Evaluating the EASY-Backfill Job Scheduling of Static Workloads on Clusters

Adam K.L. Wong and Andrzej M. Goscinski

School of Engineering and Information Technology
Deakin University
Geelong, Vic 3216, Australia
{aklwong, ang}@deakin.edu.au

Abstract— **This research aims at improving our understanding of backfilling job scheduling algorithms. The most frequently used algorithm, EASY-backfilling, was selected for a performance evaluation by scheduling static workloads of parallel jobs on a computer cluster. To achieve the aim, we have developed a batch job scheduler for Linux clusters, implemented several scheduling algorithms including ARCA and EASY-Backfilling, and carried out their performance evaluation by running well known MPI applications on a real cluster. Our performance evaluation carried out for EASY-Backfilling serves two purposes. First, the performance results obtained from our evaluation can be used to validate other researchers' results generated by simulation, and second, the methodology used in our evaluation has alleviated many problems existed in the simulations presented in the current literature.**

I. INTRODUCTION

When a computer cluster is used for running parallel jobs in a multi-users environment, an effective job scheduling mechanism is the key to provide a high quality of service (QoS) to the users and an efficient use of computing resources to the system owner. Space-sharing is the most common job scheduling approach used in loosely coupled parallel systems such as a computer cluster: each parallel job with multiple tasks receives a partition of the parallel system for its non-preemptive execution. Such scheduling activities are conducted by a cluster batch system which usually consists of two components: a resource manager and a job scheduler [7, 8]. While the resource manager works by managing jobs in the batch queue and tracking the availability of computing nodes, the scheduler is responsible for making decisions on how to prioritize the submitted jobs and on when, where and how to run the jobs based on the information maintained by the resource manager.

There exist some production cluster batch systems such as Moab/Maui [16], LoadLeveler [11], Load Sharing Facility (LSF) [20], Portable Batch System (PBS) [21] and Sun Grid Engine (SGE) [23]. The schedulers of these systems follow the space-sharing approach. They primarily adopt (i) the most commonly used queue-priority policy – first come first serve (FCFS) – to arrange jobs in the job queue; and, (ii) the basic computer-allocation policy – all requested computers available (ARCA) – to find enough idle computers in the cluster and map processes of a parallel application onto those computers. Such simple scheduling algorithms usually suffer from fragmentation of computer nodes and thus result in a poor utilization of the systems [5, 10]. Consequently, backfilling has also been included in those systems to improve their scheduling performance.

Backfilling is an optimized computer-allocation policy introduced by Lifka [14] in which small jobs are moved forward in the prioritized job queue to utilize the idle computers provided that no starvation of jobs from the job queue would occur. To achieve this, it relies on the estimates of how long each job in the queue will run. The estimates are used to find out when additional computers will become available, and to verify that any backfilled job will terminate in time so as not to delay any previously queued jobs.

Many simulation-based studies into the performance evaluation of backfilling algorithms have been presented in the literature [4, 12, 17, 22]. Although simulation is a flexible and cost-effective method for studying the performance of computer systems, its major drawback is that it is often perceived to be not fully trustworthy because the assumptions used for the evaluation reflect an incomplete and simplistic system. We believe that the validation of any simulation study is critical for useful performance analysis. Unfortunately, there is very little published experimental result for backfilling algorithms by measuring their performances directly on a real parallel system. We are therefore motivated to carry out a performance evaluation of the EASY-backfilling algorithm on a real computer cluster using real MPI parallel applications.

This research aims at improving our understanding of backfilling job scheduling algorithms. In particular, the most frequently used implementation: EASY-backfilling was selected for our performance evaluation when static workloads of parallel jobs are executed on a computer cluster. To achieve this aim, we have developed a batch job scheduler for Linux clusters, implemented several scheduling algorithms including the ARCA and the EASY-Backfilling, and carried out their performance evaluation by running well known MPI applications on a cluster.

The rest of this paper is organized as follows. Section 2 shows the related work and our response. Section 3 presents the background of backfilling. Section 4 covers the performance evaluation of the job scheduling experiments, including the experimental test-bed, workload construction and a brief description of cluster batch scheduler developed for Linux clusters. The results of the performance evaluation

and their discussion are presented in Section 5. Finally, Section 6 presents the conclusions and our future work.

II. Related Work and Our Response

Following our study into the current literature of the backfilling algorithms and their performance evaluation, we discovered that many of the results are unclear or incomplete and many important issues are unattended. In this section, the problems of the simulation-based evaluations of backfilling algorithms are first addressed. Our approach in evaluating the backfilling algorithms is then described.

A. Related Work

Our study of the related work shows that there are the following problems in the area of the simulation-based performance evaluation of the backfilling algorithms:

1) Defections in Simulation Result: The simulation model usually used in the performance studies of backfilling algorithms [4, 12, 17, 22] is based on the queuing theory [9]. However, the common problems found in those studies come from the sources of non-unified use in notations, over-simplifying in assumptions and ambiguity in experimental methodologies. For example, the loading of a system, which is directly proportional to jobs' inter-arrival rate but inversely proportional to jobs' service rate, has a significant impact on the scheduling performance. If the inter-arrival rate is smaller than the service rate in a system, the system is extremely under utilized and therefore no backfilling is needed. Unfortunately, most of the studies have not defined properly the loading of the simulated system on which the scheduling experiment was carried out but even worse the scale of loading is different from one study to another [17, 22]; and the Poisson[1] distribution of workload is 'blindly' used in most of those studies without a clear explanation. These factors add an extra difficulty in understanding and analysing the simulation results.

