Simulation Modelling Practice and Theory 19 (2011) 482–493

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

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

A new window-based job scheduling scheme for 2D mesh multicomputers

Ismail Ababneh [a], Saad Bani-Mohammad [b,*]

[a] *Computer Science Department, Jordan University of Science and Technology, Irbid, Jordan*
[b] *Computer Science Department, Prince Hussein Bin Abdullah College for Information Technology, Al al-Bayt University, Mafraq 25113, Jordan*

A R T I C L E I N F O

Article history:
Received 1 February 2010
Received in revised form 1 August 2010
Accepted 17 August 2010
Available online 21 August 2010

Keywords:
Job scheduling
Mesh
Multicomputer
Contiguous submesh allocation
Average turnaround time
Maximum waiting delay

A B S T R A C T

Allocating submeshes to jobs in mesh-connected multicomputers in a FCFS fashion can lead to poor system performance (e.g., long job waiting delays) because the job at the head of the waiting queue can prevent the allocation of free submeshes to other waiting jobs with smaller submesh requirements. However, serving jobs aggressively out-of-order can lead to excessive waiting delays for jobs with large allocation requests. In this paper, we propose a scheduling scheme that uses a window of consecutive jobs from which it selects jobs for allocation and execution. This window starts with the current oldest waiting job and corresponds to the lookahead of the scheduler. The performance of the proposed window-based scheme has been compared to that of FCFS and other previous job scheduling schemes. Extensive simulation results based on synthetic workloads and real workload traces indicate that the new scheduling strategy exhibits good performance when the scheduling window size is large. In particular, it is substantially superior to FCFS in terms of system utilization, average job turnaround times, and maximum waiting delays under medium to heavy system loads. Also, it is superior to aggressive out-of-order scheduling in terms of maximum job waiting delays. Window-based job scheduling can improve both overall system performance and fairness (i.e., maximum job waiting delays) by adopting large lookahead job scheduling windows.

1. Introduction

Mesh-connected interconnection networks have been widely used in recent distributed-memory parallel computers [3,7,10,12,15,23]. This is mainly because these networks are regular, simple, easy to implement, and scalable. Both two-dimensional (2D) and three-dimensional (3D) meshes and tori have been used in recent commercial and experimental multicomputers, such as the Caltech Mosaic [5], the Intel Paragon [19], the IBM BlueGene/L [9,18,21], and the Cray XT3 [11,22].

In most processor allocation policies proposed for mesh-connected multicomputers, a parallel job is allocated its own submesh of processors of the size and shape it has requested [3,7,8,15,20]. This can result in high external processor fragmentation, which occurs when free processors are not allocated to a parallel job because they do not satisfy the job's shape constraint. A recurring outcome in allocation studies is that contiguous allocation suffers from low overall system utilization [3,14,23]. It can reduce system utilization to low levels. Therefore, noncontiguous allocation policies have been proposed so as to increase system utilization by allowing dispersed free processors to be allocated to a parallel job [2,12,23].

Notwithstanding the ability of noncontiguous allocation to reduce, even eliminate, processor fragmentation, an advantage of contiguous allocation is that it isolates jobs from each other, which is useful for security and accounting reasons [4]. For example, contiguous allocation is proposed for use in the IBM BlueGene/L for security reasons [4]. A BlueGene/L

* Corresponding author. Tel.: +962 2 6297000; fax: +962 2 6297051.
E-mail addresses: ismael@just.edu.jo (I. Ababneh), bani@aabu.edu.jo (S. Bani-Mohammad).

1569-190X/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2010.08.007

job is allocated a partition of processors that is isolated from partitions allocated to other jobs because of the sensitive nature of some of BlueGene's applications [4].

Another approach to improving system utilization is using job scheduling policies that are not strictly first-come-first-served. Rather than always accommodating the oldest allocation request first, allocation to more recent requests is considered in order to decrease the number of idle processors and improve overall system performance [1,8,16,20,24]. These schemes, however, have several shortcomings. In [1], jobs are considered for allocation without ever waiting for the head of the queue or earlier requests. This greedy out-of-order scheduling can lead to excessive waiting delays, including indefinite postponement, for large jobs. Because small jobs are easier to accommodate, they can block allocation to a large job that has arrived earlier. Other schemes [8,16] place small bounds on the ability of the non-FCFS scheme to bypass the head of the waiting queue. Results in [3] indicate that such bypassing bounds should increase with the system load, and they should be much larger than those considered in [8,16]. Finally, the approach proposed in [24] assumes that job execution time estimates are available upon job submission. Although many current systems require users to submit such estimates, users often provide inaccurate estimated job execution times [17]. Therefore, we have limited ourselves to the case where execution time estimates are assumed to be unavailable.

In this paper, we propose a non-FCFS job scheduling scheme for use in mesh-connected multicomputers. The scheme aims to bound job waiting delays, while reducing processor fragmentation. In the proposed scheme, it is possible to bypass the head of the waiting queue of jobs awaiting allocation, but this ability is limited to a window of consecutive jobs that starts with the head of the waiting queue. This scheduling window corresponds to the lookahead capability of the job scheduler. Bypassing the head of the waiting queue has for goal reducing processor fragmentation by finding jobs that can be accommodated, which improves system utilization and average job turnaround times. Limiting the ability to bypass the oldest waiting job to a window has for goal bounding job waiting delays. Using detailed simulations, we have evaluated the proposed scheme and compared it with previous schemes. The results show that the proposed scheme can yield good average job waiting delays (i.e., good average turnaround times) and system utilization, and superior maximum job waiting delays.

This paper is organized as follows. The following section contains a review of previous related job scheduling schemes. Section 3 contains some preliminaries and the system model assumed. The proposed scheme is presented in Section 4. Simulation results are presented and discussed in Section 5. Finally, conclusions are given in Section 6.

2. Previous job scheduling schemes

Typically, traditional first-come-first-served (FCFS) job scheduling was assumed in research studies of contiguous processor allocation in mesh multicomputers [2,7,13–15,27]. In FCFS scheduling, jobs are served strictly in their arrival order. The queue where jobs wait is FIFO, and the job at the head of the waiting queue is considered for allocation before any subsequent job. An advantage of this scheme is its fairness. However, processor submeshes may remain unallocated because they are not large-enough for the head waiting job, and although they can satisfy other waiting allocation requests. As a consequence, contiguous allocation that assumes FCFS job scheduling suffers from low system utilization and poor performance. In order to improve system performance, several non-FCFS job scheduling policies that consider allocation to subsequent jobs when allocation fails for the head of the waiting queue were proposed and evaluated. Below, we include a brief review of previous job scheduling policies proposed for mesh-connected multicomputers.

2.1. k-Lookahead processor allocation

This scheme aims to retain large free mesh regions for large allocation requests. In k-lookahead allocation, the waiting requests are scanned and the largest k requests are selected. In making the allocation decision for the head of the waiting queue, an attempt is made to leave sufficient free mesh regions for the largest k waiting requests. Using simulations, the performance of 1-lookahead (i.e., $k = 1$) was evaluated. The results show that some moderate performance gains can be obtained [8]. In the general case, this scheme would require examining as many as $k!$ allocation cases. Also, $k = 1$ limits the scheduling lookahead window to two jobs (the oldest waiting job and the largest waiting job).

