Future Generation Computer Systems 29 (2013) 451–459

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

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

Graph partitioning algorithms for optimizing software deployment in mobile cloud computing

Tim Verbelen [*], Tim Stevens, Filip De Turck, Bart Dhoedt

Ghent University – IBBT, Department of Information Technology, Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium

ARTICLE INFO

Article history:
Received 8 September 2010
Received in revised form
19 June 2012
Accepted 14 July 2012
Available online 20 July 2012

Keywords:
Distributed systems
Graph algorithms
Deployment optimization
Cloud computing
Mobile computing

ABSTRACT

As cloud computing is gaining popularity, an important question is how to optimally deploy software applications on the offered infrastructure in the cloud. Especially in the context of mobile computing where software components could be offloaded from the mobile device to the cloud, it is important to optimize the deployment, by minimizing the network usage. Therefore we have designed and evaluated graph partitioning algorithms that allocate software components to machines in the cloud while minimizing the required bandwidth. Contrary to the traditional graph partitioning problem our algorithms are not restricted to balanced partitions and take into account infrastructure heterogenity. To benchmark our algorithms we evaluated their performance and found they produce 10%–40% smaller graph cut sizes than METIS 4.0 for typical mobile computing scenarios.

1. Introduction

Nowadays, the emergence of cloud computing is leading to a new paradigm of utility computing [1], where computing power is offered on an on-demand basis. Users are able to access applications, storage and processing over the Internet, via services offered by cloud providers on a pay-as-you-use scheme. The advantages for the end users are reduced cost, higher scalability and improved performance in comparison to maintaining their own private computer systems, dimensioned for peak load conditions. Moreover the elasticity of the cloud reduces the risks of overprovisioning (underutilization) or underprovisioning (saturation) [2].

The usage of the cloud is not only beneficial for web-based applications, but can also be used for other applications composed of many service components following the service-oriented programming paradigm. Some of these service components may have high needs regarding CPU power or memory consumption, and should therefore be executed on dedicated server machines in the cloud rather than on a regular desktop PC or mobile terminal, for example recognition components in an object recognition or speech to text application.

The adoption of the cloud paradigm poses the problem where to deploy software components, given the many options in terms of available hardware nodes in even moderate scale data centers. This deployment optimization is important to both the cloud user and the cloud provider in order to reduce costs. All components need to be deployed on a machine with sufficient CPU power while the communication overhead between different machines is preferably minimized as this introduces extra latency and network load. This problem can be modelled as a graph partitioning problem where a weighted graph of software components has to be partitioned in a number of parts representing the available machines. Moreover the optimal deployment can change over time, and thus in order to realise an optimal deployment a fast algorithm is desired.

An important scenario in this respect is cloud overspilling. In this scenario, a company offloads work from its own private infrastructure to a public cloud infrastructure on peak moments, as shown in Fig. 1. This enables the company to dimension its infrastructure for the average workload instead of the peak workload, reducing the underutilization and the cost of the private infrastructure. This situation is typical in digital document processing where one faces strict month-end or year-end deadlines, and thousands of batches of documents are to be processed. Typically, document processing systems support workflows consisting of a few tens of components (including content ingest, reformatting, layouting, merging, versioning, logging, output generation and printing components). As many of these components come in different versions, and potentially need to be instantiated for each customer separately, the number of components in such a scenario quickly amounts to a few hundreds.

A second use case is situated in the area of integrated simulation tools for engineering purposes [3]. These integrated tools typically involve multi-physics simulation (e.g. structural analysis, acoustic

* Corresponding author.
E-mail address: tim.verbelen@intec.ugent.be (T. Verbelen).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.07.003

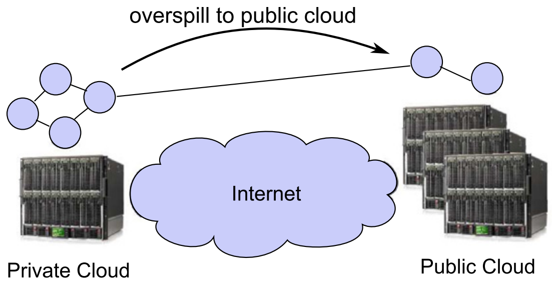

Fig. 1. Work is offloaded from private infrastructure to a public cloud on peak moments, reducing underutilization (and the cost) of the private infrastructure.

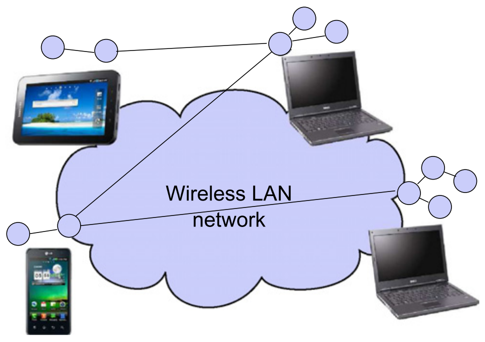

Fig. 2. A cloudlet consisting of four devices connected by a wireless network sharing their resources. Application components are distributed among all devices in the cloudlet, in order to enhance the user experience of all users.

simulation and engine dynamics simulation in the case of designing a new automobile engine), and the number of simulation tools involved can easily amount to 10–20 for small engineering projects to a few 100 individually deployable components for a realistic engineering project. In such engineering endeavours, often parameter sweeps are executed to optimize the design (or to assess the sensitivity of the resulting design performance w.r.t. these parameters), necessitating multiple instances of these simulation components running concurrently, in order to arrive at realistic design times. Again, we end up with a component graph containing a few 100–1000 components.

