ARTICLE IN PRESS

Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

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

Scheduling highly available applications on cloud environments

Marc Eduard Frîncu *

Research Institute e-Austria, Timisoara, Romania
West University of Timisoara, Timisoara, Romania

ARTICLE INFO

Article history:
Received 30 October 2011
Received in revised form
3 April 2012
Accepted 17 May 2012
Available online xxxx

Keywords:
Cloud scheduling
Multi-objective scheduling
Meta-heuristics
Nonlinear-programming
High available systems

ABSTRACT

Cloud computing is becoming a popular solution for storing data and executing applications due to its on-demand pay-per-use policy that allows access to virtually unlimited resources. In this frame applications such as those oriented towards Web 2.0 begin to be migrated on cloud systems. Web 2.0 applications are usually composed of several components that run indefinitely and need to be available to end users throughout their execution life cycle. Their availability strongly depends on the number of resource failures and on the variation in user hit rate. These problems are usually solved through scaling. A scaled application can span its components on several nodes. Hence if one or more nodes fail it could become unavailable. Therefore we require a method of ensuring the application's functionality despite the number of node failures. In this paper we propose to build highly available applications, i.e., systems with low downtimes, by taking advantage of the component based architecture and of the application scaling property. We present a solution to finding the optimal number of component types needed on nodes so that every type is present on every allocated node. Furthermore nodes cannot exceed a maximum threshold and the total running cost of the applications needs to be minimized. A sub-optimal solution is also given. Both solutions rely on genetic algorithms to achieve their goals. The efficiency of the sub-optimal algorithm is studied with respect to its success rate, i.e., probability of the schedule to provide highly available applications in case all but one node fail. Tests performed on the sub-optimal algorithm in terms of node load, closeness to the optimal solution and success rate prove the algorithm's efficiency.

1. Introduction

Cloud computing has become a popular solution for storing data and executing applications for many companies due to its on-demand pay-per-use policy which allows access to virtually unlimited resources. Public cloud providers such as Amazon, Google, Microsoft or Rackspace offer access to their data centers by means of infrastructure (e.g., Amazon EC2 [1], Rackspace cloud [2]) or platform level (e.g., Google App Engine [3], Microsoft Windows Azure [4]) APIs. There even exist solutions for companies that wish to create their own cloud systems (e.g., Eucalyptus [5], OpenStack [6]). For companies as well as for private users choosing to switch to a cloud based solution does not come without risks such as network crashes and splits, data center failures or even security breaches. A key aspect when migrating to cloud computing is to ensure application *availability* in case of failures.

For Web 2.0 applications it is essential to minimize the impact of failures throughout the application's life cycle and to constantly monitor and adapt resources to user request rates and failures through *automatic scaling* (cf. Section 4).

Web 2.0 applications usually consist of several interlinked components (e.g., web servers, databases, application logic modules or cloudlets [7], and communication servers) which need to scale according to user request rates. Due to this modular approach a further requirement, especially in case of failures, is to ensure that all required components execute on the remaining resource nodes. This guarantees that despite a possible considerable drop in the applications's performance, caused by under provisioning or node failures, the application is still able to handle a certain percent of the user requests. In this way the application's ability to handle large incoming requests is diminished – but not halted – until the components are scaled back to fit the size of the request rate.

In case of performance drops the average response time (milliseconds to seconds) increases. This causes some users to experience long response times and even timeouts. For profit oriented websites this translates into less profit being made, due to low page clicks or commercial activities.

We mentioned earlier the notion of **availability**. In web based applications this is defined as the ratio between the number of serviced requests and the total number of requests [8]. It is also generally expressed in terms of "number of nines": the more "nines" the less denial of service messages in minutes per year [9]. According to Gray et al. [9] **High Availability** (HA) means an availability of 99.999%, i.e., 5 min of unavailability per year. High-availability systems are characterized by fewer failures and faster repair times.

* Correspondence to: West University of Timisoara, Blvd Vasile Parvan, No. 4, Room 045B, 300223, Timisoara, Romania.
E-mail address: mfrincu@info.uvt.ro.

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.05.017

Two major issues arise when dealing with scalable and component based applications that need to be HA. First, as nodes cost it is undesirable for long running applications to rent extra resources as in the case of many HA cluster systems that require $N + M$ nodes to operate (cf. Section 2). Second, as applications are modular we could end up isolating particular components on several nodes. In case the hosting nodes fail our application would experience a drop in its availability while it restarts the failed resources.

In this paper we propose a solution for these two problems by introducing a method of placing each application component type on every needed node. Hence we (1) avoid allocating unnecessary extra nodes and (2) ensure with a certain degree of probability that in case only one node remains the application would still execute – although at a slower rate – as though all the required component types were available on it.

The proposed **scheduling algorithms** consider the method of scaling to be known *a priori* and focus on searching for an optimal allocation of components on nodes in order to ensure a homogeneous spread of component types on every node. Numerous current scheduling strategies (cf. Section 2) assume short-to-medium lived tasks (e.g., bag of tasks, MapReduce or parameter sweep applications) and ignore the Web 2.0 applications which are inherently long (or infinitely) running. Despite the apparent simplicity of such schedules it is often the case that we need to periodically migrate components in order to optimize node load, minimize communication costs between nodes and ensure HA.