2.2. Reservation-based non-FCFS job scheduling

In this scheme [16], a waiting allocation request has a counter that contains the number of times that it has been overtaken by subsequent requests. When a new job arrives, allocation is attempted for the job provided that the counter of the current FIFO waiting queue head does not exceed some fixed value, called MAX_PRI. If this allocation attempt fails, reservation of a large-enough submesh is attempted for the new job, and if this allocation/reservation process fails the new job is appended to the waiting queue and its counter is set to zero. An issue with this policy is that reservation can decrease system utilization. Therefore, reservation degrades performance when the load is heavy, as can be seen in the simulation results in [16]. Another issue is that only small MAX_PRI values were considered because high MAX_PRI would mean excessive waiting delays for large jobs and unfairness to such jobs. A problem with this reasoning is that large MAX_PRI values can improve overall system utilization and waiting delays, which may also yield good maximum job waiting delays.

2.3. Delay-based non-FCFS job scheduling

In [20], another non-FCFS job scheduling policy was proposed and evaluated using simulations. In this policy, a dynamic threshold value is imposed on the waiting delay of the head of the waiting queue. When this threshold is reached, no other job may be considered for allocation until the head job is served. The threshold value is equal to λW, where λ is the arrival rate of jobs, and W is the average waiting delay of running jobs. The scheduler monitors both parameters. An issue with this scheme is that the values of λ that can be accommodated depend on job characteristics. For example, the system saturates for relatively smaller λ values when jobs are larger in size. In [14], a non-FCFS scheduling policy where the waiting head can be overtaken provided it has waited for less than a preset time period was proposed. However, their simulation results show that the system utilization under this scheme is only slightly better than that under FCFS [14].

2.4. Aggressive out-of-order job scheduling

This is a greedy scheme where jobs are considered for allocation based on their arrival sequence without having to wait for the head of the waiting queue [1]. An issue with this scheme is that the scheduling lookahead window is unbounded. Therefore, a large job at the head of the waiting queue may suffer excessive delays if small allocation requests keep arriving. Also, indefinite postponement is possible.

In this research, we hypothesize that we can achieve good overall system performance and fairness (i.e., maximum job waiting delays) by adopting a large bounded lookahead job scheduling window.

3. Preliminaries

3.1. Target system

The target system in this paper is the 2D mesh-connected multicomputer. It will be denoted as $M(W, H)$, where W is the width of the mesh and H is its height. A processor (node) in column x and row y is represented by the coordinates (x, y), where $1 \leqslant x \leqslant W$ and $1 \leqslant y \leqslant H$. Nodes are interconnected as shown in Fig. 1. The size of the mesh, N, is the number of processors it has, where $N = WH$.

A $w \times h$ submesh S in $M(W, H)$ is represented by (x_b, y_b, x_e, y_e), where (x_b, y_b) is the base (lower-left corner) of S, (x_e, y_e) is its end (upper-right corner), $w = x_e - x_b + 1$, $h = y_e - y_b + 1$, $1 \leqslant w \leqslant W$, and $1 \leqslant h \leqslant H$. A submesh is said to be free if all its processors are unallocated. For example, the submesh (4, 1, 5, 3) in Fig. 1 represents the free 2×3 submesh S_1, where (4, 1) is the base node of S_1 and (5, 3) is its end node.

Definition 1. A maximal free submesh is a free submesh that is not a proper subset of any other free submesh.

Example 1. The maximal free submeshes in Fig. 1 are (1, 1, 5, 2) and (4, 1, 5, 3). The free submesh (1, 1, 3, 2) is not maximal as it is a proper subset of (1, 1, 5, 2).

3.2. Submesh allocation scheme

The function of the job scheduling algorithm is to select which job can be considered for allocation and execution next, and the function of the submesh allocation scheme is to find and choose the submesh that is to be assigned to the selected job [2,7,13,14]. The submesh allocation scheme used in this research is the contiguous allocation scheme, maximum peripheral length (MPL), proposed in [3]. In MPL, preference is given to allocating a free submesh that has a maximum boundary length aligned with the periphery of the mesh system. For example, given the system state shown in Fig. 1, a request for the allocation of a 2×1 submesh may be allocated the submesh (4, 1, 5, 1), which has the maximal mesh peripheral length of 3. The node (5, 1) is located on *two* mesh edges and the node (4, 1) is located on *one* mesh edge. When no large-enough periph-

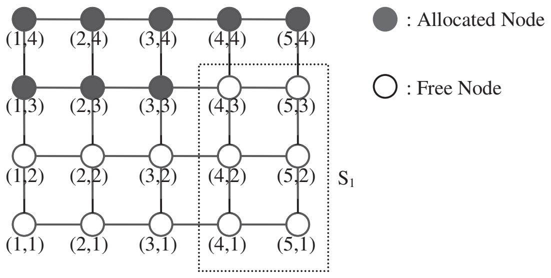

Fig. 1. A two-dimensional M(5, 4) mesh.

eral submesh is available, a request is allocated at the base of the first suitable internal free submesh that can be found; the peripheral length of an internal submesh is equal to zero. Although job scheduling schemes can be used with differing allocation schemes, we have chosen MPL because it performed well relative to other promising contiguous allocation schemes [3].

4. Proposed job scheduling scheme

The job scheduling scheme we propose in this paper uses a K-way window of consecutive jobs from which it can select jobs for allocation and execution. This window starts with the current oldest job awaiting allocation, and it contains up to K jobs that have arrived successively. Allocation to a job that is outside the scheduling window is possible only when the window's head job is assigned the submesh it has requested, the window is advanced to the next oldest job awaiting allocation, and the outside job falls within the new K-way scheduling window.

Definition 2. A K-way job scheduling window is a group of up to K successive jobs that start with the oldest job still waiting for allocation, and from which the scheduling algorithm can select jobs for allocation and execution.

Example 2. Given the target system in Fig. 1 and no waiting allocation requests, assume $K = 4$ and the next scheduling events are five job arrivals that respectively request the allocation of a 5×3 submesh, a 5×2 submesh, a 2×1 submesh, a 4×3 submesh, and a 3×3 submesh. The first job must wait because there is no free 5×3 submesh. It forms the first element in a new scheduling window. The second job is allocated the submesh $(1, 1, 5, 2)$, the third job is allocated the submesh $(4, 3, 5, 3)$, and allocation fails for the fourth job. The second, third and fourth job join the scheduling window successively behind the first job. Allocation is not attempted for the fifth job because it is outside the scheduling window. When allocation to the current window head (i.e., the first job) succeeds, the next job that is awaiting allocation becomes the head of the scheduling window, and allocation to the fifth job then becomes possible.

When a job departs and the submesh it is allocated is released, those window jobs that are still awaiting allocation (e.g., jobs 1 and 4 in Example 2) are considered for allocation in their arrival order. When allocation to the window head succeeds, the window is moved forward to the next waiting allocation request. In Example 2, if allocation succeeds for the first job, the window is moved past jobs 2 and 3 because they were served, the fourth job becomes the new window head, and the fifth job falls within the new scheduling window. That is, job 5 becomes a candidate for allocation.

