Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy

William A. Ward, Jr., Carrie L. Mahood, and John E. West
Computer Sciences Corporation
Attention: CEERD-IH-C, U.S. Army Engineer Research and Development Center
Major Shared Resource Center, 3909 Halls Ferry Road
Vicksburg, Mississippi 39180, USA
{William.A.Ward.Jr, Carrie.L.Mahood, John.E.West}@erdc.usace.army.mil
http://www.erdc.hpc.mil

Abstract

Backfill is a technique in which lower priority jobs requiring fewer resources are initiated before one or more currently waiting higher priority jobs requiring as yet unavailable resources. Processors are frequently the resource involved and the purpose of backfilling is to increase system utilization and reduce average wait time. Generally, a scheduler backfills when the user-specified runtimes indicate that executing the lower priority jobs will not delay the anticipated initiation of the higher priority jobs. This paper explores the possibility of using a relaxed backfill strategy in which the lower priority jobs are initiated as long as they do not delay the highest priority job too much. A simulator was developed to model this approach; it uses a parameter ω to control the length of the acceptable delay as a factor of the user-estimated run time. Experiments were performed for ω = 0, 1, 2, 3, and ∞ with both user-estimated run time and actual run time using workload data from two parallel systems, a Cray T3E and an SGI Origin 3800. For these workloads, queue wait time is typically shortest for $\omega = \infty$ and the effect of poor user run time estimates is relatively small. More experiments must be performed to determine the generality of these results.

1 Scheduling Policies

Many practical job scheduling policies, whether for uniprocessor or multiprocessor systems, incorporate the notion of job priority. Perhaps the simplest example of a priority scheme is setting a job's priority to elapsed time in the queue; if this priority scheme is used to dictate the order of job initiation, then a "first-come, first-served" (FCFS) policy results. Other, more elaborate schemes based on the number of processors requested and user estimates of runtime are, of course, possible. A second important concept involves how to use the resulting prioritized list of jobs. If, when a job completes, the prioritized list is searched for the first job that will run using the available number of processors, then a "first-fit, first-served" (FFFS) policy results; there are also "best-fit, first-served" (BFFS) (to fit the available number of processors the tightest) and "worst-fit, first-served" (WFFS) (to fit the most jobs) versions of this policy. Performance of these and other policies has been discussed in[1, 5, 10].

Another approach to using the resulting prioritized job list is to treat the highest priority job as the one that must execute next, and then save resources as other jobs complete until there are sufficient resources to run that job. Depending on the workload, this may lead to underutilization of the system, increased wait times, and also to job "starvation," where a job is waiting to execute but is never initiated[4, p. 38]. A partial solution to this is to execute lower priority jobs, but then pre-empt them when necessary to execute the highest priority job. This is implemented by virtually all operating systems on uniprocessors and tightly-coupled multiprocessors; very often the job will remain memory resident while pre-empted so that it may be easily restarted at the next time quantum. Implementing this capability on a large-scale parallel system requires a job checkpointing capability; some parallel systems, e.g., the Cray T3E, have it while others, particularly cluster systems, do not. Even on those that do, it is sometimes undesirable to operate the system in that mode because of the resources wasted by frequently changing from one job to another. In fact, on highly parallel systems where the processors required to run the job are the critical resource, the scheduler will often allocate processors to a job for the job's lifetime and simply allow the job to run to completion, never pre-empting it[12].

This is known as "variable partitioning"[3] or "pure space sharing"[1].

A second way to resolve this issue is to allow the scheduler to use "backfilling," a technique in which lower priority jobs requiring fewer resources are initiated before one or more currently waiting higher priority jobs requiring as yet unavailable resources[6, 11]. Here, users supply a number of processors required to run their jobs along with an estimate of the time required. Based on this information, lower priority jobs are allocated idle processors and allowed to run as long as they complete before the earliest feasible time for running the highest priority job. Apparently, this scheme depends on user time estimates for its effectiveness. However, there is some evidence that poor estimates do not adversely affect overall performance of this scheduling policy, although individual jobs may be delayed[7, p. 91][13, pp. 140-141]. Generally, use of backfill significantly improves system utilization and reduces job wait time versus the same scheduling policy without backfill. Not all scheduling policies are amenable to the use of backfill; the strategy that repeatedly dispatches the highest priority job that fits until no more jobs can be started is an example.