2) Static Workload vs. Dynamic Workload: Most of the simulation studies in the performance of parallel job scheduling have carried out experiments using a dynamic workload model for either real workload traces or synthetic workloads [15]. The major difference between a static workload and a dynamic workload is that the set of jobs is available at the beginning of the evaluation and no additional jobs arrive afterwards in the former whereas additional jobs arrive continuously during the evaluation in the latter. Jobs in a dynamic workload are modelled to arrival with bursts of different intensities depending on the distribution of their inter-arrival times. The arrival rate and the service rate of jobs have profound implications on the performance of any scheduler on a parallel system since they affect directly the number of jobs in the ready queue of the system. However, the arrival process of parallel jobs has received not enough attention by researchers which is indicated by the fact that the Poisson distribution model is assumed in all previous simulation studies. For example, many simulation studies use real workload traces obtained from some well known supercomputer centres. A workload trace normally recorded the details of jobs submitted to the supercomputer centre for execution over a long period of time (varies from 0 to 12 months). To perform the job scheduling simulation using a workload trace, one must set the loading on the simulated system by adjusting jobs' inter-arrival time of the workload trace. Although different performance evaluations can be repeated at different system loads set for a simulation study, such performance result cannot quantify precisely the relationship between the number of jobs in ready queue and a value of system load. Consequently, those performance studies using a dynamic workload can only show a 'macroscopic' overall performance of a scheduling system and the results convey an incomplete knowledge about the impact of job burstiness that occurs at different times throughout a workload evaluation on the performance.

3) Lack of Experimental Results for Real Systems: The majority of the experimental studies in parallel job scheduling were carried out by simulation. Although these studies have contributed invaluable knowledge to this research area, its major drawback is that it is often perceived to be untrustworthy and incomplete because of the lack of a full system implementation when evaluated. When a job scheduling experiment is conducted by discrete event simulation that does not employ a detailed architecture simulator for running parallel applications, no actual execution of real parallel applications is performed; and the assumptions made neglect the application-specific attributes such as granularity of parallelism, communication pattern, memory model as well as network bandwidth. These attributes however directly affect the execution behaviour of parallel programs and therefore have significant impact on the scheduling evaluation. The existing evaluation on parallel job scheduling done in a real environment [1, 2, 6, 14] is very limited qualitatively and quantitatively as compare to those done in simulation [4, 12, 17, 22].

4) Estimated Execution Time: The fundamental mechanism of backfilling relies on the estimates of jobs' runtime which are used for making a schedule such that some jobs, even though they are at the back of the queue, can be run earlier as long as they will terminate on time without delaying other jobs which are ahead in the queue. However, it is well documented [3] that user estimates are rarely accurate. Although users generally believe that a tight estimation (a relatively smaller requested time) could let their programs to run earlier because of a higher chance in backfilling, they still tend to over-estimate to make sure that their jobs will not be killed. The extent of users' over-estimations can be as large as 300 times of a program's actual runtime which is evidenced by the workload traces from the most popular supercomputing centres [19]. Whether accurate estimates would improve the performance when backfilling is used has been attended by researchers. On one hand, some simulation studies have

[1] The Poisson model uses an exponentially distributed inter-arrival time for jobs to a system over a given time interval.

shown that more accurate estimates have only minimally affected the average slowdown and inaccurate estimates even can produce a slightly better result [17]. On the other hand, other studies have shown that users who provide more accurate estimates can expect improved performance, even if other jobs do not provide more accurate estimates [3]. Nevertheless, we believe that this research issue is still far from being fully explored.

B. Our Approach

We have carried out an evaluation for the EASY-backfilling algorithm on a real system in order to compare and validate other similar simulation-based evaluations. Our approach that distinguishes this work from previously presented works is as follows:

1) Although some results from running simulation experiments on real systems have been published, the scope of the studies is usually small and specific. We carried out a detailed evaluation on EASY-backfilling by scheduling MPI parallel applications on a real cluster and our scope of study is comparable to that presented in most simulation studies.

2) A static workload can be considered as a snapshot of a dynamic workload. Under a static workload, a scheduler can be evaluated by how well it handles a given job mix. By repeatedly evaluating the scheduler under different static workloads (we used a workload size of 10, 50 and 100 jobs), we have captured the behaviour of the scheduler as if it is under different job bursts of a dynamic workload. This issue is difficult to explore in an evaluation using a dynamic workload. By using static workloads in our scheduling experiments, we have studied the influence of the number of job in the waiting queue of parallel jobs.

3) Many studies have found that the relative effectiveness of different scheduling strategies actually depends very much on the job mix of one's evaluation. The major factors that affect the scheduling of a job are the length – execution time of a job, width – number of nodes requested by a job, and the users' estimation of the job length. Srinivasan et al. has classified a few job mixes of the real workload traces[2] based on the length and width of parallel programs into different categories and interpreted the scheduling performance for those categories [22]. The problem of their evaluation lies in the fact that the different workload traces were made up of completely different programs where program characteristics such as the communication and memory requirements of jobs and the inter-arrival time of jobs are difference. Therefore, the result cannot conclusively demonstrate the implication of different workload compositions on the job scheduling performance. Instead, we constructed a set of static workloads with different compositions out of a confined set of programs from the NAS parallel benchmark suite [18].

4) We believe that the issue of more accurate estimates in backfilling is still far from fully explored. Previous studies [3, 17, 22] have conducted evaluations concentrating only on the impact of a change in the magnitude of users' estimates on the scheduling performance. However, the major factors that affect the scheduling of a job are the length, the width and users' estimates. Different compositions of a workload might react differently to a change in the magnitude of users' estimates. For example, jobs with long execution time (or at least estimated execution time) in a workload can have a very large over-estimation that creates a large hole for backfilling smaller jobs. In our evaluation, we study the impact of a change in the magnitude of users' estimates on the scheduling performance under different workload compositions.

III. BACKFILLING ALGORITHMS

A parallel job scheduling algorithm for clusters usually works in two phases. In the first phase, a queue-priority policy is used to prioritize the submitted jobs in a batch queue. In the second phase, a computer-allocation policy is then employed to decide when the jobs are scheduled and how the computers are allocated. We demonstrate with the following scheduling examples the mechanisms of the ARCA and backfilling computer-allocation policies. Static space sharing is assumed which means once the process-to-processor mapping is done this mapping remains the same for the entire duration of a program's execution. A cluster of six computers is used to run the following parallel jobs: J_1, J_2, J_3, J_4 and J_5. Table 1 lists the attributes of each job.