This overspilling problem also arises in the context of mobile computing, where the cloud can be used to enhance the capabilities of a mobile device. Due to the restricted CPU power of mobile devices, the idea is to offload parts of the application at runtime to a cloud infrastructure [4]. The question then is which parts to offload and to deploy on which machines in the cloud – and how many – in order to spread the load while keeping the needed bandwidth low. In this case, the complexity of the partitioning problem depends on the granularity of the offloading, as offloading can be done on component [4], Java class [5] or method [6] level, resulting in graphs of tens, hundreds or thousands of components.

A special case of deployment optimization occurs when multiple mobile devices connected via a wireless network share their resources in order to enhance the user experience of all users, in a so called "cloudlet" [7], as shown in Fig. 2. This is the case when no internet uplink is available for offloading to the cloud, or when cloud offloading is not beneficial due to a high WAN latency. Because the bandwidth is a scarce resource and shared between all devices in the wireless LAN, a global optimization is needed taking into account all application components of all devices.

As an use case, we mention a mobile augmented reality application. When casting such an application into a component framework, the number of independently deployable components amounts to 5–10 [7], with different components for tracking camera movements, building a 3D map of the environment, recognizing objects, detecting collisions between objects, rendering a 3D overlay, etc. Other applications, such as 3D games, are reported to consist of 10–20 components [8]. Assuming a few tens of users connected to the same cloudlet and hence sharing computing and network resources, the number of components easily exceeds 100. As these users are connected through heterogeneous devices (a mix of low-end and high-end devices), an optimal deployment guaranteeing a minimal quality of experience for all users should be aimed for.

In this paper we present algorithms to partition a software application, composed of a number of components, on a number of interconnected machines in the cloud with different capacities while minimizing the communication cost between the components. In Section 2 related work regarding graph partitioning and task allocation on the grid is discussed. Section 3 more formally describes our problem and in Section 4 algorithms are proposed that solve the problem. In Section 5 the different algorithms are evaluated and compared regarding solution quality and execution time, and the influence of different parameters is discussed. In view of the use cases mentioned in this introduction, we focus on graphs containing 100–1000 components for this evaluation. We also compare our solutions to METIS 4.0 for partitioning graphs in k balanced partitions and show the applicability of our algorithms in the mobile offloading scenario. Finally Section 6 concludes this paper.

2. Related work

2.1. Graph partitioning

Graph partitioning is a fundamental problem in many domains of computer science, such as VLSI design [9], parallel processing [10] and load balancing [11]. The graph partitioning problem tackles the problem of dividing a graph in k equal sets while minimizing the edges between the sets. When $k = 2$ this is also referred to as the min-cut bipartitioning problem. Finding a good solution for this problem is known to be NP-Hard [12]. In the following we give a brief overview of the state-of-the-art and recent advances in graph partitioning. For a more detailed survey of graph partitioning techniques we refer to [13,14].

A first class of algorithms are the so called move-based approaches, which try to iteratively improve the partition by vertex moves or swaps between the parts, such as the Kernighan–Lin (KL) algorithm [15]. By choosing moves that introduce a cost reduction of the graph cut this algorithm converges to a local optimum. Fiduccia and Mattheyses introduced a number of optimizations to the KL algorithm which led to a linear time algorithm for graph partitioning [16]. These move-based algorithms can be combined with stochastic methods such as simulated annealing [17], particle swarm optimization [18] or ant colony optimization [19] in order to escape from local optima. The biggest disadvantage of iterative improvement methods is that their performance deteriorates as the graphs get larger.

In order to partition large graphs the multilevel approach has become widely adopted [20,21,17–19,22]. The main idea is to iteratively coarsen the initial graph by merging vertices according to a matching until a small graph with a similar structure remains. This graph can then be partitioned with a spectral method [23,20] or a greedy graph growing algorithm [24]. Next the graph is again iteratively uncoarsened and a local improvement heuristic such as the KL algorithm is applied at each level. The multilevel scheme is also used in state-of-the-art graph partitioning libraries such as METIS [24], SCOTCH [25] and JOSTLE [26].

Recent work in graph partitioning explores methods based on diffusion [11] or maximum flow [27]. Also the combination of known techniques can result in new heuristics. Chardaire et al. use a PROBE (Population Reinforced Optimization Based Exploration) heuristic, combining greedy algorithms, genetic algorithms