2 Backfill Variants

There are two basic approaches to backfill: conservative backfill, which allows a lower priority job to run only if it will not delay *any* of the higher priority waiting jobs, and aggressive backfill, which will allows a lower priority job to run if it will not delay the highest priority job[8]. Regardless of which approach is used, there are further subvariants distinguished by how the backfilled job is chosen. Possibilities include (i) selecting the highest priority job that will fit, (ii) selecting the job that fits the best, and (iii) selecting a combination of jobs that fit the best. All three of these approaches are essentially implementations of a greedy strategy that "makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution"[2, p. 329]. Optimal performance is not achievable since it would require knowledge of jobs yet to be submitted.

Obviously the interpretation of "fit" allows for variations. An easy approach considers only the number of processors requested; however, this may not be optimal in terms of system utilization. For example, suppose two jobs requesting the same number of processors may be backfilled, and the first job has a higher priority and a lower time estimate than the second. If, after running the first job, no other jobs may be backfilled, then system utilization over that period will be lower. If the number of processors over time is treated as a two-dimensional space to be filled with jobs, then packing algorithms that consider all waiting jobs as candidates may be applied.

This notion of backfill fit has been extended by Talby and Feitelson in their concept of "slack-based backfilling"[12]. In this approach, three parameters – a job's individual priority, a tunable systemwide "slack factor," and the system average wait time – are used to compute a job's "slack." The system slack factor is used to control how long jobs will have to wait for execution; e.g., a slack factor of 3 implies that jobs may wait up to 3 times the average wait time. Once priorities and slacks have been calculated for all waiting jobs, then it is possible to compute a cost for a particular schedule of these jobs. Selecting the least costly schedule from all possible schedules is analogous to the knapsack problem[9, pp. 3, 261] and is, as noted in [12], an NP-hard problem. Talby and Feitelson provide several heuristics to reduce the search space of candidate schedules, and then use a simulator implementation of their method to demonstrate its effectiveness.

A different perspective on selecting backfill candidates is found in [14] and [15]; there, the authors contrast the use of accurate runtime estimates in backfill algorithms with user-supplied overestimates. A parameter R is used to specify the overestimate, with $R = 1$ corresponding to the actual runtime. Their tests included two actual and one artificial workload traces. Significantly, they observed decreasing average wait time with increasing R, particularly for the actual workloads. (For convenience, this will be referred to as the "ZK" method after the last names of the authors in [14].)

Instead of using the standard aggressive backfill criterion, this paper proposes a "relaxed" backfill technique in which jobs are backfilled as long as they do not delay the highest priority job "too much." This approach is similar in spirit to the slack-based approach in that the highest priority job may not be scheduled at the earliest possible time, and similar in technique to the ZK method in that it increases the size of the backfill time window. A tunable system parameter, ω, expressing the allowable delay as a factor of the user-specified runtime of the highest priority job, controls the degree of backfill relaxation used by the scheduler. For example, if $\omega = 1.2$, then a job may be backfilled as long as it does not cause a delay greater than 20 percent of the highest priority job's user-specified runtime. This relaxed backfill approach is illustrated in Fig. 1.

Increasing ω has an effect similar to increasing R in the ZK approach in that it increases the backfill time window in which jobs may be scheduled. Setting the relaxation parameter to ∞ allows several interesting cases, depending on the priority calculation used; this will be discussed below. Other variants include making ω relative to CPU hours requested or to priority instead of runtime. For example, in the latter case, a job would be eligible for backfilling if there were sufficient processors to run it and if its priority when multiplied by ω was greater than the priority of the highest priority job.

3 Job Priority

As has already been noted, use of backfill requires some notion of job priority, since that determines the highest priority job for which resources are being reserved and affects the choice of backfilled jobs. Intuitively, as a job waits longer, its priority should increase; e.g., the priority calculation should include a factor such as $(t - t_q)^\alpha$, where t is the current time and t_q is the time the job was queued. Next, although this is somewhat arbitrary, jobs with shorter user time estimates may be favored over ones estimated to run longer, corresponding to a factor of w^β, $\beta < 0$, where w is the user-estimated walltime for the job.

