Towards an Environment Supporting Resilience, High-Availability, Reproducibility and Reliability for Cloud Applications

Vlado Stankovski, Salman Taherizadeh
University of Ljubljana
Ljubljana, Slovenia

Ian Taylor, Andrew Jones
Cardiff University
Cardiff, United Kingdom

Carlo Mastroianni
Italian National Research Council
Cosenza, Italy

Bruce Becker
Council for Scientific and Industrial Research
Pretoria, South Africa

Heru Suhartanto
Universitas Indonesia
Jakarta, Indonesia

***Abstract*—This paper presents a design study of an environment that would provide for resilience, high-availability, reproducibility and reliability of Cloud-based applications. The approach involves the use of a resilient container overlay, which provides tools for tracking and optimizing container placement during the course of a scientific experiment execution. The system is designed to detect failure and current performance bottlenecks and be capable of migrating running containers on the fly to servers more optimal for their execution. This work is in the design phase and therefore in this paper, we outline the proposed architecture of system and identify existing container management and migration tools that can be used in the implementation, where appropriate.**

***Index Terms*—Cloud, containers, resilience, high-availability.**

I. Introduction

The Software as a Service (SaaS) approach, as well as distributed computation and computational pipelines, are routinely used in most scientific disciplines to provide a systematic mechanism for describing the science that needs to be performed and how. With the drastic increase of raw data volume in every domain, Cloud computing plays an increasingly critical role to assist scientists in organizing and processing their data, and to leverage High-Performance Computing (HPC) and High-Throughput Computing (HTC) resources. The same applies to business applications involving long-running transactions or serving many end-users or devices. The lifecycle of such widely used Cloud applications requires management techniques which cover a range of aspects relating to the creation and execution of jobs, such as resource discovery and mapping, scheduling, monitoring, logging, provenance, etc. Such algorithms generally assume there is a reasonably stable infrastructure in which dealing with failures is an exception, not the rule. In this work we aim to address some user requirements that are difficult to satisfy due to the inherent instability of infrastructures, and we are particularly motivated by the problems which arise in developing countries, where instability is sometimes particularly severe.

This instability can be caused by a number of issues, such as component failures, power outages and Internet outages. If an imminent failure is suspected, the potential problem should be resolved, instead of the application being disrupted. This means that Cloud-based applications must be resilient. Resilience is the ability of a server, network, storage system, or an entire data center to continue operating even when there has been an equipment failure, power outage or other disruption. Resilience, in fact, should be the underpinning of any distributed system, as Tanenbaum [1] emphasises: "a distributed system is a collection of independent computers that appears to its users as a single coherent system … and should be permanently available (even though parts of it may not be)".

Tools and technologies are therefore needed to allocate and manage computing resources dynamically, making it possible to maintain and restore a desired level of Quality of Service (QoS) when failures occur. To achieve resilience, it is necessary to distribute redundant implementations of computing and storage resources across physical locations. Computing and storage resources need to be pre-configured, so that if one computing process becomes deficient, the processing can be automatically continued at another redundant computing resource.

End-user QoS requirements can include processing time (especially in the case of long-running scientific experiments), downtime in cases of outages, latency, reliability, business continuity, and so on. The end-user QoS requirements vary considerably, depending on the characteristics of the applications. Accordingly, a rigorous QoS methodology is required when deciding upon the optimal allocation of resources, which can be achieved through extensive profiling of use cases and applications.

While resilience is a property of the Cloud system, which is mainly sought by Cloud providers, end-users' requirements frequently also include the following:

- **Availability** is the degree to which a service is operable; that is, capable of producing responses to submitted requests. Stronger definitions of availability (e.g. high-availability) may include constraints with regard to the time window allowed for any response to arrive, or the time window allowed for the system not to be operable (e.g. relevant for business applications involving many transactions).
- **Reproducibility** is the ability of a computational experiment to be reproduced, either by the researcher or by someone else working independently. This is another property of distributed applications that is frequently required also by researchers in developing countries, as much as anywhere else, but it is made difficult by the instability of their systems.
- **Reliability** is a property where applications consistently perform according to their specifications. In the context of a long-running application, this means that both its QoS requirements and its computational integrity are satisfied.