TABLE 1
ATTRIBUTES OF THE PARALLEL JOBS IN A FCFS QUEUE

Job ID	Order in Queue	Number of Computers Requested	Execution Time
J_1	1	3	4
J_2	2	5	5
J_3	3	2	3
J_4	4	6	2
J_5	5	1	12

A. The ARCA Policy

ARCA is the simplest computer-allocation policy used for job scheduling on clusters. Under this policy, if there are enough computers to run a job (i.e., the number of computers requested <= the number of available idle computers in the cluster), the computers are allocated and the job is started. However, if there are not enough computers available, the job must wait until some computers are released. As illustrated in Fig.1, J_2 must wait until J_1 has finished, and consequently J_3 must wait until J_2 has finished and so on. This policy for parallel job scheduling usually produces a low utilization rate of the system and a poor program response time to the users especially when the job queue and the size of jobs are large.

[2] Real workload traces obtained from different supercomputer centres are available from Feitelson's archive [19].

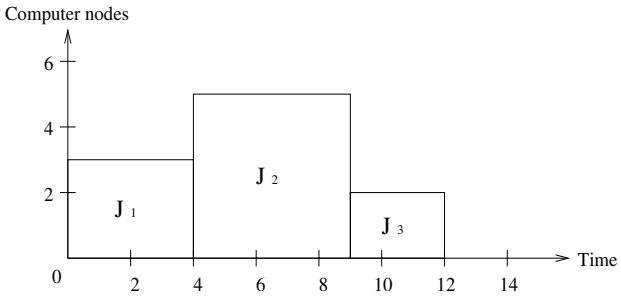
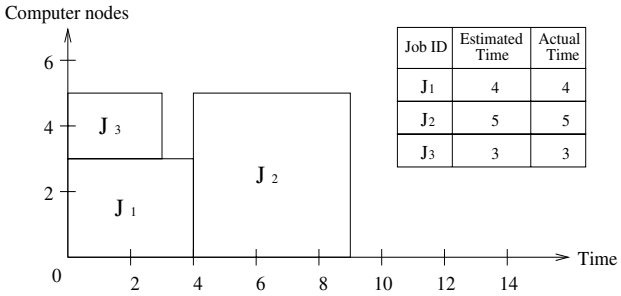


Fig. 1 Jobs follow a first come first serve order and the ARCA policy

B. The Backfilling Policy

Although some performance evaluations [10, 12] on parallel job scheduling have shown that the utilization of a parallel system can be improved up to 15% by switching from the basic ARCA policy to backfilling policy, a major drawback of backfilling is that the programs' execution time must be known or otherwise estimations must be made. If an estimation of a program's execution time must be made, its accuracy will affect the scheduling performance of the system [3, 17]. An over estimated value can lead to poor performance as backfilling of jobs may not be applied at all but an under estimated value can lead to a forced termination of jobs as it is treated in the simulation studies discussed earlier. For cluster and grid computing, it is a common situation that parallel programs may be developed in one environment but execute on another; and different executions of the same program may use different number of computers. It is a very tight restriction to know the execution time of parallel jobs in order to achieve good performance.

The backfilling policy uses the estimated execution time of programs to figure out when additional computers will become available and thus to construct a schedule that aims at optimizing the utilization of a cluster. Under this policy, a job can move forward in the prioritized batch queue to utilize idle computers if it can be fitted and its execution will not prevent the execution of other previously queued jobs. Fig. 2a shows that J_2 must wait until J_1 has finished in order to obtain enough idle computers for its execution but J_3 can be started at time 0 jumping over J_2 which is started at time 4. As a result, both the response time of J_3 and the system utilization have improved.

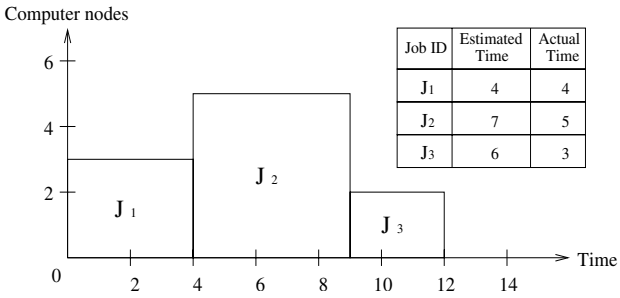


Job ID	Estimated Time	Actual Time
J_1	4	4
J_2	5	5
J_3	3	3

Fig. 2a. Backfilling with a perfect-estimation of programs' execution time

Although the backfilling policy was proposed to alleviate the problem of the ARCA policy, it has a major limitation, as we stated earlier that its performance depends very much on the accuracy of the estimated programs' execution time. We demonstrate its limitation with the following two scenarios: over-estimation and under-estimation of the programs' execution time.

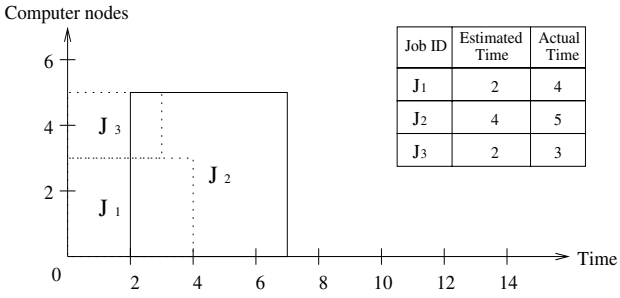
In Fig. 2b, although there are enough idle computers for J_3 to start at time 0, it won't be started since it has an estimated execution time of 6. If it has been started at time 0, it is expected to still be running at time 4 when J_1 has finished and will delay the starting of J_2. Thus, an over-estimation of programs' execution time can prohibit an optimization which would be made if the actual programs' execution time were known.



Job ID	Estimated Time	Actual Time
J_1	4	4
J_2	7	5
J_3	6	3

Fig. 2b Backfilling with an over-estimation of programs' execution time

In Fig. 2c, both J_1 and J_3 are 'blindly' started at time 0 based on their estimated programs' execution time. When J_2 is scheduled to run at time 2, both J_1 and J_3 are actually still running. A simple solution to this situation is to kill J_1 and J_3 immediately and they have to be re-started at a later time. Although a better solution would be to pre-empt the job and re-schedule it at a later time, it could be very costly to do so if the size of the job is large that has scattered on a large number of computer nodes. Thus, under-estimation of programs' execution time in backfilling may cause forced-termination of programs.