Further, the number of processors may be taken into account in calculating job priority. In a similar spirit to the estimated walltime, one might assign a higher priority to a job requesting fewer processors. However, running jobs requiring only one or a few processors is often considered not the best use of expensive, highly parallel systems, and so one might be led to the opposite conclusion and assign a lower priority to such a job. When selecting a backfill candidate, this latter alternative has the attractive property of tending to favor jobs that fit the backfill space tighter. This recommends inclusion of the factor n^γ, where n is the user-requested number of processors. Note than even with a low priority, a job requiring few processors still makes a good backfill candidate and tends to be initiated relatively quickly.

Finally, the local queue structure may also be reflected in the priority calculation; e.g., a site having four queues – background, primary, challenge, and urgent – might assign factors of 1, 2, 4, and 8, respectively, to reflect the increasing importance of jobs in those categories. More generally, this factor could be of the form r^δ, where r represents the relative importance of the queues and δ is the queue number.

Suppose time is measured in seconds, a typical job uses 32 processors, and that the queues are numbered as follows: background: -1, primary: 0, challenge: 1, and urgent: 2. Then a possible job priority calculation is of the form

$$P = \left(\frac{t - t_q}{3600}\right)^\alpha \left(\frac{w}{3600}\right)^\beta \left(\frac{n}{32}\right)^\gamma r^\delta. \qquad (1)$$

Thus, a typical 32-processor job having an estimated 1-hour runtime that has waited in the primary queue for 1 hour has a current priority of 1.

Coupled with the backfill scheme described above, several policies are possible. If $\alpha = 1$ and $\beta = \gamma = \delta = \omega = 0$, then an FCFS algorithm results. $\alpha = 1$, $\beta = \gamma = \delta = 0$, and $\omega = \infty$ yields an FFFS policy. If $\alpha = \beta = \delta = 0$ and $\omega = \infty$, then $\gamma = 1$ produces a BFFS policy, while $\gamma = -1$ produces a WFFS policy. Setting α, β, and δ to nonzero values specifies a family of interesting subvariants of these policies. Furthermore, the tunability of the parameters allows for emphasis of job aspects that are most important for a particular system.

4 Simulation Results

A scheduler simulator written in Perl was used to compare the effectiveness of the proposed approach with several other scheduling policies. The algorithm used in the simulator is shown in Fig. 2. The key element of this algorithm is the backfill window calculation. If $t + hpwait$ is the earliest time that the highest priority job could run, then the backfill window is $bfwindow = \omega * hpwait$ and backfill candidates are jobs for which there are enough CPUs and which will complete before $t + bfwindow$. The priority calculation in Eq. 1 with $\alpha = 1$, $\beta = -1$, $\gamma = 1$, and $r = 10$ was used to assign job priorities. In particular, the $\left(\frac{n}{32}\right)^\gamma$ term places jobs using larger numbers of processors higher in the priority list; this tends to somewhat counteract the tendency to backfill with numerous small jobs. Finally, priorities for waiting jobs were recalculated once every simulated minute.

Utilization data from two systems at the U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC) were used in this study. The first machine is a Cray T3E currently running version 2.0.5.49 of the UnicosMK operating system (OS) and version 2.1p14 of the OpenPBS scheduler; the scheduler has been highly customized for use at the ERDC MSRC. The T3E was initially configured with 512 600-MHz Alpha processors, each with 256 Mb of memory. During 5-15 August 2001, part of the period under study, this system was out of service while it was reconfigured to include an additional 256 675-MHz Alpha processors, each with 512 Mb of memory. This change in number of processors is reflected in this study. These processors are termed "application processing elements" (PEs). There are additional "command PEs" available for interactive login and "OS PEs" for OS processes, but because they are not part of the processor pool available to the scheduler, they are not included in this study.

The second machine is an SGI Origin 3800 (O3K) currently running version 6.5 of the IRIX OS and version 5.1.4C of the PBS Pro scheduler. The O3K has 512 400-MHz MIPS R12000 processors grouped into 4-processor nodes; each node has 4 Gb of memory. On 2 February 2002, this system was reconfigured to implement cpusets with the net effect being to reduce the number of available processors from 512 to 504, with 8 processors being reserved for the OS. This change is also reflected in the study.