Not being able to satisfy these key requirements may result in significant financial consequences, impacting customer trust in the applications they use, and impacting the application providers' trust in the Cloud services upon which they rely.

Our aim is therefore to address the basic instability of Cloud services that can be caused by problems such as component failures in highly distributed environments, frequent power outages and Internet outages. Our goal is to provide an overall architecture along with a set of open source tools that contribute to resilience, high-availability, reproducibility and reliability of Cloud-based services.

II. Concept and Approach

The term virtualization was first coined in the 1960s to refer to the use of Virtual Machines (VMs). Today, virtualization is understood by many as a common name for Cloud computing technology, and it can be achieved both through the use of VMs and containers. Key techniques for achieving resilience in Cloud computing environments include checkpointing, migration and replication [2]. Here, we plan to develop an architecture within which these techniques are combined effectively to maintain QoS. Although the overhead for these techniques is often undesirable when the resources and infrastructure are sufficiently reliable, in the context of our present work, replication is an important technique for mitigating the kinds and extent of failure encountered in highly distributed systems.

The key concept is to provide a virtual overlay over existing Clouds to enable mobility for long running jobs and to sustain resilient execution, through the use of previously-mentioned resilience techniques, in order to make efficient use of the underlying computational resources as and when they are available. At the crux of this architecture, we will use containers to provide a lightweight mechanism for deploying applications and moving them within and between Clouds; the more traditional Virtual Machine-based techniques are too heavyweight to achieve the agility we require. The next section will provide the rationale for this approach. We will then explain the proposed architecture, first describing why a component-oriented, microservice-based architecture will provide the flexibility we require, and then describing the high-level architecture in more detail. Within this architecture, some of the components will be specifically container-related, while others will be Cloud-related; we enumerate these in the following sections. In addition, there needs to be a way in which users can interact effectively with this system; we outline requirements for a Dashboard that will provide this facility. A testbed will be needed, and we outline what this will provide.

III. High Level Architecture

Our aim is to take advantage of the existing global technologies. This architecture is specifically designed as a means of achieving resilient operation of federated, heterogeneous Clouds, and will use Open Source solutions wherever possible.

The architecture will be realised at several levels: tools and services at the different levels of container, Cloud and end-user tools, which can be used from the Web. Before we provide an overview of the components in more detail in the following subsections, we first provide a snapshot of the underlying strategy. Cloud applications will provide their application code, their data and their QoS requirements. Then we will broadly proceed through the following steps:

- The code for the application will be containerised, i.e. a container (with preloaded tools) will be extended to include all of the libraries required for the target application and the application itself.
- The application dataset, which for long running applications is typically very large, will be hosted on a scalable distributed storage and retrieval system, which is a key backbone of the middleware stack. This dataset will be therefore accessible from any container on the network.
- The resulting container will be deployed in the container hub, ready for use. It will also be made available through the dashboard to the user for configuration and execution.
- The QoS parameters will be attached to the application, using the dashboard, at which point the application is ready to execute.
- Once running, the checkpoint scheduler will schedule both file system (using the hub) and live running

checkpoints (using the data repository) to checkpoint within the QoS constraint interval.

- Upon the detection of a failure, the Live Migration tool will check for the last memory and file-system checkpoint and provision a different server to run a container using this last checkpointed state. It will then inform the backend DB of this update so the user can track the migrated application.
- The live migration tool (if the user chooses) can also notify the user of this failure and request permission to move the application before migrating to a new container.