Job ID	Estimated Time	Actual Time
J_1	2	4
J_2	4	5
J_3	2	3

Fig. 2c Backfilling with an under-estimation of programs' execution time

C. Conservative VS. EASY Backfilling

There are two major backfilling strategies: conservative and aggressive (EASY). Conservative backfilling moves jobs forward only if they do not delay any previously queued jobs. EASY backfilling takes a more aggressive approach and allows jobs to skip ahead provided that they do not delay just the job at the head of the queue.

In Fig. 3, J_3 is backfilled and it can fulfil the definition of both the conservative and EASY backfilling. However, J_5 is backfilled but it can only fulfil the definition of EASY

backfilling. As shown in the figure, after J_1 and J_3 are scheduled, J_2 is in the head of the waiting queue, although the backfilling of J_5 does not delay J_2, it does delay J_4 and forces it to start later at time 12 rather than at time 9.

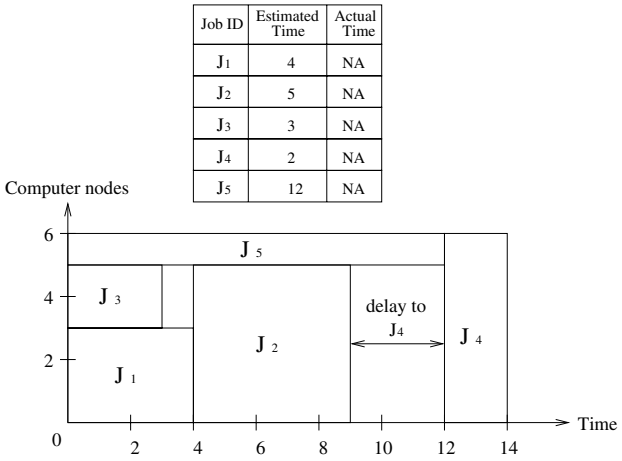


Fig. 3. Conservative VS. EASY Backfilling

There are of course many pros and cons for these two backfilling strategies. For example, conservative backfilling has a higher capability than EASY backfilling in predicting when each job will run and giving users execution guarantees. On the other hand, EASY backfilling provides better opportunities for jobs to be backfilled especially if the number of jobs in the waiting queue is large. There is no consensus on which of these two backfilling strategies is better. In fact, previous studies [4, 17, 22] have shown that the relative performance of the two backfilling strategies is workload and metric dependent and that no final conclusion could be reached.

IV. Performance Evaluation

The performance of a computing system is assessed differently by users and a system owner. Users usually care when their jobs can be run and how long it takes to have a solution. The owner or administrator concerns more on the utilization and efficiency of the cluster. We therefore used both user- and system-centric metrics to evaluate the performance of our cluster batch scheduler. The user-centric performance metrics selected are job waiting time, job execution time, job response time and job slowdown, whereas the system-centric performance metrics selected are makespan and throughput.

For the purpose of this paper, the performance metrics used are defined as follows:

1. Waiting Time – The amount of time that a job has to wait in the batch queue until it is being scheduled. To simplify the measurement of our experiments, we assume, for each scheduling experiment, the first job being scheduled has a waiting time of zero.
2. Execution Time – The amount of time that a job takes to be executed once it is scheduled.
3. Response Time – The amount of time that a job takes to complete its execution on the cluster. It is the summation of the waiting time spent in the batch queue and the execution time of the job.
4. Slowdown – The slowdown is defined as the response time normalized by the execution time: Response Time / Execution Time. It shows how much a job has been delayed.
5. Makespan – The amount of time that it takes from the first job to the last job in a workload to be finished on the cluster.
6. Throughput – The number of jobs completed on the cluster per unit of time.

A. Experimental Testbed

The hardware considered in this work is a cluster of 16 Pentium Class computers, each with 383 Mbytes of main memory. The computers are connected together by a 100Mbit/s Fast Ethernet network. All of the computer nodes are running the Red Hat Linux operating system. Also, LAM/MPI [13] is used to support the development and execution of MPI parallel applications on the cluster.

The parallel applications used to represent user batch jobs come from the NAS Parallel Benchmarks (NPB). The NPB suite has been widely used to measure and compare the performance of parallel computer systems. It is a set of eight programs, which were derived from computational fluid dynamics codes. In particular, the NAS Parallel Benchmarks 2.4 [24] can be used as MPI applications and run on parallel computers such as computer clusters.

TABLE 2
CLASSIFICATION ON ATTRIBUTES OF THE SELECTED NAS PROGRAMS (EXTRACTED FROM [25])

Program	Attributes		
	Computation	Communication	
		Volume	Pattern
EP	Computation-Bound	Negligible	Point-to-point
LU	Computation-Bound	Medium	Point-to-point
MG	Communication-Bound	High	Collective

We have found that out of the eight NAS programs, EP (Embarrassingly Parallel), LU (LU solver) and MG (MultiGrid) can represent three distinct ranges of parallel applications with program attributes that can commonly be found in the real world. Thus, these three programs are selected for our experiments. The attributes of the three selected NAS parallel programs are summarized in Table 2.

B. Workload Construction

All of the selected NAS parallel programs were compiled for the Class A size defined in the NPB suite with specific number of computers (n) to be used. Table 3 shows the execution time of EP, LU and MG measured in our experimental test-bed using various numbers of computers.

TABLE 3
EXECUTION TIMES OF SELECTED NAS PARALLEL PROGRAMS FOR DIFFERENT NUMBER OF COMPUTERS

Program	Execution Time using different number of computers, n (s)			
	n = 2	n = 4	n = 8	n = 16
EP	279	141	71	36
LU	1082	543	273	141
MG	80	46	22	12

Based on the number of computers used, jobs are classified as narrow (for n = 2 and 4) and wide (for n = 8 and 16). Table 4 shows a categorization of different workload composition types based on the ratio of wide job to narrow job in a workload.

TABLE 4
A CATEGORIZATION OF WORKLOAD COMPOSITION TYPES

Composition Type	Job Ratio – Narrow : Wide
Even-Distribute	1:1
Wide-Dominate	3:7
Narrow-Dominate	7:3

A static workload for our cluster batch scheduler is defined as the total number of parallel jobs arrived at the batch queue within a fixed period of time. We first defined three different static workload sizes based on the number of jobs: Small workload of 10 jobs, Medium workload of 50 jobs and Large workload of 100 jobs. We then further defined three different workload composition types based on the width characteristic of the jobs in the workload: Even-Distributed, Wide-Dominated and Narrow-Dominated. A total of nine combinations of the workload size and the workload composition type are presented in Table 5. Each of these workload combinations was generated by a random mix of the selected NAS parallel programs (EP, LU and MG) and ten instances were generated for each of the workload types.