The following steps represent the scheduling process for a new job:

1. If the size of the current scheduling window, *win_size*, is not less than K, append the new job at the tail of the FIFO job waiting queue.
2. If *win_size* < K, attempt allocation for the new job, increment *win_size* and insert job descriptor at the tail of the K-way scheduling window. The job descriptor includes the job's submesh requirement, and its allocation state (done or pending).

When a job terminates, the de-allocation job scheduling process described below is initiated:

1. Scan the scheduling window for the next pending allocation request.
2. If the *pending* request is the window head and allocation succeeds, move the window head to the next *pending* request in the window, if any. If no window request is still *pending*, move the window head to the waiting allocation request that arrived following the window tail. In both cases, adjust *win_size* and re-start step 1.
3. If the *pending* request is not the window head and allocation succeeds, change the state of the request to *done*.
4. Go to step 1 so as to continue with the next *pending* request, if any. Otherwise, return.

5. Simulation results

We have evaluated, using simulations, the window-based scheme by comparing it against the FCFS scheduling, aggressive out-of-order (OO) scheduling [1], and a variant of the scheme proposed in [16]. The variant adopted here is that which does not support reservation. As stated earlier, a problem with processor reservation is that it can result in idle processors. As a result, reservation degrades performance under high loads, as can be seen in the simulation results in [16]. We refer to this reservation-free variant as out-of-order with constant bound (OOCB) scheduling scheme. In this research, we use Max-Bound = 8 (OOCB-8) because this value is a good representative of the values considered in [16]. The latter scheme is a representative of policies that bound the number of times that subsequent requests can overtake the head of the FIFO waiting queue. When Max-Bound is equal to eight, up to that many subsequent jobs can be allocated the submesh they have requested before having to serve the current waiting queue head, however these jobs are not bound to a fixed-size window. That is, any more recent job can overtake the oldest job in the waiting queue provided this has not happened more than Max-Bound times. Note that FCFS corresponds to Max-Bound = 1 and OO corresponds to Max-Bound = ∞.

5.1. Synthetic workload models

The workload models we use in this paper include synthetic workloads used in previous research efforts [6,16,20,23]. The target system $M(L, L)$ is square with a side length of L. Jobs are generated independently according to a Poisson process. They have exponential interarrival times, and their service times are distributed exponentially with a mean of one time-unit. The time units for synthetic workloads are simulation time units, expressed by floating point numbers, not hours, minutes, or seconds [26]. A job requests the allocation of a submesh upon arrival, where the widths and heights of submeshes requested are generated independently using one of two distributions. The first distribution is uniform over $[1, L]$, and the second distribution is a uniform-decreasing distribution that is based on four probabilities p_1, p_2, p_3, and p_4, and three integers l_1, l_2, and l_3, where the probabilities that the width/height of a request falls in the ranges $[1, l_1]$, $[l_1 + 1, l_2]$, $[l_2 + 1, l_3]$, and $[l_3 + 1, L]$ are p_1, p_2, p_3, and p_4, respectively. The side lengths within a range are equally likely to occur. In our simulations, the number of jobs per run is 10,000, and the simulation runs are repeated enough times so that the relative error of measured turnaround times does not exceed 5%. Unless it is specified otherwise, the simulation results are for $L = 32$, $p_1 = 0.4$, $p_2 = 0.2$, $p_3 = 0.2$, $p_4 = 0.2$, $l_1 = 4$, $l_2 = 8$, and $l_3 = 16$. This uniform-decreasing distribution represents a case where most jobs are small relative to the size of the system.

5.2. Results for synthetic workloads

In Figs. 2–5, we show the average turnaround time and waiting delay results for the synthetic workload models. In the figures, the proposed policy is denoted as Window-K, and results for three window sizes ($K = 10$, $K = 120$, and $K = 240$) are displayed. These values are chosen as representatives of small, medium and large window sizes. Results obtained for other K values are presented and discussed later. It can be seen in Figs. 2 and 4 that OO produces the best average turnaround times. This is because it is aggressive in looking for a job that can be served, which reduces external processor fragmentation and improves overall system performance. However, the maximum waiting delays in OO can be very high under heavy loads when many jobs are small, which is the case in Fig. 3. This is because a large job at the head of the waiting queue can be bypassed many times when small jobs keep arriving. In Fig. 5, OO performs well because there are relatively fewer small jobs as job sizes are distributed uniformly in this Figure. As expected, FCFS saturates first as the load increases because a large job at the head of the waiting queue may block subsequent jobs until a large-enough free submesh becomes available for allocation to the head job. In the meantime, a large fraction of processors may remain idle. As a result, although FCFS is a

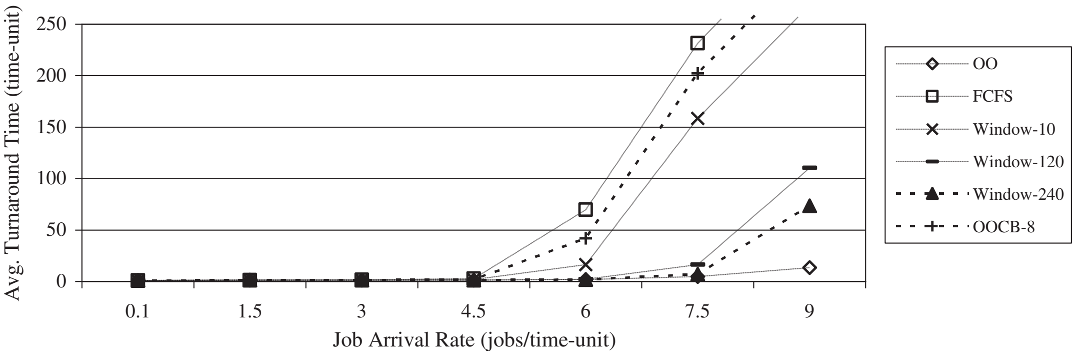

Fig. 2. Average turnaround time for the uniform-decreasing job size distribution in a 32 × 32 mesh.

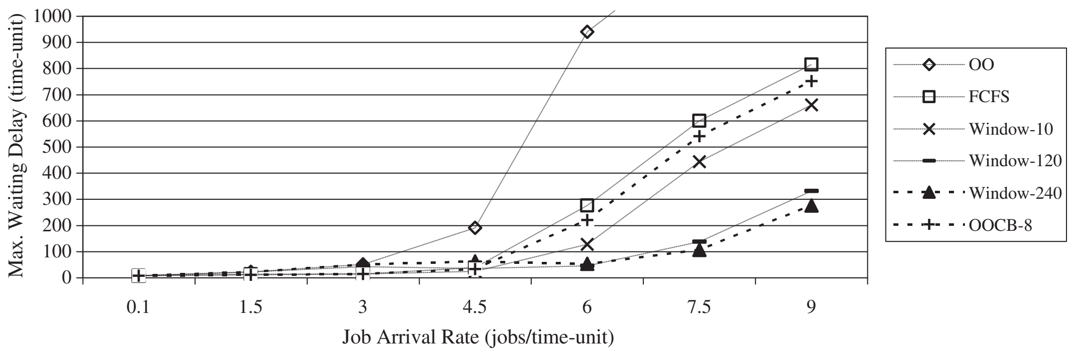

Fig. 3. Maximum waiting delay for the uniform-decreasing job size distribution in a 32 × 32 mesh.