A. Container-related components

In many research experiments, it is necessary to save the state of multiple processes in order to be able to restore them later on the same or different host. To accomplish this, an application checkpointing and migration mechanism is needed that is capable of migrating applications from one host that may become disconnected from the network, to another host that can continue where it left off. Migration can also be used to improve availability by evacuating applications to new locations before host maintenance so that they continue to run with minimal downtime. Some in-container mechanisms like CRIU (Checkpoint/Restore In Userspace) [3] could be used in this process, since CRIU provides the capability to migrate live Linux containers. In-container tools could play the role of an agent that pushes information out to a service that maintains QoS by appropriate replication, etc.

The functionalities that will be implemented at container-level include the following: (1) live checkpointing (e.g. CRIU) for snapshotting the memory. To determine checkpoint frequency, QoS constraints will be used from the Cloud applications side; (2) interfaces for applications to snapshot themselves (i.e. write files out periodically according to their own QoS requirements); (3) introspection facilities, so that containers can report state (e.g. to the dashboard), reconfigure themselves, etc.; (4) mechanisms to create and store snapshots of multi-container configurations. The nature of this functionality is more static than the previous functionalities.

B. Cloud related components

In situations when an application is unreachable or unresponsive, it is necessary to save the container's file system along with the state of given container (all the processes and their resources) into disk files; then the files are copied to another server; and the container finally is restarted. A checkpointing mechanism can minimize the amount of lost computation when such migration occurs. Accordingly, we will design a container tracking and migration tool that is capable of tracking containers, and periodically storing a snapshot of the file system and its state, so that any application can resume from its last saved checkpoint. In other words, we will develop a process of checkpointing and restarting which makes it possible to move a running application (in a container) from one server to another without a reboot and recommencement from the beginning of the computation. The underlying goal is to provide a virtual overlay over existing Clouds which will provide mobility for long running jobs to sustain resilient execution, through the use of dynamic checkpointing and migration, in order to make efficient use of the underlying computational resources as and when they are available. As far as possible, the operation of this virtual overlay will be transparent to the user, so that details of its operation will be visible and controllable on demand, but under normal circumstances the computation will proceed automatically towards completion, without user intervention, regardless of failures and outages. The following components are required:

- ***Checkpoint Scheduler*** is an asynchronous process for scheduling checkpoints for each application. It receives pushes from containers for memory snapshots and periodically snapshots containers - e.g. it can use the Docker commit operation to snapshot a Docker container to a Docker hub;
- ***Container Hub*** is a repository, e.g. Docker hub, for interfacing with the container implementation for taking snapshots;
- ***Container Repository*** is a library of pre-built containers for specific applications;
- ***Data Repository*** exposes a REST API for storing memory snapshots from containers. This component creates a "time machine" view of the container snapshots and then allows an application to restart at any point along the snapshot continuum;
- ***Live Migration Service*** queries the data repository to find current running instances and timestamps. If a timestamp indicates that there is failure to satisfy the QoS specified for that particular application instance, the application will be migrated from the last known snapshot and will be moved elsewhere to continue execution;
- ***Distributed file system*** that allows applications on containers to interact with large data sets, etc. This also needs to be resilient, like the containers. Many distributed file system implementations already exist, e.g. iRODS [4], which potentially meet our requirements. Interaction with distributed storage infrastructures such as EUDAT is also possible.

C. Dashboard-related components

A Web-based Dashboard will be provided, which will allow users to interact with and monitor the real-time availability of their applications and the underlying Cloud infrastructure. Availability metrics will be sent to a dashboard to display dynamic real-time information on the state of the application. When failure occurs, the dashboard can interface with users in cases where any decision-making is required. The dashboard will be capable of visualising how the applications are working, present notifications to inform users that something failed, and provide options for configuring and supporting software migration, and so on. The Dashboard is therefore essentially a gateway to online provider tools and resources. It will model and assess the overall availability of the Cloud infrastructure. Practically, the dashboard will provide a comprehensive view of IT infrastructure and services with respect to multiple management aspects, based upon an evaluation of multiple resources including servers, storage, networking, power, etc.