TABLE 5
A CLASSIFICATION OF THE SIMULATED WORKLOADS

Workload Size	Workload Composition Type		
	Even-Distributed	Wide-Dominated	Narrow-Dominated
Small	Small & Even-Distributed	Small & Wide-Dominated	Small & Narrow-Dominated
Medium	Medium & Even-Distributed	Medium & Wide-Dominated	Medium & Narrow-Dominated
Large	Large & Even-Distributed	Large & Wide-Dominated	Large & Narrow-Dominated

C. The Cluster Batch Scheduler

The cluster batch scheduler has been developed for Linux clusters[3] and for the scheduling of MPI parallel programs. Its implementation framework is designed to provide a high flexibility to administrators by allowing them to choose different queue-priority policies and computer-allocation policies for scheduling workloads of different characteristics. Such framework can also enhance the scheduler by allowing new policies to be introduced. For the scope of the performance evaluation presented in this paper, we have implemented the FCFS queue-priority policy; and the ARCA and EASY-backfilling computer-allocation policies.

The development of the cluster batch scheduler is made based on the following assumptions:

1. Only batch parallel jobs are considered in this scheduler[4].
2. Parallel jobs are submitted by users and they are stored in a batch queue of the cluster batch system.
3. A parallel job is represented by an MPI configuration file which specifies information such as the name and location of an MPI program, the number of computers required and the process-to-processor mapping.
4. A scheduling decision is made when the scheduler is first started and at the time whenever a parallel job has finished its execution.
5. The queue-priority policy and the computer-allocation policy of the scheduler are set by the administrator of the cluster.

The cluster batch scheduler has two major components: the Master and the Workers. The Master process is the main driving module of the scheduler that provides both the queue-priority and computer-allocation policies for making a scheduling decision. A set of Worker processes is created dynamically (one Worker per job) by the Master process for invoking and monitoring the execution of parallel jobs. These processes are synchronized and their accesses to shared data structures are controlled by semaphores and other synchronization tools of the Linux OS.

D. Experiments

We used our cluster batch scheduler to evaluate the performance of the two computer-allocation policies: ARCA and EASY-Backfilling as described in Section 3. Each of them was tested with the FCFS queue-priority policy. Although it is well known that any backfilling computer-allocation job scheduling policy is far more superior then the basic ARCA policy, none of previous literatures has presented a detailed evaluation and comparison on them. The most relevant result comparing the basic ARCA and backfilling policies is listed in [10]. It describes batch job scheduling and utilization of the supercomputers in NASA; and shows that the utilization of the supercomputers can be improved up to 15% by switching from the basic ARCA policy to backfilling policy in most of the commercial cluster batch schedulers.

The objective of our scheduling experiments is two fold. First, we would like to find out how much better is the EASY-Backfilling than the ARCA? Second, we would like to study the influence of static workload, workload composition and

[3] A port of the current implementation to the Java Virtual Machine is being developed.

[4] It can be extended to an on-line mode parallel job scheduler. Its development is being carried out.

accuracy of users' estimates on the performance of the EASY-backfilling scheduling strategy on a computer cluster.

All of the ten instances created for each of the nine workloads listed in Table 5 were scheduled by our cluster batch scheduler with the computer-allocation policies: ARCA and EASY-backfilling. We measured the job waiting time, job execution time, job response time, slowdown, makespan and throughput for each of the scheduling experiments.

For those experiments studying the influence of the user estimates on the scheduling performance of the EASY-Backfilling, we first used the actual programs' execution times recorded in Table 3 and treated them as the perfect estimates, Tp. We then created imperfect estimates, Te = (1 + k)Tp for k = 0.5, 1, 2 and 5, for the programs to represent a workload with an over estimation in the execution time in a random range of 0 to 50%, 0 to 100%, 0 to 200% and 0 to 500%.

V. Results and Analysis

Due to space limitations, we only show the first instance of scheduling traces of Workload: Small & Even-Distributed for computer-allocation policies: ARCA and EASY-backfilling (with perfect estimation) in Table 6 and 7, respectively. Each of the jobs in the workload instance is denoted by a program name and the number of computers requested separated by the '.' character. The jobs stored in the batch queue using FCFS queue-priority policy are as follows:

Workload: Small & Even-Distributed = {MG.8, LU.2, MG.4, MG.4, EP.16, MG.2, LU.4, LU.16, MG.8, MG.4}

Each entry (row) of Table 6 and 7 stores the measured performance metrics of a job upon its termination. The mean values of the waiting time, execution time, response time and slowdown of the 10 jobs, the makespan and the system throughput are also calculated for each workload instance.

Tables: 8a, 8b and 8c show the slowdown and throughput of averaging the measurements of all instances of each of the different workload types for the ARCA and EASY-backfilling computer-allocation policies.

TABLE 6
PERFORMANCE OF THE ARCA COMPUTER-ALLOCATION POLICY

Job	Requested Computers	Waiting Time (s)	Execution Time (s)	Response Time (s)	Slowdown
MG	8	0	22	22	1.00
MG	4	0	45	45	1.00
MG	4	22	45	67	1.49
LU	2	0	1066	1066	1.00
EP	16	1066	37	1103	29.81
MG	2	1103	79	1182	14.96
LU	4	1103	529	1632	3.09
LU	16	1632	144	1776	12.33
MG	8	1776	22	1798	81.73
MG	4	1776	45	1821	40.47
Mean Value		847.8	203.4	1051.2	18.69
Makespan = 1821s			Throughput = 0.33job/min.		

TABLE 7
PERFORMANCE OF THE EASY-BACKFILLING (PERFECT ESTIMATION) COMPUTER-ALLOCATION POLICY

Job	Requested Computers	Waiting Time (s)	Execution Time (s)	Response Time (s)	Slowdown
MG	8	0	21	21	1.00
MG	4	0	45	45	1.00
MG	4	21	45	66	1.47
MG	2	0	80	80	1.00
MG	4	45	45	90	2.00
MG	8	90	21	111	5.29
LU	4	21	530	551	1.04
LU	2	0	1065	1065	1.00
EP	16	1065	37	1102	29.78
LU	16	1102	146	1248	8.55
Mean Value		234.4	203.5	437.9	5.21
Makespan = 1248s			Throughput = 0.48 job/min.		

