The Journal of Systems and Software 118 (2016) 101–114

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Dynamic auto-scaling and scheduling of deadline constrained service workloads on IaaS clouds

Elias De Coninck*, Tim Verbelen, Bert Vankeirsbilck, Steven Bohez, Pieter Simoens, Bart Dhoedt

Department of Information Technology, Ghent University - iMinds, Technologiepark-Zwijnaarde 15, Ghent B-9052, Belgium

A R T I C L E I N F O

Article history:
Received 22 October 2015
Revised 4 May 2016
Accepted 5 May 2016
Available online 6 May 2016

Keywords:
Cloud computing
Deadline constrained workflow scheduling
Dynamic resource allocation

A B S T R A C T

Cloud systems are becoming attractive for many companies. Rather than over-provisioning the privately owned infrastructure for peak demands, some of the work can be overspilled to external infrastructure to meet deadlines. In this paper, we investigate how to dynamically and automatically provision resources on the private and external clouds such that the number of workloads meeting their deadline is maximized. We specifically focus on jobs consisting of multiple interdependent tasks with a priori an unknown structure and even adaptable at runtime. The proposed approach is model-driven: knowledge on the job structure on the one hand; and resource needs and scaling behavior on the other hand. Information is built up based on monitoring information and simulated 'what-if'-scenarios. Using this dynamically constructed job resource model, the resources needed by each job in order to meet its deadline is derived. Different algorithms are evaluated on how the required resources and jobs are scheduled over time on the available infrastructure. The evaluation is carried out using synthetic workloads.

1. Introduction

The availability of reliable public cloud infrastructures has shifted large amounts of computational work from privately owned clusters to these public infrastructures. However, for security concerns and cost reasons, it is often beneficial to consider a hybrid cloud solution, where a privately owned system seamlessly interoperates with one or more external infrastructure services Sotomayor et al. (2009). The normal workload and/or security sensitive jobs are executed on the private infrastructure, while peak loads are offloaded to the public part. The private cloud investment cost is thus limited to the capacity needed to handle average workloads, without having to compromise on flexibility to respond to peak demands.

In this paper we propose the design of a system supporting the dynamic resource allocation in multiple clouds for jobs, with an uncertain adaptable task structure and an unknown execution time, needing completion before a given deadline. We consider workflow-oriented jobs, consisting of multiple tasks ordered in a directed acyclic graph (DAG) modelling the input/output dependency. Our system will schedule jobs, dynamically adapting the number of allocated resources in order to meet the deadlines of all jobs without knowing the DAG structure itself and without any information of the execution time.

The makespan of a single job depends on the number of resources allocated for this job and the number of components running in parallel, which is decided by the framework based on the given deadline. All jobs are scheduled onto a deadline ordered job queue and the job scheduler is responsible for selecting the correct job(s) to execute. The main contribution of this paper is to provide a workflow management system that supports existing scheduling algorithms with learned knowledge on job type, job structure, job makespan and available resources. Simulating jobs for a varying number of resources to accelerate the acquisition of more knowledge, allows our system's schedulers to cope with jobs having a priori an unknown execution time and with changes of the job's internal structure over time.

Our proposed system has been developed on top of OpenStack and AIOLOS Bohez et al. (2014) which allows on-demand provisioning of VMs on multiple cloud providers and distributing OSGi components. For each submitted job, our system follows six steps:

1. Queue the newly submitted job onto a deadline ordered job queue.
2. Look-up the knowledge in the model for the job type, allowing to derive the resource set required to meet the job's deadline.

* Corresponding author.
E-mail addresses: elias.deconinck@ugent.be, elias.deconinck@intec.ugent.be (E. De Coninck), tim.verbelen@ugent.be (T. Verbelen), bert.vankeirsbilck@ugent.be (B. Vankeirsbilck), steven.bohez@ugent.be (S. Bohez), pieter.simoens@ugent.be (P. Simoens), bart.dhoedt@ugent.be (B. Dhoedt).

http://dx.doi.org/10.1016/j.jss.2016.05.011
0164-1212/© 2016 Elsevier Inc. All rights reserved.

