CONTENTS

1. Introduction

The rising demand for computing time by scientists from different fields of research is an ongoing trend. This demand has been answered with more and more powerful computing systems. Nevertheless, the demand for further resources persisted, so the number of resources (like available CPU cores) has increased dramatically over the last several years.

To enable easy resource usage for such systems, a set of techniques was introduced. Such techniques are e.g., Portal-Systems, Grid and Cloud services. These techniques enable scientists with moderate knowledge about computer science to use huge amounts of resources by providing easier access. A drawback of the techniques is the decreased observability of the executed processes (which we call jobs). The reasons for this are the many introduced abstraction layers such as middlewares, batch systems, service layers, virtualization, etc. – where each layer hides information to allow for easier usage. As a result, it is unclear how efficient the resources are used, and silent errors during job execution remain mainly unobserved.

One solution dealing with the additional layers of job execution is job-centric monitoring; this offers online job observation and automatic post mortem analysis. In [11, 12], we showed how to build an infrastructure to handle job-centric monitoring data for huge installations. The analysis of monitoring data is a common Big Data challenge. Therefore, we started by studying related work [14], where we identified a set of analysis techniques like genetic algorithms [6, 7, 17, 29], machine learning [2, 19–21], sequence comparison [1, 9, 24, 25], intrusion detection [5, 26, 27], and statistic of events [4, 18, 28]. The most promising technique is a similarity comparison [8, 13]. However, this method is complex and computing-intensive.

Thus, we need to check how less-computing-intensive analysis performs and whether it is possible to execute a pre analysis to reduce the number of jobs that must be handled by more-complex algorithms. The analysis of series of measurements from job-centric monitoring works in principle like the following: for a group of jobs that are expected to have similar behavior, a reference is defined. Such a group of jobs can be based on a user running similar types of applications or of a specific application or service that is executed by different users. In each scenario, varying input data is used. Jobs behaving the same as the reference are marked as error-free execution. Outliers must be analyzed further.

In the following section – Section 2 – we give examples of two typical usage scenarios where job-centric monitoring can be helpful. A description of the domains and environments for the examples is also given. Afterwards. we present the fundamentals of job-centric monitoring – Section 3. In Section 4, we explain the test data based on two so-called basic jobs. To test the detection potential of the analysis methods, we varied the basic jobs by applying aberrations. The aberrations represent possible faults in job execution or changes in the execution environment. Afterwards, we describe the statistical functions and exemplify the potential of detecting faults in job execution in Section 5. The description is based on basic jobs and the applied

aberrations. An outlook to further test data is given in Section 5.6. To classify the detection potential of the statistical functions, we present a comparison of an analysis method based on similarity functions in Section 6. Section 7 refers to the examples from Section 2 and demonstrates the usefulness of job-centric monitoring in three use cases. In the last section – Section 8 – we give a conclusion and point out further work.

2. Domains for job-centric monitoring

In the following, we describe domains that can benefit from job-centric monitoring. In section 7, we give more-detailed information for real-world examples of this domain. The first domain is the German Grid infrastructure (one of our customers we supported as resource provider). As a resource provider, we allowed other users to access computing and storage systems. Therefore, we provided access via different grid middlewares; e.g., Globus[1]. The grid middlewares used a mapping mechanism to map grid users to generic/internal users of the computing center. This is needed to submit a user's job to the batch system. The batch system cares about allocating hardware for the user and moves the user's job to the operating system of the allocated hardware.

Furthermore, this is needed to invoke an additional abstraction layer; therefore, we see a lot of abstraction layers just to start a job. It is clear that the job needs computing resources (hardware) and an operating system. In addition, most jobs need some software libraries and interpreters like Perl or Java. In our example, the job wants to run a special program to simulate parts of the cardiovascular system to prepare for a medical operation. This program is maintained by a group that builds services for health personnel. The developers of the program make up a scientific research group. The services for health personnel is based on additional hardware (not maintained by us as grid resource provider) and additional layers of software.

Back to our example; to enable very easy access for the health personnel, web browser-based access to a portal is provided. The portal allows the user to access different storage locations on the grid for uploading data and accessing different programs (like the one from above) as well as a workflow editor for combining multiple programs (e.g., transforming the uploaded medical data, so that it can be used as input for a fluid dynamic's simulator, to run multiple simulations for the cardiovascular system of a patient, and interpreting the results).

To sum up our small example, we have a lot of different abstraction layers and different groups of people involved in the process; thus, the system is quite complex and error-prone. A missing library, a poorly configured workflow, or invalid input data can stop the system from working properly. And a completed workflow execution dose not mean the absence of silent faults or near-optimal usage of the resources. A way to make such a complex system more transparent is job-centric monitoring.

[1] `http://toolkit.globus.org/toolkit/`