Comparative Evaluation of the Non-Contiguous Processor Allocation Strategies Based on a Real Workload and a Stochastic Workload on Multicomputers

S. Bani-Mohammad  M. Ould-Khaoua  I. Ababneh  Lewis M. Mackenzie
Glasgow University  Glasgow University  Al-al-Bayt University  Glasgow University
Computing Science  Computing Science  Computing Science  Computing Science
Glasgow G12 8RZ  Glasgow G12 8RZ  Mafraq 25113  Glasgow G12 8RZ
UK.  UK.  Jordan.  UK.
saad@dcs.gla.ac.uk  mohamed@dcs.gla.ac.uk  ismail@aabu.edu.jo  lewis@dcs.gla.ac.uk

Abstract

The performance study of the existing non-contiguous processor allocation strategies has been traditionally carried out by means of simulation based on a stochastic workload model to generate a stream of incoming jobs that are submitted to and run on a given message passing parallel machine for a period of time. To validate the performance of the existing allocation algorithms, there has been need to evaluate the algorithms’ performance based on a real workload trace. In this study, we evaluate the performance of several well-known processor allocation and job scheduling strategies based on a real workload trace and compare the results against those obtained from using a stochastic workload. Our results reveal that the conclusions reached on the relative performance merits of the allocation strategies when a real workload trace is used are in general compatible with those obtained when a stochastic workload is used.

1. Introduction

Efficient processor allocation and job scheduling are critical to harnessing the full computing power of a multicomputer [1, 5]. The goal of job scheduling is to select the next job to be executed while the goal of processor allocation is to select the set of processors on which parallel jobs are executed [1, 3].

Most existing allocation strategies employed in a multicomputer are based on contiguous allocation, where the processors allocated to a parallel job are physically contiguous and have the same topology as that of the interconnection network of the multicomputer [1, 4, 5, 6, 19]. Contiguous processor allocation strategies often result in high external processor fragmentation, as has been shown in [19]. External fragmentation occurs when there are free processors sufficient in number to satisfy the number requested by a parallel job, but they are not allocated to it because the free processors are not contiguous or they do not have the same topology as the multicomputer.

Several studies have attempted to reduce external fragmentation [2, 4, 13, 15, 18]. One suggested solution is to adopt non-contiguous allocation [2, 13, 15, 18]. In non-contiguous allocation, a job can execute on multiple disjoint smaller sub-networks rather than always waiting until a single sub-network of the requested size and shape is available. Although non-contiguous allocation increases message contention in the network, lifting the contiguity condition is expected to reduce processor fragmentation and increase processor utilization [2, 13, 15, 18].

Most existing research on contiguous and non-contiguous allocation has been carried out in the context of the 2D mesh [1, 2, 4, 5, 6, 7, 13, 15, 18, 19]. The mesh has been one of the most common networks for multicomputers due to its simplicity, scalability, structural regularity, and ease of implementation [1, 6].

Most existing allocation strategies employed in a multicomputer have been evaluated by means of simulation using a stochastic workload [1, 2, 3, 5, 6, 7, 13, 14, 15, 18, 19], based mostly on probabilistic distributions. In this workload, the researchers have used exponential distribution to generate inter-arrival times. Job sizes have been generated using a variety of probabilistic distributions including uniform, uniform increasing, uniform decreasing, normal, and
exponential distributions while jobs execution times have been generated with exponential distribution.

In this study, the performance of the existing non-contiguous allocation strategies is evaluated using a real workload trace and the results are compared against those obtained through a stochastic workload. A real workload trace is a record of execution of parallel jobs submitted to run on a practical parallel machine, in which each job arrives in the system, requests a particular sized partition of the system’s processors and executes on the partition for a period of time. Although messages from other jobs may pass through the new partition, the new job holds the partition exclusively until it finishes running. At this time, the job departs the system and its processors are freed for use by another incoming job [12, 17]. The partitions requested by the jobs typically include job size (the number of processors requested by these jobs) and the execution time. Moreover, each job in the workload is associated with an arrival time, indicating when it is submitted to the scheduler for consideration. A real workload trace can potentially provide a very high level of realism when used directly in performance evaluation experiments [17]. Our results reveal that the conclusions reached on the relative performance merits of the allocation strategies when a real workload trace is used are compatible with those obtained when a stochastic workload is used.

The rest of the paper is organized as follows. Section 2 provides some preliminaries. Section 3 contains a brief overview of the allocation strategies considered in this study while Section 4 contains a brief overview of the scheduling strategies. Section 5 presents simulation results. Finally, Section 6 concludes this study.

2. Preliminaries

The target system is a \( W \times L \) 2D mesh, where \( W \) is the width of the mesh and \( L \) is its length. Every processor is denoted by a pair of coordinates \((x, y)\), where \( 0 \leq x < W \) and \( 0 \leq y < L \) [15]. Each processor is connected by bidirectional communication links to its neighbour processors. The following definitions have been adopted from [15].

