinfra [3], P-Grade [4], and the Virtual Imaging Platform [5]. We also consider that the files involved in workflow executions are accessed through a single file catalog but that storage is distributed. Files may be replicated to improve availability and reduce load on servers.

The gateway may take decisions on file replication, resource provisioning, and task scheduling on behalf of the user. Performance optimization is a target but the main point is to ensure that correctly-defined executions complete, that performance is acceptable, and that misbehaving runs (e.g. failures coming from user errors or unrecoverable infrastructure downtimes) are quickly detected and stopped before they consume too many resources.

Our ultimate goal is to reach a general model of such a scientific gateway that could autonomously detect and handle operational incidents. In this work, we propose a healing process for workflow activities only. Activities are modeled as Fuzzy Finite State Machines (FuSM) [6] where state degrees of membership are determined by an external healing process. Degrees of membership are computed from metrics assuming that incidents have outlier performance, e.g. a site or a particular invocation behaves differently than the others. Based on incident degrees, the healing process identifies incident levels using thresholds determined from platform history. A specific set of actions is then selected from association rules among incident levels. We specifically study the long-tail effect issue, and propose a new algorithm to control task replication.

Section 2 presents related work. Our approach is described in Section 3 (general healing process), Section 4 (metrics used to quantify incident degrees), Section 5 (identification of incident levels), and Section 6 (actions). Experimental results are presented in Section 7 in production conditions.

## 2. Related work

Managing systems with limited intervention of system administrators is the goal of autonomic computing [7], which has been used to address various problems related to self-healing, self-configuration, self-optimization, and self-protection of distributed systems. For instance, provisioning of virtual machines is studied by Nguyen et al. [8] and an approach to tackle service overload, queue starvation, "black hole" effect and job failures is sketched by Collet et al. [9].

An autonomic manager can be described as a so-called MAPE-K loop which consists of monitoring (M), analysis (A), planning (P), execution (E) and knowledge (K). Generic software frameworks were built to wrap legacy applications in such loops with limited intrusiveness. For instance, Broto et al. [10] demonstrate the wrapping of DIET grid services for autonomic deployment and configuration. We consider here that the target gateway can be instrumented to report appropriate events and to perform predefined actions.

Monitoring is broadly studied in distributed systems, both at coarse (traces, archives) and fine time scales (active monitoring, probing). Many workload archives are available. In particular, the grid observatory [11] has been collecting traces for a few years on several grids. However, as noted by Iosup and Epema [12], most existing traces remain at the task level and lack information about workflows and activities. Application patterns can be retrieved from logs (e.g. bag of tasks) but precise information about workflow activities is bound to be missing. Studies on task errors and their distributions are also available [13,14], but they do not consider operational issues encountered by the gateways submitting these tasks. Besides, active monitoring using tools such as Nagios [15] cannot be the only monitoring source when substantial workloads are involved. Therefore, we rely on traces of the target gateway, as detailed in Section 5. One issue in this case is to determine the timespan where system behavior can be considered steady-state. Although this issue was recently investigated [16], it remains difficult to identify non-stationarities in an online process and we adopt a stationary model here.

Analysis consists in computing metrics (a.k.a. utility functions) from monitoring data to characterize the state of the system. System state usually distinguishes two regimes: properly functioning and misfunctioning. Zhang et al. [17] assume that incidents lead to non-stationarity of the workload statistics and use the Page-Hinkely test to detect them. Stehle et al. [18] present a method where the convex hull is used instead of hyper-rectangles to classify system states. As described in Section 5, we use multiple threshold values for a given metric to use more than two levels to characterize incidents.

Planning and actions considered in this work deal with task scheduling and file replication. Most scheduling approaches are clairvoyant, meaning that resource, task, error rate and workload characteristics are precisely known [19,20]. The heuristics designed by Casanova et al. [21] for the case where only data transfer costs are known are an exception, on an offline problem though. Quintin and Wagner [22] also propose an online task scheduling algorithm where only some characteristics of the application DAG are known. Camarasu-Pop et al. [23] propose a non-clairvoyant load-balancing strategy to remove the long-tail effect in production heterogeneous systems, but it is limited to Monte-Carlo simulations.

The general task scheduling problem is out of our scope. We assume that a scheduler is already in place, and we only aim at performing actions when it does not deliver expected performance. In particular, we focus on site blacklisting and on dynamic task replication [24] to avoid the long-tail effect.

Task replication, a.k.a. redundant requests is commonly used to address non-clairvoyant problems [2], but it should be used sparingly, to avoid overloading the middleware and degrading fairness among users [25]. In this work, task replication is considered only when activities are detected blocked according to the metric presented in Section 4. An important aspect to be evaluated is the resource waste, a.k.a. the cost of task replication. Cirne et al. [2] evaluate the waste of resources by measuring the percentage of wasted cycles among all the cycles required to execute the application.

File replication strategies also often assume clairvoyance on the size of produced data, file access pattern and infrastructure parameters [26,27]. In practice, production systems mostly remain limited to manual replication strategies [28].

## 3. General healing process

An activity is modeled as an FuSM with 13 states shown on Fig. 1. The activity is initialized in `Submitting Invocations` where all the tasks are generated and submitted. Tasks consist of 4 successive phases: initialization, inputs download, application execution and output upload. They are all assumed independent, but with similar execution times (bag of tasks). `Running` is a state where no particular issue is detected; no action is taken and the activity is assumed to behave normally. `Completed` (resp. `Failed`) is a terminal state used when all the invocations are