Ordered from lowest to highest priority, there are "background," "primary," "challenge," and "urgent" queues on both systems. A guiding operational tenet is to give jobs in these latter two categories premium service. Additionally, there is a "dedicated" queue that allows a particular group of users exclusive access to the system. In preparation for running in dedicated mode, the scheduler stops submitting jobs from the other queues and running jobs are allowed to complete. Then, while in dedicated mode, only jobs in the dedicated queue are run. At the end of this period, the other queues are restarted. While in this mode a system is essentially operating outside of the normal scheduling regime because only jobs in one particular queue may run and other jobs, regardless of their priority and of system resource availability, may not. Consequently, 80 out of 66,038 jobs on the T3E and 362 out of 71,500 jobs on the O3K run in this mode were eliminated from the study.

This left 65,958 jobs run on the T3E between 10 October 1999 and 10 April 2002 and 71,138 jobs run on the O3K between 13 November 2000 and 9 April 2002; these were used as input data to the simulation study. Five different backfill relaxation levels were studied using the run logs from both systems: ω = 0 (no backfill), 1 (aggressive backfill), 2, 4, and ∞. For each of the two systems and for each of the five ω levels, two scenarios were simulated: one using the user-estimated runtime, and one using the actual runtime. In each case the runtime (estimated or actual) was used in *both* the priority calculation (Eq. 1) and the backfill algorithm. In each of these 20 runs, total job wait time was computed by month and compared against actual wait time. The results are presented in Tables 1-4. Each simulation run covered the full time period noted above for each system, but only results for the period 1 April 2001 to 31 March 2002 are shown; the results for months not shown are quite similar in character.

As expected, scheduling without backfill performs the poorest. With few exceptions, as the backfill criterion is relaxed wait time decreases. More specifically, considering the T3E results based on user estimated runtimes in Table 1, if a value of $\omega = \infty$ had been preselected, then relaxed backfill would have shown a reduction versus actual wait time for every month, and the average monthly decrease in wait time would have been 46 percent. Versus standard aggressive backfill, there would also have been a reduction for every month with an average monthly decrease of 67 percent. Furthermore, relaxed backfill is an improvement over standard aggressive backfill as implemented here for every $\omega > 1$ tested. Obviously, the observed reduction in wait time must be achieved by increasing the number of jobs backfilled. This is shown in Table 2 where, again considering the results based on user estimates, the number of jobs backfilled consistently increases for increasing values of ω for every month.

An possible drawback to relaxed backfilling is that as the scheduler is given a larger pool of backfill candidates from which to choose, lower priority are scheduled earlier and the priority scheme may be undermined. More specifically, it is possible that the observed improvement in overall wait time may have come at the expense of high priority jobs. Table 3 shows that this is not the case. The systems at ERDC have four queues, background, primary, challenge, and urgent. With the exception of April 2001, the wait times for challenge and urgent jobs decrease for increasing values of ω and the times for the $\omega = \infty$ case are less than the actual wait times.

Remarkably, these improvements were obtained in spite of consistently poor user runtime estimates; the average ratio of actual runtime to user-estimated runtime on the T3E for this period is 29 percent. Another measure of the approach's robustness is its performance relative to that obtainable using exact runtimes. Referring again to Table 1 and considering the $\omega = \infty$ case, the monthly wait times obtained using user estimates are actually less than those obtained using exact runtimes for 6 of the 12 months shown.

The method's performance on the O3K, as shown in Tables 4 and 5, is similar. Considering the results obtained using user-estimated runtimes and a value of $\omega = \infty$, relaxed backfill would have outperformed both the actual scheduler and aggressive backfill every month with average monthly decreases in wait time of 87 and 62 percent, respectively. Again, these improvements were obtained in spite of user-estimated runtimes being high by an average factor of 4, and were an average of only 9 percent more than what would have been achievable using actual runtimes. Table 6 shows, as was the case with the T3E, that decreased wait times do not come at the expense of high priority jobs.

5 Conclusions

This study indicates that accurate user run time estimates are not necessary for a backfill scheme to be effective and that use of exact run times is not always superior to coarse estimates. Although the formulation of this approach is somewhat different from the ZK methodology, the results here support similar findings in [14] and [15] in the sense that as the size of the backfill time window is increased, more jobs are backfilled and overall wait time decreases. More specifically, the ZK parameter R is analogous in function to the ω used here.