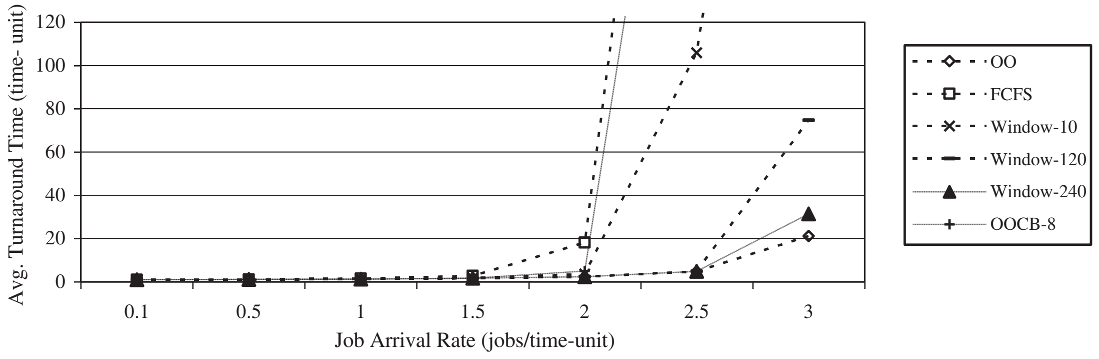

Fig. 4. Average turnaround time for the uniform job size distribution in a 32 × 32 mesh.

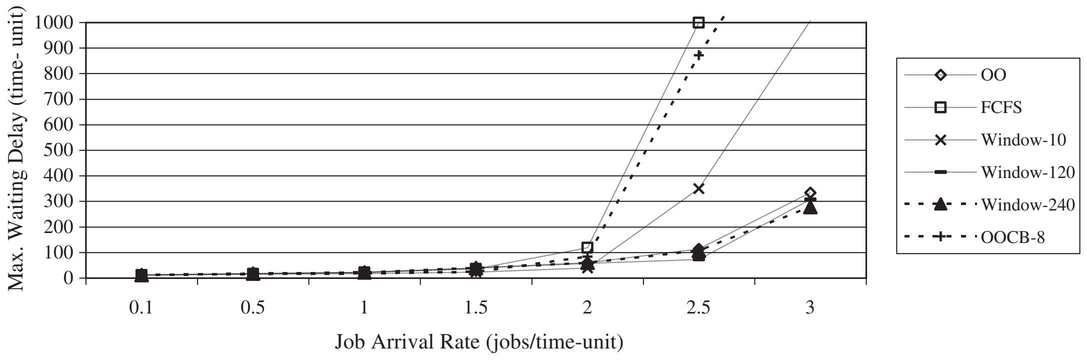

Fig. 5. Maximum waiting delay for the uniform job size distribution in a 32 × 32 mesh.

fair policy, it results in the longest waiting delays under moderately high loads (2 and 2. 5 jobs/time-unit) in Fig. 5. The scheme OOCB-8 does not perform well because a Max-Bound of eight is too small [3]. Its performance advantage over FCFS is moderate in all figures. Likewise, small K values lead to poor performance under high loads, although they are sufficient under moderate loads as the number of waiting jobs is small when the system load is moderate.

Considering both average turnaround times and maximum waiting delays, the proposed window-based job scheduling scheme achieves the best overall results across the synthetic workload models considered. This happens for $K = 240$. Window-240 produces average turnaround times that are second to those of OO, while it can achieve superior maximum waiting delays when many jobs are small, as can be seen in Fig. 3. That is, this scheme is the best at avoiding excessive waiting delays when a large fraction of jobs are small. This is a common case as can be seen in the workload traces used in this work and others [25]. Waiting delays are a major concern when the load is not light. They are small when the load is moderate, but they can be very long under heavy loads.

In addition to avoiding excessive waiting delays, Window-240 can achieve good system utilization. In Fig. 6, FCFS saturates for system utilization values of about 55%, whereas the system utilization of Window-240 reaches 78.38%. This is very

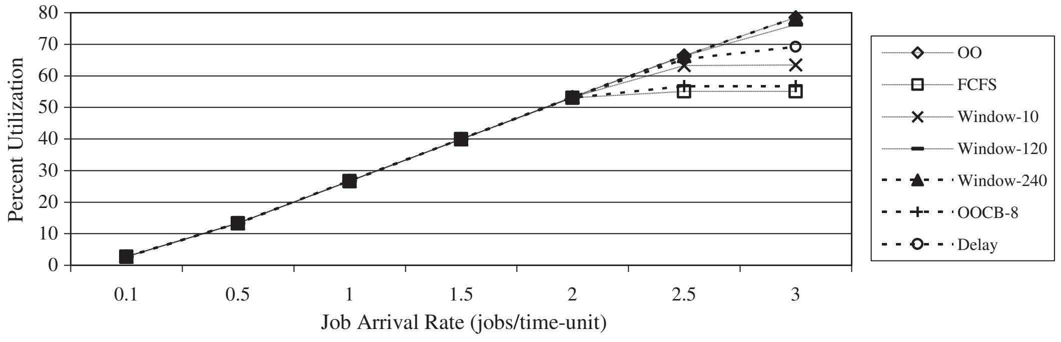

Fig. 6. Percent system utilization for the uniform job size distribution in a 32 × 32 mesh.

close to the maximum utilization value of 78.43% that OO achieves. The maximum utilization values achieved by OOCB-8 and the delay-based scheme (Delay) [20] are 56.7% and 69.2%, respectively. These values are substantially inferior to the 78.38% utilization that Window-240 can achieve. As stated earlier, the problem with OO is that it can result in very large maximum waiting delays when many jobs are small. In Fig. 7, the maximum utilization values achieved by Window-240, OO, Delay, OOCB-8 and FCFS are 72.5%, 73%, 69.1%, 52.7%, and 51%, respectively. The maximum system utilization of Window-240 is again very close to that of OO, yet Window-240 is substantially superior to OO in Fig. 3. The maximum utilization values achieved in Fig. 7 are smaller than those in Fig. 6 because many jobs are small when the job side lengths follow the uniform-decreasing distribution.

As *K* increases, the system utilization and average turnaround time of Window-*K* improve until they reach the performance of OO. The rate of improvement is high initially, and it becomes moderate as *K* increases further, as can be seen in Fig. 8. However, the maximum waiting delay initially decreases sharply, reaches a minimum, then it starts increasing slowly towards the maximum waiting delay of OO, as can be seen in Figs. 9–11. In these figures, we plot the maximum waiting de-

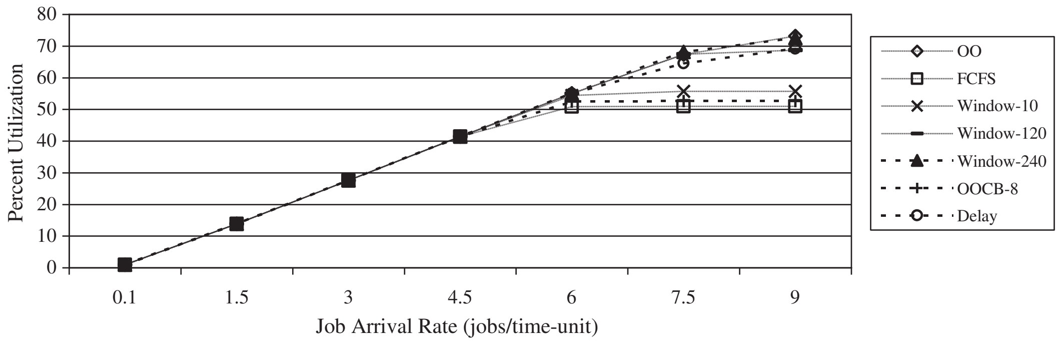

Fig. 7. Percent system utilization for the uniform-decreasing job size distribution in a 32 × 32 mesh.