TABLE 8A
PERFORMANCE OF ARCA AND EASY-BACKFILLING IN THE WORKLOAD SIZE OF 10

Performance Metric	Workload Composition	Computer Allocation Policy					
		ARCA	EASY-backing				
			Perfect	Over Estimation			
				0 – 50%	0 – 100%	0 – 200%	0 – 500%
Slowdown	Even-Distributed	15.72	10.13	11.17	10.20	10.28	10.65
	Wide-Dominated	15.43	11.49	11.36	11.57	11.29	11.50
	Narrow-Dominated	7.08	5.25	5.35	5.26	5.33	5.40
Throughput (job/min)	Even-Distributed	0.48	0.56	0.53	0.56	0.56	0.53
	Wide-Dominated	0.65	0.76	0.75	0.76	0.75	0.76
	Narrow-Dominated	0.43	0.51	0.47	0.47	0.44	0.50

TABLE 8B
PERFORMANCE OF ARCA AND EASY-BACKFILLING IN THE WORKLOAD SIZE OF 50

Performance Metric	Workload Composition	Computer Allocation Policy					
		ARCA	EASY-backing				
			Perfect	Over Estimation			
				0 – 50%	0 – 100%	0 – 200%	0 – 500%
Slowdown	Even-Distributed	54.70	26.24	30.22	31.48	29.70	33.25
	Wide-Dominated	76.03	37.84	39.29	40.49	42.85	45.50
	Narrow-Dominated	39.71	19.37	20.53	19.77	21.29	22.31
Throughput (job/min)	Even-Distributed	0.46	0.77	0.72	0.68	0.68	0.70
	Wide-Dominated	0.52	0.91	0.82	0.81	0.82	0.82
	Narrow-Dominated	0.48	0.82	0.69	0.76	0.74	0.76

TABLE 8C
PERFORMANCE OF ARCA AND EASY-BACKFILLING IN THE WORKLOAD SIZE OF 100

Performance Metric	Workload Composition	Computer Allocation Policy					
		ARCA	EASY-backing				
			Perfect	Over Estimation			
				0 – 50%	0 – 100%	0 – 200%	0 – 500%
Slowdown	Even-Distributed	114.55	51.34	52.01	53.68	59.45	63.78
	Wide-Dominated	148.93	74.78	76.38	76.32	81.88	86.33
	Narrow-Dominated	89.26	39.90	39.38	42.23	50.16	49.06
Throughput (job/min)	Even-Distributed	0.48	0.89	0.82	0.88	0.80	0.82
	Wide-Dominated	0.51	0.89	0.85	0.83	0.85	0.82
	Narrow-Dominated	0.52	0.93	0.90	0.86	0.86	0.86

In order to analyse the performance of the EASY-backfilling policy, the values of the slowdown and throughput obtained for each of the workloads above were normalized. This was done by calculating the relative improvement of both the slowdown and throughput for EASY-backfilling (also for all variants of different estimation ranges) with respect to the slowdown and throughput obtained for the ARCA policy. These results are shown in Fig. 4(a and b), Fig. 5(a and b) and Fig. 6(a and b).

As shown in Tables 8a, 8b and 8c, for each of the workloads as listed in Table 4, EASY-backfilling (and the variant) outperforms ARCA in both the slowdown and throughput; and the improvement increases as the workload increases from 10, 50 to 100. This observation is also true for all cases of workload composition types: Even-Distributed, Wide-Dominated and Narrow-Dominated. A possible explanation to this observation is that a larger static workload size means a larger number of jobs are in the ready queue and therefore a high chance (more choices) for any job to be backfilled.

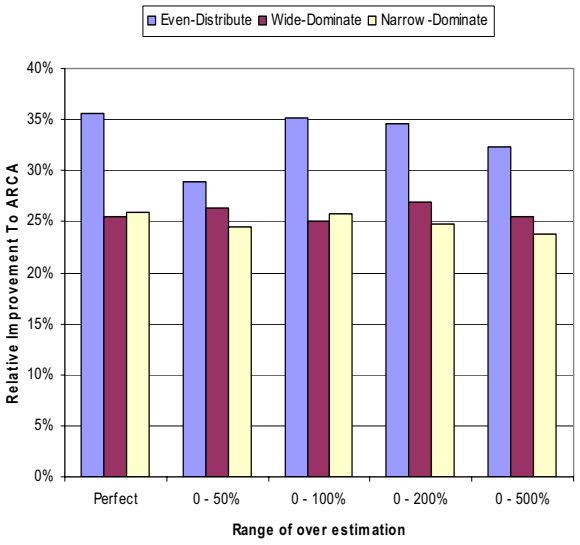


Fig. 4a. Improvement in Slowdown of EASY-Backfilling (Workload Size of 10)

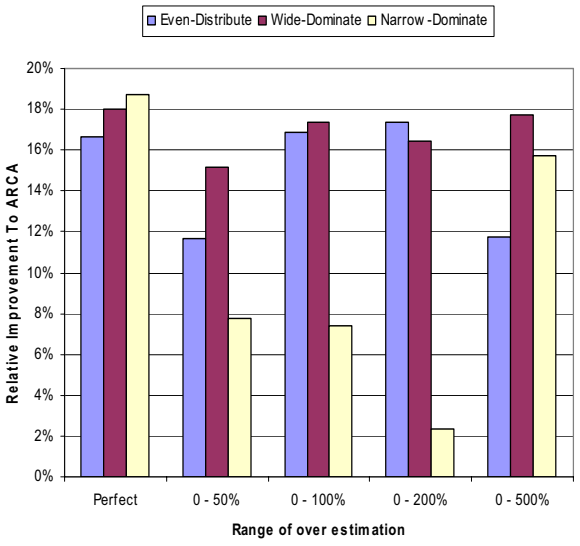


Fig. 4b. Improvement in Throughput of EASY-Backfilling (Workload Size of 10)

