Journal of Computational Science xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Journal of Computational Science

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

A Sandpile cellular automata-based scheduler and load balancer

Jakub Gąsior*, Franciszek Seredyński

Department of Mathematics and Natural Sciences, Cardinal Stefan Wyszyński University, Warsaw, Poland

ARTICLE INFO

Article history:
Received 4 April 2016
Received in revised form 13 July 2016
Accepted 9 August 2016
Available online xxx

Keywords:
Cellular automata
Distributed systems
Self-organization
Scheduling and load-balancing

ABSTRACT

We present in this paper a novel load balancing and rescheduling approach based on the concept of the *Sandpile* cellular automaton: a decentralized multi-agent system working in a critical state at the edge of chaos. Our goal is providing fairness between concurrent job submissions in highly parallel and distributed environments such as currently built cloud computing systems by minimizing slowdown of individual applications and dynamically rescheduling them to the best suited resources. The algorithm design is experimentally validated by a number of numerical experiments showing the effectiveness and scalability of the scheme in the presence of a large number of jobs and resources and its ability to react to dynamic changes in real time.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Cloud computing (CC) is one of the emerging developments in distributed, service-oriented, trusted computing. It offers the potential for sharing and aggregation of different resources such as computers, storage systems, data centers and distributed servers. The goal of the cloud-based architecture is to provide a form of elasticity; the ability to expand and contract capacity on-demand. That means there needs to be some mechanism in place to balance requests between two or more instances of client's applications.

We propose a decentralized and self-organizing multi-agent system based on the *Sandpile* cellular automata (CA) model. Given a set of agents, which have only partial knowledge of local resources and submitted workload, it is possible to design a set of simple local rules, so that a smart behavior emerges from them at a global system level. Its role is to: (a) balance the workload within each local neighborhood by identifying source and destination node pairs and determining the amount of workload to be transferred between them, and (b) balance the workload across the entire system.

By localizing the balancing domain to a single neighborhood we are able to reduce the overhead of the balancing process, as well as ensure a balanced workload for the entire platform. This is accomplished by introducing overlapping domains, whereby excess workload can diffuse from more heavily loaded neighborhoods into lightly loaded ones. Potentially, more accurate migration strategies are made possible by larger neighborhoods. However, larger balancing domains may increase the aging period of information and cause the load balancing overhead to be more unevenly distributed.

Because of the resource heterogeneity and communication overheads existing in CC systems, we take into account features such as processing power of individual CC nodes and communication latency between them. The main goal of our algorithm is to reduce the average response time of arriving jobs by equalizing the *Slowdown* between neighboring CC nodes. The performance of our scheme is evaluated in terms of several performance metrics in relation to multiple variations of arrival and processing times as well as the number of submitted jobs.

The remainder of this paper is organized as follows. In Section 2, we present the works related to the distributed scheduling and load balancing in the grid and cloud computing systems. In Section 3, we describe the proposed cloud system model. Section 4 presents the proposed solution of dynamic load balancing and scheduling scheme based on the *Sandpile* CA model. The experimental evaluation of the proposed approach is given in Section 5. We end the paper in Section 6 with some conclusions and indications for future work.

2. State of the art

Distributed scheduling has been widely studied in the context of real-time systems, when jobs have deadline constraints. Among others, in [1] authors proposed a distributed algorithm to solve general constraint optimization problem with a guaranteed

* Corresponding author.
E-mail address: j.gasior@uksw.edu.pl (J. Gąsior).

http://dx.doi.org/10.1016/j.jocs.2016.08.005
1877-7503/© 2016 Elsevier B.V. All rights reserved.

convergence using only localized, asynchronous communication between agents involved in this process. However, in large-scale systems, one cannot hope to reach such an optimality *on-the-fly* and would rather employ heuristics inspired by emergent peer-to-peer techniques [2]. To cope with node volatility, several mechanisms have been proposed, such as checkpointing and migration [3,4]. These mechanisms allow to efficiently manage computing resources that are likely to fail at any given time. Therefore, the application of robust distributed architectures and adaptation of peer-to-peer systems are becoming some of the most widespread methods in that area [2,5,6].

Due to the distributed nature of such systems, several concurrent jobs originating from different users are likely to compete for the resources. Traditionally, schedulers aim at minimizing the overall completion time of a job [6]. However, in a multi-user setting, it is important to maintain some fairness between users: we do not want to favor a user with a large number of small jobs compared to another with fewer larger jobs. Similarly, if jobs can be submitted at different entry points of the distributed system, we do not want that the location of the user to impact his experienced running time [2].

There are several studies complementary to our work, which focus on how to share the available resources among several users. For example, in [7] authors employed a scheduling policy utilizing the solution to a linear programming problem in order to maximize the system capacity. Closer to our problem, Viswanathan in [8] proposed a distributed scheduling strategy specifically designed to handle large volumes of computationally intensive and arbitrarily divisible workloads submitted for processing involving multiple sources and processing nodes. In [9] authors proposed a distributed scheduling solution ensuring a fair and efficient use of the available resources by providing a similar share of the platform to every application through stretch optimization.