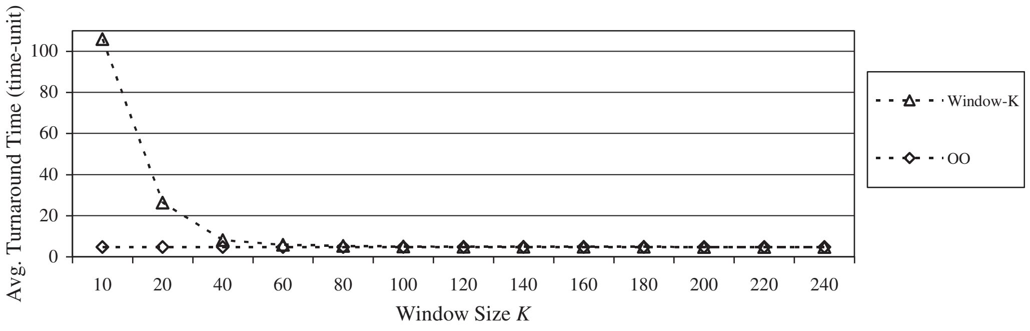

Fig. 8. Average turnaround as a function of window size in Window-K under the load of 2.5 jobs/time-unit and the uniform job size distribution in a 32 × 32 mesh; 200 simulation runs.

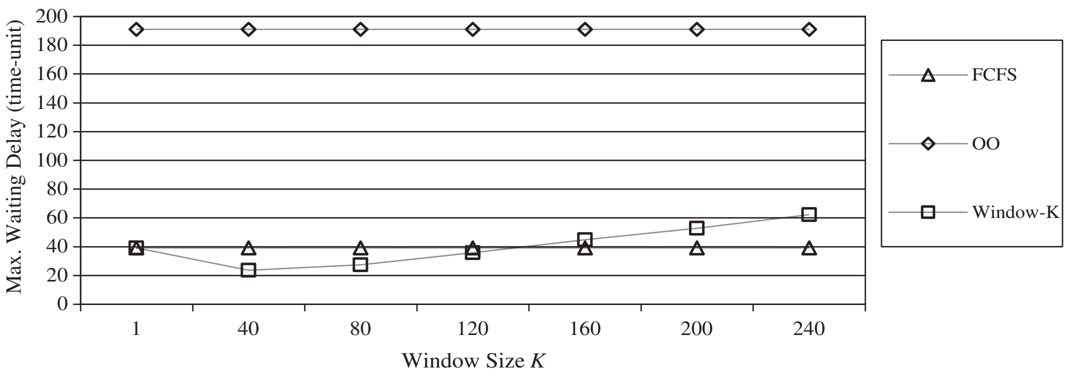

Fig. 9. Maximum waiting delay under the load of 4.5 jobs/time-unit and the uniform-decreasing job size distribution in a 32 × 32 mesh; 100 simulation runs.

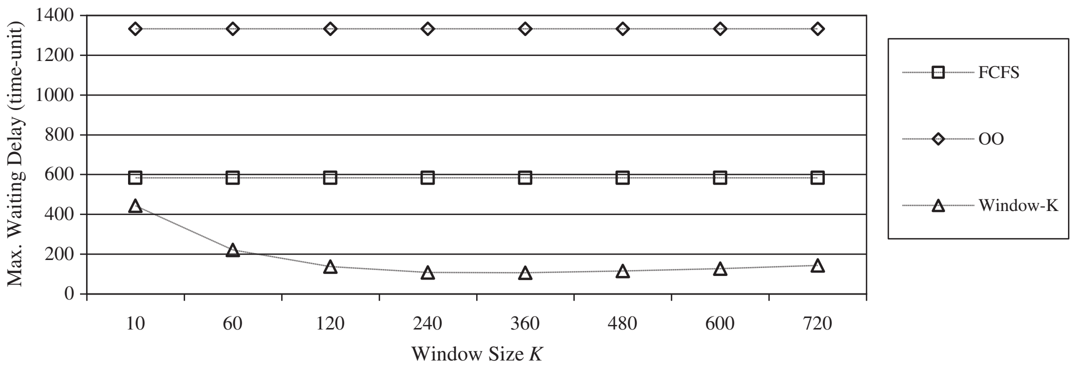

Fig. 10. Maximum waiting delay under the load of 7.5 jobs/time-unit and the uniform-decreasing job size distribution in a 32 × 32 mesh; 200 simulation runs.

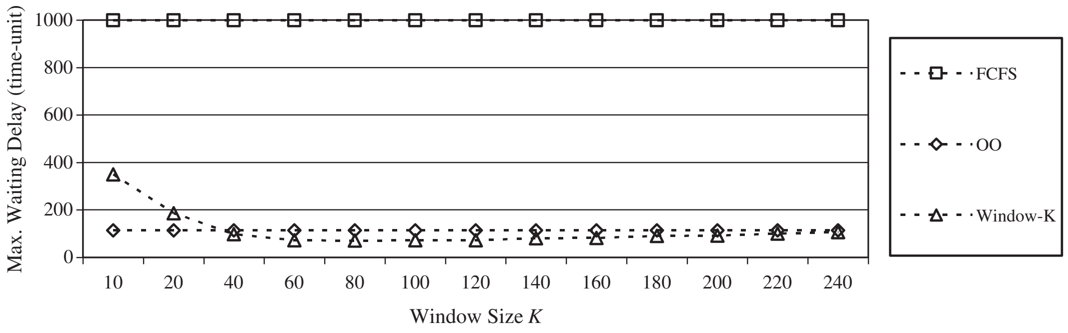

Fig. 11. Maximum waiting delay under the load of 2.5 jobs/time-unit and the uniform job size distribution in a 32 × 32 mesh; 200 simulation runs.

lay for different window sizes and selected system loads. It can be verified that the maximum waiting delay decreases initially as K increases, then it starts increasing slowly until it reaches OO values for larger values of K. These trends, when taken together, lead to choosing large scheduling window sizes. As a result of this observation and because waiting delays are short for all policies when the system is lightly loaded, it is recommended that a large lookahead window sizes be used by a system administrator who would use this scheme.

5.3. Performance impact of mesh system size

In this section, we investigate the effect of the size of the mesh system on the performance of the scheduling strategies. Mesh side lengths varying from $L = 16$ to $L = 96$ were considered. However, varying mesh side lengths did not change the relative performance of the scheduling strategies for both the uniform and uniform-decreasing job size distributions. As example, we show in Fig. 12 the mean turnaround times of the scheduling schemes against the mesh side length for the uniform job size distribution and a heavy system load of 2 jobs/time-unit.