The Dashboard will include the following tools and features:

- user accounts for authorization and authentication, allowing users' role-based access to the system;
- an application submission interface;
- an application instance timeline for monitoring progress;
- inspection/monitoring/access of the running containers to enable users to view the application output in real-time (e.g. using a command style dashboard widget), log into a running container; visualise application state, and to provide a remote desktop to the container, depending on requirements;
- a time machine mechanism to allow the user to go back to any snapshot, log in and inspect the application state at that time. This helps to identify any execution issues at various stages;
- mechanisms and interfaces exposed for notifying the application authors when something has happened, e.g. to give them the opportunity to control the migration process;
- monitoring views of the applications e.g. CPU, load, disk space, etc. of the running container.

D. Technical Issues

Given the types of Cloud applications for which resilience and high availability must be provided, a number of issues at a technical level will be investigated as follows.

- When to migrate containers. Live migration of containers from unreachable or unstable servers to provide resilient Cloud services is a fundamental goal. A crucial decision that must be made in this situation relates to determining the best time to migrate containers to maximise resilience, while still providing high availability.
- Which containers to migrate. Once a decision to migrate containers from a server is made, one or more containers must be selected from the full set of containers running on one server, to be migrated and reallocated on a new server. The problem consists of determining the best subset of containers to migrate that will provide the most beneficial system reconfiguration.
- Where to migrate the containers selected for migration, and where to place new containers. Determining the best placement of new containers or the containers selected for migration to other servers is another essential aspect that must be considered by the system.
- How to minimise time to migrate and place a container. Rapid migration is needed in order to minimise delays.
- Number of checkpointing intervals. One of our challenges is to compute the optimal number of checkpointing intervals in the context of Cloud computing for minimizing the inevitably extremely large size of collective memory usage, when we set equidistant checkpoints. Both the QoS limits and underlying physical limitations of the networks must be included in this calculation.
- Checkpoint image size. The solution will build highly specific and highly optimised container images tuned for minimal size and management overhead. This is important because actual memory usage is one of the performance requirements.
- Which metrics to monitor. A monitoring tool will collect resource usage data including throughput, response time, application, power, cooling and so on. This information must then be processed (filtered, converted, aggregated, etc.) to generate availability metrics using a set of availability models.

IV. CONCLUSIONS

The pilot Cloud applications that are to be made resilient by using the environment rely strongly on the Software-as-a-Service (SaaS) delivery model. The SaaS approach is currently used extensively in research, education and business purposes.

Over the past several years, resilience has become a major issue in distributed high-performance systems, especially in the wake of large Petascale systems and future Exascale ones. With millions of CPU cores running billions of threads, Exascale systems are highly likely to experience a number of faults many times per day. Current approaches to resilience are built for homogeneous Clouds, so to perform automatic or application level checkpoint-restart will not work because of the incompatibilities between the underlying migration mechanisms and security infrastructures that surround them. This reality leaves the application community with a difficult challenge: to find new approaches successfully to run applications until their normal termination despite the essentially unstable nature and heterogeneity of the Cloud resources available regionally and nationally. We intend to addresses these issues by taking a structured, pragmatic approach to innovate and contribute to the community in a number of innovative ways.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 643963 (SWITCH project).

REFERENCES

[1] Tanenbaum, A and Van Steen, M: Distributed Systems, Principles and Paradigms, Prentice-Hall, 2002.

[2] Kalyan, R. and Kumar, A.: "Trends towards Failover Techniques for Cloud Computing Environment",International Journal of Advanced Research in Computer Science and Software Engineering, Volume 5, Issue 1, January 2015.

[3] CRIU team, "Checkpoint/Restore In Userspace", http://criu.org/, visited September 2015.

[4] iRODS, "Integrated Rule-Oriented Data System", http://www.irods.org/, visited September 2015.