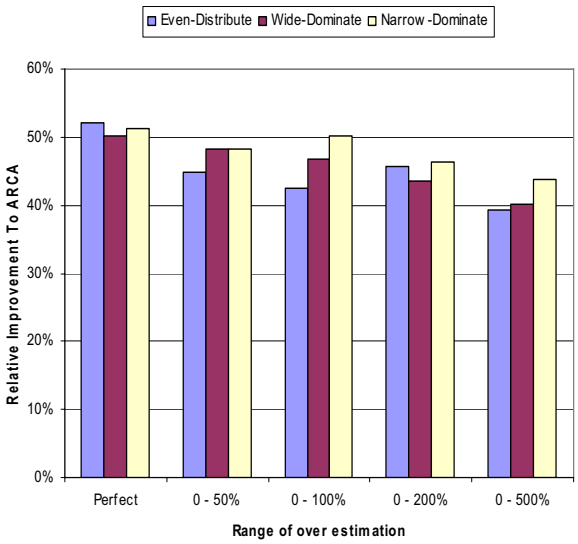


Fig. 5a. Improvement in Slowdown of EASY-Backfilling (Workload Size of 50)

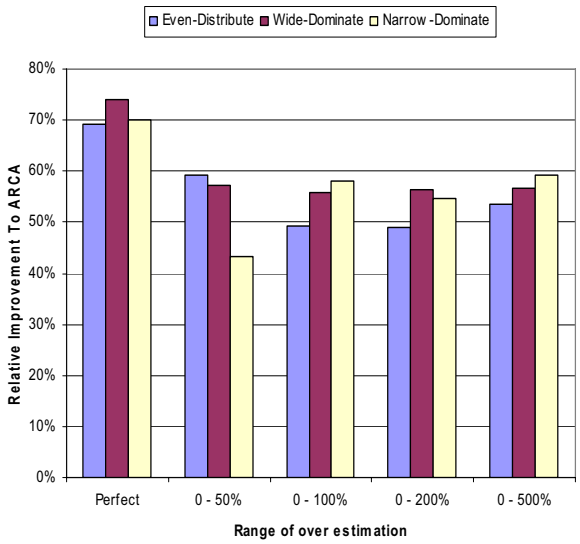


Fig. 5b. Improvement in Throughput of EASY-Backfilling (Workload Size of 50)

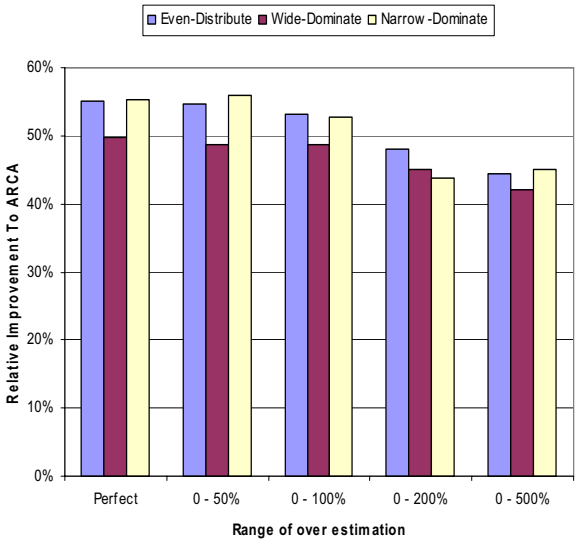


Fig. 6a. Improvement in Slowdown of EASY-Backfilling (Workload Size of 100)

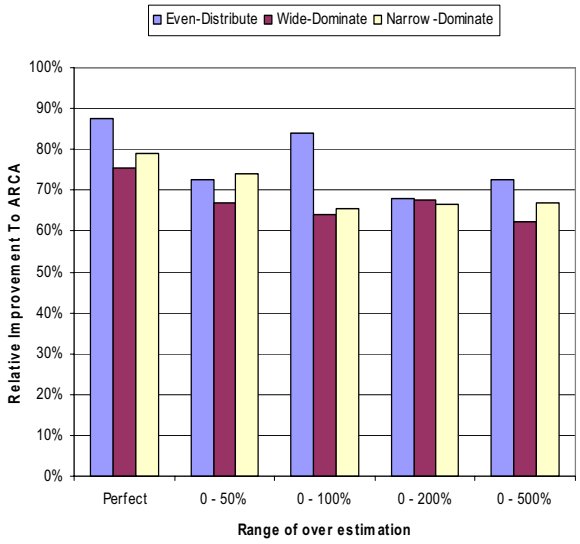


Fig. 6b. Improvement in Throughput of EASY-Backfilling (Workload Size of 100)

For each of the workload sizes: 10, 50 and 100, it is shown that EASY-Backfilling generated a smaller improvement in slowdown for the Wide-Dominated workload than for the Even-Distributed and Narrow-Dominated workloads. This observation is true only for the case where the perfect estimation of programs' execution time is considered. For the others cases where imperfect programs' estimation time is used, no such pattern can be observed. A possible explanation to this observation is that wide jobs find it difficult to backfill since they cannot find enough free computers easily. If the workload is Wide-Dominated, less narrow jobs which are waiting in the ready queue are available. This explanation is based on our experimental assumption that jobs (both wide and narrow) are randomly distributed in the ready queue.

Finally, it can also be seen that EASY-backfilling with perfect estimation outperforms any forms of EASY-backfilling with different magnitudes of over-estimation: 0-50%, 0-100%, 0-200% and 0-500% for all of the different workload sizes and most of the different workload composition types. The results for larger workload sizes of 50 and 100 also show that the improvement in slowdown decreases as the magnitude of over estimation increases. However, there is no clear trend in the improvements of throughput as the magnitude of over estimation increases and for all cases of workload composition types: Even-Distributed, Wide-Dominated and Narrow-Dominated.

VI. Conclusions and Future Work

We have improved our understanding of the backfilling job scheduling algorithm, in particular, of the most frequently used algorithm, EASY-backfilling. We have achieved this by performing its performance evaluation using static workloads of parallel jobs executed on a real computer cluster. For this purpose we have developed a batch job scheduler for Linux clusters, implemented several scheduling algorithms including ARCA and the EASY-Backfilling, and carried out their performance evaluation by running well known MPI applications on a cluster. The results we have achieved can be used not only to validate the simulation results but also to alleviate the problems found in most simulation studies. Thus, we have achieved the aim of this research.