5.4. Real workload traces

In addition to the synthetic workloads, we consider real workload traces based on data collected form several parallel computer systems. The traces are available on a public web site [25], where the raw trace data is converted to a standard format that includes relevant scheduling and allocation information, such as job submission times, run times, and number of allocated processors. In this paper, we drop job trace entries that have a negative run time or a negative number of allocated processors. Moreover, because we are interested in 2D meshes, we convert the number, n, of processors requested by a job into a width, w, and a height, h. The conversion is based on a square transformation, unless it is explicitly stated otherwise. In this transformation, we have $n = wh$, $w \leqslant W$, $h \leqslant H$, and $|w - h|$ is minimum. In choosing w and h, we have $w \leqslant h$. For example, in a 12 × 12 mesh a request for fifteen processors is transformed into a request for a 3 × 5 submesh. When n cannot be converted for the target mesh, it is changed to the smallest larger integer that can be transformed for the target. For example, 17 is changed to 18 (3 × 6) in a 12 × 12 system because $H < 17$.

In what follows, T_e denotes the average of the run times of jobs, J_{size} is the average of the number of processors they are allocated, and T_i is the average of their interarrival times. The workload traces we use in this paper were collected from the

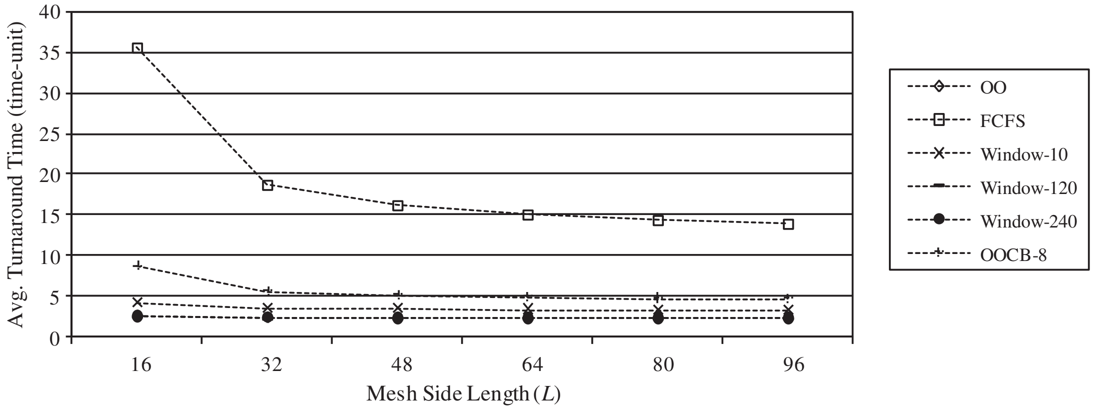

Fig. 12. Average turnaround time against mesh side length for the uniform job size distribution and a load of 2 jobs/time-unit.

systems below. The primary reason for choosing these traces is that they cover long periods of time on a parallel computer system.

- *SDSC Blue Horizon*: This San Diego Supercomputer Center IBM SP computer system consisted of 144 nodes. Each node was an 8-processor SMP. The workload trace used consists of 223,407 jobs and covers the period from April 25, 2000 to January 1, 2003. The jobs have the following characteristics: T_e = 4381.67 s, J_{size} = 5.212 nodes, and T_i = 378.64 s. In the simulation, we view the system as a 12 × 12 2D mesh-connected computer.
- *SDSC Intel Paragon*: This San Diego Supercomputer Center Intel Paragon 2D mesh-connected parallel computer included a batch partition of 352 processors that we view as a 22 × 16 mesh. In the simulations, we consider only trace jobs that

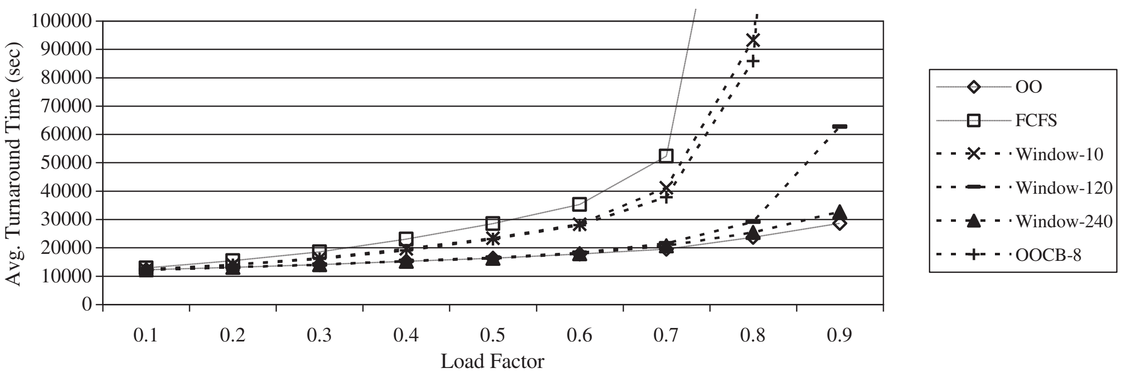

Fig. 13. Average turnaround time in seconds for the scheduling algorithms using the sdsc-95 trace workload.

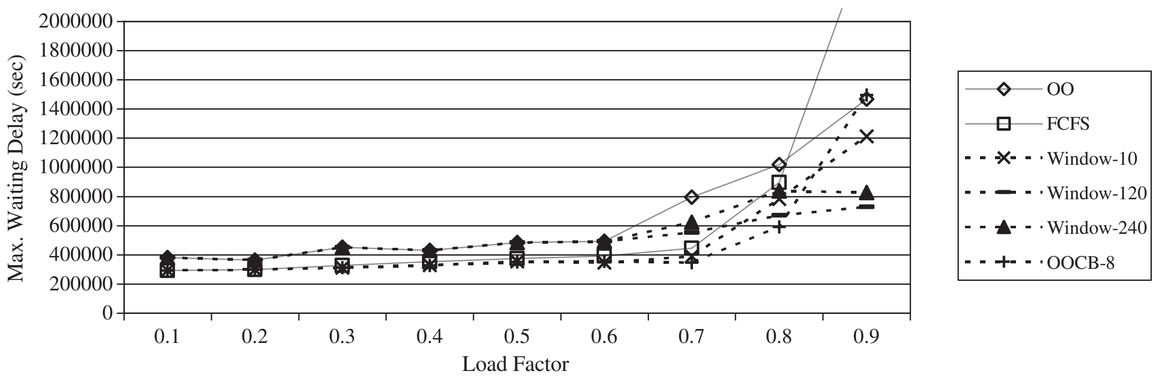

Fig. 14. Maximum waiting delay in seconds for the scheduling algorithms using the sdsc-95 trace workload.

were submitted to this partition. These traces are subdivided into two parts. One is for the year 1995 and the other is for 1996. A characteristic of these workload traces is that a large percentage of jobs have a power-of-two size, presumably because the Network Queuing System (NQS) scheduler which queues jobs according to power-of-two job sizes was used for this Intel Paragon partition. The 1995 trace consists of 19,403 jobs with the following characteristics: $T_e = 11354.8$ s, $J_{size} = 31.83$ processors, and $T_i = 1632.1$ s. The 1996 trace consists of 18,432 jobs that have the following characteristics: $T_e = 14792.5$ s, $J_{size} = 25.93$ processors, and $T_i = 1736.87$ s. As in [10], a job size n that is a multiples of 16, is transformed into a 2D submesh of the form $w \times 16$, where $n = 16w$.The square transformation with $w \geqslant h$ is used for the remaining job sizes.

