Available online at www.sciencedirect.com

ScienceDirect

Future Generation Computer Systems 25 (2009) 364–370

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/fgcs

An efficient adaptive scheduling policy for high-performance computing

J.H. Abawajy*

Deakin University, School of Engineering and Information Technology, Geelong, VIC. 3217, Australia

Received 25 March 2006; accepted 20 April 2006
Available online 26 July 2006

Abstract

The advent of commodity-based high-performance clusters has raised parallel and distributed computing to a new level. However, in order to achieve the best possible performance improvements for large-scale computing problems as well as good resource utilization, efficient resource management and scheduling is required. This paper proposes a new two-level adaptive space-sharing scheduling policy for non-dedicated heterogeneous commodity-based high-performance clusters. Using trace-driven simulation, the performance of the proposed scheduling policy is compared with existing adaptive space-sharing policies. Results of the simulation show that the proposed policy performs substantially better than the existing policies.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Distributed systems; Commodity cluster computing; Space-sharing; Job scheduling; Heterogeneous systems; Performance analysis

1. Introduction

The processing power of a single computer system has become inadequate for certain problems of a global scale, and the use of a super-computer is not always an option for many researchers. With advances in hardware, software and computer networks, the pre-dispositions in high-performance computing system design and deployment have shifted from the conventional parallel and distributed super-computer onto network-based distributed systems such as commodity-based cluster computing [11]. Clusters are now recognized as popular high-performance computing platforms for both scientific and commercial applications [5]. These commodity-based high-performance clusters have better price–performance ratios for a given computing problem than alternative high-performance computing platforms and have raised parallel and distributed computing to a new level.

In this paper, we address the problem of job scheduling for commodity-based *high-performance clusters (HPC)* with an emphases on *space-sharing* scheduling policy. The motivation for addressing this problem is that, while commodity-based HPC clusters offer tremendous computing power, this potential power is not exploited effectively [12,2]. Also, as commodity clusters become more commonly used for large-scale applications that pose tremendous processing and/or storage demands on the system, resource management and scheduling becomes an important issue for the efficient deployment of commodity-based HPC clusters [2,12,6].

Since commodity-based HPC clusters are commonly operated in space-sharing mode [6], the *space-sharing* scheduling policy is an appropriate policy for such platform. Space-sharing is the short-term partition of processors in the system into varying size sets and then the allocation of each set to different jobs. There are three main forms of space-sharing disciplines: *static space-sharing*, *adaptive space-sharing*, and *dynamic space-sharing*. Under *static space-sharing* policy, the processors are divided into a fixed number of disjoint sets, each of which is allocated to individual jobs. The problem with the static approach is that it can lead to a problem known as *processor fragmentation*, in which there is a mismatch between the allocated processor size and the processor requirements of the jobs [1]. Also, short jobs can easily be blocked by long jobs for a long time before being executed. However, in practice short jobs usually demand a short turnaround time. The *adaptive space-sharing* policy configures each job to execute on a subset of the total available processors, the size of which is based on the job's processor requirements as well as the current system load conditions. However, adaptive approaches are not sensitive to subsequent system changes. This shortcoming

* Tel.: +61 3 52271376.
E-mail address: jemal@sunet.com.au.

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.04.007

is addressed in the *dynamic space-sharing* approach, where processors can be taken away from running jobs, or jobs can be given extra processors at run-time. This pre-emptive capability, however, entails substantial overhead, as such dynamic space-sharing policies are commonly used in shared memory systems, while *adaptive space-sharing* policies are the preferred choice in distributed systems over the *static space-sharing* policy.

