Future Generation Computer Systems 29 (2013) 1885–1900

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Hopfield neural network for simultaneous job scheduling and data replication in grids

Javid Taheri [a,*], Albert Y. Zomaya [a], Pascal Bouvry [b], Samee U. Khan [c]

[a] Centre for Distributed and High Performance Computing, School of Information Technologies, The University of Sydney, Sydney, NSW 2006, Australia
[b] Department of Electrical and Computer Engineering, University of Luxembourg, Luxembourg
[c] Department of Electrical Engineering, North Dakota State University, Fargo, USA

HIGHLIGHTS

- Simultaneous job and data allocation in grid environments.
- Calculate near optimal solution for all sorts of grid.
- Designed for real worlds jobs instead of the traditional simplistic view of jobs.
- Significant outperformance in comparison with current algorithms.
- Fast convergence speed; usually less than a minute for a medium-sized grid.

ARTICLE INFO

Article history:
Received 12 October 2011
Received in revised form
22 April 2013
Accepted 22 April 2013
Available online 9 May 2013

Keywords:
Job scheduling
Network aware scheduling
Data file migration policies
Grid environments

ABSTRACT

This paper presents a novel heuristic approach, named JDS-HNN, to simultaneously schedule jobs and replicate data files to different entities of a grid system so that the overall makespan of executing all jobs as well as the overall delivery time of all data files to their dependent jobs is concurrently minimized. JDS-HNN is inspired by a natural distribution of a variety of stones among different jars and utilizes a Hopfield Neural Network in one of its optimization stages to achieve its goals. The performance of JDS-HNN has been measured by using several benchmarks varying from medium- to very-large-sized systems. JDS-HNN's results are compared against the performance of other algorithms to show its superiority under different working conditions. These results also provide invaluable insights into scheduling and replicating dependent jobs and data files as well as their performance related issues for various grid environments.

1. Introduction

Grid computing has matured into an essential technology that enables the effective exploitation of diverse distributed computing resources to deal with large-scale and resource-intensive applications, such as those found in science and engineering. A grid usually consists of a large number of heterogeneous resources spanning across multiple administrative domains. The effective coordination of these heterogeneous resources plays a vital key role in achieving performance objectives. Grids can be broadly classified into two main categories: computational and data, based on their application focus. In recent years, the distinction between these two classes of grids is much blurred, mainly due to the ever increasing data processing demand in many scientific, engineering, and business applications, such as drug discovery, economic forecasting, seismic analysis, back-office data processing in support of e-commerce, Web services, etc. [1].

In a typical scientific environment such as in High-Energy Physics (HEP), hundreds of end-users may individually or collectively submit thousands of jobs to access peta-bytes of distributed HEP data. Given the large number of tasks resulting from splitting these bulk submitted jobs and the amount of data being used by them, their optimal scheduling along with allocating their demanding data files becomes a serious problem for grids—where jobs compete for scarce compute and storage resources among available nodes. The Compact Muon Solenoid (CMS) [2] and the Large Hadron Collider (LHC) [3] are two well known case studies for such applications and are used as a motivation to design many systems including the algorithm in this article. Both systems constantly submit thousands of parallel jobs to access many shared data files. In such systems, each job is an acyclic data flow of hundreds of tasks in which CMS/LHR executable modules must

* Corresponding author. Tel.: +61 290369718.
E-mail addresses: javid.taheri@sydney.edu.au (J. Taheri), albert.zomaya@sydney.edu.au (A.Y. Zomaya), pascal.bouvry@uni.lu (P. Bouvry), samee.khan@ndsu.edu (S.U. Khan).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.04.020

Table 1
Typical job characteristics in CMS [5].

Number of simultaneously active users	100–1000
Number of jobs submitted per day	250–10,000
Number of jobs being processed in parallel	50–1000
Turnaround time for jobs	0.2 s–5 months
Number of datasets that serve as input to a sub job	0–50
Average number of datasets accessed by a job	250–10,000 K
Average size of the dataset accessed by a job	30 GB–3 TB

run them in parallel [4]. Table 1 shows a typical number of jobs from users and their computation and data related requirements for CMS jobs [5].

Grid schedulers are mainly divided into two types: (1) job-oriented and (2) data-oriented systems. In job-oriented systems, data files are fixed in location and jobs are scheduled, usually adhering to some objective such as power consumption [6,7]. In this case, the goal is to schedule jobs among Computational Nodes (CNs) to minimize the overall makespan of executing all jobs in the whole system; here, it also is assumed that the overall transfer time of all data files is relatively negligible compared to executing jobs. The speed and number of available computer resources in different CNs and the network capacity between CNs and Storage Nodes (SNs) are typical considerations taken into account in such systems. For data-oriented systems, on the other hand, jobs are fixed in location and data files are moved and/or replicated in the system so that their accessibility by their relevant jobs is increased. In contrast to the previous mode, here it is assumed that transfer time of all data files is much more time consuming than executing their dependent jobs. As a result, jobs will need less time to download the associated data files to execute and therefore, the execution time (i.e., makespan of executing jobs plus transfer time of data files) of the system is reduced. The available storage in SNs and the capacity of interconnected network links between CNs and SNs are typical considerations in such allocations. From a practical point of view, neither of these two system types is adequate to deal with cases in which both computational jobs and data files are equally influential factors for efficient system utilization. Therefore, inappropriate distribution of resources, large queues, reduced performance, and throughput degradation for the remainder of the jobs are some of the drawbacks of assuming systems fit into just one of these two types.