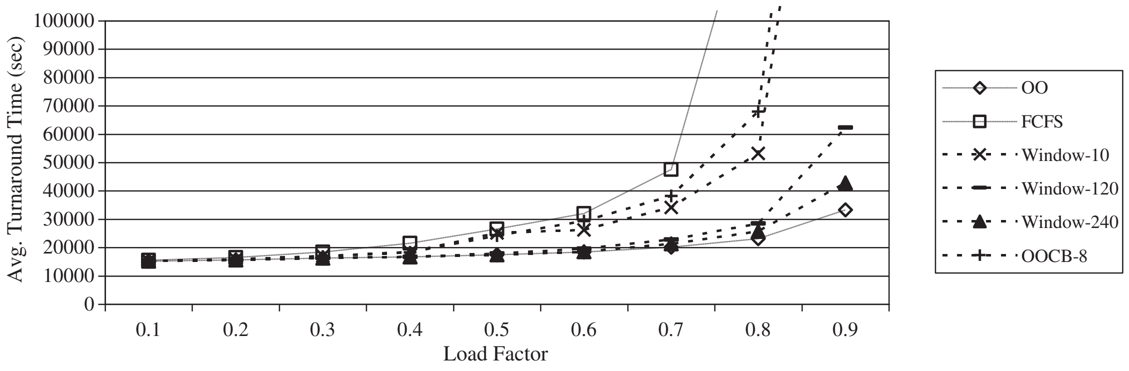

Fig. 15. Average turnaround time in seconds for the scheduling algorithms using the sdsc-96 trace workload.

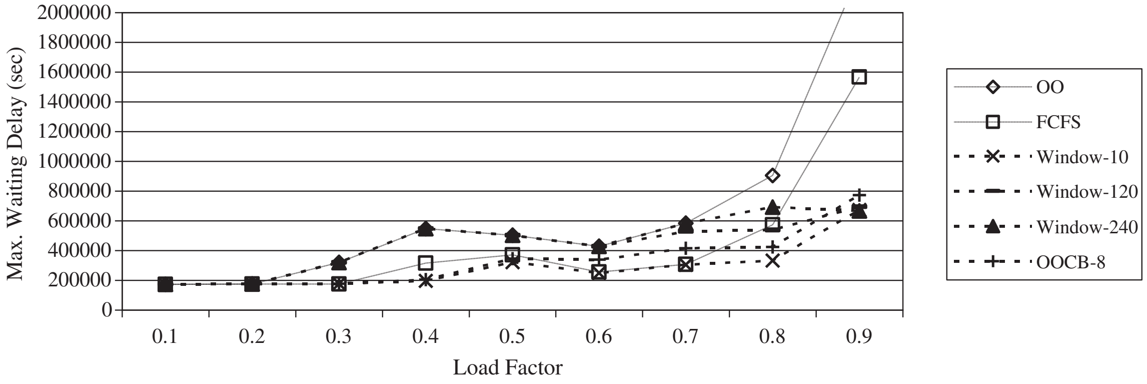

Fig. 16. Maximum waiting delay in seconds for the scheduling algorithms using the sdsc-96 trace workload.

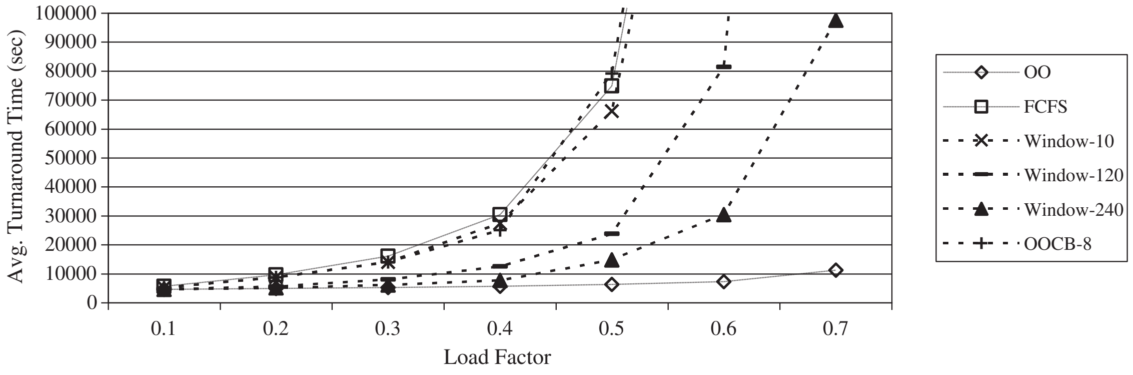

Fig. 17. Average turnaround time in seconds for the scheduling algorithms using the Blue Horizon trace workload.

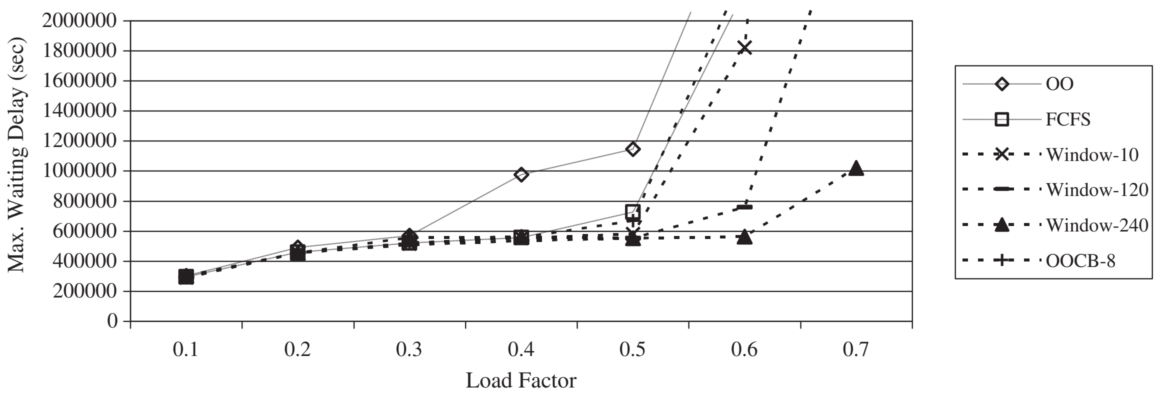

Fig. 18. Maximum waiting delay in seconds for the scheduling algorithms using the Blue Horizon trace workload.