In [10] authors proposed a skeleton for dynamic load balancing through *gossiping*: rather than a fully-operative scheduling system, the authors aim at illustrating the application potentials of gossiping protocols. They demonstrated the utility of applying an averaging component along with an overlay component to obtain high scheduling performance with respect to the amount of transferred workload that is often indistinguishable from the optimal case. Similarly, in [11] authors described a distributed load balancing algorithm based on building a consensus between nodes. To reach the consensus nodes communicate within a homogeneous architecture via gossiping protocol.

Finally, in [12] authors employed a *Sandpile* model for non-clairvoyant load-balancing of jobs in large-scale decentralized systems. This approach has a strong connection with the one presented here: it works with two different interconnection topologies, based on a ring and a small-world graph and aims to minimize the sizes and quantities of the avalanches by using a gossiping-based version of the multi-agent system. Instead of propagating a real avalanche, the gossiping protocol forwards the avalanche virtually until a new state of equilibrium is found. The proposed solution was found to reduce the overhead of intermediate migrations and increase the overall throughput of the system, however it employed a simple First In, First Out (FIFO) local allocation scheme and did not consider the impact of resulting rescheduling events on the overall performance of the considered platform.

3. Cloud model

3.1. System & user model

Our system model is based on the architecture introduced in [13] and consists of a set of geographically distributed cloud nodes M_1, $M_2, \ldots, M_m$, which are connected to each other via a wide area network. Each node M_i is described by a parameter m_i, which denotes the number of identical processors P_i and its computational power s_i, characterized by a number of operations per unit of time it is capable of performing. Fig. 1(a) depicts an exemplary set of parallel machines in the CC system.

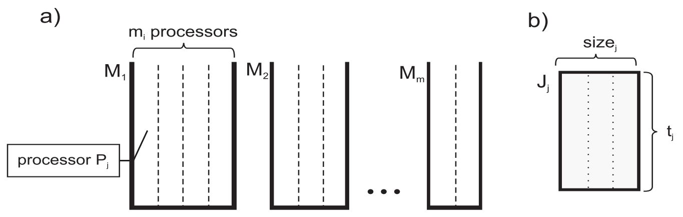

Fig. 1. Example of the CC system. A set of parallel machines (a) and the multi-threaded job model (b).

Individual users $(U_1, U_2, \ldots, U_n)$ submit their jobs to the system, expecting their completion before a required deadline. Job (denoted as J_k^j) is jth job produced (and owned) by user U_k. J_k stands for the set of all jobs produced by user U_k, while $n_k = |J_k|$ is the number of such jobs. Each job has varied parameters defined as a tuple $< r_k^j, size_k^j, t_k^j, d_k^j >$, specifying its release date r_k^j; its size $1 \le size_k^j \le m_m$, that is referred to as its processor requirements or *degree of parallelism*; its execution requirements t_k^j defined by a number of operations and deadline d_k^j.

Rigid jobs require a fixed number of processors for parallel execution: this number is not changed until the job is completed. We assume that job J_k^j can only run on machine M_i if $size_k^j \le m_i$ holds, that is, we do not allow multi-site execution and co-allocation of processors from different machines. We assume a space sharing scheduling approach, therefore a parallel job J_k^j is executed on exactly $size_k^j$ disjoint processors without preemptions, while $p_k^{i,j} = t_k^j / s_i$ defines job's J_k^j execution time on machine M_i. Fig. 1(b) shows an example of the multi-threaded job model.

3.2. Problem formulation

In this section, we formally define the problem we target. Our goal is to design a dynamic and fully decentralized scheduling architecture oriented to on-line distributed platforms. We are interested in a proactive scheduling scheme, meaning that it should not interfere with the actual assignment of jobs unless the workload is detected to be in a non-equilibrium state [12]. We are interested in modeling a distributed scheduling scheme on such a complex CC system, i.e., allocating a limited quantity of independently managed resources to a specific number of independent user jobs (Virtual Machines (VMs) or applications/jobs to be run, etc.) in a limited operation time, without the need of any centralized coordination and control facility.

The goal of the scheduler is to find the allocation and the time of execution for each job. The distribution of the jobs must be done in such a way that the system's throughput is optimized. To do so, an optimal trade-off between the processing overhead and the degree of knowledge used in the balancing process must be sought. All scheduling and load balancing decisions must be taken locally by the agents assigned to local cloud resources. Formally, the objective is to allocate a batch of local jobs to the available cloud nodes M_i and minimize the global system *Slowdown* ς_{max} thereby enforcing a fair trade-off between all submitted jobs. We consider minimization of the time $\varsigma^i_{\max}$ on each cloud node M_i over the system in such a way

Download English Version:

https://daneshyari.com/en/article/4951014

Download Persian Version:

https://daneshyari.com/article/4951014

Daneshyari.com