European Journal of Operational Research 000 (2016) 1–18

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Adaptive large neighborhood search heuristics for multi-tier service deployment problems in clouds

Anders N. Gullhav[a,*], Jean-François Cordeau[b], Lars Magnus Hvattum[a], Bjørn Nygreen[a]

[a] Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
[b] HEC Montréal, 3000 chemin de la Côte-Saint-Catherine, Montréal, Québec H3T 2A7, Canada

A R T I C L E I N F O

Article history:
Received 15 December 2015
Accepted 1 November 2016
Available online xxx

Keywords:
Metaheuristics
Cloud computing
Replication
Local search

A B S T R A C T

This paper proposes adaptive large neighborhood search (ALNS) heuristics for two service deployment problems in a cloud computing context. The problems under study consider the deployment problem of a provider of software-as-a-service applications, and include decisions related to the replication and placement of the provided services. A novel feature of the proposed algorithms is a local search layer on top of the destroy and repair operators. In addition, we use a mixed integer programming-based repair operator in conjunction with other faster heuristic operators. Because of the different time consumption of the repair operators, we need to account for the time usage in the scoring mechanism of the adaptive operator selection. The computational study investigates the benefits of implementing a local search operator on top of the standard ALNS framework. Moreover, we also compare the proposed algorithms with a branch and price (B&P) approach previously developed for the same problems. The results of our experiments show that the benefits of the local search operators increase with the problem size. We also observe that the ALNS with the local search operators outperforms the B&P on larger problems, but it is also comparable with the B&P on smaller problems with a short run time.

1. Introduction

An increasing proportion of enterprise and business software, such as customer management systems, email systems and time management applications, are run as web services in clouds through the software-as-a-service (SaaS) model. However, Marston, Li, Bandyopadhyay, Zhang, and Ghalsasi (2011) identify the lack of quality of service and availability guarantees as one of the major weaknesses of adopting cloud software services. Even though frameworks and software systems offering fault tolerance management in clouds have been proposed (Cully et al., 2008; Distler, Kapitza, Popov, Reiser, & Schröder-Preikschat, 2011; Jhawar, Piuri, & Santambrogio, 2013), there exist very few optimization models considering fault tolerance by introduction of redundancy (Avižienis, Laprie, Randell, & Landwehr, 2004) in the literature. Distler et al. (2011) present a fault tolerance approach based on active-passive replication, where passive backup replicas are run in a paused state, from which they can be activated rapidly. The passive replicas do not serve demand while being paused, and the replicas consume considerably less resources than corresponding demand-serving active replicas. We take these ideas into account when regarding the service deployment problem of a SaaS provider (SP). In this problem we consider decisions related to the replication of the SaaS services simultaneously with placement decisions. In previous work (Gullhav & Nygreen, 2015), we presented two mathematical models for the problem: one that considers service placement in a hybrid cloud, and another one that only considers placement in the private cloud of the service provider. We refer to Mell and Grance (2011) for definitions of the different types of clouds.

The SP offers a set of SaaS services, modeled as multi-tier services, to its clients. A multi-tier service is a service composed of multiple tiers that collaborate to deliver a service to the clients, and a typical example of a multi-tier service is a three-tier web service composed of a web server, an application server and a database server. Fig. 1 illustrates a three-tier service. When deployed in a cloud environment, each of the tiers, referred to as components of the service run in separate virtual machines (VMs). In turn, the VMs are placed on physical machines, which we refer to as nodes. In our work, we assume that the service provider owns and operates one or more data centers forming a private cloud, and can lease additional VMs in a public cloud when needed. Furthermore, the services of the SP are required

* Corresponding author.
E-mail addresses: anders.gullhav@iot.ntnu.no (A.N. Gullhav), jean-francois.cordeau@hec.ca (J.-F. Cordeau), lars.m.hvattum@himolde.no (L.M. Hvattum), bjorn.nygreen@iot.ntnu.no (B. Nygreen).

http://dx.doi.org/10.1016/j.ejor.2016.11.003
0377-2217/© 2016 Published by Elsevier B.V.

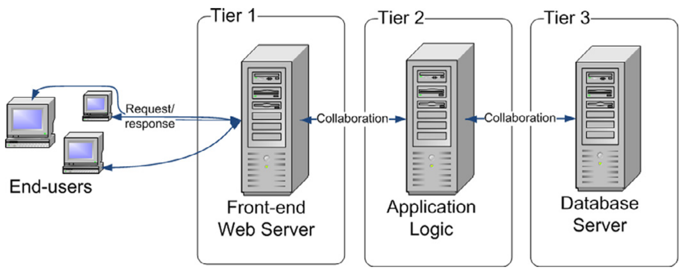

Fig. 1. Illustration of a three-tier web service.