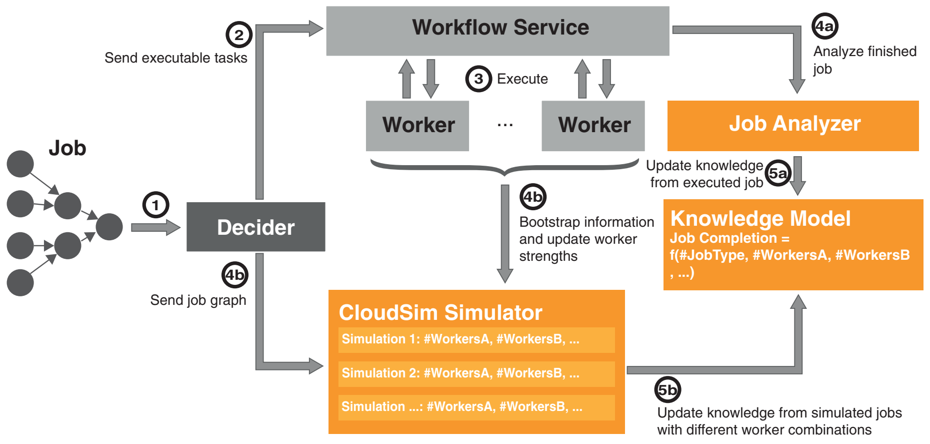
Workflow Service
Send executable tasks
Execute
Worker
Worker
Analyze finished job
Job Analyzer
Job
Decider
Update knowledge from executed job
Bootstrap information and update worker strengths
Knowledge Model
Job Completion = f(#JobType, #WorkersA, #WorkersB , ...)
Send job graph
CloudSim Simulator
Simulation 1: #WorkersA, #WorkersB, ...
Simulation 2: #WorkersA, #WorkersB, ...
Simulation ...: #WorkersA, #WorkersB, ...
Update knowledge from simulated jobs with different worker combinations

Fig. 1. Functional flow to increase knowledge from monitoring information and simulations. To measure infrastructure strength worker performance is monitored and bootstrapped into the simulations.

3. Allocate the required resources from an available private or public IaaS provider with spare capacity.
4. Schedule the tasks of each job on these allocated resources, such that the deadline of the job is met.
5. Simulation of the job execution for a number of varying resources, allowing to increase the knowledge model to improve future job planning.
6. Update knowledge model with monitoring information from the executed and simulated jobs.

After completion of each job, performance metrics are collected on the job completion time and the actual resource strength and usage. These metrics are used to update the knowledge model that the system maintains and will be used to more accurately plan and schedule future jobs of the same job type. Infrastructure performance metrics measure the capacity of VMs, which are used during job simulations to accurately update the knowledge model and to adapt for interference between VMs. Fig. 1 shows the basic flow of the system to build up knowledge.

This paper is structured as follows. Practical use case examples are provided in Section 2 to show the need for this framework. In Section 3, we describe the architecture of our framework showing how the major components of the system interact. Section 4 describes the scheduling algorithms investigated in combination with the resource allocation strategy. Section 5 explains the simulation component in detail, focussing on the additions made to an existing cloud simulation framework, needed to handle the scenarios relevant for this work. In Section 6, we highlight a number of design considerations of the framework and evaluation results are presented in Section 7. Section 8 discusses related work on scheduling workloads and criteria constrained workloads and finally Section 9 concludes this paper.

2. Application case studies

2.1. Manage, fuse and serve geospatial data

The first use case application is designed to manage, fuse and serve geospatial data. The goal is to efficiently store and retrieve geospatial data of a certain area to/from any database, service or file. The geographic information systems application (GIS) consists of three main components:

Server: the data server serves the geospatial data and relevant meta-data from a tile repository. The tile structure and depth are a priori unknown because areas with more detail are represented by more tiles. The data can be accessed by any interested component.

Client: the client accesses the data server to visualize a part of the map.

Engine: the engine is the computational worker to efficiently calculate and store the input data into the tile repository. A tile is the unit of geospatial data for storage and retrieval. It represents a patch of data corresponding to a location on the planet. Tiles exists at different levels of detail so visualization can be optimized by the requested detail level. In Fig. 2 the pyramid representation of detail levels is shown.

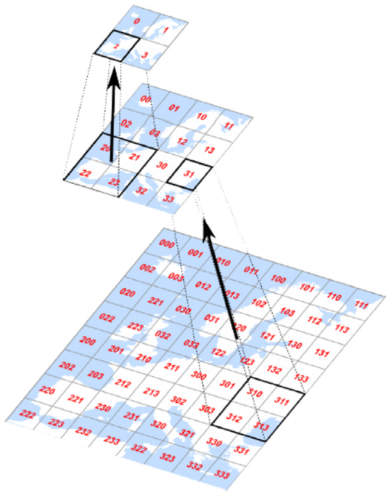

Fig. 2. A tile pyramid of Europe. A tile at a certain detail level spans exactly four tiles of a more detailed level.

To implement this application on top of the framework we need to represent the tile pyramid as a directed graph of tasks (see Fig. 3) and group them into a single job type. To enable the framework's functionality a Decider and one or more Workers are required. The Worker components can be automatically scaled by our framework.

GIS decider: the decider keeps track of the jobs internal structure. Tasks are submitted to the framework in order to fork, render or merge tiles. It decides if for a certain detail level four new tasks have to be spawned or if the tile has to be calculated.

Store: the store component stores tiles on the file system.