Relaxed backfill with $\omega = \infty$ is equivalent to a variant of some first-, best-, or worst-, fit policy. Thus, all of these approaches are part of a larger family of methods and are obtainable by selecting appropriate values for ω and for the parameters in the priority scheme. It is possible that this parameter space could form the basis for a taxonomy of scheduling policies.

The results gathered here indicate that if relaxed backfill is allowable from a policy perspective, then it (particularly the $\omega = \infty$ version) may be quite effective in reducing total job wait time versus aggressive backfill. Although the time span and size of the input data, as well as the use of data from two different systems, give credence to this conclusion, further study is necessary to provide additional confirmation of the method's applicability. This would include use of data from other systems or other user populations, a different priority assignment scheme, a different backfill scheme, or use of ω values relative to processor hours or priority instead of hours.

More specifically, assuming the technique is widely applicable, sensitivity analyses should be conducted to determine how the method behaves for various priority parameters and to determine the best settings for a given system and workload. An important aspect of this would also be to determine if the $\omega = \infty$ version is superior for all priority schemes. Finally, an appropriate modification to this approach should be developed to guarantee that jobs do not wait indefinitely.

6 Acknowledgments and Disclaimer

The authors gratefully acknowledge the inspiration for this approach provided by Dr. Daniel Duffy, Computer Sciences Corporation (CSC), ERDC MSRC. This work was supported by the U.S. Department of Defense (DoD) High Performance Computing Modernization Program through the ERDC MSRC under contract number DAHC94-96-C-0002 with CSC.

The findings of this article are not to be construed as an official DoD position unless so designated by other authorized documents. Citation of trademarks herein does not constitute an official endorsement or approval of the use of such commercial products, nor is it intended to infringe in any way on the rights of the trademark holder.

References

[1] K. Aida, H. Kasahara, and S. Narita. Job scheduling scheme for pure space sharing among rigid jobs. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 98–121, Berlin Heidelberg New York, 1998. Springer-Verlag.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[3] D. G. Feitelson. A survey of scheduling in multiprogrammed parallel systems. Research Report RC 19790 (87657), IBM T.J. Watson Research Center, October 1994.

[4] R. Finkel. *An Operating System Vade Mecum*. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[5] R. Gibbons. A historical application profiler for use by parallel schedulers. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 58–77, Berlin Heidelberg New York, 1997. Springer-Verlag.

[6] Intel Corp. *iPSC/860 Multi-User Accounting, Control, and Scheduling Utilities Manual*, May 1992. Order Number 312261-002.

[7] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the Maui scheduler. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 87–102, Berlin Heidelberg New York, 2001. Springer-Verlag.

[8] D. A. Lifka. The ANL/IBM SP scheduling system. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 295–303, Berlin Heidelberg New York, 1995. Springer-Verlag.

[9] B. M. E. Moret and H. D. Shapiro. *Algorithms from P to NP*. Benjamin/Cummings, Redwood City, California, 1991.

[10] E. W. Parsons and K. C. Sevcik. Implementing multiprocessor scheduling disciplines. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 166–192, Berlin Heidelberg New York, 1997. Springer-Verlag.

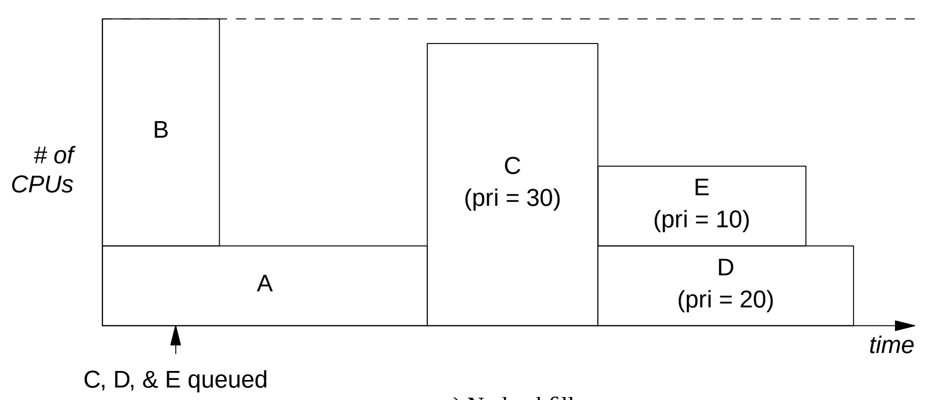
[11] D. D. Sharma and D. K. Pradhan. Job scheduling in mesh multicomputers. In *Intl. Conf. Parallel Processing*, volume II, pages 1–18, August 1994.