to have a certain level of quality of service (QoS), as specified in service level agreements (SLAs), i.e., contracts between the SP and its clients. The QoS might be specified in terms of bounds on the performance, e.g., the response time, and bounds on the dependability, e.g., the availability or downtime. To obtain a satisfactory performance, each service component can be replicated into a number of load-balanced replicas (in separate VMs) that serve the clients in parallel. However, when one is to provide services run on a failure-prone infrastructure, such as the underlying hardware of clouds, one should take actions to limit the effects of the faults on the services. A key technique to make services fault-tolerant is standby redundancy. The concept of standby redundancy can be explained by the following slightly simplified but illustrative example. Consider a three-tier service with two load-balanced web server replicas, two load-balanced application logic replicas and a single database server. If the database server fails due to a fault, the service would obviously be counted as unavailable by the clients until the VM running the database server is restarted. Instead, if one of the two web server replicas fails due to a fault, there would only be one web server replica serving the clients until the second web server replica is restarted, and the clients would experience a lower performance (e.g., longer response times) in this time window. In both of these cases, running passive backup replicas (also denoted standby replicas) that could be activated faster than the time it takes to restart a replica would reduce the unavailability and improve the performance. In the following, the load-balanced replicas are denoted active replicas while the backup replicas are denoted passive replicas.

The overall objective of the problem is to find the minimum cost deployment while respecting the QoS requirements of the SLA and other technical requirements, such as node resource capacities. In general, there is a non-linear relationship between the numbers of active and passive replicas of each component and the QoS of a service. To maintain a linear (mixed-integer) optimization model, Gullhav and Nygreen (2015) introduced a modeling structure called *replication patterns* that specifies a number of active and passive replicas of each component of a service such that the QoS requirements are satisfied. The replication level decisions are handled by selecting one replication pattern for each service, and hence, the details of the QoS requirements and models are not explicit in the optimization model, but instead handled when specifying replication patterns. By the use of the analytic queuing models of Gullhav, Nygreen, and Heegaard (2013) it is possible to evaluate the QoS of different replication patterns for a given service, and thus, give several replication patterns for each service as input to the model. The reason to specify several replication patterns for each service is that when a service provider offers multiple services and these services are deployed on the same infrastructure, the cheapest way to replicate the components of a service is dependent of how other services are replicated and deployed. This is because a cost-efficient packing of the nodes is dependent on how well the VMs of the components of the different services fit together, and the number of VMs of each component of each service is governed by the replication patterns. Therefore, the increased flexibility that arise by specifying several replication patterns can result in more cost-efficient deployment. Note that all replication patterns for a given service are *minimal*, meaning that if one removes one active or passive replica from any tier, the QoS will no longer be satisfactory.

The literature proposing related placement problems is discussed in our previous paper (Gullhav & Nygreen, 2015) and a recent survey on resource management in clouds is given by Jennings and Stadler (2015). Goudarzi and Pedram (2011) and Ardagna, Panicucci, Trubian, and Zhang (2012) propose resource allocation models for deployment of QoS-constrained multi-tier services. Their models do not concern backup replication and placement of backup replicas as very few models in the service placement literature do. However, Bin et al. (2011) propose a solution method for a placement problem of an infrastructure-as-a-service (IaaS) provider, where some VMs require one or more backup locations to which they can be migrated in case of a failure. Another problem related to ours is the redundancy allocation problem, where the goal is to find the minimum cost allocation of parallel components to different subsystems in series, while maintaining a reliability higher than a given level (Kuo & Wan, 2007).

In Gullhav and Nygreen (2015), we modeled the problem as a direct mixed-integer program (MIP). In addition, the problem was reformulated as a stronger pattern-based model. The latter formulation was solved by an a priori column generation algorithm, also called pre-generation, where a subset of the feasible patterns were given to the master problem in advance of the optimization. Furthermore, in Gullhav and Nygreen (2016), we proposed a branch and price (B&P) algorithm, where patterns were generated dynamically instead of a priori. The B&P algorithm outperformed the pre-generation algorithm.

The contribution of this paper is to introduce two novel adaptive large neighborhood search (ALNS) algorithms for two variants of the service deployment problem. In addition to destroy and repair operators, which are part of the standard ALNS framework, a novel feature of the algorithms is a local search layer on top of the repair operators. A key question we seek to answer in the computational study of this paper is what benefits the local search operators could bring to the ALNS. Another special feature of the proposed algorithms is a MIP-based repair operator, in addition to other heuristic insertion operators. Since the repair operators vary with respect to their time-performance trade-off, we score the operators according to both their performance and time consumption in the adaptive operator selection. Furthermore, the computational study also compares the speed and solution quality of the ALNS algorithms and the previously proposed B&P algorithm for the two versions of the service deployment problem.

The outline of the paper is as follows. In the next section, we present a description of the service deployment problem, and in Section 3, we repeat the direct MIP formulation of Gullhav and Nygreen (2015). In Section 4, we give a description of the components of the proposed ALNS algorithms. The computational study is presented and discussed in Section 5, before Section 6 concludes the paper. Appendix A gives a summary of the main mathematical symbols used in the mathematical formulations and the description of the ALNS, while some additional details from the computational experiments are shown in Appendix B.

2. Problem description

Let $\mathcal{S}$ be the set of multi-tier services, and let $\mathcal{Q}_i$ denote the set of components of service $i \in \mathcal{S}$. For brevity, we will denote component $q \in \mathcal{Q}_i$ of service i as the pair (i, q). The VMs running the service components might run in a public cloud or on the set of nodes, $\mathcal{N}$, in the private cloud of the service provider, and each