In term of the analysis of our performance evaluation, we learned that EASY-backfilling (and the variants) outperforms ARCA in both of the slowdown and throughput; and the improvement increases as the workload increases. We observed that EASY-Backfilling generated a smaller improvement in slowdown for the Wide-Dominated workload than for the Even-Distributed and Narrow-Dominated workloads and that is true only for the case where the perfect estimation of programs' execution time is considered. We also observed that EASY-backfilling with perfect estimation outperforms any form of EASY-backfilling with different magnitudes of over-estimation for all of the different workload sizes and most of the different workload composition types. However, we discovered that studying the impact of workload compositions, i.e., Even-Distributed, Wide-Dominated and Narrow-Dominated on performance, is a non-trivial task. We believe that more experiments with a wider set of job compositions types should be carried out.

Our on-going work includes the porting of our job scheduler to the Java Virtual Machine such that most non-dedicated clusters which run the Windows OS can be utilized; the design and development of new parallel job scheduling algorithms for clusters in the direction of dynamic space sharing as well as the development of an on-line job scheduler for clusters and grids.

Acknowledgement

The authors wish to express their gratitude to Mr Michael Brock and the reviewers of Cluster 2007 for their constructive comments.

Reference

[1] B. Bode, D. M. Halstead, W. Hall and D. Jackson. The Portable Batch Scheduler and the Maui Scheduler on Linux Cluster. In Proceedings of the 4th Linux Showcase and Conference, Atlanta, GA, October, 2000.

[2] N. Capit, C. D. Costa and Y. Georgiou et al. A Batch Scheduler with High Level Components. In Proceedings of the IEEE Int'l Symposium on Cluster Computing and the Grid, pages 776-783, May, 2005.

[3] S. Chiang, A. Arpaci-Dusseau and M.K. Vernon. The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance. Lecture Notes in Computer Science – Workshop on Job Scheduling Strategies for Parallel Processing, Springer, No. 2537, pages 103-127, 2002.

[4] S. Chiang and C. Fu. Re-evaluating Reservation Policies for Backfill Scheduling on Parallel Systems. In Proceedings of the 16th IASTED Int'l Conf. on Parallel and Distributed Computing and Systems (PDCS), Cambridge, MA, Nov. 2004.

[5] D.G. Feitelson and L. Rudolph. Parallel Job Scheduling – A Status Report. Lecture Notes in Computer Science – Workshop on Job Scheduling Strategies for Parallel Processing, Springer, No. 3277, pages 1-16, 2004.

[6] M. Hanzich, P. Hernandez, E. Luque, F. Gine and F. Solsona. 3Dbackfilling: A Space-Sharing Approach for Non-Dedicated Clusters. In Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems, Phoenix AZ, page 131-138, November, 2005.

[7] S. Iqbal, Y. Fang, K. Priddy and B. Bryce. Workload Management and Job Scheduling on Platform Rocks Cluster. Dell Power Solutions, November, pages 35-39, 2005.

[8] S. Iqbal, R. Gupta and Y. Fang. Planing Considerations for Job Scheduling in HPC Clusters. Dell Power Solutions, February, pages 133-136, 2005.

[9] R. Jain. The Art of Computer Systems Performance Analysis. John Wiley & Sons, 1991.

[10] J.P. Jones and B. Nitzberg. Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization. Lecture Notes in Computer Science – Workshop on Job Scheduling Strategies for Parallel Processing, Springer, No. 1659, pages 1-16, 1999.

[11] S. Kannan, M. Roberts, P. Mayes, D. Brelsford and J.F. Skovira. Workload Management with LoadLeveler. IBM, November, 2001. URL: http://www.redbooks.ibm.com, last access: March, 2007.

[12] H.D. Karatza. A Simulation Model of Backfilling and I/O Scheduling in a Partitionable Parallel System. In Proceedings of the 32nd Conference on Winter Simulation, Orlando, Florida, pages 496-505, 2000.

[13] LAM/MPI. URL: http://www.lam-mpi.org, last access: March, 2007.

[14] D. Lifka. The ANL/IBM SP Scheduling System. Lecture Notes in Computer Science – Workshop on Job Scheduling Strategies for Parallel Processing, Springer, No. 949, pages 295-303, 1995.

[15] V. Lo, J. Mache and K. Windisch. A Comparative Study of Real Workload Traces and Synthetic Workload models for Parallel Job Scheduling. Lecture Notes in Computer Science – the Fourth Workshop on Job Scheduling Strategies for Parallel Processing, Springer, No. 1459, pages 25-46, 1998.

[16] Moab Cluster Suite and Maui Cluster Scheduler. URL: http://www.clusterresources.com, last access: March, 2007.

[17] A.W. Mu'alem and D.G. Feitelson. Utilization, Predictability, Workloads and User Runtime Estimates in Scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel and Distributed Systems, Vol. 12, No. 6, pages 529-543, June, 2001.

[18] NAS Parallel Benchmarks. URL:http://www.nas.nasa.gov/Software/NPB/, last access: March, 2007.

[19] Parallel Workload Archive by Dror Feitelson. URL: http://www.cs.huji.ac.il/labs/parallel/workload/, last access: March 2007.

[20] Platform Computing Inc. Platform LSF. URL: http://www.platform.com/Products/Platform.LSF.Family/, last access: March, 2007.

[21] Portable Batch System. URL: http://www.openpbs.org/, last access: March, 2007.

[22] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Selective reservation strategies for backfill job scheduling. Lecture Notes in Computer Science – Job Scheduling Strategies for Parallel Processing, Springer, No. 2537, pages 55-77, 2002.

[23] Sun Microsystems Inc. Sun Grid Engine. URL: http://gridengine.sunsource.net/, March, 2007.

[24] R.F. Van der Wijngaart. The NAS Parallel Benchmarks 2.4. NAS Technical Report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, 1995.

[25] A.K.L. Wong and A.M. Goscinski. Execution Environments and Benchmarks for the Study of Applications' Scheduling on Clusters. Lecture Notes in Computer Science – Distributed and Parallel Computing, Springer, No. 3719, pp. 204-213, 2005.