Journal of Computer and System Sciences 81 (2015) 943–957

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Priority-grouping method for parallel multi-scheduling in Grid

Goodhead Tomvie Abraham *, Anne James, Norlaily Yaacob

Distributed Systems and Modelling Group, Coventry University, Priory Street, Coventry, CV1 5FB, United Kingdom

ARTICLE INFO

Article history:
Received 3 July 2014
Received in revised form 20 September 2014
Accepted 1 October 2014
Available online 30 December 2014

Keywords:
Grid computing
Scheduling
Parallel programming
Multicore-systems
Multi-scheduling

ABSTRACT

This article presents a method of enhancing the efficiency of Grid scheduling algorithms by employing a job grouping method based on priorities and also grouping of Grid machines based on their configuration before implementing a suitable scheduling algorithm within paired groups. The Priority method is employed to group jobs into four groups, while two different methods, SimilarTogether and EvenlyDistributed, are employed to group machines into four groups before implementing the MinMin Grid scheduling algorithm simultaneously. Implementing the scheduling algorithms simultaneously within paired groups (multi-scheduling) ensures a high degree of parallelism, increases throughput and improves the overall performance of scheduling algorithms. Two sets of controlled experiments were carried out on an HPC system. Analysis of results shows that the Priority Grouping method improved the scheduling efficiency by very large margins over the non-grouping method.

1. Introduction

As the multicore technology becomes ever more pervasive in our daily lives and as Grid computing continues to grow, it becomes ever more appealing for a method that exploits one of the technologies for the benefit of the other. In an age characterised by multicore systems, scheduling of Grid jobs without considering the advantages of parallelism achievable from the evolving multicore hardware does not augur well for the current trend in computing hardware. Neglecting the underlying hardware in scheduling Grid jobs will hamper the growth of the Grid as the technology continues to evolve because sequential scheduling constitutes a bottleneck when the number of jobs increases. Current Grid scheduling algorithms fall short of exploiting the advantages of the underlying multicore systems, hence are inadequate to address the future of the Grid. To achieve any commensurate gain in Grid scheduling in tandem with advances in hardware, Grid scheduling algorithms need to be designed to exploit the available multicores. A parallel multi-scheduling method that considers the underlying multicore hardware of the scheduler and Grid machines will not only ensure the sustenance of the projected growth of the Grid but will also reverse the neglect by most Grid scheduling algorithms on the need for a paradigm shift towards the exploitation of the advantages of the multicore system in Grid scheduling.

This work is intended for implementation on multicores. It uses a priority-based grouping method to split jobs into four priority groups and splits machines into same number of groups before implementing the MinMin scheduling algorithm between paired job group and machine group in parallel. A priority group in this context means a group containing a set of jobs split on some priority attribute. A machine group refers to a list containing machines (Grid resources) in the same category based on our grouping method. Two methods, SimilarTogether and EvenlyDistributed, are used alternatively to split

* Corresponding author.
E-mail addresses: abrahamg@uni.coventry.ac.uk (G.T. Abraham), a.james@coventry.ac.uk (A. James), n.yacoob@coventry.ac.uk (N. Yaacob).

http://dx.doi.org/10.1016/j.jcss.2014.12.009
0022-0000/

machines into groups. The factors that determine whether a machine has been sorted into a particular group are either the similarity in configuration or the non-similarity in configuration. Scheduling of jobs is carried out within the groups, hence we can have several instances of threads executing scheduling algorithms independently. Multi-scheduling implies a method of scheduling in parallel from within the matched groups of jobs and machine groups. Parallelism enables the optimised use of available processing components within a system.

This paper is organised as follows. Section 1 introduces the Priority Based Parallel Multi-scheduler for Grids. Section 2 provides some background for our research and Section 3 places the research in the context of related work and current trends. Section 4 discusses our Priority grouping method for jobs and our Machine grouping methods. Section 5 exposes our simulation for the experiment and Section 6 describes the experimental design. Section 7 presents and discusses the results, analysing the performance of the Priority grouping method. A conclusion and future thoughts are provided in Section 8.

2. Background

The backbone of the Grid [1] is the already established Internet, powerful supercomputers, multiple computing clusters, large scale distributed networks and the connectivity of these resources. The aggregation and integration of these powerful computing systems, clusters, networks and resources, implemented with policies that ensure the delivery of computing services to users' specifications or requirements, is what represents the Grid.

The Grid computing environment enables jobs submitted by users to be executed at Grid sites that may be located thousands and millions of miles apart. This demands reliability of the hardware and software, efficiency in time consumption and effectiveness in the utilisation of resources. It also requires that schedules are as optimal as possible. Finding the optimal schedule for a set of jobs is an NP-complete problem and so heuristics are typically used. Alternatively, non-deterministic algorithms such as genetic algorithms can be used. However if the scheduling algorithm becomes too complex, the benefits of obtaining an optimal solution are outweighed by the time it takes to schedule. Our aim is to reduce scheduling time by using parallelism in the multicore environment.

2.1. Pervasive Grid and multicores

With advances in chip technology, current computer systems are now imbued with multiple cores [2–4]. These advances are due largely to the paradigm shift in hardware design and call for a retrospective paradigm shift in software design technology if the potential gains of multicore computing are to be achieved. The requirement on Grid utility that jobs submitted by users from several and different locations are processed at other locations millions of miles away demands an effective and fundamental scheduling algorithm [5,6]. Scheduling of Grid jobs and the growth of the Grid will be hampered if Grid scheduling algorithms are not designed to benefit from the hardware technology of the day [5,7].

For Grid scheduling to gain from these advances in hardware technology, meet its projected growth and the challenges of the future and also make good the revolution in computer hardware design, a paradigm shift is imperative in software programming model [8]. This is because sequential programs do not scale with multicore systems and therefore do not benefit from parallelism [9–13]. Hence, it will be a welcome development for effort to focus on methods that ensure the gains in computing hardware are translated generally to gains in software production and in particular Grid scheduling. This is a necessity because Grid scheduling can become a bottleneck when the number of jobs increases and the scheduling process remains sequential. It is therefore important to exploit parallelism in scheduling and make use of multicore availability to improve throughput in the scheduler.

2.2. Some impediments to the impact of multicore systems

The advent of multicore systems created a gap in application designs as execution of jobs in sequence in the midst of several processors does not optimise utilisation of available processors. Some of the impediments to legacy systems and applications in the multicore era according to Gurudutt Kumar [9] include:

- *Inefficient parallelisation* – which is an impediment in legacy systems or applications that fail to support multi-threading and in some cases too many threads,
- *Serial bottlenecks* – which are mostly common to applications that share a single data source among contending threads, or serialisation of data-accessing processes to maintain integrity,
- *Over-dependence on operating system or runtime environment* – which arises when too much is handed to the operating system or runtime environment to scale and optimise the application,
- *Workload imbalance* – where the tasks are unevenly spread to the various cores,
- *I/O bottlenecks* – which occur due to disk I/O blocking and
- *Inefficient memory management* – which is a performance inhibitor caused by the sharing of memory by several CPUs.

To correct or eliminate these impediments, many scientific and engineering platforms have reacted appropriately by embracing and implementing mechanisms that utilise multicores to greater benefit [7,11,14–16] while other researchers have