[12] D. Talby and D. G. Feitelson. Supporting priorities and improving utilization of the IBM SP2 scheduler using slack-based backfilling. In *13th Intl. Parallel Processing Symp.*, pages 513–517, April 1999.

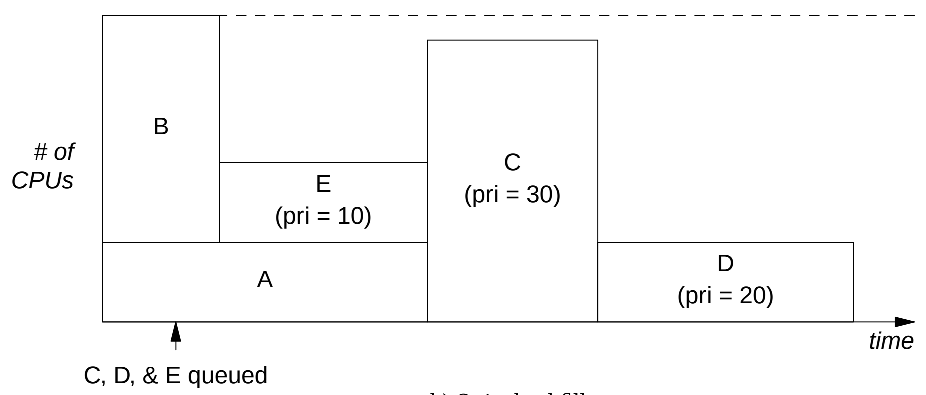
[13] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramanian. An integrated approach to parallel scheduling using gang-scheduling, backfill, and migration. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 133–158, Berlin Heidelberg New York, 2001. Springer-Verlag.

[14] D. Zotkin and P. J. Keleher. Job-length estimation and performance in backfilling schedulers. In *8th High Performance Distributed Computing Conf.* IEEE, 1999.

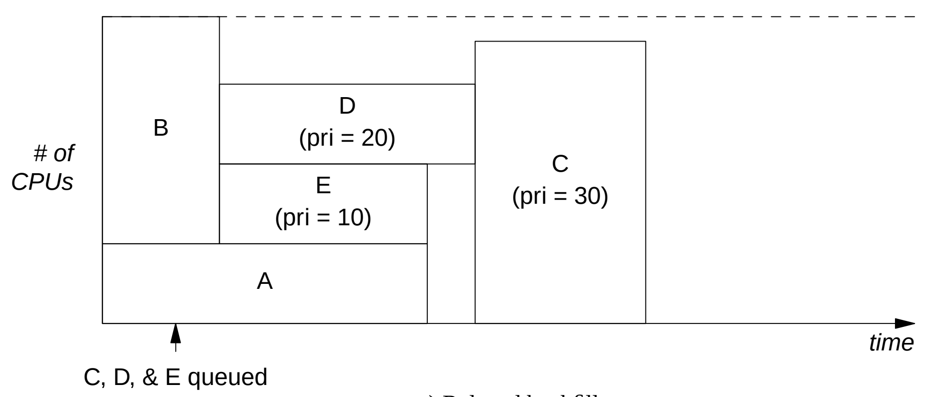
[15] D. Zotkin, P. J. Keleher, and D. Perkovic. Attacking the bottlenecks of backfilling schedulers. *Cluster Computing*, 3(4):245–254, 2000.



a) No backfill



b) Strict backfill



c) Relaxed backfill

Figure 1. Schedule for running five jobs under three scheduling policies

```
SPECIFY number of CPUs in model
SPECIFY queues (e.g., background, primary, challenge)
SPECIFY dt (time interval for checking queue status)
SPECIFY alpha, beta, gamma, delta, and omega for priority calculation
SET t = 0
SET twake = 0

WHILE ( more input OR jobs in ready queue OR jobs in run queue )

   IF ( t >= twake AND more input )
      READ next command
      IF ( command is "queue <job>" )
         PUT job in ready queue
      ELSE IF ( command is "sleep <tsleep>" )
         SET twake = t + tsleep
      END IF
   END IF

   REMOVE any completed jobs from run queue
   RECALCULATE priorities of jobs in ready queue
   SORT ready queue in descending order by priority

   IF ( any jobs added to ready queue OR any jobs completed )
      WHILE ( enough CPUs )
         START highest priority job (top of ready queue)
      END WHILE
      SET hpwait = time until highest priority job can run
      SET bfwindow = omega * hpwait
      WHILE ( more jobs in ready queue to check )
         IF ( enough CPUs AND job will complete before t+bfwindow )
            START backfill job
         END IF
      END WHILE
   END IF

   SET t = t + dt
END WHILE
```