5.5. Results for real workload traces

In Figs. 13–18, we show the results for the trace workloads. In these figures, we varied the interarrival times of jobs by multiplying the job arrival (submission) times included in the trace files by the inverse of a *load factor*. This factor is increased by 0.1 increments from a low value of 0.1 to a maximum value that is past the knee of the corresponding turnaround time curve. It can be seen in these figures that OO again produces the best average turnaround times because it carries out aggressive lookahead, which reduces processor fragmentation and job turnaround times. However, its maximum waiting delays are excessive under heavy loads for all trace workloads considered. A job can be overtaken many times by smaller subsequent jobs, which increases its waiting delay. Also, it can be seen in the figures that FCFS, OOCB-8 and Window-10 perform poorly in terms of average turnaround times. They lead to system saturation under smaller system loads. The proposed scheme Window-240 has the second best average turnaround time (and mean waiting delay) results. However, it has substantially better maximum waiting delays than OO. Considering both average turnaround times and maximum waiting delays, the proposed scheme has the best overall results across the trace and synthetic workloads considered when *K* is large.

6. Summary and conclusions

In this paper, we have proposed a window-based out-of-order job scheduling scheme for mesh-connected systems. In the scheme, denoted as Window-*K*, the oldest job waiting for allocation can be bypassed by a subsequent job provided the subsequent job is within a window of *K* consecutive jobs that start with the oldest waiting job. In particular, when all jobs within the window have been accommodated, out-of-order scheduling is suspended until a large-enough submesh becomes available for allocation to the oldest job, at which point the window is moved forward to the next oldest waiting job. Using extensive simulations, we have compared the proposed scheme with previous schemes, including the extreme schemes FCFS ($K = 1$) and greedy out-of-order scheduling (unbounded *K*). The results show that the proposed scheme can achieve good tradeoff between fairness, represented by maximum waiting delays, and overall system performance. It can produce good maximum waiting delays and average turnaround times under high system loads when *K* is large. Because waiting delays are relatively short for all policies when the system is lightly loaded, it is recommended that large lookahead window sizes be used by a system administrator who would use this scheme. In particular, the results in this paper show that $K = 240$ is a good choice that performs well across a wide range of job characteristics, represented by the synthetic and real workload traces considered in this paper.

This research paper has examined the effect of our new scheduling scheme on the performance of contiguous allocation. As a future continuation of this research, it would be interesting to study the effect of this scheme on noncontiguous allocation.

References

[1] I. Ababneh, Job scheduling and contiguous processor allocation for three-dimensional mesh multicomputers, AMSE Advances in Modelling & Analysis 6 (4) (2001) 43–58.
[2] I. Ababneh, Availability-based noncontiguous processor allocation policies for 2D mesh-connected multicomputers, Journal of Systems and Software 81 (7) (2008) 1081–1092.
[3] I. Ababneh, On submesh allocation for 2D mesh multicomputers using the free-list approach: global placement schemes, Performance Evaluation 66 (2) (2009) 105–120.
[4] Y. Aridor, T. Domany, O. Goldshmidt, J.E. Moreira, E. Shmueli, Resource allocation and utilization in the BlueGene/L supercomputer, IBM Journal of Research and Development 49 (2/3) (2005) 425–436.
[5] W.C. Athas, C.L. Seitz, Multicomputers: message-passing concurrent computers, IEEE Computer 21 (8) (1988) 9–24.

[6] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, A new processor allocation strategy with a high degree of contiguity in mesh-connected multicomputers, Journal of Simulation Modelling, Practice & Theory (SIMPRA), Elsevier Science 15 (4) (2007) 465–480.
[7] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, L. Mackenzie, Comparative evaluation of contiguous allocation strategies on 3D mesh multicomputers, Journal of Systems and Software 82 (2) (2009) 307–318.
[8] S. Bhattacharya, W.-T. Tsai, Lookahead processor allocation in mesh-connected massively parallel multicomputer, in: Proceedings of International Parallel Processing Symposium, 1994, pp. 868–875.
[9] M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burow, T. Takken, P. Vranas, Design and analysis of the BlueGene/L torus interconnection network, IBM Research Report RC23025, IBM Research Division, Thomas J. Watson Research Center, Dec. 3, 2003.
[10] D. P. Bunde, V. J. Leung, J. Mache, Communication patterns and allocation strategies, Sandia Technical Report SAND2003-4522, Jan. 2004.
[11] Cray, Cray XT3 Datasheet, 2004.
[12] C.-Y. Chang, P. Mohapatra, Performance improvement of allocation schemes for mesh-connected computers, Journal of Parallel and Distributed Computing 52 (1) (1998) 40–68.
[13] G.-M. Chiu, S.-K. Chen, An efficient submesh allocation scheme for two-dimensional meshes with little overhead, IEEE Transactions on Parallel and Distributed Systems 10 (5) (1999) 471–486.
[14] H. Choo, S.-M. Yoo, H. Youn, Processor scheduling and allocation for 3D torus multicomputer systems, IEEE Transactions on Parallel and Distributed Systems 11 (5) (2000) 475–484.
[15] D. Das Sharma, D.K. Pradhan, Submesh allocation in mesh multicomputers using busy-list: a best-fit approach with complete recognition capability, Journal of Parallel and Distributed Computing 36 (2) (1996) 106–118.
[16] D. Das Sharma, D.K. Pradhan, Job scheduling in multicomputers, IEEE Transactions on Parallel and Distributed Systems 9 (1) (1998) 57–70.
[17] D.G. Feitelson, Workload Modeling for Computer Systems Performance Evaluations, 2007. <http://www.cs.huji.ac.il/~feit/wlmod/wlmod.pdf>.
[18] A. Gara, M.A. Blumrich, D. Chen, L.-T. Chiu, P. Coteus, M.E. Giampapa, R.A. Haring, P. Heidelberger, D. Hoenicke, G.V. Kopcsay, T.A. Liebsch, M. Ohmacht, B.D. Steinmacher-Burow, T. Takken, P. Vranas, Overview of the Blue Gene/L system architecture, IBM Journal of Research and Development 49 (2/3) (2005) 195–212.
[19] Intel Corp., Paragon XP/S Product Overview, 1991.
[20] G. Kim, H. Yoon, On submesh allocation for mesh multicomputers: a best-fit allocation and a virtual submesh allocation for faulty meshes, IEEE Transactions on Parallel and Distributed Systems 9 (2) (1998) 175–185.
[21] E. Krevat, J.G. Castannos, J.E. Moreira, Job scheduling for the BlueGene/L system, in: Proceedings of the Job Scheduling Strategies for Parallel Processing Workshop (JSSPP), 2002, pp. 38–54.
[22] M. Levine, CRAY XT3 at the Pittsburgh Supercomputing Centre, DEISA Symposium, Bologna, 4–5 May 2006.
[23] V. Lo, K.J. Windisch, W. Liu, B. Nitzberg, Noncontiguous processor allocation algorithms for mesh-connected mutlicomputers, IEEE Transactions on Parallel and Distributed Systems 8 (7) (1997) 712–725.
[24] A.W. Mu'alem, D.G. Feitelson, Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling, IEEE Transactions on Parallel and Distributed Systems 12 (6) (2001) 529–543.
[25] Parallel Workloads Archive, 2008. <http://www.cs.huji.ac.il/labs/parallel/workload/>.
[26] ProcSimity v4.3 User's Manual, University of Oregon, 1996.
[27] S.-M. Yoo, H.Y. Youn, B. Shirazi, An efficient task allocation scheme for 2D mesh architectures, IEEE Transactions on Parallel and Distributed Systems 8 (9) (1997) 934–942.