**Definition 1:** A sub-mesh \( S(w, l) \) of width \( w \) and length \( l \), where \( 0 < w \leq W \) and \( 0 < l \leq L \) is specified by the coordinates \((x, y, x', y')\), where \((x, y)\) is the lower left corner of \( S \) and \((x', y')\) is its upper right corner. The size of \( S(w, l) \) is \( w \times l \) processors.

**Definition 2:** An allocated sub-mesh is one whose processors are all allocated to a parallel job while a free sub-mesh is one whose processors are all not allocated.

**Definition 3:** A suitable sub-mesh \( S(w, l) \) is a free sub-mesh that satisfies the conditions: \( w \geq a \) and \( l \geq b \) assuming that the allocation of \( S(a, b) \) is requested.

In this study, it is assumed that parallel jobs are selected for allocation and execution using the First-Come-First-Served (FCFS) and the Shortest-Service–Demand (SSD) (i.e., shortest execution time) scheduling strategies. The FCFS scheduling strategy is chosen because it is fair and it is widely used in other similar studies [2, 5, 6, 15, 18, 19] while the SSD scheduling strategy is used to avoid potential performance loss due to blocking [11].

3. Non-contiguous Allocation Strategies

Advances in routing techniques such as wormhole routing [16], have made communication latency less sensitive to the distance between communicating nodes [2]. This has made allocating a job to non-contiguous processors plausible in networks characterised by long-diameters, such the 2D mesh. Below, we describe some non-contiguous allocation strategies that have been proposed for the 2D mesh.

**Paging:** In the Paging strategy [18], the entire 2D mesh is divided into pages that are sub-meshes with equal sides’ length of \( 2^{\text{size-index}} \), where \( \text{size-index} \) is a positive integer. A page is the allocation unit. The pages are indexed according to several indexing schemes (row-major, shuffled row-major, snake-like, and shuffled snake-like indexing). In this study, we only consider the row-major indexing scheme because using the remaining indexing schemes has only a slight impact on the performance of Paging, as has been demonstrated in [18]. The number of pages a job requests is computed by: \( \lceil (a \times b) / \text{Psize} \rceil \), where \( \text{Psize} \) is the size of the pages, and \( a \) and \( b \) are the side lengths of the requested sub-mesh. In Paging, there is some degree of contiguity because of the indexing schemes used. Contiguity can also be increased by increasing the parameter \( \text{size-index} \). However, there is internal processor fragmentation for \( \text{size-index} \geq 1 \), and it increases with \( \text{size-index} \) [18].

**Multiple Buddy Strategy (MBS):** In MBS [18], the mesh is divided into non-overlapping square sub-meshes with side lengths equal to the powers of two upon initialization. The number of processors, \( p \), requested by an incoming job is factorized into a base-4 representation of the form: \( p = d_0 \times 2^0 + d_1 \times 2^1 + d_2 \times 2^2 + \ldots + d_i \times 2^i \), where
The request is then considered for allocation according to the factorized number, where \( d_i \) blocks of size \( 2^i \times 2^i \) are required. If a required block is unavailable, MBS recursively searches for a larger block and repeatedly breaks it down into four buddies until it produces blocks of the desired size. If that fails, the requested block is broken into four requests for smaller blocks and the searching process is repeated [18]. An issue with MBS is that it may fail to allocate a contiguous sub-mesh although one exists. In fact, contiguous allocation is explicitly sought in MBS only for requests with sizes of the form \( 2^{2n} \), where \( n \) is a positive integer.

**Greedy Available Busy List (GABL):** The GABL strategy combines the desirable features of both contiguous and non-contiguous allocation, and partitions requests based on the sub-meshes available for allocation [13]. In GABL [13], when a parallel job is selected for allocation a sub-mesh suitable for the entire job is searched for. If such a sub-mesh is found it is allocated to the job and the allocation is done. Otherwise, the largest free sub-mesh that can fit inside \( S(a,b) \) is allocated. Then, the largest free sub-mesh whose side lengths do not exceed the corresponding side lengths of the previously allocated sub-mesh is searched for under the constraint that the number of processors allocated does not exceed \( a \times b \). This last step is repeated until \( a \times b \) processors are allocated. Allocated sub-meshes are kept in a busy list. Each element in this list includes the \( id \) of the job the sub-mesh is allocated to. When a job departs the sub-meshes it is allocated are removed from the busy list and the number of free processors is updated. Note that allocation always succeeds if the number of free processors is \( \geq a \times b \). Moreover, it can be noticed that the methodology used for maintaining contiguity is greedy. GABL attempts to allocate large sub-meshes first.

