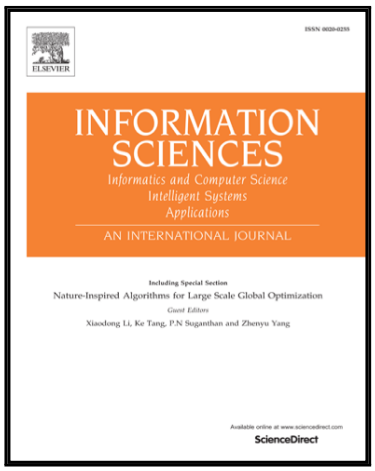

Self managed virtual machine scheduling in Cloud systems

Stelios Sotiriadis, Nik Bessis, Rajkumar Buyya

PII: S0020-0255(17)30827-7
DOI: 10.1016/j.ins.2017.07.006
Reference: INS 12972

To appear in: *Information Sciences*

Received date: 29 March 2016
Revised date: 14 June 2017
Accepted date: 6 July 2017

Please cite this article as: Stelios Sotiriadis, Nik Bessis, Rajkumar Buyya, Self managed virtual machine scheduling in Cloud systems, *Information Sciences* (2017), doi: 10.1016/j.ins.2017.07.006

Self managed virtual machine scheduling in Cloud systems

Stelios Sotiriadis

The Edward Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Bahen Centre for Information Technology St. George Campus, 40, Toronto, ON M5S 2E4, Canada

e-mail: s.sotiriadis@utoronto.ca

Nik Bessis

Department of Computer Science, Edge Hill University, St Helens Rd, Ormskirk, Lancashire L39 4QP, UK

e-mail: Nik.Bessis@edgehill.ac.uk

Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Parkville VIC 3010, Australia

e-mail: rbuyya@unimelb.edu.au

Abstract

In Cloud systems, Virtual Machines (VMs) are scheduled to hosts according to their instant resource usage (e.g. to hosts with most available RAM) without considering their overall and long-term utilization. Also, in many cases, the scheduling and placement processes are computational expensive and affect performance of deployed VMs. In this work, we present a Cloud VM scheduling algorithm that takes into account already running VM resource usage over time by analyzing past VM utilization levels in order to schedule VMs by optimizing performance. We observe that Cloud management processes, like VM placement, affect already deployed systems (for example this could involve throughput drop in a database cluster), so we aim to minimize such performance degradation. Moreover, overloaded VMs tend to steal resources (e.g. CPU) from neighbouring VMs, so our work maximizes VMs real CPU utilization. Based on these, we provide an experimental analysis to compare our solution with traditional schedulers used in OpenStack by exploring the behaviour of different NoSQL (MongoDB, Apache Cassan-