The rest of the paper is structured as follows: Section 3 describes the application model by introducing the mathematical formalism used throughout the rest of this work. Section 4 presents two scheduling algorithms for achieving HA in the context of this paper. We first address the problem of component and node scaling (cf. Sections 4.1 and 4.2) as they are essential aspects in the proposed algorithms. Section 4.1.1 deals with the special case in which the components' loads are known and gives an optimal solution to the number of components each node can accommodate. Section 4.2 presents a pro-reactive algorithm for node scaling based on the user hit rate. Section 4.3 then presents the two algorithms used to schedule components on nodes and to reactively allocate nodes if no suitable node exists—i.e., the pro-active algorithm failed to allot the necessary nodes. The optimal solution can be used in case the loads of every component are known and is depicted in Section 4.3.1, while Section 4.3.2 presents the algorithm for the general case. The model used to measure the success rate of the proposed algorithms is addressed in Section 4.3.3. Section 5 describes the testing scenarios and includes discussions on the provided success rate, node load and closeness to the optimal solution. The paper concludes by reviewing some of the main achievements of this study (cf. Section 6).

2. Related work on high available systems and cloud scheduling

The issue of achieving HA systems has gathered a lot of attention with work ranging from cluster and grid to utility computing. The overall problem can be reduced to the problem of placing virtual machines on a limited physical node so that the number of physical resources is minimized. This is also known as the *bin packing problem* [10] which is known to be *NP-hard*.

A popular technique towards HA systems is to use virtual resources that are migrated in case of failures. Several commercial solutions including VMware [11] and Xen [12] use it but take opposite approaches: VMware is a reactive solution which restarts virtual machines on other resources in case of errors. This leads to temporal delays in the application uptime. Xen on the other hand uses a proactive method based on monitoring data and migrates virtual machines before the predicted failure occurs. However this approach has also a major drawback as failures are difficult to predict in advance.

For Internet based services solutions include deploying load balancers which in case one or more servers fail redirect the traffic to the remaining ones [13–16].

Loveland et al. [17] study how virtualization techniques can augment HA and propose a simple redundant method for placing virtual machines on multiple nodes.

Besides migration there are also solutions based on clustering. Database servers such as MySQL [18] and key value stores like Amazon's Dynamo [19] rely on it. In this approach processes are distributed to replication servers by using DNS Round Robin [18] or key consistent hashing [19].

One interesting aspect concerning the migration based methods depicted earlier is that they separate the HA from the allocation problem. This raises an important question regarding their overall efficiency when virtual machines contain dependent applications.

Some work which considers HA as part of the allocation algorithm has been done. Recently Machida et al. [20] proposed an optimal solution for achieving HA systems by using redundant nodes. The authors also show that for some cases their algorithm uses less than the $N + M$ nodes needed when classic and simple approaches such as First-Fit Decrease [21] are used.

Gottumukkala et al. [22,23] propose a reliability-aware allocation algorithm for achieving HA in large scale computing systems. In their work they study the possibility of using only reliable nodes when allocating virtual machines and show their solution to offer a greater improvement in terms of waste time and completion time than the classic Round Robin approach.

All of the previous examples have relied on software to achieve HA systems, yet hardware solutions exist as well. For instance, in their work, Aggarwal et al. [24] propose a low-level isolation for fault containment and reconfiguration for chip multiprocessors. Their method partitions the chip into multiple failure zones which can be exploited by redirecting power from failed chip multiprocessor components to remaining ones.

Our proposed scheduling approach is different from these strategies in the sense that we address a specific problem – that of providing HA long running applications – and also because we operate on top of virtual machines and schedule *co-located* application level component processes. In contrast other approaches – from industry, e.g., Google App Engine [3], IBM Cloudburst [25]; or from academia, e.g., [20,22] – schedule virtual machines by asserting that one component instance runs isolated on its own virtual machine.

Another difference in our approach is that we take advantage of the inherit scalability property of Web 2.0 based applications and of the component based (workflow-like) structure of these applications. Cloud systems rent resources on an hourly basis and for long running applications this induces regular costs. Therefore applying traditional solutions which allocate redundant resources only increases the overall cost. Due to the fact that scalable applications usually require several components of the same type to coexist in order to cover the requests a solution would be to spread the component types homogeneously on every needed node. In this way we ensure that in case all nodes except one fail we still have all the application components running. So we do not only achieve HA without needing extra nodes but also minimize the actual number of used nodes.

Virtualization is a key aspect in cloud computing as it allows us to harvest the full potential of the "unlimited" resource pool of these systems. Since we have already mentioned attempts to integrate HA with allocation (scheduling) algorithms have been made. However most scheduling solutions used in the industry still use simple scheduling heuristics and either offer HA as an

Please cite this article in press as: M.E. Frîncu, Scheduling highly available applications on cloud environments, Future Generation Computer Systems (2012), doi:10.1016/j.future.2012.05.017