In this paper, we propose a new adaptive space-sharing policy for non-dedicated, heterogeneous and distributively owned HPC clusters [2]. Although the *adaptive space-sharing* policy has been shown to provide better performance than the *static space-sharing* policy for distributed memory systems [9, 3], the static space-sharing policy is the most common approach used in commodity-based HPC clusters [6,10,7]. Using trace-driven simulation, the effectiveness of the proposed policy is examined. We compare the performance of the proposed policy with two baseline policies. The results show that the proposed policy provides substantial performance improvements at system loads of interest (i.e., medium-to-high system loads). Also, an increased understanding of the ways in which a shared environment and resource heterogeneity affect space-sharing parallel job scheduling policies is discussed. From this study, we conclude that an adaptive space-sharing policy based on commodity clusters is viable, both from a conceptual point of view and quantitatively for jobs of sufficient size.

The rest of the paper is organized as follows. Section 2 discusses the background, related work and system model used in this paper. In Section 3, a new adaptive space-sharing policy is proposed to address these problems. In Section 5, the baseline adaptive space-sharing policies used to compare with the proposed policy are discussed. In Section 6, we describe the experimental setup as well as the system and workload models used in the experiments. The performance of these adaptive policies under various system and workload parameters is presented in Section 7. This section shows that the proposed adaptive policy outperforms two different similar policies. The conclusion and future directions are given in Section 8.

2. Background and problem statement

In this section, we discuss the background, related work and the system workloads. We also formulate the problem and discuss the basic structure of adaptive space-sharing policies and the problem facing these scheduling policies under a shared heterogeneous environment.

2.1. Commodity-based high-performance computing

Commodity-based HPC clusters can generally be classified into *enterprise high-performance clusters (EHPC)* and *pervasive high-performance clusters (PHPC)*. Both clusters are formed from a collection of commodity off-the-shelf hardware computers interconnected and configured to operate as a single unit. However, computers in EHPC are dedicated, homogeneous and interconnected with a local-area network. Also, the entire system is privately owned and operates within a network scope managed by a single system administrator. In contrast, pervasive computing environments are far more dynamic and heterogeneous than enterprise environments. In pervasive computing, heterogeneity occurs in many aspects: hardware, software platforms, network protocols, and processors. Also, computers in PHPC are individually owned, non-dedicated and interconnected with a local-area or wide-area network [2].

2.2. System workloads

In this paper, we focus on *pervasive high-performance clusters (PHPC)*. Users submit applications to the system for execution without being concerned about the heterogenity or the changing set of volunteer machines on which the computation is actually performed. In this paper, we refer to a program or an application as a job. There are two types of jobs in the system: local and parallel. A parallel job is composed of a set of tasks that can execute on any of the processors as *external tasks*. In contrast, the local jobs consist of a single process (task) and run as a single entity on the workstation that they are originated on as *local tasks*. Moreover, local tasks have pre-emptive priority over the external tasks, which means that, when a local task arrives at a node, the external tasks running on that node are suspended and will be resumed when the node becomes available or migrated to another available node by the *Migration Manager*.

2.3. Problem statement

The universal problem in resource allocation is the conflict between individual users trying to maximize their use of a resource and the global limitations on the availability of that resource. In shared and heterogeneous systems, in addition to the classic allocation problem (how many processors to allocate to a job), we are also faced with the placement problem (which processors to allocate to a job) and task reassignment problem (when and which tasks to reallocate to where).

Exiting scheduling policies for commodity-based HPC clusters provide only rudimentary facilities for space-sharing the processors. Existing *adaptive space-sharing* policies have been developed for traditional dedicated homogeneous multiprocessor systems [9] and have also found their way into *commodity cluster computing* environments with very little or no modifications [8]. Also, most of the previous studies focus on optimizing a single application at a time. However, the viability of running parallel and sequential applications concurrently on clusters has been demonstrated in [4].

The conventional adaptive space-sharing policy can generally be characterized by three main attributes:

1. the scheduler is activated whenever a job arrives and there are idle processors, or whenever a job departs and there are queued jobs;
2. an incoming job is assigned to a subset of the total available processors; and
3. a running job releases its partition only after it finishes execution.