The above allocation strategies have been selected for comparison in this study because they have been shown to perform well in [13, 18].

**4. Job Scheduling Strategies**

The order in which jobs are scheduled can have considerable effect on the performance of the allocation strategies in mesh multicomputers [11, 14]. In this study, we consider the FCFS and SSD scheduling strategies. In FCFS scheduling strategy, the allocation request that arrived first is considered for allocation first. Allocation attempts stop when they fail for the current FIFO queue head. The SSD scheduling strategy considers the shortest job to be the one having the shortest total processor service demand [8, 11, 14].

The performance of non-contiguous allocation can be significantly affected by the type of the scheduling strategy used. To illustrate this, the performance of the allocation strategies considered in this study is evaluated under both FCFS and SSD scheduling strategies. The results show that the SSD scheduling strategy is much better than the FCFS scheduling strategy, therefore, the scheduling and allocation strategies both have substantial effect on the performance of non-contiguous allocation strategies in 2D mesh.

**5. Simulation Results**

Extensive simulation experiments have been carried out to compare the performance of the allocation strategies considered in this study. We have implemented the allocation and deallocation algorithms, including the busy list routines, in the C language, and integrated the software into the ProcSimity simulation tool for processor allocation and job scheduling in highly parallel systems. ProcSimity has been used to investigate some of the processor allocation problems, such as fragmentation and communication overhead problems [8, 12].

In the results shown below, we model a 16 \( \times \) 22 mesh. This size was selected to closely match the size of the partition which generated the trace. Jobs are served on the FCFS and the SSD scheduling strategies. The interconnection network uses wormhole routing. Flits are assumed to take one time unit to move between two adjacent nodes, and \( t_s \) time units to be routed through a node. A flit is the smallest unit of data transmission in a wormhole routing network. Message sizes are represented by \( P_{len} \). Processors allocated to a job communicate with each other using the all-to-all communication pattern [7, 16, 18]. In all-to-all communication pattern, each processor allocated to a job sends a packet to all other processors allocated to the same job. This communication pattern is used because it causes much message collision and it is known as the weak point for non-contiguous allocation algorithms [7]. As in [18], the number of messages that are actually generated by a given job is exponentially distributed with a mean \( num_{mes} \).

Unless specified otherwise, the performance figures shown below are for a 16 \( \times \) 22 mesh, \( t_s = 3 \) time units, \( P_{len} = 8 \) flits and \( num_{mes} = 5 \) packets. The values we use for \( t_s \) and \( P_{len} \) were recommended in [12]. The main performance parameters used are the average
turnaround time of jobs and the average packet blocking time. The turnaround time of a job is the time that the job spends in the mesh from arrival to departure. The average packet blocking time is the average time that message packets spend blocked in network buffers, waiting for access to their next channel [12]. An important independent variable in the simulation is the system load, defined as the inverse of the mean inter-arrival time of jobs. Its values for the various simulation points were determined through experimentation with the simulator.

To allow for both easy comparison with previous experiments in [13] and for a realistic evaluation of allocation strategies, our performance evaluation includes the use of two different workloads. The first workload is a stochastic workload. In a stochastic workload, jobs are assumed to have exponential inter-arrival times, the execution times of jobs are not simulator inputs. They are determined by the simulator and their values depend on $t_s$, $P_{len}$, the number of messages sent, message contention, and distances messages traverse. We have used uniform distribution to generate the widths and lengths of the job requests. The uniform distribution is used over $[1, W]$ for the width of the requested sub-mesh and over $[1, L]$ for the length of the requested sub-mesh, where the width and length of a job request are generated independently. These distributions have often been used in the literature [5, 15, 18, 19]. Each simulation run consists of 1000 completed jobs. Simulation results are averaged over enough independent runs so that the confidence level is 95% and the relative errors do not exceed 5%.

The second workload is a real workload trace, a stream of 10658 real production jobs from the Intel Paragon at the San Diego Supercomputer Centre. The traced job stream is taken only from the 352 nodes [9, 10]. The workload trace had the following statistical characteristics: the mean interarrival time was 1186.7 seconds; the average job size was 34.5 nodes, and with the distribution favouring sizes that are non-powers of two. Our real workload trace uses the arrival times, job execution times and job sizes. To challenge the allocation strategies, we multiply job arrival time $s$ by a constant factor $f$. When $f < 1$, the simulated interarrival times decrease, resulting in an increased system load.

The notation <allocation strategy> (<scheduling strategy>) is used to represent the strategies in the performance figures. For example, GABL(SSD) refers to the Greedy Available Busy List allocation strategy under the Shortest-Service-Demand scheduling strategy.