There are three main phases of scheduling in such complex systems [8]: (1) resource discovery, (2) matchmaking, and (3) job execution. In the first phase, resource discovery, grid schedulers conduct a global search to generate a list of all available resources as well as their limitations and history profiles in a system. In the second phase, matchmaking, schedulers try to determine best choices for executing jobs and replicating data files. Capacities of CNs/SNs as well as quality of the network connecting them are among the basic characteristics that need to be considered by schedulers to perform this phase. In the last phase, job execution, schedulers produce commands for CNs and SNs to execute jobs and replicate data files, respectively. Here, schedulers do not interfere with details of such commands and leave CNs/SNs to perform their allocated commands, including – but not limited to – data file staging or system cleanups.

In this work, the matchmaking process of schedulers was targeted and our contribution is a holistic scheduling approach to concurrently minimize two very important performance factors of a grid system, i.e., (1) makespan for executing all jobs, and (2) transfer time of all data files. Our approach schedules jobs and replicates data files with respect to (1) characteristics of CNs/SNs in a system, (2) inter-dependences between jobs and data files, and (3) network bandwidth among CNs/SNs to host these jobs and data files.

The rest of this paper is organized as follows. Section 2 presents the related works to solve the stated problem. Section 3 presents the assumed framework for this article. Section 4 presents the problem statement. Section 5 presents preliminaries of our proposed algorithm, i.e., (i) the original algorithm to distribute a variety of stones into different jars, and (ii) fundamentals of a Hopfield Neural Network (HNN) optimizer. Section 6 presents JDS-HNN, followed by its results in Section 7. Discussion and analysis of results and conclusions are presented in Sections 8 and 9, respectively.

2. Related works

Several approaches have already been proposed to solve the bi-objective scheduling problem that is the focus of this work. Most of these methods make certain assumptions about the nature of jobs and data files to present a specific real system. Their solutions can be roughly categorized into two classes: online and batch [9]. In the online methods, it is assumed that jobs arrive one-by-one, usually following a predetermined distribution, and grid schedulers must immediately dispatch these jobs upon receiving them. In the batch methods (also known as batch-of-jobs or bulk) jobs are assumed to be submitted in a bulk and thus, grid schedulers need to allocate several jobs at the same time. Although the online mode can be a fair representation of small grids, CMS and LHR as well as many other massive systems always process their jobs in the batch mode. To date, most approaches usually use only one mode (online or batch) and only few exist that use both.

The European Data Grid (EDG) project was among the first that has created a resource broker for its workload management system based on an extended version of Condor [10]. The problem of bulk scheduling also has been addressed through shared sandboxes in the most recent versions of gLite from the EGEE project [11]. Nevertheless, these approaches only consider one of the priority and/or policy controls rather than addressing the complete suite of co-allocation and co-scheduling issues for bulk jobs. In another approach for data intensive applications, data transfer time was considered in the process of scheduling jobs [12]. This deadline based scheduling approach however could not be extended to cover bulk scheduling. In the Stork project [13], data placement activities in grids were considered as important as computational jobs; therefore, data-intensive jobs were automatically queued, scheduled, monitored, managed, and even check-pointed in this system/approach. Condor and Stork also were combined to handle both job and data file scheduling to cover a number of scheduling scenarios/policies. This approach also lacks the ability to cover bulk scheduling. In another approach [14], jobs and data files are linked together by binding CNs and SNs into I/O communities. These communities then participate in the wide-area system where the Class Ad framework is used to express relationships among the stakeholders. This approach however does not consider policy issues in its optimization procedure. Therefore, although it covers co-allocation and co-scheduling, it cannot deal with bulk scheduling and its related management issues such as reservation, priority and policy. The approach presented in [15] defines an execution framework to link CPUs and data resources in grids for executing applications that require access to specific datasets. Similar to Stork, bulk scheduling is also left uncovered in this approach.

In more complete works such as the Maui Cluster Scheduler in [16], all jobs are queued and scheduled based on their priorities. In this approach, which is only applicable for local environments (i.e., for clusters rather than grids), weights are assigned based on various objectives to manipulate priorities in scheduling decisions. The data aware approach of the MyGrid [17] project schedules jobs close to the data files they require. However, this traditional approach is not always very cost effective as the