Figure 2. Simulator algorithm for relaxed backfill

Table 1. Total wait times in hours by month under various scheduling policies using workload data from the ERDC MSRC T3E. "Est" identifies results derived using user-estimated runtimes for both the priority calculation and for backfilling. "Act" identifies results derived using actual runtimes for both the priority calculation and for backfilling

	Percent	Actual	Time	Simulated Wait Time				
Month	Util.	Wait	Used	ω=0	ω=1	ω=2	ω=4	$\omega=\infty$
2001/04	72.2	3494	Est	58735	4491	2430	1582	3050
			Act	40165	2104	1450	1351	1511
2001/05	76.4	3007	Est	22976	6337	3945	2317	1952
			Act	13432	3862	2871	2113	2057
2001/06	81.4	3224	Est	14651	5284	4003	2841	2046
			Act	8850	3946	3304	2993	2828
2001/07	63.9	2856	Est	9611	3373	2161	1551	1175
			Act	6365	2150	1646	1366	1079
2001/08	66.1	1174	Est	18141	2386	1626	1159	787
			Act	10144	1042	849	789	776
2001/09	77.8	2817	Est	68851	16222	8578	3461	2232
			Act	39327	4663	3506	3199	2857
2001/10	58.0	1403	Est	8898	2254	1436	895	481
			Act	6667	1030	781	602	680
2001/11	68.7	2282	Est	14664	4073	2747	1950	1336
			Act	9390	2073	1494	1585	1116
2001/12	62.8	2088	Est	9339	2573	1527	1108	1204
			Act	4981	1661	1084	1050	915
2002/01	61.2	43366	Est	5365	2399	1141	859	563
			Act	3425	1503	773	584	551
2002/02	72.9	4265	Est	24254	5354	3373	2488	1456
			Act	11032	2753	1963	1704	1686
2002/03	74.9	3926	Est	33039	7767	5018	4017	2257
			Act	18719	7773	4711	3331	2956
Total	69.2	73902	Est	288524	62513	37985	24228	18539
			Act	172497	34560	24432	20667	19012

Table 2. Number of JOBS backfilled for selected months under various scheduling policies using workload data from the ERDC MSRC T3E. "Est" identifies results derived using user-estimated runtimes for both the priority calculation and for backfilling. "Act" identifies results derived using actual runtimes for both the priority calculation and for backfilling

	No.	Time	Jobs Backfilled			
Month	Jobs	Used	ω=1	ω=2	ω=4	ω=∞
2001/05	2607	Est	1054	1241	1606	2582
		Act	982	1087	1418	2576
2001/07	2144	Est	425	617	758	966
		Act	390	453	627	914
2001/09	2798	Est	1420	1837	2106	2192
		Act	1710	1756	1894	2188
2001/11	2226	Est	492	631	943	1184
		Act	611	661	757	1183
2002/01	4140	Est	282	463	530	695
		Act	333	417	496	680
2002/03	2768	Est	788	1082	1218	1552
		Act	788	868	1064	1422

Table 3. Total wait times in hours by month for challenge/urgent jobs under various scheduling policies using workload data from the ERDC MSRC T3E. These results were derived using user-estimated runtimes for the priority calculation and for backfilling.