In Figures 1 and 2, the average turnaround times of jobs are plotted against the system load for the all-to-all communication pattern and the two scheduling strategies, FCFS and SSD. It can be seen from these figures that the results from both workloads ranked the scheduling and allocation algorithms in the same order from best to worst in terms of performance parameter used in these figures except that the performance of the MBS strategy based on a real workload is inferior to that of the Paging(0) strategy because the size of the sub-meshes requested by jobs in this workload is non-power of two and the contiguous allocation is explicitly sought in MBS strategy only for requests with sizes of the form $2^{2n}$, where $n$ is a positive integer. The results reveal that the GABL allocation strategy performs better than all other allocation strategies for both workloads and scheduling strategies considered in this study. In Figure 1, for example, the difference in performance in favour for the GABL(FCFS) strategy could be as large as 33% to the Paging(0)(FCFS) strategy, and 68% to the MBS(FCFS) strategy under the job arrival rate 0.004 jobs/time unit. Experiments that use larger packet sizes (16, 32, and 64 flits) based on a stochastic workload have been also conducted. Their results lead to the same conclusion on the relative performance of the allocation strategies. Moreover, the results indicate that the relative performance merits of the GABL strategy over the remaining strategies become more noticeable as the packet length increases. Results have also shown that the effects of the SSD scheduling strategy on the performance of the allocation strategies are better than that of the FCFS scheduling strategy under both workloads considered in this study.

In addition to the turnaround times, we have measured another performance parameter for the non-contiguous allocation strategies considered in this study. This is the average packet blocking time computed for all jobs. Packet blocking time is used to measure the contention in the interconnection network. Messages originate from a processor element and their flits traverse the network in pipeline fashion to their destination processor. If the header flit of a packet is routed to a busy channel, that header flit and its trailing flits stop moving and block whichever channels they occupy in the network [18]. This results in packet blocking time, due to contention, which can be measured in the simulation.

In Figures 3 and 4, the average packet blocking time is plotted against the system load for the all-to-all communication pattern and the two scheduling strategies considered in this study, FCFS and SSD. The results of the experiments from both workloads ranked the scheduling and allocation algorithms in the same
order from best to worst; GABL strategy has better packet blocking time than the remaining strategies for all loads. In Figure 3, for example, the average packet blocking times of the GABL(SSD) strategy are 81% and 51% of the average packet blocking times of the Paging(0)(SSD) and MBS(SSD) strategies, respectively, when the job arrival rate is 0.02 jobs/time unit.

*Figure 1. Turnaround time vs. system load for the all-to-all communication pattern and a real workload in a 16 × 22 mesh.*

*Figure 2. Turnaround time vs. system load for the all-to-all communication pattern and a stochastic workload based on uniform side lengths distribution in a 16 × 22 mesh.*

*Figure 3. Packet blocking time vs. system load for the all-to-all communication pattern and a real workload in a 16 × 22 mesh.*
To sum up, the above performance results demonstrate that both a real workload and a stochastic workload considered in this study gave the same ranking of the allocation strategies from best to worst.

6. Conclusion and Future Directions

This study has investigated the impact of real workload traces and stochastic workloads on the performance of the non-contiguous allocation strategies that have been proposed for 2D mesh connected multicomputers. These non-contiguous allocation strategies cover a wide range of choices, including Paging strategy (Paging(0)), Multiple Buddy Strategy (MBS), and Greedy Available Busy List strategy (GABL). The GABL allocation strategy attempts to maintain a high degree of contiguity among processors allocated to a parallel job which decreases the number of sub-meshes allocated to a job, hence decreases the distance traversed by messages, and which in turn decreases the communication overhead. The GABL allocation strategy achieves this by using a busy list whose length is often small even when the size of the mesh scales up.

In this study, two scheduling strategies have been used to avoid potential performance loss due to blocking that results from largest jobs. Simulation results have shown that the relative performance of the non-contiguous allocation strategies has not been significantly affected by the choice of a workload. In most experiments a real workload and a stochastic workload ranked the performance of the non-contiguous allocation algorithms in the same order from best to worst in terms of parameters such as average turnaround time and average packet blocking time. Moreover, the simulation results have shown that the effects of the SSD scheduling strategy on the performance of the allocation strategies are better than that of the FCFS scheduling strategy in terms of average turnaround time.

As a continuation of this research in the future, it would be interesting to assess the allocation strategies in other common multicomputer networks, such as torus networks. Another possible direction for future research is to implement the allocation strategies based on other real workload traces from different parallel machines and compare it with our results by means of simulation.

7. Acknowledgment

We would like to thank Prof. Dror Feitelson, School of Computer Science and Engineering, Hebrew University, who provided us workload traces and information regarding the traces. Also, we would like to thank Dr. Eitan Frachtenberg, Hebrew University, who helped us to get in touch with Prof. Feitelson regarding the traces.

8. References