	No.	Actual	Simulated Wait Time				
Month	Jobs	Wait	ω=0	ω=1	ω=2	ω=4	ω=∞
2001/04	1108	1845	21791	2939	1678	1190	2558
2001/05	1588	1921	11614	3686	2460	1428	1266
2001/06	600	1006	2853	2160	1851	1197	591
2001/07	806	1142	1601	737	510	398	222
2001/08	285	296	1142	447	339	298	200
2001/09	381	659	3493	1444	959	746	580
2001/10	280	667	1205	647	490	304	276
2001/11	448	741	2298	1311	1184	616	428
2001/12	593	797	1893	841	585	423	264
2002/01	656	496	893	655	398	301	134
2002/02	459	424	1821	1589	1408	1005	274
2002/03	446	891	3277	1478	1401	962	573
Total	7650	10885	53881	17934	13269	8868	7366

Table 4. Total wait times in hours by month under various scheduling policies using workload data from the ERDC MSRC O3K. "Est" identifies results derived using user-estimated runtimes for both the priority calculation and for backfilling. "Act" identifies results derived using actual runtimes for both the priority calculation and for backfilling

	Percent	Actual	Time	Simulated Wait Time				
Month	Util.	Wait	Used	ω=0	ω=1	ω=2	ω=4	$\omega=\infty$
2001/04	46.1	13755	Est	581	234	102	103	96
			Act	495	195	169	163	103
2001/05	34.4	3793	Est	76	60	56	32	24
			Act	49	34	28	26	24
2001/06	74.6	3408	Est	5844	2935	3550	1463	1043
			Act	3058	1322	1123	899	960
2001/07	42.8	2441	Est	33	32	24	22	21
			Act	32	31	31	31	21
2001/08	89.8	6351	Est	89345	15544	13575	5493	3868
			Act	38697	6204	4263	3956	3479
2001/09	71.3	15179	Est	120081	12244	16405	2231	1788
			Act	90146	4055	2426	1420	1622
2001/10	71.9	96093	Est	54436	24812	11727	1378	1006
			Act	27581	5866	2108	1488	896
2001/11	73.7	11347	Est	6850	2085	1797	1613	1144
			Act	4676	2045	1584	1415	941
2001/12	66.2	6690	Est	332	205	187	125	113
			Act	250	158	159	149	103
2002/01	78.5	11196	Est	100232	7013	4050	3444	2710
			Act	54634	4401	3855	2806	2602
2002/02	78.1	24051	Est	13003	6107	5316	3991	2788
			Act	8853	3704	3394	3154	2270
2002/03	62.9	25274	Est	4283	2566	2332	1755	954
			Act	3120	1156	965	822	859
Total	65.7	219578	Est	395096	73837	59121	21650	15555
			Act	231291	29171	20105	16329	13880

Table 5. Number of JOBS backfilled by month under various scheduling policies using workload data from the ERDC MSRC O3K. "Est" identifies results derived using user-estimated runtimes for both the priority calculation and for backfilling. "Act" identifies results derived using actual runtimes for both the priority calculation and for backfilling

	No.	Time	Jobs Backfilled			
Month	Jobs	Used	ω=1	ω=2	ω=4	$\omega=\infty$
2001/05	6732	Est	19	30	57	83
		Act	33	56	68	83
2001/07	4990	Est	4	7	10	10
		Act	6	6	6	10
2001/09	3713	Est	1329	2183	3710	3710
		Act	936	2938	2902	3709
2001/11	1992	Est	375	464	467	547
		Act	335	385	422	528
2002/01	4280	Est	2182	2358	2602	3262
		Act	1942	2341	2686	3232
2002/03	4834	Est	581	733	954	2073
		Act	524	593	754	2044

Table 6. Total wait times in hours by month for challenge/urgent jobs under various scheduling policies using workload data from the ERDC MSRC O3K. These results were derived using user-estimated runtimes for the priority calculation and for backfilling.

	No.	Actual	Simulated Wait Time				
Month	Jobs	Wait	ω=0	ω=1	ω=2	ω=4	$\omega=\infty$
2001/04	2396	4694	116	113	52	56	55
2001/05	421	1347	3	3	5	5	4
2001/06	236	1536	533	473	785	963	386
2001/07	282	284	2	2	3	3	3
2001/08	410	1776	22678	5573	5384	693	160
2001/09	595	5081	12276	3402	4956	111	101
2001/10	1026	4713	3903	2661	2282	372	231
2001/11	727	2571	1484	880	751	750	576
2001/12	532	952	59	56	61	45	51
2002/01	667	2022	8344	2474	1553	1430	534
2002/02	726	6921	2167	1745	1467	1290	549
2002/03	1166	8558	1519	911	1043	689	117
Total	8134	40455	53084	18293	18342	6407	2767